Reinforcement Learning-based Autonomous Driving with Proximal Policy Optimization

Bence Zsombor Hadlaczky
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
hadlaczkybzs@edu.bme.hu

Dávid Antal Novák
Department of Control Engineering and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
novak.antal.david@edu.bme.hu

Balázs Peisz
Department of Telecommunications and Media Informatics
Budapest University of Technology and Economics
Budapest, Hungary
peiszbalazs@edu.bme.hu

***Abstract*—In the recent decade, the development of reliable and robust Driving Assistant Systems and Autonomous Driving Systems has become increasingly important for the Automotive Industry. However, making a vehicle to drive itself is not an easy task. One approach to this problem is Reinforcement Learning. In this paper, we present an RL-based algorithm that is capable of learning and performing autonomous control of a vehicle (robot) in a simulated environment. Our implementation is at the same time an entry to the well known AI Driving Olympics (AIDO), which is a competition for autonomous driving robots in real time and simulated environments. We chose to participate in the lane following section of the mentioned competition with our agent based on the RL algorithm Proximal Policy Optimization (PPO) which has proved to be a powerful method to train AI policies in challenging environments.**

***Keywords*—reinforcement learning, autonomous driving, PPO, convolutional neural networks, simulation, deep learning**

I. Introduction

Nowadays one of the most researched topics in the field of information technology is artificial intelligence (AI). It is not only a great research topic, but also a field which has more and more real world applications such as natural language processing (NLP) in machine learning based translators or AI powered stock pricing predictions just to name a few. As years pass by there will be even more practical, everyday applications of this field, one that we are particularly looking forward to is none other than self driving vehicles. Self driving vehicles, such as autonomous cars are gaining much attention these days and of the techniques which power their growing popularity as well as effectiveness is a subfield of machine learning, which is called deep learning.

Deep learning is a family of algorithms and machine learning architectures which can be described as a specific field of supervised learning. Deep learning uses deep neural networks in order to achieve previously unseen performance at many different tasks such as image recognition, semantic segmentation or generative modelling in a way which at the relevant tasks replicates or even surpasses human performance. One of the most innovative fields of deep learning research deals with convolutional neural networks (CNN) [1] an effective architecture which is widely used in image and as well as time series related tasks. The usage of CNNs in applications related to autonomous cars is well documented and researched [2] with many great accomplishments e.g. successful implementation of lane following in different scenarios or road segmentation.

Whereas the application of CNNs is based on the paradigm of supervised learning in our implementation of robot control we based our solution on another family of algorithms, which is reinforcement learning (RL). Reinforcement learning is a whole subfield of machine learning unlike specific architectures such as CNNs with its own methodology, family of algorithms and different applications. In reinforcement learning unlike supervised or unsupervised learning an agent is set to learn only from its experiences based on its interactions with the world known as the *environment* [3]. In order to achieve higher performance than it can be done by utilizing traditional RL algorithms such as dynamic programming or temporal difference learning, we decided to use deep reinforcement learning architectures, which use the previously mentioned deep neural networks and the concept of deep learning in the field of reinforcement learning. By doing so it has already been demonstrated that agents learning as such can achieve far better performance at most of the tasks represented in the field [4]. Moreover agents trained with deep RL methods were able to achieve superhuman performances at many tasks, which proved to be unscalable for regular RL algorithms like the game of Go [5], or complex computer games such as StarCraft II [6].

In this paper we build on the foundations of the theoretical background listed below and the recent advancements mentioned above in order to propose a Proximal Policy Optimization (PPO) [7] based solution to the lane following task of the AI Driving Olympics. In order to train our agent for the specific lane following task we use the simulation environment provided by the organizers of the competition named Duckietown [8] (which was originally developed for educational purposes) and as reinforcement framework Stable Baselines.

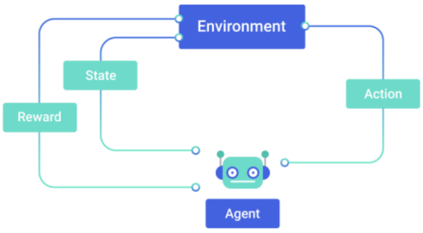
The rest of this paper is organized as follows. Section II. lays the foundations of the theoretical background of this paper and also introduces the Duckietown simulation environment, Section III. describes the previous solutions for the task, Section IV. describes our proposed solution for the task and the work we've done and finally Section V. evaluates our work as well as draws further conclusions regarding the task and our specific implementation.

II. Background

A. *Reinforcement Learning(RL)* [3]

Reinforcement learning is learning by interacting with an environment. An RL agent learns from the consequences of its actions, rather than from being explicitly taught and it selects its actions on basis of its past experiences (exploitation) and also by new choices (exploration), which is essentially *trial and error* learning. The reinforcement signal that the RL-agent receives is a numerical reward, which encodes the success of an action's outcome, and the agent seeks to learn to select actions that maximize the accumulated reward over time.

To formulate a basic Reinforcement Learning problem, we need to introduce some key terms that help to describe the basic elements of this problem. An *environment* is the physical (or simulated) world in which the agent operates. A *state* is the current situation of the agent. The *reward* is a feedback from the environment. The agent acts according to a *policy* which is a method that maps the agent's state to actions. The *value* is the future reward that an agent would receive by taking an action in a particular state.



1. Figure: Reinforcement Learning diagram

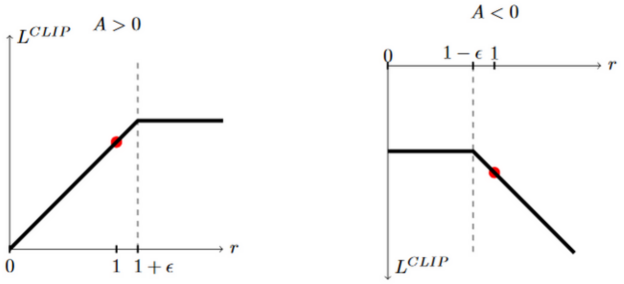
The mathematical frameworks to describe an environment in RL are the *Markov Decision Processes*. An MDP consists of a set of finite environment states *S*, a set of possible actions *A(s)* in each state, a real valued reward function *R(s)* and a transition model *P(s',s | a)*. However, real world environments are more likely to lack any prior knowledge of environment dynamics. In such cases, so-called model-free RL methods come particularly useful.

B. *Proximal Policy Optimization (PPO)* [7]

To quote from OpenAI on PPO: *'Proximal Policy Optimization, which performs comparably or better than state-of-art approaches while being much simpler to implement and tune.'* PPO is a model-free, on-policy, actor-critic, policy gradient method. This algorithm is a type of policy gradient training that alternates between sampling data through environmental interaction and optimizing a clipped surrogate objective function using stochastic gradient descent. The clipped surrogate objective function improves training stability by limiting the size of the policy change at each step. The objective function is the following:

$$L^{CLIP}(\theta) = \hat{E}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\,\hat{A}_t)]$$

where θ denotes the policy parameter, $\hat{E}_t$ is the empirical expectation over timesteps, r_t is the ratio of the probability under the new and old policies respectively, $\hat{A}_t$ is the estimated advantage at time *t* and ε is the so-called clipping ratio, which is a hyperparameter. If the probability ratio between the new policy and the old policy falls outside the range (1 — ε) and (1 + ε), the advantage function will be clipped. ε is set to 0.2 for the experiments in the PPO paper. With this clipped objective it is possible to restrict large policy updates and consequently avoiding huge drops in performance during learning.



2. Figure: Advantage function clipping [7]

In our project we mainly used the PPO2 algorithm in the Tensorflow based Stable Baselines RL Library which is the implementation of the base PPO made by OpenAI for GPUs.

C. *Duckietown Platform*

The Duckietown platform comprises self-driving vehicles (*Duckiebots*) and model urban environments (*Duckietowns*). Duckiebots are minimal autonomy platforms built to transport duckies, the citizens of Duckietown. They are equipped with a camera and all computation is done onboard. The wheels are powered with DC motors. The bots use primarily computer vision to drive down lanes, avoid pedestrians (duckies), and navigate intersections. To follow the lanes they look at the road and detect the road markings. To navigate in DuckieTowns they read signage, including road signs and traffic lights, and communicate with other robots to coordinate. The Duckietowns are structured urban environments built for Duckiebots to operate successfully. Duckietowns are modular, which means they can be expanded at will. Duckietowns are made of two layers: the floor layer and the signal layer. As long as appearance specifications are met, Duckiebots will always behave appropriately.

3. Figure: Duckietown scene

Gym-Duckietown is a simulator for the Duckietown Universe. It places an agent (Duckiebot) inside of a virtual instance of a Duckietown. The virtual Duckietown contains loop of roads with turns, straights and optionally intersections, obstacles, traffic and pedestrians. The Gym is highly customizable. It is possible to create your own map for different driving tasks: Lane Following (LF), Lane Following with Vehicles (LFV) and Lane Following with Vehicles and Intersections (LFVI). Moreover, the Duckietown Gym is mostly based on the OpenAI Gym which provides compatibility with most of the available Reinforcement Learning Libraries that makes implementing different RL algorithms less cumbersome. In this paper we only dealing with the Lane Following task.

III. Previous solutions

As we mentioned before, there are numeral solutions for AI algorithms in. One algorithm for example is the **Deep Deterministic Policy Gradient (DDPG)**. This approach was proposed by the Duckietown Team as well. However this algorithm is notoriously sensitive for hyperparameters and this trait makes hyperparameter optimization particularly difficult. A different solution, proposed in [9] was the **Deep Q-Network (DQN)**. This is Q-learning with Neural Networks that approximate Q-values for each action based on the state. Another possible solution was the synchronous, deterministic **Advantage Actor Critic (A2C)**. However **Proximal Policy Optimization (PPO)** uses ideas from A2C, so it can be considered an upgrade. So, we have decided to work with PPO.

IV. Our proposed solutions

In this section we introduce our PPO based solution for the task and compare it to other baseline models within the known reinforcement learning frameworks. Furthermore this section gives a brief summary of the conditions and flow of our work during the semester.

A. *Related baseline solutions*

At the start of our work we inspected and tested the baseline RL solution of the AIDO lane following challenge and even trained the given agent, but despite all of our efforts we could not develop it further because of the hyperparameter sensitive nature of the used DDPG algorithm. (This is depicted by the DDPG results file in our GitHub repository) Because of this we had to move on to another reinforcement learning framework and an alternative algorithm. That was the main reason that we started to experiment with PPO and eventually chose it as the base of our solution.

First we worked with the Stable Baseline3 framework which was recommended to us by our advisor. While using the framework we worked with Python 3.6 and PyTorch 1.71. on Google Colab utilizing a GPU with 12 GB RAM. Although we tried out many configurations and parameters to make the baseline PPO algorithm work, because of the limitations posed by the lack of hardware at our disposal (most notably the lack of a powerful GPU with more than 16 GB of RAM), we could not make the desired algorithms work using Stable Baselines3.

Since Stable Baselines3 was too computationally demanding to work with through Colab we downgraded the framework to the original Stable Baselines. We also changed the deep learning framework to Tensorflow 1.15 because it was the only supported solution in the RL framework. After we gained some familiarity with the original Stable Baselines we were able to train the baseline PPO agent for 100000 timesteps. The reason why we were unable to train it further, (although we acknowledge the fact that most RL agents will not learn effectively until 500000 or even 1 million timesteps) was attributed to the lack of computational power and consequently the failure of the Colab runtime. Therefore we were only able to train and to run hyperparameter optimization.

The default values for the PPO in the stable baselines are:

- Discount factor: 0.99,
- Entropy coefficient for the loss calculation: 0.01,
- Learning rate: 0.00025
- Value function coefficient: 0.5
- Maximum value for the gradient clipping: 0.5
- Factor for trade-off of bias vs variance for Generalized Advantage Estimator: 0.95
- Number of training mini batches per update:
- Number of epoch when optimizing the surrogate: 4
- Clipping parameter: 0.2

The reason why we were unable to train it further, (although we acknowledge the fact that most RL agents will not learn effectively until 500000 or even 1 million timesteps) was attributed to the lack of computational power and consequently the failure of the Colab runtime. Therefore, we were only able to train the baseline agent for 100000 timesteps. After evaluating the baseline model we got -1020.02 for the mean reward and for the standard reward 52.8774.

After running the baseline algorithm, we went on running a hyperparameter optimization using Optuna (*suggested in the Stable Baselines documentation*) to get the optimal values for each hyperparameter to increase the performance of our agent. We ran the optimization for 100000 timesteps (because of the mentioned hardware limitations) and managed to get a suboptimal solution for the task, which was better than the baseline algorithm. (The detailed steps of the optimization process can be observed in our repository.) The parameters are the following:

- Discount factor: 0.91559,
- Learning rate: 9.5622e-06
- Value function coefficient: 0.77015
- Factor for trade-off of bias vs variance for Generalized Advantage Estimator: 0.92814
- Number of epoch when optimizing the surrogate: 14
- Clipping parameter: 0.23316

Using these hyperparameter we ran a 100000 timestep long training and a 10-episode long evaluation. The results are the following: mean reward is - 267.3280 and the standard reward was 1084.8454. Although we were not able to reach positive mean reward, an improving tendency can be observed. This is only a small increase, but due to our limited resources we were unable to refine this solution.

Further into our work we tried to migrate our progress to TensorForce, a different RL framework with hopes of a better runtime experience and perhaps the chance to train our agent for more timesteps. We happened to have the same issue as before. The 12GBs of RAM in Colab happened to be insufficient, like before. The default values in the TensorForce for PPO:

- Learning rate: 0.001
- Absolute/relative fraction of batch timesteps to subsample: 0.33
- Likelihood-ratio clipping threshold: 0.25
- Discount factor for future rewards of discounted-sum reward estimation: 0.99

The training lasted for 100000 timesteps (1000 episodes with 100 timesteps). After evaluation, we got an episode mean reward of -313.9119. This is an expected value with our limited resources.

V. Conclusion

As it can be seen in the previous section we managed to train three different agents, two baselines and one optimized and showcased the effectiveness of not only the PPO algorithm but the use of hyperparameter optimization. Despite all the difficulties imposed by the hardware limitations we consider our work as a success and as a process which enabled us to learn much about both deep learning as well as reinforcement learning.

References

[1] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner. Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, 1998.

[2] M. Bojarski, D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, K. Zieba. End to End Learning for Self-Driving Cars. 2016.

[3] R. Sutton, A. Barto. Reinforcement Learning: An Introduction. The MIT Press, Second edition. 2018.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, Deep Reinforcement Learning: A Brief Survey. In IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26-38. 2017.

[5] D. Silver, A. Huang, C. Maddison *et al.* Mastering the game of Go with deep neural networks and tree search. *Nature* **529,** 484–489. 2016.

[6] O. Vinyals, I. Babuschkin, W.M. Czarnecki *et al.* Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575,** 350–354. 2019.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov. Proximal Policy Optimization Algorithms. 2017.

[8] J. Tani, L. Paull, M. Zuber, D. Rus, J. How, J. Leonard, A. Censi. Duckietown: An Innovative Way to Teach Autonomy. Advances in Intelligent Systems and Computing. 560. 2016.

[9] P. Almási, R. Moni, B. Gyires-Tóth, Robust Reinforcement Learning-based Autonomous Driving Agent for Simulation and Real World, 2020

[10] GitHub repository for the project: https://github.com/Comicboy/deep-rl-for-the-duckietown-aido_lf-challenge Downloaded