Reinforcement Learning Agents in TrackMania: A Comparative Analysis

ECE 5424 - Patrick Stock, Adam Lahouar - 03/10/24

Abstract

This paper explores the performance of three reinforcement learning agents - Soft Actor-Critic (SAC), Randomized Ensembled Double Q-Learning (REDQ), and Proximal Policy Optimization (PPO) - within the context of the racing game TrackMania. The study employs a range of evaluation criteria to determine which algorithm demonstrates proficiency, aiming to rival a human player's skill level. The comparison includes insights into the unique characteristics of each algorithm and experimentation in scenarios designed to reveal differences in adaptability, exploration vs. exploitation, and other relevant factors. This research aims to contribute to the understanding of the effectiveness of reinforcement learning approaches in the domain of real-time optimal control.

Specific Aims

The overarching goal of this project is to develop several reinforcement learning agents for TrackMania that are competitive with a human player and assess how well each agent performs, comparing their performances with one another. This project should lead to insights about which reinforcement learning algorithm is best for this specific use case.

More specifically, we will test the following reinforcement learning architectures:

1. Soft Actor-Critic (SAC)
2. Randomized Ensembled Double Q-Learning (REDQ)
3. Proximal Policy Optimization (PPO)

These reinforcement learning agents will be evaluated and compared using the criteria described below to deduce which is the best choice for optimizing inputs in TrackMania. Theoretically, any given track will have an optimal series of inputs, and the criteria aim to gauge the proximity of the agent's inputs to achieving peak performance.

Background

TrackMania is a popular racing game with several attributes that make it well-suited for training and evaluating reinforcement learning (RL) agents:

1. The game's primary objective is straightforward: reach the finish line in the shortest time possible, offering a clear metric for assessing and comparing RL agent performance.
2. Car control in TrackMania relies on four inputs: forward, backward (brake), left, and right. These inputs can be applied in a binary or analog manner, allowing for either unpressed or fully pressed actions, or smooth inputs ranging from 0% to 100%. Despite this simplicity, the game's physics and diverse track designs offer agents many opportunities to explore strategies and optimize race times.
3. The deterministic nature of TrackMania's physics ensures that the same input sequence will always result in identical in-game actions, which reduces potential sources of noise and variability during RL agent training and testing.
4. TrackMania provides a user-friendly track editor, allowing the creation of custom tracks that can be tailored to the desired level of challenge for RL agent training.

The TMRL library facilitates the interaction between RL agents and the TrackMania environment by providing access to game information as well as the ability to control the car in real time. The agents can either operate on raw screenshots of the game environment from the perspective of the car (at a user-selectable framerate) or LIDAR measurements of the nearest walls in front of the car [1]:

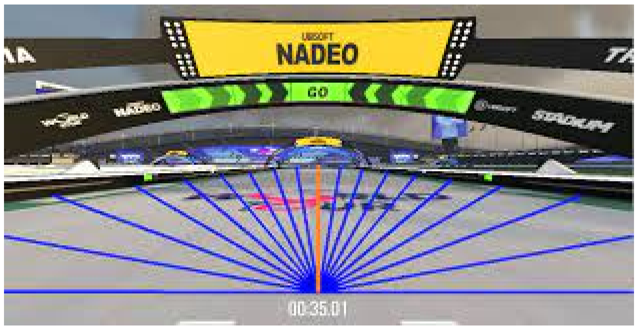


Figure 1 - TMRL library LIDAR example [1]

Previous attempts at training RL agents for TrackMania have demonstrated varying degrees of success. In [2], the performance of several agents with different architectures was evaluated, eventually landing on a generic RL agent that was able to consistently outperform the author, an experienced player likely surpassing the average human player's skill level. Conversely, in [3], employing the Soft Actor-Critic (SAC) algorithm, the agent was only able to reach 'silver medal' levels of performance, indicating proficiency below top-tier players but superior to beginners.

Research Design and Method

The selection of the three architectures (SAC, REDQ, PPO) for this project was made primarily to streamline the evaluation and comparison process between the architectures, prioritizing efficiency over the time-consuming task of troubleshooting implementation issues. The TMRL library offers prebuilt pipelines for SAC and REDQ, which is expected to greatly accelerate the initial setup process. Similarly, OpenAI provides a baseline implementation of PPO [4] that has been demonstrated to show promising results [5], further warranting its inclusion.

Soft Actor-Critic

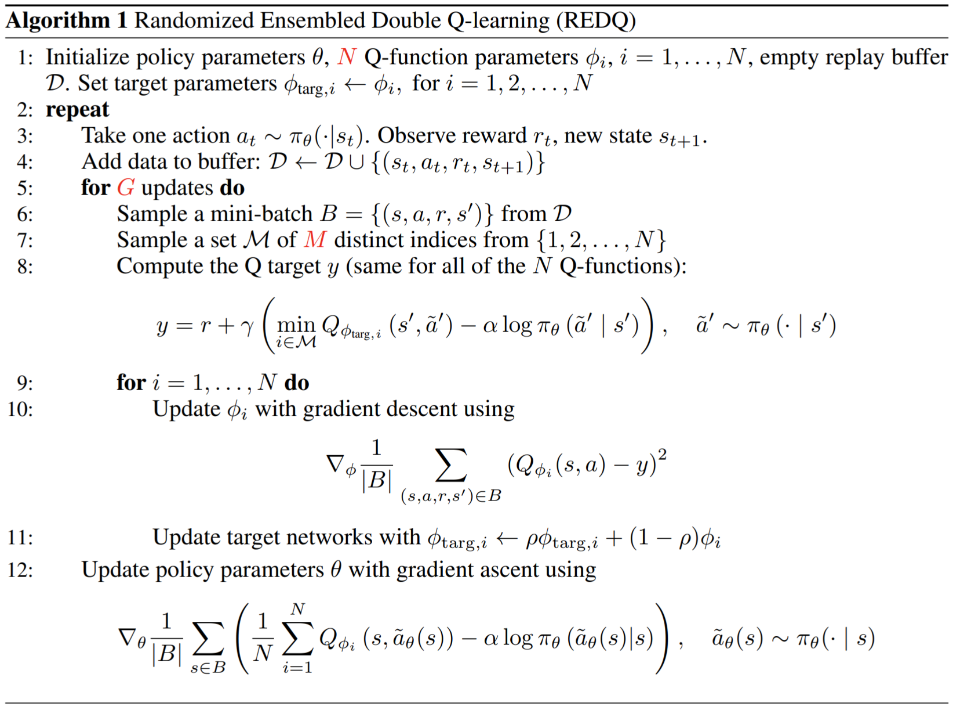
Soft Actor-Critic (SAC) is an off-policy model-free algorithm that is well-suited for environments featuring continuous action spaces, making it a viable choice for TrackMania. In SAC, two distinct artificial neural networks (ANNs) are used to handle inputs, estimate rewards, and execute actions. The actor network processes observations to generate control outputs, while the critic network uses both the observation and action to predict the expected reward from the environment with the current policy. The concurrent training of both networks involves using environment rewards to train the critic network and using the critic network to train the actor network. Additionally, the update process includes adding the entropy of the policy to the objective function, promoting exploration and resilience to variations while maintaining maximal rewards [1, 6].

```
Algorithm 1 Soft Actor-Critic
  Initialize parameter vectors $\psi$, $\bar{\psi}$, $\theta$, $\phi$.
  for each iteration do
    for each environment step do
      $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
      $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
    end for
    for each gradient step do
      $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
      $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1, 2\}$
      $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
      $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$
    end for
  end for
```

Figure 2 - Soft Actor-Critic Algorithm [6]

Randomized Ensembled Double Q-Learning

Randomized Ensembled Double Q-Learning (REDQ) is a modification of the SAC algorithm that replaces the critic network with several randomly initialized networks that are randomly sampled for reward estimates. During training, the critic networks are all updated in parallel with the same target. Additionally, REDQ includes a minimization step over a user-selectable, fixed-size random subset of the critic networks to reduce variance in the Q-function estimates [1, 7]. While the authors note in [7] that REDQ can be applied to many off-policy model-free algorithms, SAC was used for the given pseudocode:



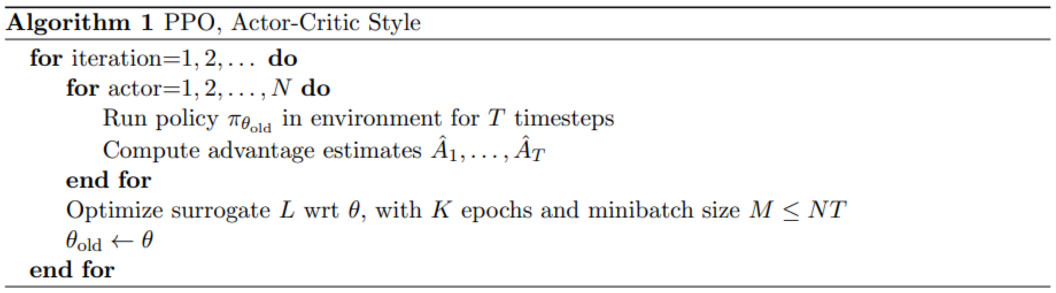
Algorithm 1 Randomized Ensembled Double Q-learning (REDQ)

```
1: Initialize policy parameters $\theta$, $N$ Q-function parameters $\phi_i$, $i = 1, \ldots, N$, empty replay buffer
   $\mathcal{D}$. Set target parameters $\phi_{\text{targ},i} \leftarrow \phi_i$, for $i = 1, 2, \ldots, N$
2: repeat
3:     Take one action $a_t \sim \pi_\theta(\cdot|s_t)$. Observe reward $r_t$, new state $s_{t+1}$.
4:     Add data to buffer: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
5:     for $G$ updates do
6:         Sample a mini-batch $B = \{(s, a, r, s')\}$ from $\mathcal{D}$
7:         Sample a set $\mathcal{M}$ of $M$ distinct indices from $\{1, 2, \ldots, N\}$
8:         Compute the Q target $y$ (same for all of the $N$ Q-functions):
               $y = r + \gamma \left( \min_{i \in \mathcal{M}} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' \mid s') \right), \quad \tilde{a}' \sim \pi_\theta(\cdot \mid s')$
9:         for $i = 1, \ldots, N$ do
10:            Update $\phi_i$ with gradient descent using
                   $\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \left( Q_{\phi_i}(s, a) - y \right)^2$
11:            Update target networks with $\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i$
12:    Update policy parameters $\theta$ with gradient ascent using
           $\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left( \frac{1}{N} \sum_{i=1}^{N} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) \right), \quad \tilde{a}_\theta(s) \sim \pi_\theta(\cdot \mid s)$
```

Figure 3 - Randomized Ensembled Double Q-Learning Algorithm [7]

Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a collection of policy gradient methods for reinforcement learning. These methods "alternate between sampling data through interaction with the environment, and optimizing a 'surrogate' objective function using stochastic gradient ascent" [5, p. 1]. In [5], they introduce a new objective function that allows multiple epochs of "minibatch" updates rather than only performing one gradient update per data sample. PPO has the data efficiency and reliable performance of trust region policy optimization (TRPO) while being easier to implement, more generalizable, and having better sample complexity. It also performs similarly to Actor-Critic with Experience Replay (ACER) while being much less complex. The pseudocode for PPO can be seen below:



Algorithm 1 PPO, Actor-Critic Style

```
for iteration=$1, 2, \ldots$ do
    for actor=$1, 2, \ldots, N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \le NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

Figure 4 - Proximal Policy Optimization Algorithm [5]

Datasets and Experiments

Given the nature of the project and the fact that TrackMania provides a dynamic and interactive environment that allows for straightforward testing, the project will primarily focus on experimentation rather than traditional datasets. The agents will be evaluated on a selection of prepared tracks within the TrackMania environment. These tracks encompass a diverse range of conditions, consisting of corners with varying angles and speeds, straight sections, gradual uphill and downhill slopes, and a small handful of dirt track surfaces.

During the evaluation process, each agent will undergo a testing trial consisting of 100 attempts per track (subject to change), and their performance will be assessed based on the following key metrics:

- Best Time: The lowest completion time achieved by the agent during the test trial.
- Average Time: The mean completion time across the test trial.
- Consistency: The standard deviation of completion times across the test trial.
- Failure Rate: The percentage of attempts in which the agent failed to complete the track before a cutoff timer (to be determined later).

In addition to the standard performance evaluation metrics, the agents will also be subjected to a series of trials to determine key differences between their underlying architectures:

- Learning Rate: The performance of the agent over the course of training.
- Adaptability/Robustness: The performance of the agent when evaluated on a track containing different road conditions and more extreme corners and slopes.
- Exploration vs. Exploitation: Evaluation of the agent's decision-making in a track containing both a short and risky route and a longer, safer route.

Timelines

The project will consist of two main activities, coding and testing. First, an agent must be coded such that it can make use of its respective reinforcement learning algorithm to learn how to play TrackMania. Next, the agent must be tested to determine whether it can succeed in completing several test tracks within a reasonable amount of time. Then, tweaks are made to the code until the agent passes all of the tests. This process is repeated across all agents until they pass the test tracks in a reasonable amount of time. Then, the agents will be evaluated using the various criteria above.

We will aim to have the first agent coded, tweaked, and tested by the end of March. This will give us enough time to learn the TMRL library, practice coding a reinforcement learning algorithm, and refine our agent-making process. In April, we will begin coding the other agents and finish by evaluating and comparing the agents.

Key References

[1] S. Ramstedt, AndrejGobeX, Pius, Y. Bouteiller, and E. Geze, “Trackmania-RL/TMRL: Reinforcement learning for real-time applications,” GitHub, https://github.com/trackmania-rl/tmrl (accessed Mar. 9, 2024).

[2] Yosh, “Training an unbeatable AI in Trackmania,” YouTube, https://www.youtube.com/watch?v=Dw3BZ6O_8LY (accessed Mar. 5, 2024).

[3] T. Potočnik, “TrackMania Reinforcement Learning,” TUDelft, https://github.com/Ladrigon/TMRL/commits/main/REPORT.pdf (accessed Mar. 8, 2024).

[4] Dhariwal, Prafulla and Hesse, Christopher and Klimov, Oleg and Nichol, Alex and Plappert, Matthias and Radford, Alec and Schulman, John and Sidor, Szymon and Wu, Yuhuai and Zhokhov, Peter, “OpenAI Baselines,” GitHub, https://github.com/openai/baselines (accessed Mar. 10, 2024).

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” arXiv.org, https://arxiv.org/abs/1707.06347 (accessed Mar. 10, 2024).

[6] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” arXiv.org, https://arxiv.org/abs/1801.01290 (accessed Mar. 9, 2024).

[7] X. Chen, C. Wang, Z. Zhou, and K. Ross, “Randomized ensembled double Q-learning: Learning fast without a model,” arXiv.org, https://arxiv.org/abs/2101.05982 (accessed Mar. 9, 2024).