Building a Robust AI Agent in TORCS using Trust-Region Policy Optimization

Anonymous Authors[1] Kostya Sebov

1. Introduction

1.1. Synopsis

The Open Racing Car Simulator (TORCS) is an environment accessible via OpenAI's gym and is perfectly suited for building and testing reinforcement learning algorithms that can enable an agent to learn how to drive virtually. My partner and I are especially interested in developing a robust learning algorithm via trust-region policy optimization /citeSchulman et al., 2015), policy-based learning approach that will ultimately lead to the agent learning to drive in a manner that minimizes both the completion time on the race course and the likelihood of collisions and going off-course.

Other attempts have been made to train an agent to properly race in this environment, including the popular paper written by Google's DeepMind (Silver et al., 2016) on continuous control that used deep deterministic policy gradients (DDPG) to attain noteworthy results. However, there is still ground to be covered to attain consistent stability and seamless maneuvering in this environment and those are the gaps we will attempt to close.

1.2. Importance

Given the rise of self-driving car technology, mostly spearheaded by Tesla, it is natural for students of machine learning and artificial intelligence to ponder and probe the various complications in developing such technology, primarily the technicalities in determining how the agent optimally balances exploration and exploitation without exposing the passenger to avoidable danger.

We do not want to productionize autonomous vehicles that choose actions purely stochastically (completely randomly) but rather one that is risk-averse and avoids exploring actions such as flooring the gas that would lead to dangerous scenarios. In order to be both a safe and effective driver, these agents must be highly robust. Hence, this project is a potential step towards realizing this greater goal.

2. Approach

For reference, the state-space in this environment consists of the following:

- Angle between the car direction and the direction of the track axis. Ranges from [-180, 180] degrees.
- A Vector of 19 range finder sensors: each sensor returns the distance between the track edge and the car within a range of 200 meters
- Distance between the car and the track axis. The value is normalized w.r.t. to the track width: it is 0 when the car is on the axis, values greater than 1 or -1 means the car is outside of the track.
- Speed of the car along the longitudinal axis of the car. This is the central term in the reward function.
- Speed of the car along the transverse axis of the car.
- Speed of the car along the z-axis of the car.
- Vector of 4 sensors representing the rotation speed of wheels.
- Number of rotation per minute of the car engine.

Additionally, the action-space consists of:

- **Steering**: .
- **Acceleration**:
- **Brake**:

2.1. High-Level Methodology

We started by reproducing the results on TORCS for deep deterministic policy gradients with experience replay as done in the aforementioned continuous-control paper by DeepMind. From there, we train the agent on different tracks to prevent overfitting and tune the hyperparameters until satisfactory performance is reached. In particular, we start out with the easier tracks with level roads and minimal curves so the agent can focus learning specifically how to accelerate and brake and then train on progressively more difficult tracks so the agent learns how to steer while simultaneously adjusting speed with the accelerator and brake. We also adjust different hyperparameters including batch-size (number of frames per episode) and the reward function, which in its vanilla form is simply the observed longitudinal velocity of the car.

Then, we implement trust-region policy optimization instead and proceed to follow the same set of steps outlined in the previous paragraph until performance stabilizes. Finally, once the TRPO-based model is fully trained, we want to compare the two approaches (DDPG and TRPO) in terms of how quickly the agent reaches the finish line as well as the number of collisions and other blunders on various test tracks. Hopefully, TRPO validates our hypothesis and yields a more robust driving agent.

2.2. Current Status

Currently, we have the deep deterministic policy gradient implementation up and running. This implementation includes experience replay via re-sampling from a maintained buffer, target Q-network for stability during training process, and a Ornstein-Uhlenbeck process for exploration. The Ornstein-Uhlenbeck process is a stochastic process with built-in mean-reverting properties with tunable parameters controlling rate of reversion to the mean and rate of drift from the mean. Sufficient training has been done to the extent that the agent is now able to drive relatively well and usually complete the tracks it was trained on. Occasionally, the agent crashes or gets stuck somewhere on the track and cannot recover.

One major change we made to the reward function to get better performance was penalize the agent for driving in the traverse direction. This did in fact reduce the training time required and slightly better steering. However, it is still a work in progress and may need different modifications with the TRPO implementation.

2.3. Future Objectives

First things first, we need to ensure that when an agent gets stuck it finds a way to get back onto the track. Even better, we can prevent the agent from getting trapped in the first place by having it learn to hit the brake stochastically in a similar fashion to human drivers. In other words, the agent will learn to ”feel out” the brake and consequently execute better turns and avert crashes.

Secondly, we need to actually implement trust-region policy optimization end-to-end. The appeal behind this approach over others is that it’s designed for robustness and empirically tends to result in monotonic policy improvement. Further, it tends to require very little tuning of hyperparameters to function properly.

Of course, implementing TRPO, training, and model-tuning in order to get the agent to perform will require substantially more time and effort. We may need to make several tweaks along the way to ensure TRPO functions as intended and ultimately converges to a robust policy.