Self-driving in Duckietown environment
Önvezető autózás Duckietown környezetben

Kohlmann Dániel **Köpeczi-Bócz Ákos Tamás** **Széles Katalin**

Abstract

Deep Reinforcement Learning is a well-established method for developing self-driving algorithms. It is advisable to start the training of the self-driving algorithm in a simulation environment. We present the training of our “Reinforcement Learning” based self-driving algorithm implemented in Duckietown simulation environment. In the first step, we got familiar with the simulation environment. We can interact with the environment with an action signal and get the current status in the form of an RGB image. The training is based on the Q-Learning method, for which we did not use a separate Deep Q-learning specific framework in order to gain a deeper insight into the DQN method. During the training, the "policy" function is estimated with a neural network solution. The input of this network is the previously mentioned RGB image (from 5 consecutive moments) and the outputs are the Q-values of the selectable actions (forward, right, left). As a result of our work, we could train a model that is suitable for detecting and following traffic lanes under limited conditions.

A Deep Reinforcement Learning egy jól használható módszer sávkövető algoritmusok fejlesztéséhez. Az önvezető algoritmus tanítását célszerű szimulációs környezetben kezdeni. Duckietown szimulációs környezetben megvalósított "Reinforcement Learning" alapú önvezető algoritmusunk tanítását mutatjuk be. Az első lépésben a szimulációs környezettel ismerkedtünk meg. A környezettel egy beavatkozó jellel tudunk kommunikálni és az aktuális állapotot egy RGB kép formájában kapjuk meg. A tanítás a Q-Learning metóduson alapján valósul meg, melyhez külön Deep Q-learning specifikus keretrendszert nem használtunk annak érdekében, hogy mélyebb belátást nyerjünk a DQN-be. A tanítás során a "policy" függvényt becsüljük neurál hálós megoldással. Ennek a hálózatnak a bemenete a korábban említett RGB kép (5 egymás utáni időpillanatból) a kimenetek pedig a választható akciók Q-értékei (előre, jobbra, balra). Munkánk során sikerült egy olyan modellt tanítanunk, mely alkalmas a sávok érzékelésére és korlátozott körülmények között a sáv követésére.

1 Introduction

The Duckietown environment [5] is an educational and research framework specially created for self-driving solutions using deep learning. It is easy and cheap to construct since it's made of exercise mats and tape. The Duckiebot is the driving instance in this framework that has a single camera and two wheels. The Duckiebot is equipped with a compact computer (such as a Raspberry Pi) to carry out the calculations. The Duckietown framework comes with an OpenAI-gym based simulation environment where one can make and train an initial prototype before trying it out in a real world scenario. As it was shown by several groups, like [1] or [2]. This is especially true for the reinforcement learning approach that we also used during our work.

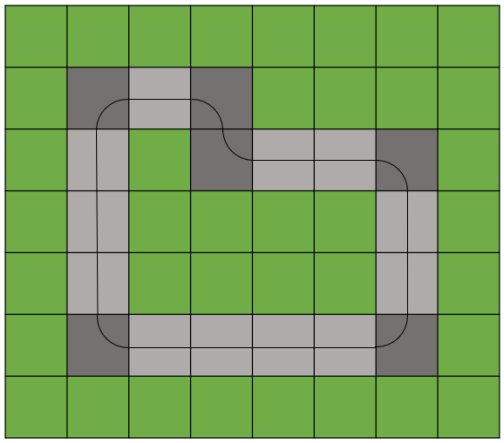

Figure 1: Our custom map

2 Reinforcement learning

The reinforcement learning [3] is a training process consisting and agent that is acting in an environment. The environment provides the actual state for the agent along with the actual reward corresponding to that state. The agent takes the actual state and based on a policy function makes an action. We used the DQN implementation which means we calculate the expected Q-value for each action-state pair and train the policy network in order to minimize the difference between the resultant Q-values and the expected Q-values. The reinforcement learning has been widely used for playing and controlling video games and simulation environment [4][7].

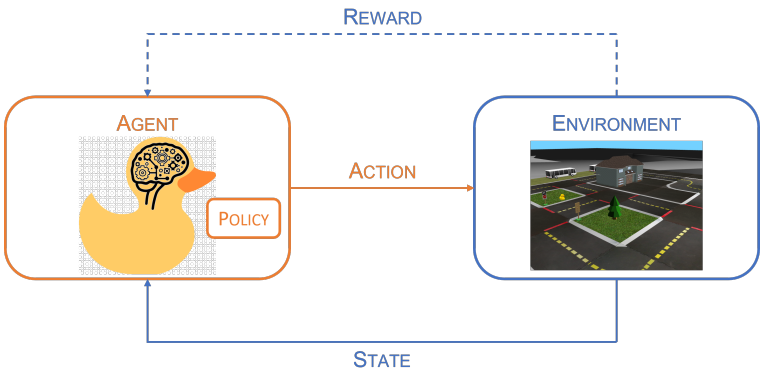

Figure 2: The reinforcement learning model

3 Policy and Target networks

Since our state space is multi-dimensional(RGB pictures), we need a complex function to make a correct decision for each input state. In order to fulfill this, we used a convolutional neural network to process the images and make decisions [9]. Firstly, we wanted to implement a transfer learning solution using a pre-trained model, but due to computational capacity limitations we decided to take a different approach. During our research we have found the solution of [1] which seemed suitable for our need and decided to use a similar network in order to fulfill our task. We have found in several approaches that it is beneficial to use a separate target network when computing the target Q-values for training stability. The target network is an exact copy of the policy network except that its weights are not trained and after 3 set of training the weights of the policy network are copied to the target network.

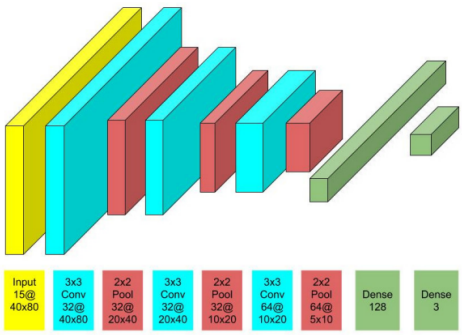

Figure 3: CNN policy network

4 Implementation

During the implementation of the Reinforcement Learning algorithm we tried to avoid "plug n' play" Reinforcement Learning frameworks so we could gain a deeper insight into the actual internal dynamics of the system. We were aware from the beginning that probably this would result in a less efficient self-driving solution.

4.1 Data input and preparation

In each episode we used a random seed to avoid repetitive domains and to avoid over fitting [10]. The first step was the preparation of the input images. We get one RGB images after each step. We made a stack of five consecutive images in order to capture the motion and not only the actual state. For efficiency purposes we cut off the unimportant part of the image (upper part, mainly the sky) and re-scaled the picture size to 80x40 pixels. Following the re-scaling we segmented the yellow and white colors. These are the colors of the lines. We separated the resultant segments into different color layers. We don't use the third channel of the RGB image, but we didn't want to remove it (although we could since it is not carrying any useful information right now) because we might need it for future upgrade. Also it wasn't a bottleneck in our method so that's the second reason we decided to keep it. We scaled down the values by 255 so the input information is ranging from 0-1. With this resolution and midifications it is possible to complete the calculations efficiently even on a CPU.

Figure 4: Original RGB image

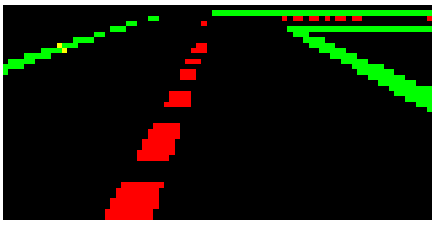

Figure 5: Rescaled and segmented image

4.2 Training

- We create replay memory inside the agent class (This class contains the neural network and the training alg. also)
- We initialize the networks
- For each episode:
  - Starting state initialization
  - For each run:
    - Selecting action (epsilon-greedy)
    - Applying the selected action
    - Reading the state (camera view) and the reward
    - Putting those into the memory
  - Training on collected data
    - Picking a batch randomly
    - State propogation based on batch
    - Passing the calculated states to the target network
    - Loss: Q-Q target
    - Updating weights
  - Mirror policy network weights to target network after every 3 episodes

4.2.1 Q-learning

We have chosen the DQN [8] method which is based on the classical Q-learning [6]. Q-learning is a model-free reinforcement learning algorithm. The main purpose is to learn the best action for every state. In classical cases the state-space is low dimensional and this method can be carried out by configuring the values in a Q-table. In case of our high dimensional state space we swap the Q-table with a CNN and optimize it in a way that it gives the expected Q-values for each state. The expected Q-value is based on the Bellman equation shown here:

$$q_*(s,a) = R_{t+1} + \gamma \text{max} q_*(s',a') \tag{1}$$

The q_* is the target Q-value, the R_{t+1} is the reward resulting from taking action a in state s. The s' and a' are the state-action pairs for the next state. γ is the discount-rate.

4.2.2 Action choosing

In order to decide between the exploration and exploitation we have decided to use the epsilon-greedy algorithm. We take a random number between 0 and 1. We constantly increase the threshold above which we exploit rather than explore. This way in the begining our agent takes random actions and by the time passes we rely more on the trained policy network.

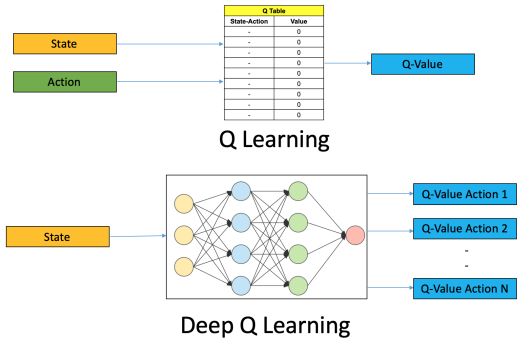

Figure 6: Q-learning and DQN

4.2.3 Memory and episode

A sampling consists 1 episode which has 5 runs. After each episode we train the policy network from randomly picked 256 experiences. We store all the experiences gained during these runs into a list, consisting the state, next state, chosen action, gained reward and other data. If the episode ended before we could gain 256 experiences then we don't train, since our agent didn't meet with enough scenarios.

4.2.4 Reward

So far we have used the default reward system of the duckietown-gym environment. We believe that most of our problem lay in this reward function, we have tried to tweak with this reward function, but couldn't construct a well working system. Right now the reward system takes into consideration the distance from the middle of the right hand side lane and the relative angle between the tangent of the lane and our agent.

4.3 Evaluation

The evaluation of the training is done in multiple ways. First we constantly render the environment so we can follow the evolution. During training we take 20% of the random experiences as a validation split to avoid overfitting. We use early stopping during the training. During the process we constantly monitor the loss after each training and store the reward for each run and we plot the improvement of the episode reward after 70 episodes.

4.4 Test and results

The test is always carried out by using the *–test* and *–load-weight* arguments. This way one can observe the behaviour of the final policy network. Our best result was a promising lane following policy that is capable of consistently following the lane and take the right turns. We have experienced that it learnt to follow the yellow line. Also it is capable of avoiding running off the circuit when it hits a white line after a left turn, but it gets stuck in an oscillation after meeting with a white line following a left turn.

5 Future improvements

As mentioned before we have experienced an oscillational local minimum after the left turns. We have tried several parameter settings to overcome this problem but we were unsuccessful so far. We believe that the main factor that is impacting it is the reward function for which we have found several propositions. The other direction for improvements is the hiper-parameter optimization which we would like to introduce but first we want to test other reward functions.

References

[1] P. Almási, R. Moni, B. Gyires-Tóth, *et al.*, "Robust Reinforcement Learning-based Autonomous Driving Agent for Simulation and Real World," in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020.

[2] A. Kalapos, Cs. Gór, R. Moni, I. Harmati, *et al.*, "Sim-to-real reinforcement learning applied to end-to-end vehicle control" *23rd International Symposium on Measurement and Control in Robotics (ISMCR)*, 2020.

[3] R. S. Sutton, A. G. Barto, *et al.*, "Reinforcement Learning: An Introduction second edition," in *Cambridge, MA: The MIT Press*, 2018.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, *et al.*, "Playing Atari with Deep Reinforcement Learning," in *NIPS Deep Learning Workshop*, 2013.

[5] L. Paull, J. Tani, H. Ahn, J. Alonso-Mora, *et al.*, "Duckietown: An open, inexpensive and flexible platform for autonomy education and research," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[6] J. Beakcheol, K. Myeonghwi, H. Gaspard, W. Jong, *et al.*, "Q-Learning Algorithms: A Comprehensive Classification and Applications," *Proceedings of the IEEE*, pp. 133653 - 133667, 2019.

[7] O. Vinyals, *et al.*, "Grandmaster level in StarCraft II using multi-agent reinforcement learning" *Nature*, 2019.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[9] M. Bojarski, D. Del Testa, D. Dworakowski, *et al.*, “End to end learning for self-driving cars,” *arXiv preprint arXiv:1604.07316*, 2016.

[10] J. Tremblay, A. Prakash, D. Acuna, *et al.*, “Training deep networks with synthetic data: Bridging the reality gap by domain randomization,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 969–977, 2018,