Deep Reinforcement Learning for Autonomous Systems

Piero Macaluso - s252894
Candidate
Politecnico di Torino

Prof. Pietro Michiardi
Supervisor
EURECOM

Prof. Elena Baralis
Supervisor
Politecnico di Torino

This document represents a summary of the master thesis project. The source code of this work is publicly available at `https://github.com/pieromacaluso/Deep-RL-Autonomous-Systems`

1. Introduction

Because of its potential to drastically change mobility and transport, autonomous systems and self-driving vehicles are attracting much attention from both the research community and industry. Recent work has demonstrated that it is possible to rely on a comprehensive understanding of the immediate environment while following simple high-level directions, to obtain a more scalable approach that can make autonomous driving a ubiquitous technology. However, to date, the majority of the methods concentrate on deterministic control optimisation algorithms to select the right action, while the usage of deep learning and machine learning is entirely dedicated to object detection and recognition.

Recently, we have witnessed a remarkable increase in interest in Reinforcement Learning (RL). It is a machine learning field focused on solving Markov Decision Processes (MDP), where an agent learns to make decisions by mapping situations and actions according to the information it gathers from the surrounding environment and from the reward it receives, trying to maximise it. As researchers discovered, it can be surprisingly useful to solve tasks in simulated environments like games and computer games, and RL showed encouraging performance in tasks with robotic manipulators. Furthermore, the great fervour produced by the widespread exploitation of deep learning opened the doors to function approximation with convolutional neural networks, developing what is nowadays known as deep reinforcement learning.

1.1. Objective

In this thesis, we argue that the generality of reinforcement learning makes it a useful framework where to apply autonomous driving to inject artificial intelligence not only in the detection component but also in the decision-making one. The focus of the majority of reinforcement learning projects is on a simulated environment. However, a more challenging approach of reinforcement learning consists of the application of this type of algorithms in the real world.

After an initial phase where we studied the state-of-the-art literature about reinforcement learning and analysed the set of possible alternatives about technologies to use, we started our project from the ideas in [2], where the authors were able to train a self-driving vehicle by using Deep Deterministic Policy Gradient (DDPG) [3] by tuning hyper-parameters in simulation. Our approach was different: we researched an algorithm suitable for real-world experiments and capable of operating correctly without an expensive hyper-parameter tuning made in simulators. We found in Soft Actor-Critic (SAC) [1] the algorithm we needed.

Therefore, our thesis consisted of two main contributions:

1. Designing of the control system to let all components and technologies involved interact.
2. Setting up the experimental framework with SAC algorithm and carrying out an experiment to analyse the strengths and weaknesses of this approach.

2. Design of the control system

We based our project on Cozmo, a little toy robot produced by Anki, whose developers offered a granular and fully-featured Python SDK with many interfaces to allow direct control of the robot. We found it to be suitable for our experimental needs: it mounts a grayscale camera with 60° field of view (FoV) and has two tracks to steer and drive. We aimed to apply deep reinforcement learning algorithms, so we decided to use PyTorch as deep learning framework and the standardised approach provided by OpenAI Gym to build up the reinforcement learning environment for the experiments.

Our idea was to build a system where the human holds complete control over the experiment progress: he can start the episode and suspend it when the robot reaches a fatal state to reposition it in the closest stable situation and restart

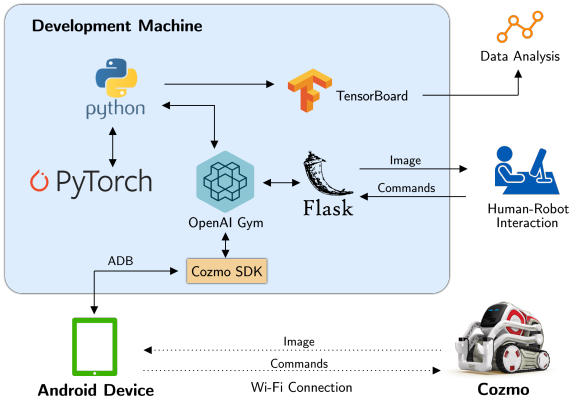

Figure 1. Outline of the control system that shows the most crucial technologies and component involved.

the loop. In this scenario, the human has total responsibility for the robot training: he is the one who decides whether an action is dangerous or not by disengaging reinforcement learning algorithms decision.

To obtain this configuration, we designed a simple and intuitive user interface that prompts the user when a new experiment starts. This interface provides a live stream from Cozmo on-board camera and a set of keys with the related function. It works through Javascript to communicate to a Flask server that interfaces directly to Cozmo SDK and the OpenAI Gym environment to provide information for the user (e.g. images, learning information) and the robot (e.g. commands). We used TensorBoard to gather data for analysis and representations. The outline of the system we obtained is available in figure 1.

CozmoDriver-v0 is the Cozmo environment we designed. It represents the result of continuous development and testing to solve all the problems and particularities that we have found along the way. We implemented all the requirements and needs dictated by the type of experiment, putting in place all the compromises to manage the available RAM and make experiments manageable by reducing the learning time between one episode and the next one. One of the most crucial features we implemented is the rescue system, useful in a context where experiments last tens of hours and unexpected errors can occur. It consists of two central parts. The first one is a volatile backup system: it stores the system state before each episode to allow the user to restore faulty ones. The second one is a checkpoint system to save the state of the experiment and allow the user to restore it in the following days.

We spent some time to search for the most reliable way to build a transportable track where to train Cozmo efficiently without influencing it through the presence of reflections, but providing a path as close as possible to reality. We also designed it by using grey cotton fabric as a baseline and a white paper tape of width equal to 2.5cm to draw the lane.

Before starting the experiment, we formalised the problem as an MDP obtaining the following setup:

State Space The current state observation consists of two 64×64 grayscale images which represent the situation before and after the action respectively. We decided to resize images provided by Cozmo SDK for memory reason due to the experience replay memory needed for off-policy algorithms.

Action Space We used an array of two real values to represent how the robot can interact with the surrounding environment: the first value represents the desired speed with a range between 0 and 1, while the second one describes the vehicle steer with a range between -1 and 1. Both values are the result of a simplification useful in the learning phase. In practice, the system designed converts these values to correctly interact with the robot. The maximum speed reachable is 150mm/s.

Reward Function The reward function defines which is the purpose of the task to solve. After an analysis of the literature available, we decided to formalise this function as the total length of track crossed by the robot after each action taken. Despite its simple formulation, this choice revealed to be effective. Furthermore, this choice leads to match reward with length covered, useful fact to support the developer in quantifying agent improvements and have better feedback. The episode concludes when the robot reaches an irrecoverable situation: for this reason, the reward of the last action of the episode is equal to zero.

3. Experiments

The path that led to the final implementation of the algorithm allowed us to detect some specific requirements and to overcome them with appropriate countermeasures. We decided to firstly implement a simplified environment to test functionalities and reinforcement learning algorithms we aimed to use in Cozmo environment. We used the inverted pendulum swing-up problem, a classic problem in the control literature and available in OpenAI Gym. The original implementation of this environment consisted of observations with values related to the current angle and speed of the pendulum. For this reason, we decided to build a wrapper for the original environment to receive observations as raw pixels, since the goal was to apply the same considerations and the same convolutional neural networks that we would use in the Cozmo environment.

The results of the experiments carried out using hyperparameters taken from the available literature showed that SAC algorithm has a better performance than the DDPG one, both in terms of stability and number of episodes to achieve a working policy. Therefore, we decided to implement the SAC algorithm to carry out experiments in the Cozmo environment.

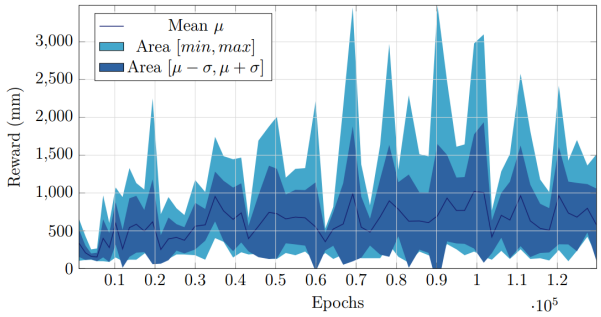

Figure 2. SAC CozmoDriver-v0 Test Reward Plot. The graph reports mean, standard deviation range and min-max range of the average reward obtained from 10 test episodes every 50 episodes.

We opted for a neural network with three convolutional layers with 16 filters of 3×3 dimension, a stride of 2, zero padding and two fully-connected layers with 256 features. We applied batch normalisation after each convolutional layer and used the Rectified Linear Unit function (ReLU) as non-linearity.

We managed to complete a whole set of 3000 episodes exploiting the SAC algorithm to solve the autonomous driving task with Cozmo. Taking into account waiting times between episodes and charging times, we managed to complete one experiment in almost one week, after about 1.3×10^5 epochs of learning.

The agent reached the most significant results in the testing phase presented in figure 2 where we outlined minimum and maximum values obtained in every set of ten episodes together with the mean and the standard deviation. Following this approach, we noticed a performance increase with a maximum mean of almost 1 metre. Furthermore, the maximum value reached among all tests episodes was equal to almost 3.5 metres which equals more than one complete tour of the track. It is noticeable that the results are not stable as we expected from the experiments with the inverted pendulum environment: the reward values do not improve uniformly with increasing epochs. However, carrying out the experiments episode by episode, we noticed a marked improvement in the performance obtained in the tests. The robot learned to approach turns and to stay on the lane of a straight road.

4. Conclusions

The plot reports a visible improvement in the maximum length of track travelled before the disengagement of the user. Despite these improvements, the agent was not able to learn how to drive securely and stably, as we can notice from the unstable growth of the mean reward. These facts made us reflect on the critic points of our experiment setup that may have had a role in the instability of the results obtained.

We localised two major problems which, in our opinion, have had a particular influence on the results obtained. The first factor was the amount of RAM available in the development machine. This limitation forced us to decrease the size of the replay memory and a consequent early deletion of less recent episodes. Analysing the plots, we noticed that this fact translated in the increase of the temperature parameter: this symptom underlines the need for the algorithm to explore more the solution space. The second major problem was the limitation of the camera sensor on the robot, particularly its viewing angle. The features offered by the Anki Cozmo camera proved to be inadequate to observe the track we designed. We noticed this fact after many episodes when the robot started to improve its performance: it began to adopt a wave behaviour on the straights, interpreting the vision of a single road line as a curve.

4.1. Future Work

Our proposals about future improvements to the project stem from the weakness in our approach. It could be interesting to execute these algorithms on a device with a bigger RAM, but also to design this approach with a Variational Auto-Encoder (VAE) to reduce the dimensionality of the information retrieved during experiments.

It may be useful to enhance sensors installed in the self-driving robot. In addition to the possibility to build up a custom Donkey Car, we believe that one valuable alternative to Anki Cozmo could be Anki Vector, which mounts a 720p camera with 120° Ultra Wide FoV. It could be interesting to perform reinforcement learning algorithms with the usage of the renewed front camera together with the infrared laser scanner on-board to investigate approaches to data fusion in reinforcement learning.

Another intriguing research path consists of an investigation about the application of model-based reinforcement learning algorithms to autonomous driving. A more in-depth review of the literature to better understand the feasibility of this approach, focusing on its strengths and weaknesses compared to model-free ones, can be the right choice to make the next step in this research field.

References

[1] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[2] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.