Twin Delayed DDPG on Car Racing V2

David B. Hoffmann, Jan Henrik Bertrand
Cooperative State University Baden-Wuerttemberg
Mannheim
{s212408, s212402}@student.dhbw-mannheim.de
Matr. Nr. 2571020, 8556462

July 11, 2024

Contents

Acronyms

RL Reinforcement Learning

TMRL Trackmania Reinforcement Learning

ASHA Asynchronous Successive Halving Algorithm

DDPG Deep Deterministic Policy Gradient

PPO Proximal Policy Optimisation

SAC Soft Actor Critic

TD3 Twin Delayed DDPG

List of Tables

List of Figures

Twin Delayed DDPG on Car Racing V2 *

David B. Hoffmann, Jan Henrik Bertrand
Cooperative State University Baden-Wuerttemberg
Mannheim
{s212408, s212402}@student.dhbw-mannheim.de
Matr. Nr. 2571020, 8556462

July 11, 2024

Abstract

This project explores the application of the Twin Delayed DDPG (TD3) algorithm to the CarRacing-v2 environment using OpenAI's Gym. Originally, we considered the Trackmania environment but switched to CarRacing-v2 due to resource limitations. We aimed to train an autonomous agent to navigate a randomly generated race track with continuous control inputs.
The project underscores the importance of hyperparameter tuning and demonstrates the practical deployment of reinforcement learning algorithms for complex control tasks. The project repository can be found at `https://github.com/adavidho/racing-rl-project/tree/main`

***K*eywords** Reinforcement Learning · Twin Delayed DDPG · Car Racing V2

1 Introduction

1.1 Motivation

Machine learning models have matched human capabilities in many digitised tasks in recent years. When it comes to the interaction of machine learning models with the chaotic real world, they still often fall behind human capabilities due to the lack of available labelled data and the complex environment they have to face.
Autonomous driving is one example where we see constant research progress. Motivated by this scientific progress we developed our own autonomous driving agent.

Initially, we started development with the Trackmania car racing game as our environment and the Trackmania Reinforcement Learning (TMRL) package [2]. After the initial setup, we found that the high resource requirements for training agents would not allow for extensive experiments or sufficient hyperparameter tuning.

This leads us to the less complex Car Racing V2 [1] environment. It is one of the native gymnasium Box2D environments that allow the training of an agent on top-down image data of a randomly generated racing track.

1.2 Observation Space

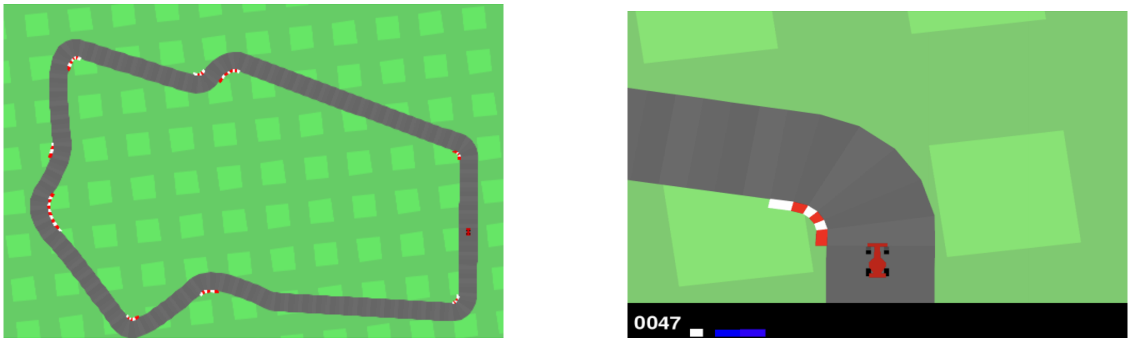
The Car Racing V2 gymnasium environment generates a random race track at initialisation such as the one shown in Figure 1b. The agent can observe the state of the environment through a close-up 96 by 96 coloured image such as the one shown in Figure 1b. As such the observation space is discrete and high dimensional with values between 0 and 255 for each of the $96 \times 96 \times 3 = 27648$ pixel values. As the value range is relatively large and integral we can assume that an agent will treat it as continuous and not make clear categorical case distinction between two different values.

1.3 Action Space

The Car Racing V2 environment has a discrete and continuous action space. The former has five possible actions such that the agent (car) can either do nothing, steer left, steer right, accelerate or brake. In the continuous case, the action is

*<u>*Citation*</u>: **David B. Hoffmann, Jan Henrik Bertrand. Twin Delayed DDPG on Car Racing V2**

Figure 1: Car Racing V2 Environment [1]



(a) Full Track View Example (b) Sample Observation

determined by a vector of length three, where the first element $\in [-1, 1]$ controls steering (-1 full left, 0 neutral, 1 full right), the second element $\in [0, 1]$ controls the gas (0 no acceleration, 1 represents full throttle) and the last element controls the break (0 not breaking, 1, full break).
We chose the continuous action space for the following two main reasons. Firstly, it offers a more precise level of control over the car through continuous values for steering, gas and break. Secondly, we hypothesise that it will be more natural and therefore easier for the neural network-based Reinforcement Learning (RL) agent to learn continuous targets in opposition to categoricals.

2 The Agent

2.1 Algorithm Selection

When choosing the RL algorithm we have to consider constraints opposed by the state and the action space. As both of them can be considered continuous, we can exclude all algorithms that only support discrete observations and actions (e.g. Multi-Armed Bandits, Q-Learning, SARSA or other Tabular algorithms) as they are most likely not well adapted to this problem.

There remain a variety of different DeepRL algorithms to choose from such as Deep Deterministic Policy Gradient (DDPG) [3], Proximal Policy Optimisation (PPO) [4], Soft Actor Critic (SAC) [5], and TD3 [6] each with its advantages and disadvantages.
In Car Racing V2 we want the agent to find an optimal route through a given track. Once it finds this route the optimal behaviour will be to stick to it and not include random variation. It follows that a deterministic policy where the best action is taken should be able to lead to optimal performance. This allows us to avoid unnecessary complexity and leaves us with a choice between DDPG and TD3. The latter is based on the former, but benefits from multiple improvements such as double learning in the q-function network (critic) which stops maximisation bias, and delays in the policy network (only updated every n-steps) to remedy the fact the q-values are initially bad, and introducing noise to output actions and target actions which facilitates exploration, we choose to use TD3 for our experiments. We choose the Stable-Baselines3 [7] implementation for our experiments.

2.2 Hyperparameter Tuning

Hyperparameters can help with better convergence and stable training which is a big issue in RL. For the selected TD3 algorithm we chose to optimise the learning rate α which most likely has the biggest impact on convergence, the Polyak update coefficient τ and the discount factor γ, with the ranges specified in Table 1.

Hyperparameter	**Notation**	**Distribution**	Optimum
Learning Rate	α	log uniform(1e-8, 0.1)	5.6672e-05
Polyak Update	τ	log uniform(1e-8, 1)	7.2524e-06
Discount Factor	γ	uniform(0.9, 0.999)	9.7588e-01

Table 1: Hyperparameter Tuning Results for TD3

Tuning is implemented using the syne tune package for large-scale hyperparameter optimisation [8] in combination with the Asynchronous Successive Halving Algorithm (ASHA) [9] optimiser, which is a distributed version of hyperband [10]. For this training, a reporter callback is added to the training loop to facilitate communication between the tuning workers and the ASHA scheduler. This setup allowed for distributed tuning over eight NVIDIA Tesla V100-SXM2-16GB GPUs and evaluated 293 sampled hyperparameter configurations within 17 hours. The resulting optimal hyperparameter values are also documented in Table 1.

2.3 Training

For the final training, the reporter callback is removed from the training loop. Instead, two additional callbacks are introduced. An evaluation callback monitors agent performance in an evaluation environment and then saves checkpoints when the model reaches a new best performance.
A second callback is used for early stopping in case a mean episode reward of 500 is exceeded, which did not happen during our training.
The optimal hyperparameters found during tuning were then used for model training. In our final training run, we set the number of time steps for training to one million.

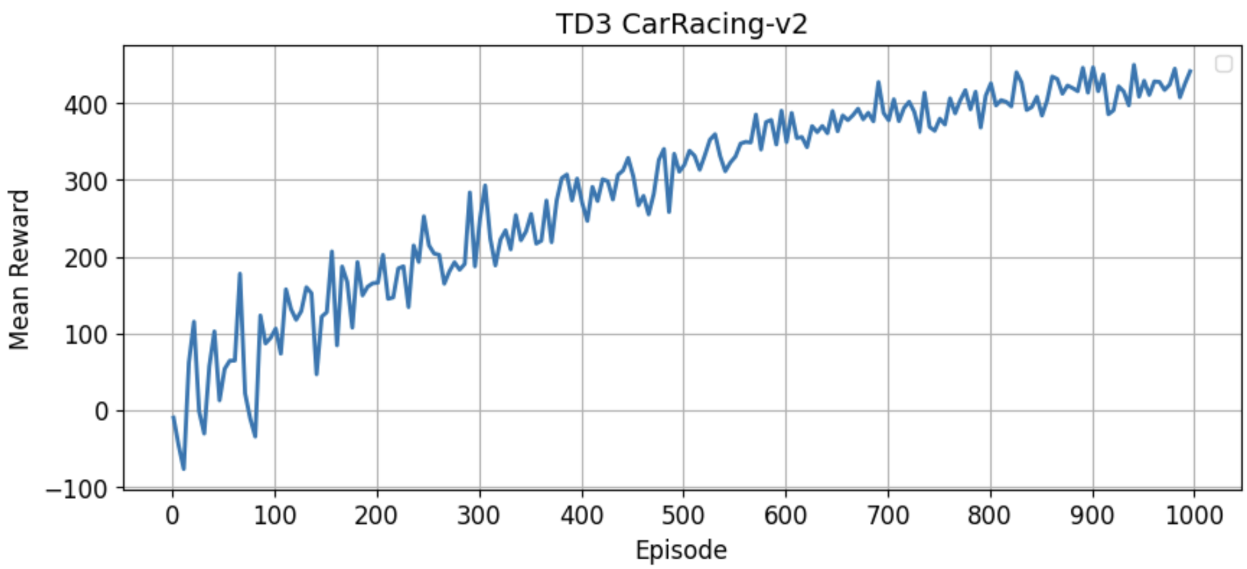


Figure 2: Training Curve for TD3 on Car Racing

Figure 2 shows that the agents' mean reward increases relatively stabley, due to the optimised hyperparameters. As we can see, the model is not fully converged, indicating that with an increased number of time steps, it should be possible to attain even better performance.

3 Discussion

Due to extensive hyperparameter tuning, we were able to successfully train the TD3 algorithm on the CarRacing-v2 environment. However, we encountered several challenges in the process of doing so.
The primary problem we were facing in the beginning was convergence. In the first experiments that did not use optimal hyperparameters, we found that the model neither converged nor diverged but rather stagnated at a constant mean reward of -80 with small fluctuations. This was even the case when increasing training up to 600,000 time steps. When visualising the behaviour of this agent we found that it was not performing any action but rather stayed stationary in one spot, collecting a negative reward for each time step.
The initial intuition of increasing the amount of random noise added to the action did not yield the desired results. Instead, we hypothesised that suboptimal hyperparameters may keep the model from learning.
However, hyperparameter tuning itself came with a challenge. Where training a single model already took hours, we were estimating that fully training a representative number of models for hyperparameter tuning would take multiple days or weeks. This leads us to use ASHA which as a distributed version of Hyperband [10], does not train all models up to full convergence but uses successive halving to prioritise well-performing configurations.

Even with this highly optimised way of approximating optimal hyperparameters training we initially allocated four days for tuning on a p3.16xlarge EC2 instance with eight NVIDIA Tesla V100-SXM2-16GB GPUs. As we had to run everything in a notebook, we used nbconvert inside a screen session to facilitate long runtimes. Initially, this seemed to work. GPU utilisation stopped after 4 days, however, the nbconvert did not write the finished notebook to a file within the next two days. After repeatedly decreasing the overall tuning time nbconvert successfully terminated with 17 hours of hyperparameter tuning.
The optimised hyperparameters lead to a vastly improved training performance that reaches between a mean episode reward of 370 and 450 depending on the run.
Overall, the project improved our understanding of how to tackle challenges with reinforcement learning and improve training using general machine learning practices like hyperparameter tuning or RL specific principles like the exploration versus exploitation trade-off.

References

[1] Chris Campbell. Top-down car physics - Box2D, 2014.

[2] Yann Bouteiller, Edouard GEZE, GobeX, and pius. Trackmania-RL, 2024.

[3] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[6] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[7] 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.

[8] David Salinas, Matthias Seeger, Aaron Klein, Valerio Perrone, Martin Wistuba, and Cedric Archambeau. Syne tune: A library for large scale hyperparameter tuning and reproducible research. In *International Conference on Automated Machine Learning, AutoML 2022*, 2022.

[9] Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A System for Massively Parallel Hyperparameter Tuning. Technical Report arXiv:1810.05934, arXiv, 2020.

[10] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. Technical Report arXiv:1603.0656, arXiv, 2018.