Self-driving vehicle in Duckietown environment
Duckpropagation

Attila Pethő **Olivér István Farkas** **Gyula Faragó**

Abstract

In this project we tried to bring a solution to driving a small robot with vision-based reinforcement learning in a simulated environment. The robot relies on images from its front-facing camera. Our models generate continous actions that control a differential-drive robot vehicle. To train the policy we have selected two state-of-the-art algorithms (A2C and PPO). A2C is an Actor Critic model, while PPO uses Proximal Policy Optimization. In order to train these models we have to feed in the images, which we preprocess first. To train the model better for real-world conditions we used domain randomization. Further analysis of our work is discussed in the document.

Kivonat

Ebben a projektben megpróbáltunk megoldást találni egy kis robot vezetésére látásalapú megerősített tanulással egy szimulált környezetben. A robot az elöl elhelyezkedő kamerájának képeire támaszkodik. Modelljeink folyamatos cselekvéseket generálnak, amelyek egy differenciálhajtású robotjárművet irányítanak. A stratégia betanításához két modern algoritmust választottunk (A2C és PPO). Az A2C egy Actor Critic modell, míg a PPO a Proximal Policy Optimizationt használja. A modellek a környezettől kapott képkockák alapján tanulnak, amelyeket először előfeldolgozunk. Ahhoz, hogy a modell valós környezetbe átültetését elősegítsük, domain randomizálást használtunk. Munkánk további elemzését a dokumentumban tárgyaljuk.

1 Introduction

Self driving vehicles are clearly the future of transportation. If you look at Waymo's [**?**] safety report you can see two things. First it is obvious that AI driven vehicles are more safe than human driven ones. The other thing which - makes development hard - is the amount of data needed for the training of the AI. The report mentions that the AI used in the cars have learned from over 20 million driven miles. It is quite a big number and an astounding achievement given the fact that the car fleet only had 8 bigger accident during this time. The report shows that all accident had a similar cause, human error. In three times the car was stationary and someone ran into it. This clearly shows the potential of self driving vehicles.

For us, 20 million miles driven seems a lot, but to train well an AI it is not that much. This is the reason Waymo chosen to train its AI in simulation as well. The driven miles in the simulation were 20 billion. That means in order to archive better results, using a simulator is a good idea. Duckietown provides an excellent learning ground for those who want to understand the theory of reinforcment learning, and apply the theory in a real environment. It has started in a class at MIT in 2016. Duckietown has grown so much since then, it is now a worldwide initiative to learn AI and robotics. Our goal was to deep dive into the world of reinforcement learning and develop efficient learning strategies for our duckiebots.

2 Methods

2.1 Reinforcement Learning

In a reinforcement learning setting, an agent interacts with its environment by choosing actions (a_t) at every timestep, based on information called state (s_t), and receives back the next state (s_{t+1}) and the reward for the current action (r_{t+1}), which will be used to select the next action (a_{t+1}). Figure 1 summarizes this process. The agent tries to gradually improve its policy by exploring new ways to behave in the environment and observing the rewards received for its actions. The goal of the agent is to maximize the expected future rewards over a finite amount of steps. There are many ways of achieving this goal, which is reflected in the number of different reinforcement learning algorithms. A fairly simple one is Deep Q-Networks (DQN) which chooses actions based on an estimate of the action-value function that is calculated from previous steps in the environment. Despite its beginner friendly nature, DQN would not be the best choice for this type of problem, because it can only utilize discrete actions, which could potentially cause wobbling and unstable behaviour [TODO 14]. For this reason, we chose two algorithms, Proximal Policy Optimization (PPO) and a synchronous, deterministic variant of Asynchronous Advantage Actor Critic (A2C), which both support continuous actions.

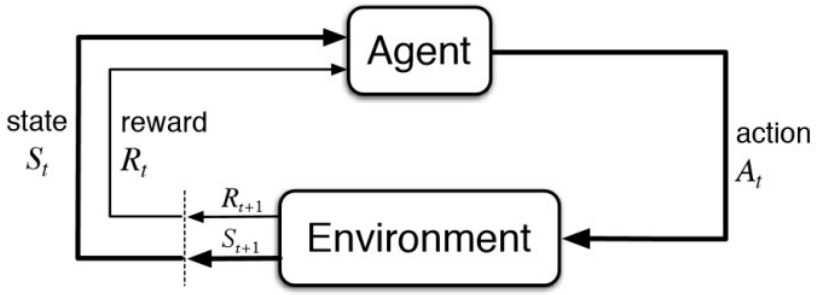

Figure 1: Illustration of Reinforcement Learning

Implementing the above mentioned algorithms would be very time consuming, so we decided to use the Stable Baselines 3 library [TODO], which is a simlified version of OpenAI baselines. The SB3 library contains several implementations of state-of-the-art RL algorithms and useful tools to train, retrain and evaluate the models. [TODO: about the policy network]

2.2 Environment

The environment gives us an 640x480 RGB image. We have to preprocess this image, because feeding the original images to the CNN would be a waste of resources because it makes training process much slower, and the network can learn from smaller images just as well.

2.2.1 Observations

Preprocessing actions are applied to the observations. These can be resizing the image, cropping the image, grayscale the image, segment the colors in it or stack the last n number of the frames.

Figure 2: (a)

Figure 3: (b)

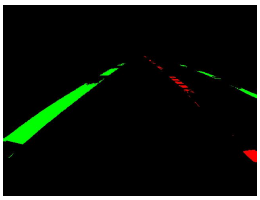

Figure 4: (c)

Figure 5: Observations from the environment. (a) shows a raw observation, (b) shows a grayscaled image, (c) shows a color segmentated image

2.2.2 Actions

Since the vehicle doesn't have turnable wheels, the turning must be achived by modifying the speed of the individual wheels. In the real life this speed is expressed as the PWM (Pulse Width Modulation) value of the applied voltage on the motor. For simplicity this value is normalized to 1 both in the simulation and the real life. For easier calculation the PWM value is not calculated dynamically it is statically coded at fix values which equates to three individual states. These states are forward, right turn, left turn.

2.2.3 Rewards

Just like any other engineering problem choosing optimal rewards are tricky and require lot of testing and empirical estimation. A poorly chosen reward function can be exploited by the agent. In our first test runs the agent got reward for staying on the path. The reward was increasing and we got long episodes which looked promising. After examining the data we discovered that the agent was rotating around in one place. It realized that the easiest way to maximize reward was not what we initially intended as a goal. To correct this we had to take in account the angle, position from the middle and speed of the agent.

2.3 Evaluation

There are many ways to measure the performance of the trained agents, and for the sake of simplicity we chose two metrics for this purpose: survival time (in timesteps) and rewards received from the environment. Survival time tells us how many steps the agent took before either it went out of the road or the episode finished. A well-trained agent should stay on the road for the whole evaluation episode. Altough this provides valuable information, this metric does not tell much about performance on its own. Hence the rewards are also stored and evaluated to gain richer knowledge about the agents perfomance. The models were trained on the 'zigzag_dists' map, because it offers a good balance between left and right turns and straight sections. In order to test how well the models learned to generalize, evaluation was done on three different maps: 'zigzag_dists', 'loop_empty', and 'udem1'.

2.4 Hyperparameter Optimization

Reinforcement learning models are extremely sensitive to hyperparameter settings. A poor choice of hyperparameters can cause unstable performance or it could prevent convergence completely. Findig the right parameters for training can be a difficult job, and it is usually not a good idea to start tuning the parameters manually. Instead you can use an optimization library, that sets up sophisticated experiments to find the right hyperparameters for your usecase.

For our project we used the Optuna library, which offers several tools to dynamically explore the hyperparameter-space in a define-by-run style. The library features state-of-the-art alorithms for sampling and pruning, easy parallelization of experiments and visualization tools. In order to optimize our models, we used the Tree-structured Parzen Estimator (TPE) for sampling and the Median pruner for early termination of less promising trials. The studies were conducted with a budget of 100 trials and a maximum of 50000 steps, and evaluations at every 10000 steps. For optimizating the PPO model, we used an RTX 6000 workstation with 24GB VRAM and 46GB RAM, but for the A2C study, we had to use a GTX1060OC with 6GB of VRAM and 16GB RAM which was not enough to finish the study, so in that case the results are based on 47 finished trials. Tables 1(b) and 1(a) shows the optimized hyperparameters for the models.

Learning rate schedule	constant
Learning rate	0.1324
Norm. advantage	False
n_steps	64
Entropy coef.	0.0015
Use RMS prop.	False
Gae. lambda	0.98
Max grad. norm.	0.7
vf coef.	0.7373
Activation	tanh

1(a) A2C

Learning rate schedule	linear
Learning rate	0.8764
Batch size	256
n_steps	32
Entropy coef.	6.5699
Clip range	0.2
Epochs	5
Gae. lambda	0.92
Max grad. norm.	0.8
vf coef.	0.2425
Activation	relu

1(b) PPO

Table 1: Optimized model hyperparameters

3 Results

The PPO and the A2C agents were trained in the gym-duckietown simulator, with concatenating the last 3, grayscaled observations, and using the 'steering' actions and the orientation rewards described in subsubsection 2.2.2 and subsubsection 2.2.3. The agents were trained on the 'zigzag_dists' map for one million steps, using only 1 environment. Each episode lasted a maximum of 500 steps, because this proved to be enough to take at least a full round on the map for a reasonable agent. In order to examine the effects of the hyperparameter optimization, first we trained the models with mostly the default SB3 parameters, the only exception was the learning rate, which was set to a constant 0.0005 value. This was ment to be a baseline training to compare it with the optimized models. The main difference - besides the optimized parameters - was the use of 4 parallel environments and the 2 million steps long training. Table 2, Table 3 and Table 4 shows the evaluation results on the different maps.

Table 2: Evaluation results on the 'zigzag_dists' (seen) map

	PPO (default)	A2C (default)	PPO (optimized)	A2C (optimized)
Mean episode reward	11.51	74.24	128.19	74.24
Mean episode length	71.5	308.0	452.9	308.0

Table 3: Evaluation results on the 'small_loop' (unseen) map

	PPO (default)	A2C (default)	PPO (optimized)	A2C (optimized)
Mean episode reward	14.83	61.21	93.78	61.21
Mean episode length	69.8	219.5	311.6	219.5

Table 4: Evaluation results on the 'udem1' (unseen) map

	PPO (default)	A2C (default)	PPO (optimized)	A2C (optimized)
Mean episode reward	6.01	98.27	70.79	98.27
Mean episode length	38.2	359.6	262.7	359.6

The evaluation results show us, that in the case of the PPO model, hyperparameter optimization significantly helped achieving higher rewards and longer runs, both in previously seen and unseen environments. But regarding the A2C model, the optimization seemingly did not help at all. It is unclear why this has occured, which means further examination of the training and evaluation conditions are needed for the A2C agent. An other interesting result to consider, is the fact that the A2C model performed better on a map that it has never seen, which is also a more complex version of the 'small_loop' map, with intersections and previously unseen objects like houses and ducks in the background. Furthermore, the A2C model achieved much better results with its "default" hyperparameters than the PPO, which could mean it needs less optimization trials.

4 Conclusion

In this work two reinforcement learning algorithms were trained to drive a two-wheel robot in a simulated environment. The models were compared with each other and their optimized versions using basic metrics from the simulation. It was found that optimization did help regarding the PPO model, but the A2C agent performed the same way. In order to understand the reasons behind this phenomena more in-depth experiments are needed. To further improve the results, longer optimization trials are necessary, with at least 100000 steps, and a variety of maps to train on, which can improve generalization and help avoid overfitting to a single map. More evaluation methods could be used to elaborate further on the models performance, like transversal and angular deviations from the right lane centerline, or distance travelled in the ego lane.

(Disclaimer: This is not the final version of the document. Please see a more recent commit.)