Self Driving In Duckietown

Önvezető autó tanítása Duckietown környezetben

Dániel Veress
Budapest University of Technology
and Economics
verdani19@gmail.com

Marcell Dancsó
Budapest University of Technology
and Economics
dancsomarci@gmail.com

Milán Konor Nyist
Budapest University of Technology
and Economics
nyist.milan78@gmail.com

Abstract

Autonomous driving has been a prevalent problem in the field of Artificial Intelligence. Many environments have been created to solve this issue in the virtual space. With the help of the **Duckietown** [1] environment, we have trained 2 **reinforcement** learning and 1 **imitation** learning model. We used state-of-the-art algorithms PPO [2] and DQN [3] to train our reinforcement learning agents. Our goal was to train an agent that could solve the lane following problem. We think that our mission was accomplished during our research and all of **our agents are able to drive for an extended amount of time without leaving the track**.

Az autonóm vezetés korunk egyik legizgalmasabb kérdése a Mesterséges Intelligencia területén. Sok olyan környezetet hoztak létre, melyben virtuálisan tudjuk szimulálni az önvezetést. A kutatásunk során a Duckietown környezet segítségével 2 megerősítésés tanulású és 1 imitációs tanulású modellt tanítottunk be. A két megerősítéses tanulás során a legkorszerűbb algoritmusokat használtunk, név szerint a PPO valamint, az DQN algoritmusokat. Célként tűztük ki, hogy az ágenseink sikeresen megoldják a vonalvezetés problémáját. Úgy gondoljuk, hogy teljesítettük célunkat és kimondhatjuk, hogy mindegyik ágensünk képes hosszabb ideig az úton haladni anélkül, hogy letérne arról.

1 Introduction

As of this year (2022), autonomous driving is a great challenge to tackle with artificial intelligence. To overcome this task we need huge amounts of data.
The two main approaches to autonomous driving are gathering data from the real world, and a much cheaper option to gather data in a virtual environment, which can be later used in real-life circumstances. One of these virtual environments is called Duckietown [1], developed at MIT. In this environment, the user is able to drive a duckie-bot virtually and gather data. The created data set can be used for imitation learning where the agent learns to drive like the user who used the virtual environment.
Another option is to utilize supervised learning. In that case, the agent learns to drive in the virtual environment by trying to get the highest reward possible. We are responsible for determining what action is worth a positive reward and what action deserves a negative one.

2 Background And Motivation

The decision to work on this project came to all of us very easily. Every member of the research team follows closely the day-to-day advances of autonomous driving. We felt like we had a good grasp on the subject and wanted to contribute to the field in a small but meaningful way.
Experimenting with the famous environment of gym Duckietown [4] and an opportunity to submit our own agent to the lane following competition made it even more appealing. Therefore, we made several agents with different learning methods in order to get a respectable placement in the competition.
We went as far as to test our agents against other agents that were trained with state-of-the-art algorithms such as Proximal Policy Optimization [2].

3 Implementation

Coming up with an architecture seemed straightforward for the imitation learning method. After managing to build our own data set, (more on that in Data Gathering And Cleaning) we had to construct a neural network that could handle image inputs. We used convolutional layers [5] to make that happen. Another question was about the action space and whether we should train using classification or regression. In order to get the best results we experimented with both models and came to the conclusion that the classification model learns faster and yields a smoother experience.
The architectures for reinforcement learning were predetermined since we used ready-made algorithms. The task here was to decide which ones to test out. We landed on PPO [2], A2C [6]. Despite being similar [7] PPO produced far greater results, so we decided to give a shot to another type of algorithm: DQN [3], as we wanted to try out both of the big branches of reinforcement learning: q-learning, and policy learning. More details about the models can be found in Training.

3.1 Data Gathering And Cleaning

As mentioned above, the machine learning algorithms we worked with are imitation learning and reinforcement learning. The latter does not require any input data, since it will learn in the environment while driving.
On the other hand, imitation learning is all about input data and its quality. The input data consists of two parts. A snapshot of the environment that was taken while the user drives, and a label that tells what actions were performed at that moment. An action consists of forward velocity and steering angle. Both values are in the range of $\left[-1.0, 1.0\right]$.
Each snapshot contains a lot of noise and unnecessary data that would hold back the agent's learning progress, you can see an example in Figure 1. The vital information in the image is the lane and the position of the car.

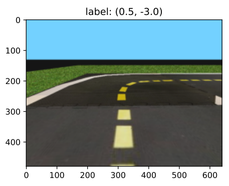

Figure 1: Original snapshot with the label, taken in the Duckietown environment.

With a preprocessing pipeline, we managed to remove all unnecessary data. The pipeline has four stages: downscaling, cropping, grayscaling, and thresholding. Running the original images through the pipeline results in the output pictures shown in Figure 2.

After constructing this architecture, the research team started gathering data by driving in the environment and saving every frame which was then processed. We gathered around 30 Gbs of raw data which was reduced to 11 Gbs after going through the preprocessing pipeline.

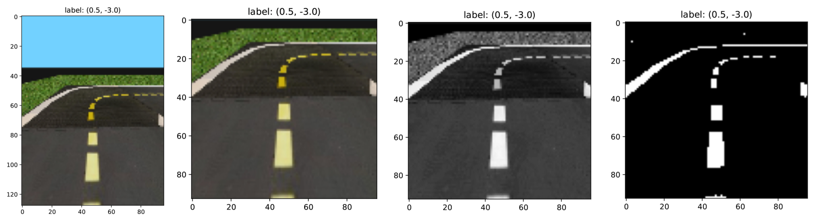

Figure 2: The preprocessing timeline shown in each iteration. Downscaling, cropping, grayscaling, thresholding in this order.

3.2 Training

3.2.1 Imitation learning training

The data was gathered with a script that took discrete input from the human driver. As a result the driving experience is a bit choppy. So first we tried building a regression model for predicting the velocities and steering angles, as in theory it can interpolate between the different inputs (go forward, turn right and left), and yield to an overall smoother experience. The results were great; on paper it interpolated perfectly between the actions, but the numbers were always slightly different than the labels. This resulted in not turning fast enough in corners despite correctly identifying the situation.

To correct this we tried predicting the actions (forward, right, left), with a classification model. This approach was a lot more successful.

3.2.2 Baseline algorithms training

For these algorithms, namely PPO and DQN, the training process is fairly straight forward with the provided stablebaselines3 [8] python library. Tweaking the environment, however presents a much harder task, as reinforcement learning algorithms are extremely sensitive to their hyperparameters.

3.3 Evaluation

3.3.1 Imitation learning evaluation

The classification model could not only navigate the training map, but also other unseen maps even with moving obstacles, but also with a reduced data set of only a 100MB.

Of course, the generalizing capabilities dropped with this reduction, but it was still good enough to tackle the challenges mentioned above most of the time.

Furthermore two popular hyperparameter optimization technique was used to enhance the already working model, namely Hyperband [9] and Randomsearch [9] which managed to push the accuracy scores a few percent higher. Out of the two Hyperband clearly outperformed the RandomSearch, but took significantly more time to complete.

3.3.2 Baseline algorithms evaluation

We have found PPO much more suited to the task, as even after a few hundred thousand iterations it was already showing promising results, so we put a lot more computing time into further optimizing it's parameters and the final, best model after 1.4million cycles is able to navigate the maps with ease.

The hyperoptimization algorithms tried with imitation learning were way too computation heavy to run with reinforcement learning, so we could only optimize by hand.

Out of all the parameters the reward function was the most critical. The default reward function of the duckietown environment isn't so sofisticated it turns out, so we took inspiration from other people competing in the AIDO olympics.[10] One particularly tricky situation was when the agent found a bug in the reward system and only learned to turn continuously, as result it only produced the same data every frame, resulting in an endless loop even after a million iterations. A solution we found was to not let the agent stop, always giving the bot some speed, or to choose between actions like in the classification model in imitation learning.

Another parameter was the processing of frames. We tried both our pipeline method mentioned above, and other solutions [10] as well, but it didn't seem to make much of a difference.

3.4 Testing

3.4.1 Imitation learning testing

The models were tested on a separated testing data set. The results were quite promising for both regression and classification models, but the real test was the run in the environment. At first glance one might even say that the results of the regression model were better than the classification, because it only makes slight mistakes, whereas the classification model's predictions are abruptly wrong at least 90% of the time. but as it turns out it drives better.

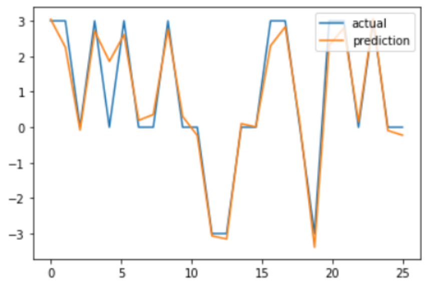

Figure 3: Regression model results

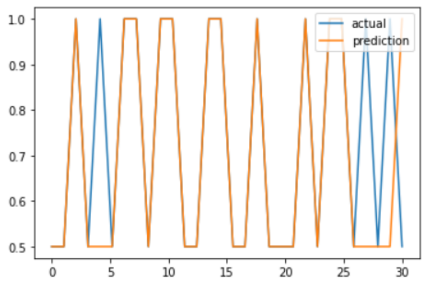

Figure 4: Classification model results

3.4.2 Baseline algorithms testing

Tensorboard was used to track the training itself, but the numbers here don't always mean that the model will perform well, only that it somewhat satisfies the reward function we gave. So every hundred thousand iterations we tested the models in the simulator.

4 Future Plans, Summary

With all the work behind us, we still feel like there is room for improvement and a lot of routes we can explore. Some of the plans we have listed are training an agent that can maneuver in a multiplayer environment, meaning that other duckie-bots are present. We are aiming to submit at least one agent to the AI-DO competition. Experimenting with new algorithms and more in-depth hyperparameter tuning is also a goal of ours.
With all these plans for the future, we mustn't forget how far we've come. Getting to learn the environment of Duckietown [1] and being able to successfully train multiple agents with different algorithms is something we are proud of. Our current agents are able to tackle the obstacles presented by lane following and are a perfect baseline for more complex autonomous driving problems.

5 Bibliography

[1] Liam Paull, Jacopo Tani, Heejin Ahn, Javier Alonso-Mora, Luca Carlone, Michal Cap, Yu Fan Chen, Changhyun Choi, Jeff Dusek, Yajun Fang, Daniel Hoehener, Shih-Yuan Liu, Michael Novitzky, Igor Franzoni Okuyama, Jason Pazis, Guy Rosman, Valerio Varricchio, Hsueh-Cheng Wang, Dmitry Yershov, Hang Zhao, Michael Benjamin, Christopher Carr, Maria Zuber, Sertac Karaman, Emilio Frazzoli, Domitilla Del Vecchio, Daniela Rus, Jonathan How, John Leonard, and Andrea Censi. Duckietown: An open, inexpensive and flexible platform for autonomy education and research. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1497–1504, 2017.

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

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[4] Maxime Chevalier-Boisvert, Florian Golemo, Yanjun Cao, Bhairav Mehta, and Liam Paull. Duckietown environments for openai gym. `https://github.com/duckietown/gym-duckietown`, 2018.

[5] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[6] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. 2016.

[7] Shengyi Huang, Anssi Kanervisto, Antonin Raffin, Weixun Wang, Santiago Ontañón, and Rousslan Fernand Julien Dossa. A2c is a special case of ppo, 2022.

[8] 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.

[9] Tom O'Malley, Elie Bursztein, James Long, François Chollet, Haifeng Jin, Luca Invernizzi, et al. Kerastuner. `https://github.com/keras-team/keras-tuner`, 2019.

[10] András Kalapos, Csaba Gór, Róbert Moni, and István Harmati. Vision-based reinforcement learning for lane-tracking control. *ACTA IMEKO*, 10(3):7–14, 2021.