[S24] Reinforcement Learning. Trackmania AI racer

Vladimir Bazilevich Mikhail Rudakov

May 2024

Abstract

In this project, we train agent to drive in a 3D car racing game TrackMania. Distributed RL TMRL library and GYM environment are used. We try RL soft actor-critic (SAT) as well as basic ad-hoc training algorithms. For the underlying model, try complex RNN-based model in both LIDAR- and CNN-based environments on premade racing maps. Comparing the results, we observe that RL-based RNN approaches converge slower, they outperform simpler heuristic driving. Our code along with demos and training stats is available on github[1].

1 Introduction

Hold on tight as we merge the high-octane world of car racing with the cutting-edge capabilities of Reinforcement Learning, creating an unstoppable force on the Trackmania race tracks! ...well, not as exciting as ChatGPT puts it, yet we found an idea of training RL algorithm on a real car racing game quite fascinating!

In this project we aim to train an AI to play ***TrackMania*** 3D car racing videogame. This is an arcade racing game with a large community of players and many regimes available. Maps tend to be not only flat but also have some jumps, loops, different kinds of surfaces and daytime lighting.

Indeed, the topic of self-driving in a simulated environment is not new at all: Kiran et al. [1] provide comprehensive overview of CV and RL methods used in such scenario. Specifically, popular RL methods such as DDQN [2], REDQN [3] are actively utilized. Moreover, as balancing exploration and AI safety is a must in real world environment, careful reward shaping such as potential-based reward shaping [4] are often used. To conclude, there are a lot of advancements made in autonomous driving area, and we aim just to touch the basics safely with Trackmania AI driving task.

Our goals for the project are the following:

- Set up TrackMania AI racer environment via TMRL;

[1] `https://github.com/vBazilevich/S24-RL-project-trackmania`

- Experiment with SAC algorithm, dive into detail of the implementation
- Implement basic ad-hoc driver in LIDAR environment
- Try to improve RL models with RNNs
- Conduct training experiments and compare the results for obtained models and algorithms

2 RL Problem Statement

We first need to properly formulate the task of "driving a car" in RL terms. This section describes key components of RL problem being solved: agent and its actions, environment information available, and designed reward structure. Figure 1 outlines the components and their features. In practice, we use flexible TMRL rtgym environment.

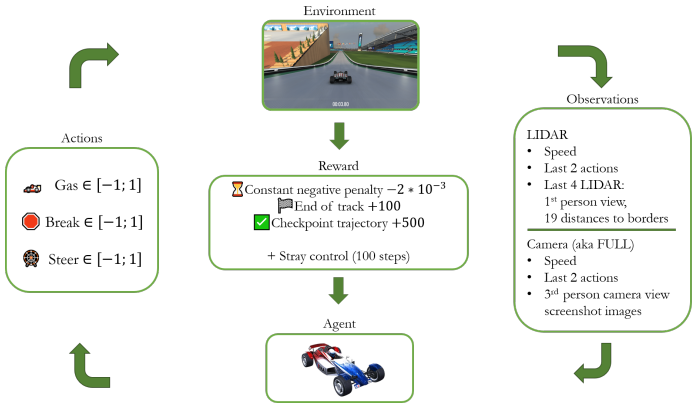


Figure 1: Trackmania RL Problem Description.

2.1 Agent

Obviously, our agent is a car in the Trackmania game. As in the real car, agent has three actions available:

- Gas: float value $gas \in [-1; 1]$, where -1 corresponds to no acceleration, and $+1$ to full acceleration;
- Break: float value $break \in [-1; 1]$, where -1 corresponds to no braking, and $+1$ to full braking;
- Steer: float value $steer \in [-1; 1]$, where -1 is full left turn, $+1$ is full right turn, and 0 is going straight.

(a) Train map (b) Test map

Figure 2: Trackmania maps environment examples

Agent can set each of these three variables. Note that due continuity of actions standard MDP-based methods are hardly applicable in our case (not to mention the track size).

2.2 Environment & Observations

Environment for the Trackmania is a game map, including racing track, start and finish gates, and the car itself. In some advanced custom maps, track may not be flat, e.g. may contain jumps, loops and dead ends. Also custom map modules contain accelerating or deccelerating panels, which additionally complicate the race. After few initial experiments we decided to stick to the plain maps, both due to agent training and computational complexity. Figure 2 presents examples of the maps we use.

We use TMRL[2] library as a connection layer between the game itself and our models and learning procedures. The library (with the use of extension OpenPlanet) interacts with the game screen and logs, retrieving necessary information. In such environment, a single step is taking action from a given position, i.e. setting gas, break and steer variables. Steps happen with 20Hz frequency, each 0.05 seconds. An episode in this environment is a sequence of agent steps until the finish is reached or car does not arrive at new checkpoints during fixed amount of steps *MAX_STRAY*.

In particular, we experiment with two types of available gym environments, going by nicknames LIDAR and FULL.

2.2.1 LIDAR environment

The core of observations in this environment type are light detection and ranging (LIDAR) observations of the map. They are collected automatically from the cockpit view (with invisible car) to the nearest black pixel borders in 19 directions. The catchy detail is that for maps with different color schemas this technique may be not stable, thus we use standard maps. The environment reports last 4 LIDAR observations, along with current speed (taken from OpenPlanet logs) and 2 last actions taken.

[2] `https://github.com/trackmania-rl/tmrl`

This environment is lightweight and relatively easy to learn from, but, of course, less generalizible and powerful.

2.2.2 FULL (Camera-based) environment

The full environment outputs the game screen directly. In our case, each image is grayscale 64x64 image (resized from original 256x128). Similarly to LIDAR, observations include last 4 screen observations, current speed and 2 previous actions.

Game screenshots give more information about the game and car position, but it takes more processing and training time, as more complex models than MLP for LIDAR are to be used (like convolutional neural networks, CNN). Lastly, CNN-based training takes at estimate x10-30 times more to train than LIDAR one. Nevertheless, we conduct CNN+RNN experiments in such an environment.

2.3 Reward Structure

Reward design in a car-racing game is not a simple $+\infty$ for finish and constant negative discount per time step - this way, agent would take ages to complete the track even once with such random evaluation.

To address the natural complexity of the environment, we use a (custom) pre-recorded trajectory as a foundation for our reward function. Trajectory is split into multiple checkpoints along the path, as in Figure 3. The reward is proportional to the number of checkpoints passed since past frame. We assume that checkpoint is passed if it's the closest one along 500 next checkpoints. However, to allow the model to go backwards we give it a smaller reward if it hasn't approached any of the next checkpoints but came closer to one of the 10 previous checkpoints. This way the model might learn to go backwards when it's necessary.

As a time discounting reward, a fixed negative low reward is applied at each time step. If model crosses finish line it gets a higher reward. If, however, model does not perform any intelligent actions for a few frames the episode is being terminated.

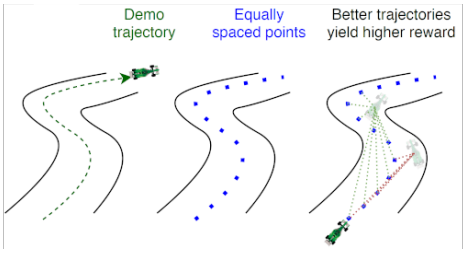


Figure 3: Track trajectory checkpoints. Source: TMRL

3 Model architecture

Due to TMRL structure we are limited in layers we could use. Arguably, one of the most useful ones which we can't use is batch normalization.

3.1 Ad-hoc Driving

As a baseline model, we tried to implement ad-hoc driving based on our human experience of driving a car. Our ad-hoc algorithm interacts with LIDAR environment. The algorithm contains several if statements for corner cases and adjusting speed and steering based on the road situation. Particularly:

- If all LIDAR distances are very high, drive straight with $gas = 1, break = 0, steer = 0$;
- If LIDAR sees an edge approaching (either from the left or right), reduce speed with linear dependence on the distance;
- If a border is too close (overwrites previous case), then start steering to the direction where the border distance is larger (turn direction), with decreasing speed.
- If distances just in front of the car are very small, we set $gas = 0, break = -1, steer = 0$ to recover from a bump;

As we can see, ad-hoc algorithm contains a lot of bare comparisons and constants, which are tuned by hand by simulations. Indeed, policy is deterministic and does not require training.

3.2 Feed forward network (FFN)

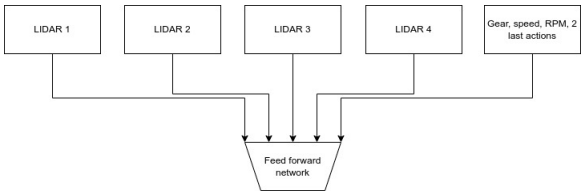


Figure 4: Feed-forward network for LIDAR environment

Simplest model: just concatenate all observations and pass them through fully-connected network.

3.3 RNN

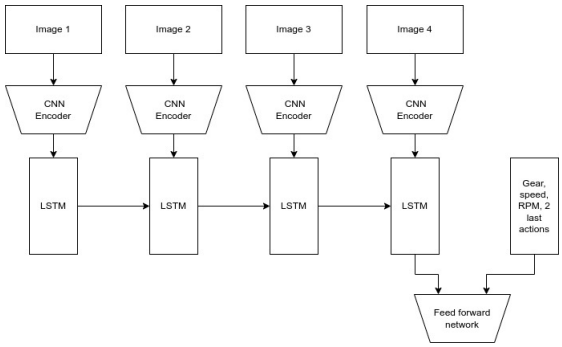


Figure 5: RNN-based model for LIDAR environment

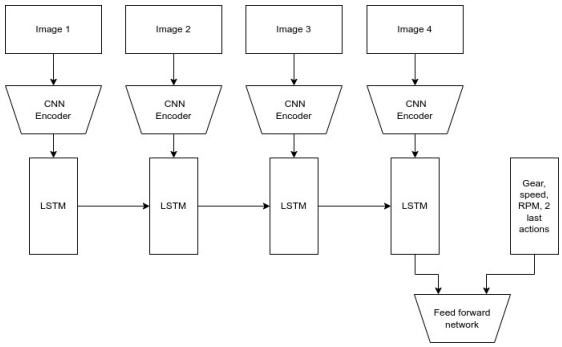


Figure 6: RNN-based model for images environment

To address the sequential nature of the problem we decided to utilize capabilities of LSTM to model the dynamics of the car and thus propose a better action.

In LIDAR environment we pass the LIDAR observations via LSTM, concatenate last hidden states and pass them through fully-connected network.

In images environment we use the same CNN to encode the images and feed those encoded images through LSTM. The rest of the model is the same as in images environment.

In both environments we used bi-directional LSTM.

4 SAC training

The Soft Actor Critic (SAC) algorithm bridges the gap between stochastic policy optimisation and DDPG-style approaches by optimising a stochastic policy in an off-policy manner. Although it was released almost simultaneously with TD3, it is not a straight replacement. However, it does use the clipped double-Q

method, and because SAC policies are inherently stochastic, it also ends up benefiting from target policy smoothing.

A key component of SAC is regularisation of entropy. The goal of policy training is to optimise the trade-off between expected return and policy entropy, which is a measure of randomness. This is closely related to the exploration-exploitation trade-off, whereby higher entropy leads to greater exploration, which can expedite subsequent learning. It may also keep the strategy from reaching a poor local optimum too soon.

5 Experiments & Discussion

All configurations, training scripts and videos are available on our GitHub [3].

5.1 Experiment setup

We launch a Steam version of Trackmania 2020 on our machine (described in Table 1). TMRL connects to Trackmania over OpenPlanet API to collect information about the current speed and RPM. Image capturing is performed via window capture on top of X-server which is also being used to adjust the window to parameters defined in experiment configuration. LIDARs are computed with the use of OpenCV library. TMRL uses vgamepad to send control signals to the game.

Component	Specification
OS	Manjaro Linux
CPU	AMD Ryzen 5900X
GPU	Nvidia RTX 3060
RAM	32 GB
VRAM	12 GB

Table 1: Main machine description

5.2 Ad-hoc

As ad-hoc algorithm does not require training and is deterministic, we conducted an empirical hyperparameters tuning during algorithm development. As a result, we present a single training run video, available in our github. Figure 7 shows cumulative reward of that run: our car successfully passes checkpoints and completes roughly 0.75 of the track in the time of 1 min, 9 seconds, getting stuck in the sharp turn. Nevertheless, the car passes moderate turns easily. However, due to condition of keeping far from the border, it always oscillates, potentially slowing down the race.

[3] `https://github.com/vBazilevich/S24-RL-project-trackmania`

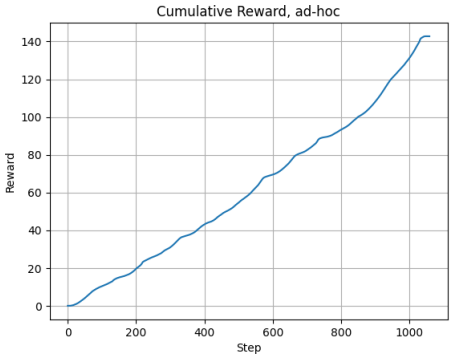
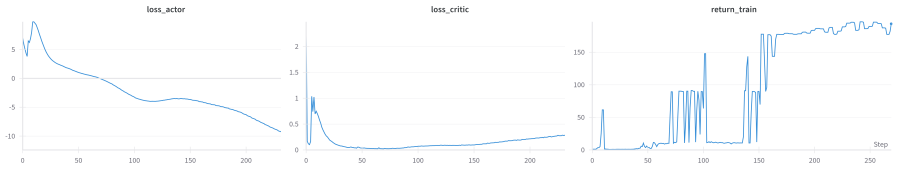


Figure 7: Cumulative reward for Ad-hoc model evaluation

Obviously this ad-hoc algorithm needs to be carefully hand-crafted for the environment. Moreover, it highly depends on the camera position and map surrounding. For instance, it would fail for another cockpit car view option, as distances to borders change in LIDAR measurements. Ad-hoc is a solid sane baseline, but not scalable and not learning from any reward or experience.

5.3 FFN

Surprisingly, FFN has shown the best result. It took around 1 hour of training in LIDAR environment for this model to learn how to ride train track from start to finish. Learning curves are on Figure 8. Training cumulative reward reaches value 196, corresponding to the finish. However, it was unable to beat Vladimir's personal record - 1 minute 5 seconds, while model fastest lap was 1 minute and 24 seconds. Likely, with extra reward (or penalty) the model might learn the more optimal trajectory.



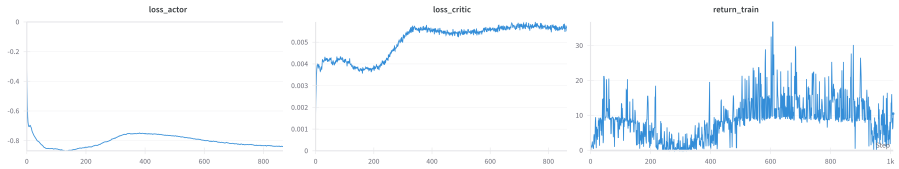
(a) Actor loss (b) Critic loss (c) Train reward

Figure 8: Performance of FFN in LIDAR environment

5.4 RNN

Overall this approach seems to be quite complicated to train both in images and LIDAR environments, with training statistics shown on Figures 9 and 10. Even after 7 hours of training car barely reaches the first turn on the map. In its best attempts model has reached fifth turn on a train map and mostly stuck

on a first U-turn. We tried different approaches to regularize the model and stabilize its training but no attempt reached ended up successfully.



(a) Actor loss (b) Critic loss (c) Train reward

Figure 9: Performance of RNN in LIDAR environment

(a) Actor loss (b) Critic loss (c) Train reward

Figure 10: Performance of RNN in Images environment

6 Conclusion

In this project, we experimented with Trackmania environment, creating an deterministic ad-hoc and improved RNN-based model for LIDAR and Image-based environments. We observe that although ad-hoc driver does arguably well with small efforts, it is not generalizable and weak at sharp turns. Complex RNN-based agents have high learning potential, but fail to train in sensible time. It turns out that standard FNN on stacked observations performs best, finishing the track in 1 minute 5 seconds.

As a possible future work, we would like to extend the TMRL library with our RNN developments, train more complex RNN agents, and try to improve ad-hoc agent on custom maps.

Bibliography cited

[1] B. R. Kiran, I. Sobh, V. Talpaert, *et al.*, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 4909–4926, 2021.

[2] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[3] X. Chen, C. Wang, Z. Zhou, and K. Ross, "Randomized ensembled double q-learning: Learning fast without a model," *arXiv preprint arXiv:2101.05982*, 2021.

[4] S. M. Devlin and D. Kudenko, "Dynamic potential-based reward shaping," in *11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, IFAAMAS, 2012, pp. 433–440.