Self-driving in the Duckietown environment

Önvezető autózás a Duckietown környezetben

Márton Kerekes
Department of Telecommunications
and Media Informatics
Budapest University of Technology and Economics
Budapest, Hungary
kerekesmarton@edu.bme.hu

***Abstract*—In the future, autonomous driving cars will be much more important to people than it is nowadays. Advanced driver-assistance systems are already in used in many vehicles and it's just a matter of time before fully self-driving cars are on the road. One of the most used algorithm types is reinforcement learning and what I used to try it myself out in this field. In this paper I present a simple Lane following task in Duckietown a well-known autonomous driving environment. Present PPO and Impala algorithm for training the self-driving model. Rewards and observation modifications in more depth. And the used Ray reinforcement learning framework/library.**

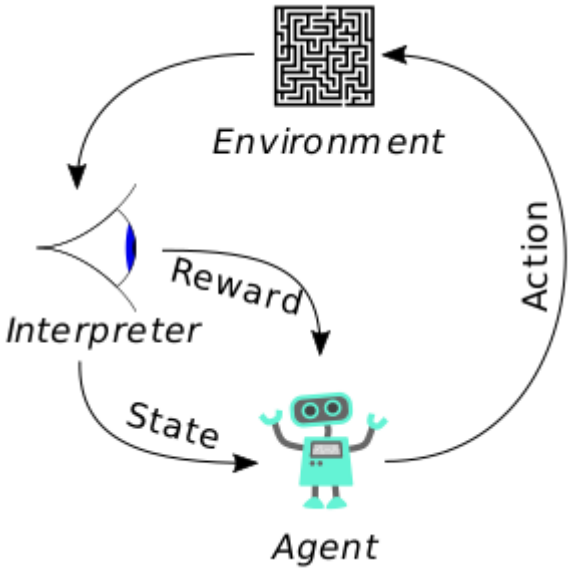
A jövőben az önvezető autók sokkal fontosabbak lesznek az emberek számára, mint manapság. A fejlett vezetéstámogató rendszereket már most is sok autóban használják, és csak idő kérdése, hogy mikor lesznek teljesen önvezető autók az utakon. Az egyik leggyakrabban használt algoritmustípus a megerősítéses tanulás, és amit én magam is használok. Ebben a cikkben egy egyszerű sávkövetési feladatot mutatok be a Duckietownban, egy jól ismert autonóm vezetési környezetben. Bemutatom a PPO és az Impala algoritmust az modell tanításához. Jutalmak és megfigyelési módosítások részletesebben. És a használt Ray megerősítő tanulás keretrendszer/könyvtárat.

Keywords—Self-driving cars, Duckietown, reinforcement learning, RL, Impala, PPO, Ray, deep learning

I. Introduction

Machine learning nowadays is one of the hottest topics in the field of researcher. It is not hard to see why since it can give a solution to problems, we thought it is impossible. One part of it is deep learning which is used mostly to solve complex problems and give a solution for it. Not just categorize images or find where something is on the image, but generate human voices, make images from nothing and to drive cars like a human or better. A computer driving a car by itself was mostly just a fantasy in the previous few decades.

Deep learning three main subcategory is supervised learning, unsupervised learning, and reinforcement learning. Reinforcement learning is algorithms, where the model learns in an environment. The algorithms are based on rewarding desired behaviors or punishing undesired ones. Training these are based on so called agents who will decide the next action. The action is sent to the environment to change its state by the given action. Then, an observer gives reward based on how good the given action from the agent was and update the agent state based on the given reward.



1. Figure: Reinforcement Learning illustration

II. Background

A. Related work

The idea of self-driving vehicle is actually pretty old. [1] [2] [3] [4] Experiments with self-driving cars started as early as 1939. Since then, artificial intelligence (AI) has been applied to the field and huge progress has been made. Reinforcement learning is used most of the time for these kinds of problems but supervised one like imitation learning [5] is used too. Lane following is a standard task in autonomous vehicles [6] [2]. More complex problems include lane following with vehicles is when other cars are on the road [6] [7] [8]. Lane following with other vehicles and intersections are the most complex where not only other vehicles are on the road, but you have intersections [9] [10] [11].

B. Duckietown environment

The Duckietown environment is a platform where you can study machine learning, AI and train/test your autonomous self-driving vehicles. Duckietown hosting

AI Driving Olympics (AI-DO) each year twice, where you can compete with your DuckieBot against competitors around the world. The goal is to "The AI-DO are a set of competitions with the objective of evaluating the state of the art for ML/AI for embodied intelligence". The environment has virtual and physical interface too.

Both virtual and physical the DuckieBot are equipped with camera, DC motors, and the minimal computation power needed for driving. The only way the bot get around is with the camera since it is the only connection is having to the outside world. The actions are calculated by the onboard chip based on the camera output. Gym-duckietown is the name of the virtual environment. It is built so that you can use the OpenAI gym with it for easier training. This helps to use Reinforcement Learning libraries like Ray.

III. Implementation

For implementation the Ray reinforcement learning library(Rllib) was used and the Ray Tune for fine tuning the model with PyTorch and Python 3.8.

A. *Wrappers*

The observation space is the image we get from the camera, and actions are based on it. These wrappers are around the observation/action space, and they change the output of it. Most of the used wrappers are from the original Gym-Duckietown repository with slight adjustment.

Two custom wrappers have been added to the model. One of the most important aspects of RL is the reward calculation. These calculations are made by *DtRewardWrapper*. The other one is *CropImageWrapper* for remove unnecessary part of the image. The top third side of the image do not contain any necessary information since only the sky is visible.

B. *Reward*

Rewards are the only thing that the agent see, so it is one of the most important parts of RL, as mentioned. The reward is calculated as follows:

$$R = 100 * R_d + R_{cd} + R_{ca}$$

- R_d is the reward for the traveled distance
- R_{cd} reward based on how close to the center of the tracking lane
- R_{ca} reward based on the angle to the middle of the tracking lane

This calculation only works if the agent is in the right lane, otherwise the reward is -30.

C. *Actions*

A DiscreteWrapper is used for the action space. So, the action space has a discrete distribution, since you can only turn left, right, or just go straight ahead. Three actions are available from which the agent can choose.

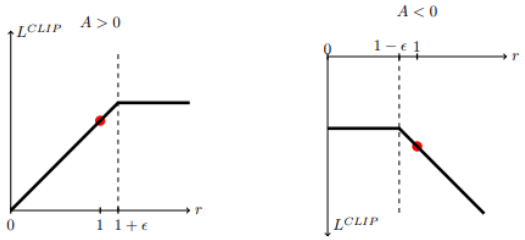
Available actions:

- 0 – Speed is half of the maximum speed, and the wheels are straight
- 1- Speed is 0.4 of the maximum speed and the wheels are full right
- 2- Speed is 0.4 of the maximum speed and the wheels are full left

IV. Algorithm

A. *Proximal Policy Optimization*

In short, PPO [12] is a policy gradient method for reinforcement learning, which alternate between sampling data through interaction with the environment and optimizing the reward function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample. PPO have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity. PPO use clipped surrogate objective function to improves training stability by limiting the size of the policy change at each step:



2. Figure: IMPALA illustration [12]

$$J^{\text{CLIP}}(\theta) = \widehat{E_t}\left[\min\left(r_t(\theta)\widehat{A_t}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_t}\right)\right]$$

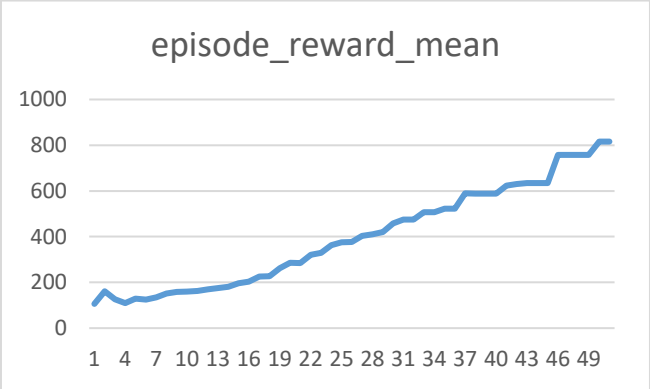
- θ is the policy parameter
- $\widehat{E_t}$ denotes the empirical expectation over timesteps
- r_t is the ratio of the probability under the new and old policies, respectively
- $\widehat{A_t}$ is the estimated advantage at time t
- ε is a hyperparameter, usually 0.1 or 0.2

1) *Training*

Tuned parameter for training:

- Learning rate: 0.0001- 5e-6
- Batch size: 1500 | 2000
- The PPO clipping: 0.28 - 0.32
- SGD batch size: 150 | 200

Average reward during training:

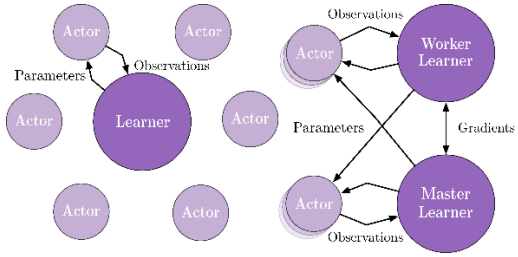


3. Figure: Rewards per iteration

It was only trained for a short amount of time, but we can still see how the agent learns.

B. *Importance Weighted Actor Learner Architecture*

In short, IMPALA [13] is an off-policy actor-critic framework that decouples acting from learning and learns from experience trajectories using V-trace. Not only uses resources more efficiently in single-machine training but also scales to thousands of machines without sacrificing data efficiency or resource utilization. Most efficient when multiple actors are used.



4. Figure: IMPALA illustration [13]

V-trace is an off-policy agent-critic reinforcement learning algorithm that helps tackle the lag between when actions are generated by the actors and when the learner estimates the gradient.

1) Traning

Unfortunately, Rllib was unable to fully train the model, because not matter what parameter I used after a specific timestep (around ~100k) it stop with a *"Error raised reading from queue"* it is a bug maybe in Ray itself since I saw many issues on GitHub with same problems.

V. Conclusion

In this paper, two reinforcement learning algorithm was used to train with the Rllib on the DuckieTown environment. Testing/Training with the PPO and IMPALA algorithms. A discrete action space for faster training with only 3 possible actions. Custom observation space and rewards. Used Ray Tuning for better fine tuning the agents. Most of my effort and time goes to figure out the environment and to connect it to Rllib. There were just too many bugs I had to find and fix like too new python version (10/11), modules incompatibility, etc.

VI. References

[1] S. a. P. D. A. Baluja, "Expectation-based selective attention for visual monitoring and control of a robot vehicle," Elsevier, Robotics and autonomous systems, 1997.

[2] S.-Y. a. L. J.-H. a. C. D.-H. Oh, "A new reinforcement learning vehicle control architecture for vision-based road following," IEEE, 2000.

[3] G. a. C. S. Antonelli, "Experiments of fuzzy lane following for mobile robots," IEEE, 2004.

[4] A. R. Bacha, "Line detection and lane following for an autonomous mobile robot," Virginia Tech, 2005.

[5] J. a. S. R. a. G. C. a. S. S. a. R. D. a. N. N. a. M. P. a. M. S. a. G. N. a. S. A. a. o. Hawke, "Urban driving with conditional imitation learning," IEEE, 2020.

[6] W. a. M. M. S. a. K. F. a. S. I. a. M. M. Tsui, "Soft-computing-based embedded design of an intelligent wall/lane-following vehicle," IEEE, 2008.

[7] B. a. J. A. a. Z. P. a. M. P. a. G. C. a. H. S. a. M. H. Osi. ski, "Simulation-based reinforcement learning for real-world autonomous driving," IEEE, 2020.

[8] A. a. H. J. a. J. D. a. M. P. a. R. D. a. A. J.-M. a. L. V.-D. a. B. A. a. S. A. Kendall, "Learning to drive in a day," IEEE, 2019.

[9] G. a. H. J. a. L. Z. a. L. L. Wang, "Harmonious lane changing via deep reinforcement learning," IEEE, IEEE Transactions on Intelligent Transportation Systems, 2021.

[10] S. a. J. D. a. W. X. Wang, "Deep reinforcement learning for autonomous driving," IEEE, arXiv preprint arXiv:1811.11329, 2018.

[11] X. a. Y. Y. a. W. Z. a. L. C. Pan, "Virtual to real reinforcement learning for autonomous driving," IEEE, arXiv preprint arXiv:1704.03952, 2017.

[12] J. a. W. F. a. D. P. a. R. A. a. K. O. Schulman, "Proximal policy optimization algorithms," arXiv , arXiv preprint arXiv:1707.06347, 2017.

[13] L. a. S. H. a. M. R. a. S. K. a. M. V. a. W. T. a. D. Y. a. F. V. a. H. T. a. D. I. a. o. Espeholt,

"IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures," PMLR, 2018.