Learning to Race: solving with Evolutionary Algorithm and Reinforcement Learning

João Pedro Knauer de Queiroz Verçosa
Electric Engineering Department
CTC - Scientific Technical Center
PUC-Rio, Pontifícia Universidade Católica do Rio de Janeiro
Rio de Janeiro, Rio de Janeiro, Brasil
Email: jpkqvercosa@hotmail.com

***Abstract*—This work presents two distinct approaches to solving a racing environment problem. The first approach utilizes a genetic evolutionary algorithm (EA) that mutates neural networks, enhancing their capabilities with each generation. The second approach implements the Twin Delayed Deep Deterministic Policy Gradient (TD3) reinforcement learning algorithm, testing its efficacy within the same environment. After a thorough evaluation of both algorithms, we conclude that both techniques are capable of achieving excellent results given the limitations of the environment. The EA is capable of consistently completing the task after a few mutations and the TD3 learns how to drive after some collected experience is saved in the replay memory.**

***Key-words*—machine learning, evolutionary algorithm, reinforcement learning, neural networks, MLP, TD3**

1. Introduction

Autonomous driving and autonomous racing are highly researched fields within the domains of machine learning and artificial intelligence. These fields aim to develop systems capable of navigating and making decisions without human intervention, leveraging various algorithms and computational techniques to achieve superior performance in dynamic and complex environments. Multiple surveys have been done showcasing different approaches for solving this kind of problem [1] [2] [3]. They aim to be a general solution for real-life problems and implement complex Convolutional Neural Networks (CNNs), Long-Short Term Memory (LSTM), image recognition, object detection systems, Light Detection and Ranging (LiDAR), and sensor reading devices.

In this article, we present two didactic and intuitive algorithms, based on simulating sensors through a Pyglet environment. Two methods were implemented: a basic evolutionary neural network controller and a more robust Actor-Critic On-Policy Twin Delayed Deep Deterministic (TD3) reinforcement learning algorithm [4] [5]. The two methods are supposed to be light to run and simple to visualize, enabling anyone to check their functioning and their pros and cons.

2. Objectives

The primary objective of this article is to investigate and compare two different artificial intelligence approaches for learning and optimizing performance in autonomous racing. Specifically, we aim to:

- Explore the effectiveness of a genetic evolutionary algorithm in evolving neural networks to enhance their decision-making capabilities.
- Implement and assess the performance of the TD3 reinforcement learning algorithm in the same racing environment.
- Compare the outcomes of both approaches to determine their relative strengths and weaknesses.

3. Environment

The environment under study is an application developed by Tomas Brezina and hosted on GitHub [6] under the CC BY-NC 4.0 License. This application generates a 5x3 grid system and employs pre-manufactured track pieces to create complete tracks. These tracks can support one or more cars, allowing users to observe their movement. Examples of randomly generated tracks are shown in Figure 1.

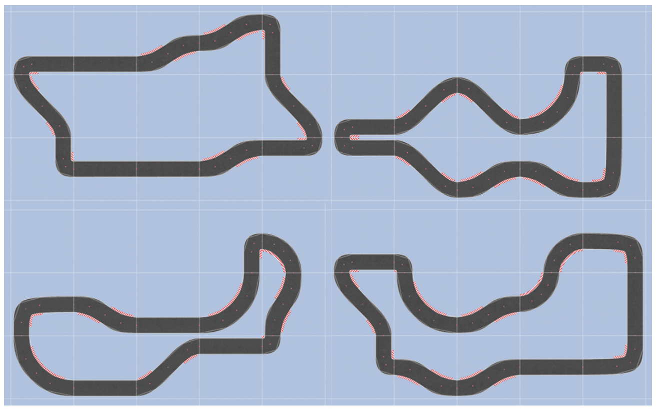
Figure 1. Four different tracks randomly generated

As illustrated in Figure 2, each car is equipped with five sensors. These sensors, represented as lines, detect the proximity of the car to the walls, which define the track's boundaries. If a car contacts a wall, its simulation is halted, rendering the car inactive for the remainder of the generation or episode.

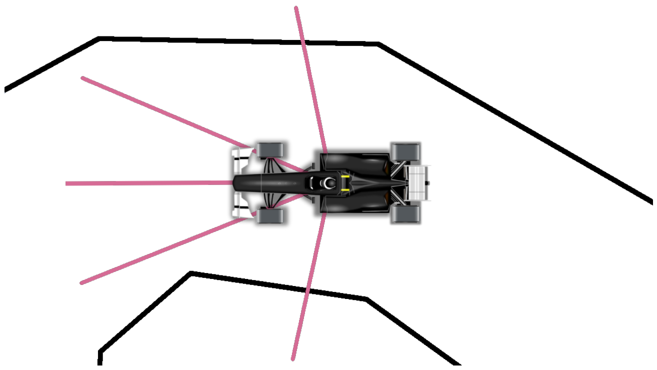
Figure 2. Car sensors showed in pink. They give feedback on how far the car is from the margin of the track. Figure adapted from [6]

Each car has an internal controller that processes six inputs: the five sensor readings and the car's speed. Based on this information, the controller outputs two values. The first value, ranging between -1 and 1, dictates the car's turning angle in the next simulation step. The second value, between 0 and 1, sets the car's acceleration for the subsequent step. While driving through the track, the car passes through checkpoints, which serve as the rewards it collects and are the metric implemented by the environment.

Each generation or episode lasts for 30 seconds, after which the active cars are reset and a new population initiates a subsequent generation or episode. At the beginning of each new generation or episode, a new track is randomly generated. This approach increases the diversity of states encountered by the cars and prevents overfitting to a specific track configuration.

Additionally, the application implements several keyboard shortcuts that enhance usability. These shortcuts allow users to move the camera, enter a debug view to observe the car's sensors, save models and configurations of the current experiment, and generate a graph displaying the reward history of the running experiment. The environment also considers a friction coefficient ($f = 0.95$), a maximum speed ($maxSpeed = 30$), and an acceleration multiplier ($a = 1$). These parameters were held constant throughout the experiments.

4. Implementations

This section contains information about implementing the two methods explored in this article. The code with all the modifications can be found on the GitHub of the author [7]. When running the app it's possible to choose between learning autonomous driving with Evolutionary Algorithm (EA) or the Reinforcement Learning algorithm.

4.1. Evolutionary Algorithm

The EA employs a simple Multi-Layer Perceptron (MLP) [8]. This feed-forward neural network is typically trained using back-propagation and optimized with stochastic gradient descent (SGD). However, for this specific problem, there is no labeled dataset available to train and evaluate the network using these conventional methods. Consequently, we opted to use an EA, utilizing the MLP as the car's "brain".

The proposed network consists of six input neurons, two hidden layers with four neurons each, and two output neurons, as depicted in Figure 3. Given the problem's simplicity, it is unnecessary to use prediction error to dictate the driving style. Instead, we evaluate a population of 40 cars, each with its own set of weights. From this population, we select the three best-performing cars and replicate their parameters with some mutation for the next generation.

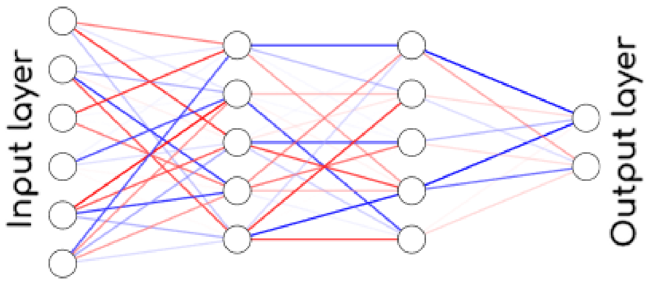

Figure 3. EA Neural Network. The six input neurons represent five sensors and one value corresponding to the car's velocity. The two output neurons control steering and acceleration. Figure adapted from [6]

The cars are evaluated based on the number of checkpoints they reach during their run. These checkpoints are located along the center of the track and are positioned following the curvature of the turns. Each track contains between 40 and 50 checkpoints, depending on its specific configuration.

It could be argued that the network is not genuinely learning since the initial weights are randomly assigned, and subsequent generations merely select and mutate the best-performing combinations. Despite this, the method remains valid, and the results, as is going to be demonstrated in section 6, are impressive.

4.2. Reinforcement Learning

Two variants of an Action-Critic technique were implemented in the app: Deep Deterministic Policy Gradient (DDPG) [9] [10] and Twin Delayed Deep Deterministic Policy Gradient (TD3) [5]. Although both algorithms can be run, this article focuses on the TD3 implementation and results, as TD3 is an evolution of DDPG.

The TD3 algorithm is an advanced off-policy actor-critic method specifically designed for environments with continuous action spaces. It addresses several key limitations inherent in its predecessor, the DDPG algorithm, including the tendency for overestimation bias in the Q-value function and training instability [5].

TD3 is built upon the deterministic policy gradient framework, where the policy (actor) directly maps states to specific actions deterministically, and the critic estimates the action-value function $Q(s,a)$. It utilizes several innovative techniques to optimize the agent's learning.

One such technique is Clipped Double Q-Learning, which uses two separate critic networks to mitigate overestimation bias. Given two critics with parameters θ_1 and θ_2, the target value for the Q-function is computed using the minimum value predicted by the two critics:

$$y = r + \gamma \min_{i=1,2} Q_{\theta'_i}(s_{t+1}, \pi_{\phi'}(s'))$$

where r is the reward, γ is the discount factor, s' is the next state, and $\pi_{\phi'}$ is the target policy network with delayed parameters ϕ'.

To further stabilize training, TD3 updates the actor and the target networks less frequently than the critics, a strategy known as delayed policy updates. Specifically, the policy and target networks are updated only every 200 iterations of the critics. This reduces the variance in policy updates and prevents the actor from adapting too quickly to the critic's changing estimates.

To address the issue of sharp changes in the target policy, TD3 also adds noise to the target action, encouraging smoother and more robust policy updates:

$$\pi_{\phi'}(s') = \pi_{\phi'}(s') + \mathcal{N}(0, \sigma)$$

where $\mathcal{N}(0, \sigma)$ is Normal distributed Gaussian noise.

Ornstein-Uhlenbeck noise was also implemented to improve the functioning of TD3. This type of noise, characterized by its temporally correlated properties, is particularly effective for continuous control tasks. By generating noise that resembles natural physical systems' variations, the Ornstein-Uhlenbeck process ensures smoother exploration in action space. This approach is beneficial for improving the stability and performance of the TD3 algorithm, as it prevents erratic changes in actions, thereby facilitating more consistent and reliable policy updates.

The controller of the car, it's again a MLP that is capable of processing inputs and generating control actions. This network comprises 6 input neurons, followed by two hidden layers, each containing 32 neurons, and the output layer consists of 2 neurons, which represent the continuous control actions for steering and acceleration. This neural network configuration is more robust than the previous one because the TD3 algorithm must effectively learn and generalize from the data, providing precise control over the car's movements.

The discount factor (γ) was held at 0.99. The noises were generated with a zero mean and standard deviation (σ) of 0.2. The rate of mean reversion (θ), which dictates how the noise process decays to zero, inside Ornstein–Uhlenbeck noise, was held at 0.15.

4.2.1. Replay Memory

Initially, a randomized sampling technique was utilized in the replay memory to store and sample transitions. This approach ensures that the learning algorithm is exposed to a diverse set of experiences, promoting stability and avoiding correlations in consecutive updates. However, randomized sampling treats all transitions equally, which was not the most efficient strategy for learning.

To address this, the replay memory was later updated to use Prioritized Experience Replay (PER). In Prioritized Experience Replay, transitions are sampled based on their importance, which is typically measured by the magnitude of their temporal-difference (TD) error. This prioritization allows the algorithm to focus more on learning from transitions that are more significant to improving the policy, thus accelerating the learning process and improving performance [11].

The priority of a transition i is typically defined as:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

where p_i is the priority of transition i and α is a hyperparameter that determines the level of prioritization. The TD error δ_i is often used to set the priority $p_i = |\delta_i| + \epsilon$, where ϵ is a small positive constant to ensure that all transitions have a non-zero probability of being sampled.

This approach not only improves the efficiency of the learning process but also enhances the stability and convergence of the algorithm by ensuring that more informative transitions are sampled more frequently, allowing the agent to learn more effectively from its experiences.

The memory size was configured to hold a maximum of 1 million transitions. A small constant (ϵ) was fixed at 1×10^{-6} to prevent any transition from having zero priority. The prioritization exponent (α), which influences the degree of prioritization, was set to 0.6, and the importance-sampling exponent (β) was initialized at 0.4. The TD3 algorithm utilizes a batch size of 512 transitions, with delayed policy updates occurring every 200 steps.

4.2.2. Reward Function

Despite the environment having its standard way of evaluating the driving performance of the car, utilizing the number of checkpoints that the car visited during a run, it was necessary to develop a special reward function to help the TD3 algorithm in the learning process. That function was designed to help the car keep a positive speed rate, and also keep its sensors away from the boundaries of the track.

The car is assigned a reward of 5 units for maintaining proximity to the center of the track while traveling at a positive velocity. In contrast, it incurs a penalty of 0.5 units for either traveling at insufficient speeds or approaching the boundaries of the track. Moreover, the vehicle accrues incremental rewards of 0.1, 0.5, 1, and 2 units as its speed progressively increases.

For comparative purposes, TD3's reward function is exclusively employed during the algorithm's training phase. However, during evaluation, we adopt the intrinsic metric of the environment to assess performance, counting the number of checkpoints reached by the car.

5. Experiments

For each algorithm under evaluation, the experimental setup involved running simulations over 1000 generations or episodes, or until the replay memory buffer reached its maximum capacity (10^6 transitions).

In the EA experiments, a population size of 40 agents was employed. At the end of each generation, the top three performing agents were selected to generate the subsequent population, utilizing a mutation rate of 0.6 to adjust the weights of their Neural Networks.

Regarding the TD3 algorithm, two distinct experiments were conducted. The first experiment involved a single car per episode, while the second experiment utilized a population size of five cars per episode. However, the efficiency of TD3 is influenced by frequent updates to the critic's and actor's neural networks once the replay memory begins to fill, resulting in decreased simulation speed as the number of cars per episode increases. This limitation constrained the number of cars that could be effectively simulated in parallel. Notably, all cars shared a common Replay Memory buffer, and the same neural network architecture was employed to instantiate cars in subsequent episodes, so, the mutation rate was zero.

The experiments recorded both the maximum and mean values of the population during runtime, alongside their corresponding 10-point moving averages. Additionally, the wall time required to achieve these results was documented. Specifically for the Reinforcement Learning algorithm, the total number of transitions stored in the replay memory at the end of each experiment was also reported.

6. Results

6.1. Evolutionary Algorithm

Following the parametrization presented in the article, the following graphs show the results of the EA, running for 200, 500, and 1000 generations.

The results of the EA, as shown in Figures 4, 5, and 6, depict the algorithm's performance across different numbers of generations. In Figure 4, the algorithm runs for 200 generations with 40 agents per generation. The best result achieved in this configuration is 125 checkpoints. The first graph in this figure illustrates the population mean and population maximum for each generation, while the second graph presents the 10-point moving average, providing a smoothed view of the algorithm's progression.

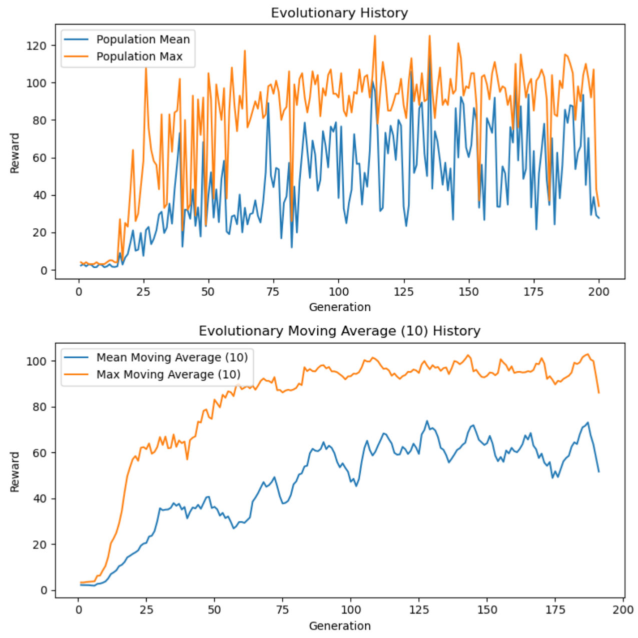

Figure 4. Results for EA, running for 200 generations with 40 agents per generation. The best result is 125 checkpoints. The first graph shows the population mean and the population max for each generation, and the second one the 10-point moving average.

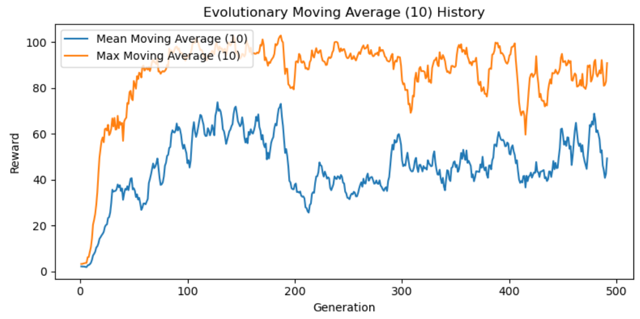

Figure 5. Results for EA, running for 500 generations. The best result is 126 checkpoints. The graph shows the population mean and the population max for the 10-point moving average.

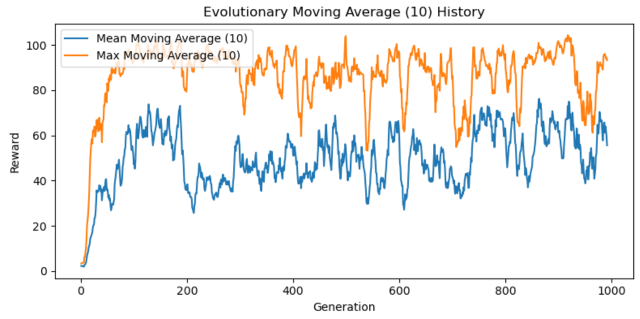

Figure 6. Results for EA, running for 1000 generations. The best result is 126 checkpoints. The graph shows the population mean and the population max for the 10-point moving average.

Figure 5 extends the evolutionary run to 500 generations, resulting in a slightly improved best result of 126 checkpoints. The graph focuses on the 10-point moving average of both the population mean and the maximum, offering insights into the longer-term trends and the stability of the evolutionary process over a more extended period.

Finally, Figure 6 shows the results for an even longer run of 1000 generations. The best result remains at 126 checkpoints, the same as observed in the 500-generation run.

6.2. Reinforcement Learning

There were two reinforcement learning experiments. The first ran for 1000 episodes, with one agent per episode, and the replay memory buffer recorded 536×10^3 transitions. The second one ran for 352 episodes with 5 agents per episode when it reached the 1×10^6 transitions recorded at the replay buffer.

6.2.1. Experiment 1

The results for the first experiment of the TD3 algorithm with one car per episode are illustrated in Figures 7, 8, and 9, displaying its performance over different numbers of episodes.

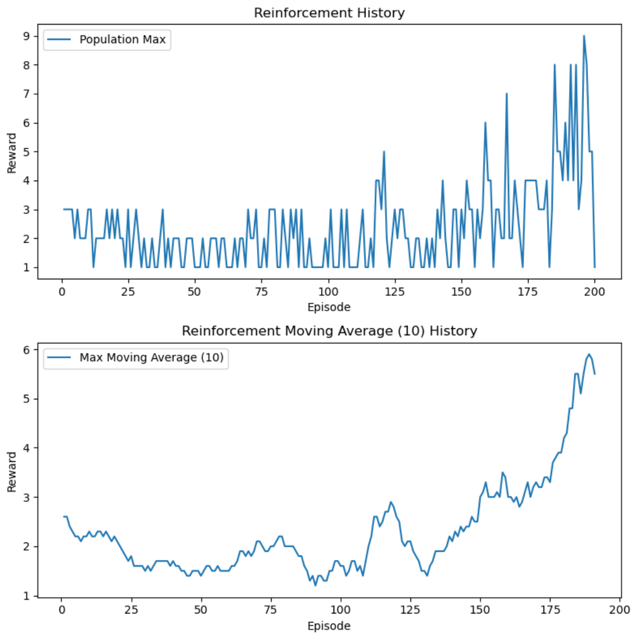

Figure 7. Results for TD3, running for 200 episodes with 1 agent per episode. The best result is 9 checkpoints. The first graph shows the agent score for each generation, and the second one the 10-point moving average.

In Figure 7, the algorithm runs for 200 episodes with one agent per episode, achieving the best result of 9 checkpoints. The first graph in this figure shows the agent's score for each episode, while the second graph presents the 10-point moving average, which helps smooth out short-term

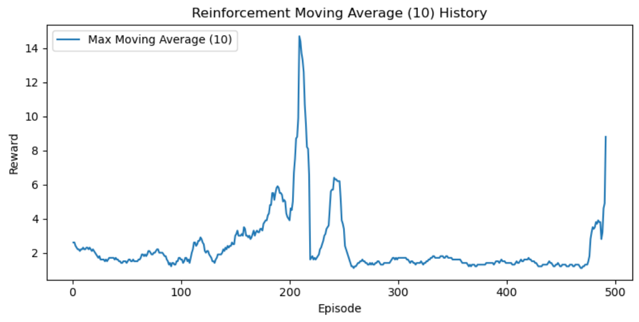

Figure 8. Results for TD3, running for 500 episodes with 1 agent per episode. The best result is 52 checkpoints. The graph shows the agent score with the 10-point moving average.

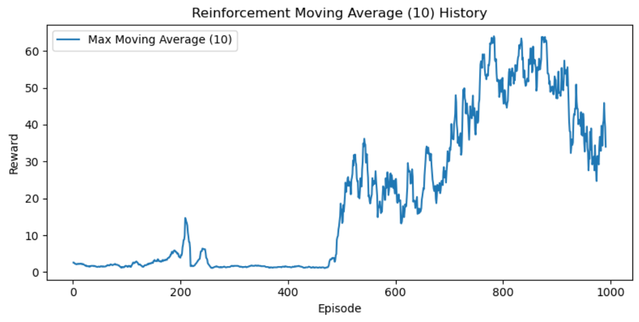

Figure 9. Results for TD3, running for 1000 episodes with 1 agent per episode. The best result is 88 checkpoints. The graph shows the agent score with the 10-point moving average.

fluctuations and highlights longer-term trends in the agent's performance.

Figure 8 extends the algorithm's run to 500 episodes, resulting in a significantly improved best result of 52 checkpoints. This figure includes a graph that focuses on the 10-point moving average of the agent's score, offering a clearer view of the performance trajectory over a larger number of episodes. It's possible to see that the agent isn't consistent yet, and despite reaching 52 checkpoints in episode 214, the 10-point moving average around this episode shows that the average score is 15. So in this moment of training it's possible to consider the best run an outlier.

Finally, Figure 9 shows the results for an even longer run of 1000 episodes. The best result achieved in this configuration is 88 checkpoints, indicating continued performance enhancement with more training episodes. The graph in this figure also presents the 10-point moving average of the agent's score, demonstrating the overall trend and stability of the TD3 algorithm's performance over an extended period. It's possible to see substantial gains from 500 to 1000 episodes underscoring the importance of extensive training for achieving high performance levels.

6.2.2. Experiment 2

The results for the second experiment are depicted in Figures 10 and 11 and show its performance with multiple agents per episode.

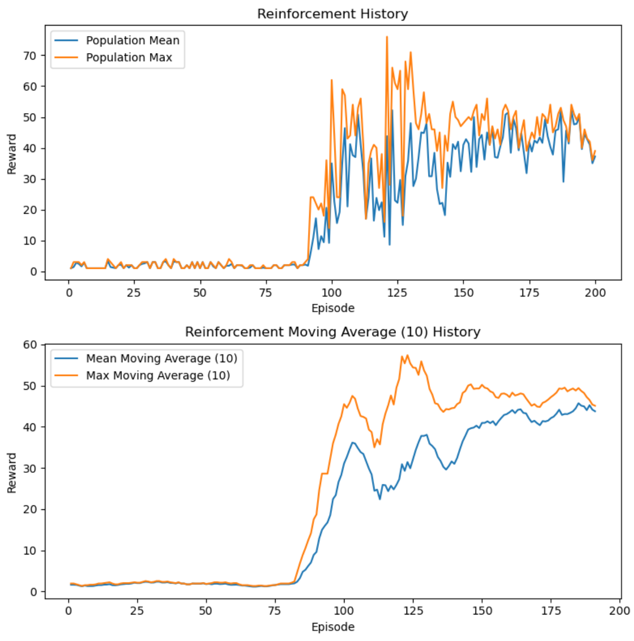

Figure 10. Results for TD3, running for 200 episodes with 5 agents per episode. The best result is 76 checkpoints. The first graph shows the agent score for each generation, and the second one the 10-point moving average.

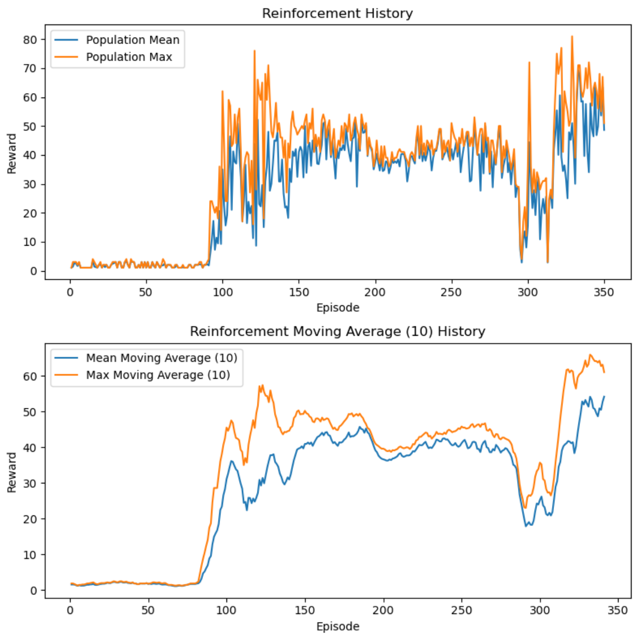

Figure 11. Results for TD3, running for 350 episodes with 5 agents per episode. The best result is 81 checkpoints. The first graph shows the agent score for each generation, and the second one the 10-point moving average.

In Figure 10, the algorithm runs for 200 episodes with five agents per episode, achieving a best result of 76 checkpoints. The first graph shows the agent scores for each episode, reflecting the performance variability among the five agents, while the second graph presents the 10-point moving average, smoothing out the scores to illustrate the overall trend more clearly.

Figure 11 extends the number of episodes to 350, resulting in a slightly improved best result of 81 checkpoints.

6.3. Time comparison

The results illustrated in Figures 12, 13, and 14 compare the runtime of the EA and the TD3 algorithm under different experiments.

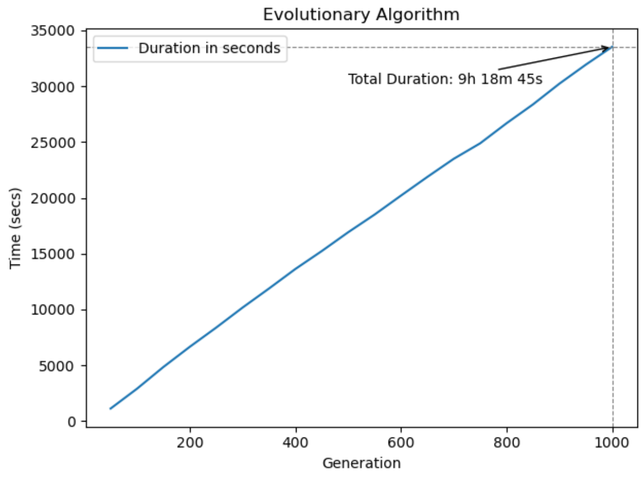

Figure 12. The EA takes 9 hours and 18 minutes and 45 seconds to run the 1000 generations

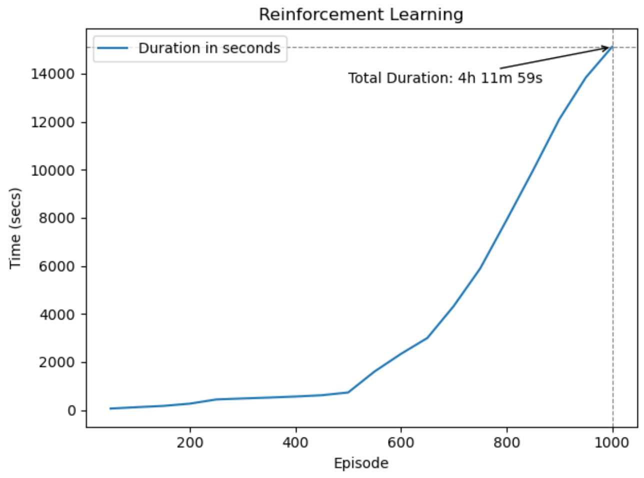

Figure 13. The TD3 with 1 car per episode takes 4 hours 11 minutes and 59 seconds to run 1000 generations

Figure 12 shows that the EA takes 9 hours, 18 minutes, and 45 seconds to run 1000 generations, with learning occurring in linear time. This indicates a steady, predictable increase in computational time as the number of generations

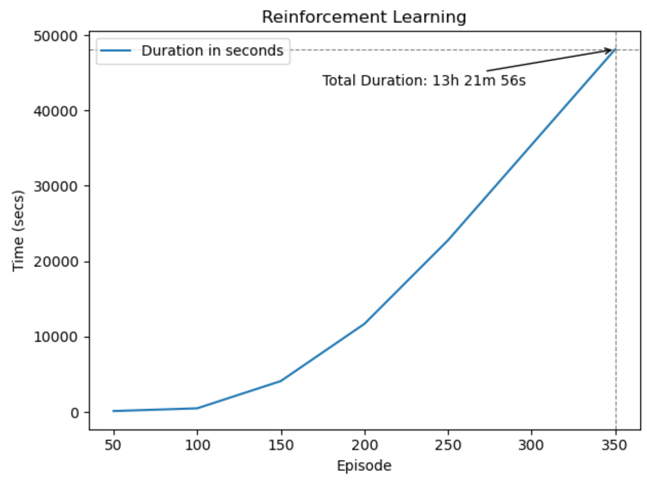

Figure 14. The TD3 with 5 cars per episode take 13 hours 21 minutes and 56 seconds to run 350 generations

increases, reflecting a consistent algorithmic efficiency over the entire run.

In Figure 13, the TD3 algorithm with one car per episode requires 4 hours, 11 minutes, and 59 seconds to complete 1000 episodes. The learning curve remains linear, but notable increases in the slope are observed after 500 and 700 episodes.

Figure 12 presents the results for TD3 with five cars per episode, taking 13 hours, 21 minutes, and 56 seconds to run 350 episodes. The time complexity grows exponentially, indicating that adding more agents per episode significantly increases the computational burden. This exponential growth highlights the greater resource requirements and complexity when training multiple agents simultaneously, which contrasts with the more linear time growth observed in the other scenarios.

7. Conclusion

Our comparative analysis of the performance of an EA and reinforcement learning TD3 algorithm across various generations and episodes provides valuable insights into their performance and efficiency in learning to race.

The results illustrated in Figures 4 to 6, the EA demonstrates a consistent improvement in performance with increased generations, peaking at 126 checkpoints by the 1000th generation due to the time limit imposed by the environment. The population mean and maximum curves are relatively far apart because every generation some cars crash instantly while others accumulate more checkpoints due to mutations occurring in each generation.

The EA shows high robustness and stability, likely because we are dealing with a simplified problem where it is possible to achieve a good combination of weights in the neural network within a few generations. It takes a longer time to reach 1000 generations because there was always an agent capable of running for the entire 30 seconds in almost every generation.

Conversely, the TD3 algorithm, as detailed in Figures 7 through 11, shows more substantial improvement as we collect more transitions into the replay buffer. The algorithm learns from its experience, requiring various states with different rewards for the actor-critic model to start converging.

In single-agent scenarios, performance markedly enhances from 9 checkpoints at 200 episodes (Figure 7) to 88 checkpoints at 1000 episodes (Figure 9). This significant progression underscores TD3's potential to leverage longer training durations to achieve superior results.

Notably, the time efficiency of TD3 with one agent per episode (Figure 13) is significantly better, requiring just over 4 hours for 1000 episodes. This efficiency arises because, because in the beginning, as there is only one agent, if it crashes, the episode ends, and a new one begins. Since initially the agent chooses its action based on a noisy distribution, the episodes are short until it learns the correct actions.

With five agents per episode, although the performance improves initially, the time required grows exponentially, culminating in over 13 hours for 350 episodes (Figure 14). This is because all agents frequently update their weights in critics and actor networks with every step they take. Additionally, only one of the agents is carried over to the next episode, meaning $\frac{4}{5}$ of the calculations done during the episode are discarded.

The findings presented suggest that while the EA is effective in optimizing checkpoint achievement over prolonged generations, TD3, particularly with a single agent per episode, offers a more time-efficient solution with competitive performance improvements. It is also evident that TD3 is effectively learning, while EA benefits from the simplified environment. In a real-world scenario, it would be more challenging to reach optimal solutions using EA.

The choice between these algorithms should therefore be guided by the specific requirements of the environment and computational efficiency within practical constraints. Future research should explore hybrid approaches and further parameter tuning to potentially combine the strengths of both methods.

An alternative to solving the issue of exponentially growing training time in TD3 could be starting with five agents, collecting their experience, and, once learning reaches an acceptable plateau, saving the model and continuing training with only one agent per episode. These steps could be explored in future work.

References

[1] Kiran, B. R., Ibrahim Sobh, V. Talpaert, P. Mannion, A. A. Sallab, S. Yogamani, e P. P'erez. "Deep Reinforcement Learning for Autonomous Driving: A Survey". IEEE Transactions on Intelligent Transportation Systems 23 (2020): 4909–26. https://doi.org/10.1109/TITS.2021.3054625.

[2] Badue, Claudine, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B. Cardoso, Avelino Forechi, Luan Jesus, et al.

"Self-Driving Cars: A Survey". Expert Systems with Applications 165 (march, 2021): 113816. https://doi.org/10.1016/j.eswa.2020.113816.

[3] Grigorescu, S., Bogdan Trasnea, Tiberiu T. Cocias, e G. Macesanu. "A survey of deep learning techniques for autonomous driving". Journal of Field Robotics 37 (2019): 362–86. https://doi.org/10.1002/rob.21918.

[4] Hessel, Matteo, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, e David Silver. "Rainbow: Combining Improvements in Deep Reinforcement Learning". arXiv, 6 de outubro de 2017. https://doi.org/10.48550/arXiv.1710.02298.

[5] Fujimoto, Scott, Herke van Hoof, e David Meger. "Addressing Function Approximation Error in Actor-Critic Methods". arXiv, 22 de outubro de 2018. https://doi.org/10.48550/arXiv.1802.09477.

[6] Tomas Brezina, *NeuralNetworkRacing*. Accessed on: 28/05. Available at: https://github.com/TomasBrezina/NeuralNetworkRacing

[7] João Pedro Knauer de Queiroz Verçosa, GitHub, *ICMS Prediction*, https://github.com/JPVercosa/LearnToRace

[8] Haykin, S. *Neural networks: a comprehensive foundation*. (1994) Prentice Hall PTR.

[9] Silver, David, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, e Martin Riedmiller. "Deterministic Policy Gradient Algorithms", [s.d.].

[10] Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, e Daan Wierstra. "Continuous control with deep reinforcement learning". arXiv, 5 de julho de 2019. https://doi.org/10.48550/arXiv.1509.02971.

[11] Schaul, Tom, John Quan, Ioannis Antonoglou, e David Silver. "Prioritized Experience Replay". arXiv, 25 de fevereiro de 2016. https://doi.org/10.48550/arXiv.1511.05952.