Deep Reinforcement Learning with Continuous Control in CARLA

Moritz Zanger[1]Florian Rottach[1]Izel Kilinic[1]

***Abstract*— Style:**

- **Written last**
- **100-250 words**
- **Past tense**

Content:

- **Condensed version of entire article**

I. Introduction

The task of teaching vehicles how to drive autonomously in urban scenarios is a challenging and complex one to solve. Not only is there the problem of finding the adequate response to a given situation but also the challenge of taking into account the surrounding factors that have an influence on the state that a vehicle is in and its possible actions. To date, most approaches focus on the manual design of behavioral policies, such as defining a driving policy through the use of annotated maps. While these solutions might work in situations which are documented by the provided mapping infrastructure, they are often difficult to generalize or scale, as they do not necessarily enable the comprehension of any given local scene. In order to make autonomous driving truly feasible in a real-world scenario it would be better to develop systems which are able to find their way without having to rely on an explicit set of rules. One possible solution to this task is provided by reinforcement learning methods. Here, the agent, i.e. the vehicle, actively searches for the optimal driving policy whilst trying to maximize a numerical reward signal. As opposed to imitation learning techniques, which have been popular in finding driving policies (1), reinforment learning algorithms enable a car to exceed human abilities, if applied correctly. In recent years, deep reinforcement learning methods have proven to be succesful in solving complex tasks such as playing GO (2) or Atari (3) and there have been efforts in tackling various problems in the field of autonomous driving, including continous control tasks (4).

However, two major drawbacks of reinforcement learning methods are their heavy depency on adequate input state representations (5) and, as with other machine learning techniques, their need of a sufficient amount of accurate sample data to train on. In order to be able to train safely on an adequate amount of data, one approach is the use of data from other domains. For this purpose, the urban driving simulator CARLA has been developed, which is used as a simulation environment for this project.

In this paper, several state-of-the-art reinforcement learning algorithms are implemented and compared, with regard to their performance considering different driving tasks. Additionally, reward functions for the respective problems are tested and several input represantations are designed and evaluated.

?? What is it exactly that we contributed that is new to already existing research??

II. Related Work

III. Background

We regard the typical reinforcement learning setting where an agent interacts with an environment $\mathcal{E}$. At each one of a number of discrete timesteps t the agent decides on taking an action a_t from a given set of actions $\mathcal{A}$. This is done based on the state s_t that the agent is currently in and following a policy π, which is a mapping of the possible states to the action space $\pi : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{A})$. In this case, π is stochastic, as it returns the probability distribution over the possible actions, and the action space is continuous.

As a result, the agent receives a reward r_t and the subsequent state s_{t+1} of his environment. The setup is assumed to follow the properties of a Markov decision process, where besides state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $r(s_t, a_t)$, we also include the transition function to the future states $p(s_{t+1}|s_t, a_t)$.

IV. Concept/Methods and Models

In the course of this project, three differnt algorithms are used. Following the work of Lillicrap et al. (4) and Mnih et al. (6), the (DDPG)- and (A3C) algorithms are implemented and compared according to their performance in the Gym environment CarRacing-v0. Later on, the PPO algorithm is applied to solve a continuous control task in the CARLA simulator. TODO: Hier unser Konzept analog zur Präsentation vorstellen (Slide mit den 4 Kasten)

A. DDPG

One very notable advance in reinforcement learning has been made by the development of the so-called "Deep Q Network" (Mnih et al., 2015). The DQN is able to solve tasks with high-dimensional observation spaces. However, it is only efficiently capable of working with discrete and low-dimensional action spaces. In order to adapt a (DQN) for the successful use with continuous control problems, as given in CarRacing-v0, a discretization of the action space has to be carried out, which can lead to two main difficulties: an explosion in the number of possible actions and the loss of important information (4).

To evade these obstacles (Lillicrap et al.) propose a new approach, called the Deep Deterministic Policy Gradient

[1]The authors are with FZI Research Center for Information Technology, Haid-und-Neu-Str. 10-14, 76131 Karlsruhe, Germany {kuhnt, zoellner}@fzi.de

(DDPG), which is a model-free, off-policy actor-critic algorithm. They adopt the advantages of (DQN) and combine them with the actor-critic framework, resulting in the stabilization of Q-learning by using a replay buffer and soft updates on the target networks of both actor and critic, through

$$\tau << 1 : \theta' \leftarrow \tau\theta + (1 - \tau)\theta' \tag{1}$$

and finding a deterministic policy.

Noise with Ornstein-Uhlenbeck

B. A3C

C. PPO

D. Reward function

The design of the reward function turned out to be a much more difficult procedure as expected initially. The already provided rewards in gym's CarRacing environment were rather simple and still led to a good result in the end. For CARLA however, it was necessary to invest more time into the engineering and designing of a suitable inducement system, because the driving behavior depended on more factors and the performance metric was not just driving as fast as possible. In the following, the process of arriving at our final reward function will be described.

In the first step, we investigated possible sensor inputs that might have an impact on the driving behavior and discussed on their impact. Table 1 summarizes the outcome.

TABLE I: Identified reward function components

Component	Description	Intention
Per frame penalty	The penalty amount subtracted from the reward for each frame	Forces agent to move in order to compensate the penalties
Lane invasion increment	Is either 0 or 1 and reflects if a lane invasion took place for the current frame.	The agent should perform as few lane changes as possible
Steering angle	The absolute angle of the steering wheel	Avoids oszillations in the driving behavior
Delta heading	The angle between current street direction and car position	Agent should drive as straight as possible relatively to road direction
Position change	The angle between current street direction and car position	Maximize travelled distance
Collision binary per frame	Penalty for crashing into other vehicles, objects or road infrastructure	Avoid crashes and improve security of driving
Velocity	The agent's absolute speed retrieved from the CARLA engine	Forces the agent to move forward
Distance to middle lane	The absolute distance to center in meters - can also be squared to improve	Drive as centered as possible

We started with incrementally adding these attributes to our reward calculation and quickly realized, that the main challenge is to adjust the weights and harmonize the contrary effects of the terms. An example would be, that giving the velocity a relatively high weight, such as 0.8, while giving the distance to center line a weight of 0.2 results in an agent that speeds over the map and pays few attention on lane invasions. On the opposite side, the agent will drive only very slowly or even not at all, if the rewards for oscillations are too high compared to the velocity. Considering, that we have not only two, but rather several possible components, this results in a complex combinatorical problem that can either be solved by trail and error or applying permutation optimization techniques. To achieve initial results, we started to discover a proper reward function "by hand".

We pruned the above list by applying the following considerations that resulted from tesing different approaches. Firstly, some components show a redundant behavior and hence one of them can be removed. An example would be the velocity and the position change - both contain the same information. Further, the attributes lane invasion and steering angle didn't affect the driving behavior in a positive way. The two most important parameters turned out be the velocity and the delta heading, which expressed the relative angle to the current street angle. Our final reward function and the aggregation weights can be found in table 2.

We want to note here, that a performance-wise comparison between the different parameter combination is hard to measure with a metric and is not the aim of this project. However, it would make sense to develop a suitable approach to categorize and assess the performance of different reward functions towards the desired outcome.

The result (in combination with tuning the reinforcement learning models) is very promising and can be summarized as follows: The agent is capable of driving smoothly within the right lane and can perform turn manuevers on most intersections. It attempts to drive around other vehicles, but is not capable of breaking. We trained different models on this reward function and all of them had a good performance compared to other incentive attempts. We assume, that event better results can be achieved when applying an improved optimization method to find out the best parameter-combination. However, this is a combinatorical problem and requires several simulation runs.

E. Input representation

Dealing with the high-dimensional environment in CARLA is one of the central challenges in implementing a well functioning RL algorithm. In consideration of an abundance of possible sensor types available in CARLA we decided to pursue increasing levels of realism which generally also correspond to increasing levels of difficulty. These aforementioned levels of realism involve the following:

- Ground truth segmented bird's eye view

TABLE II: Final reward function terms and weights

Component	Effect	Weight
Per frame penalty	Lead to a strong initial learning behavior. Already after few steps the agent kept accelerating to compensate this penalty.	-0.01
Velocity	When selected too small, slow learning and when too high, strong lane oscillations / non-smooth driving.	+0.05
Delta heading	Introducing this term improved the driving stability enormously.	-0.005
Squared distance to middle lane	Squaring had very beneficial effects, other powers were to restrictive.	-0.01
Collision binary	Might be set to an even higher value but lead to attempts to avoid other objects	-100

- Ground truth segmented front view
- Latent space generated from ground truth segmented images
- Latent space generated from rgb images

Notably, these input representations differ in dimensions and thus lead to different network architectures in the appended network architectures of the RL agent.

a) Ground truth segmented bird's eye view: The Ground truth segmented bird's eye view is a rather unrealistic scenario, in which we assume availability of a camera positioned 20m above the performing agent. It is merely imaginable in a fully observed city where autonomous driving is part of a high-level traffic control system. Albeit rather unrealistic, we decided to implement a 13-class ground truth segmented bird's eye view due to its similarity to the CarRacing environment and its obvious advantages in containing information central to navigational tasks. The full list of the included classes are described by dosovitskiy et al.[**?**]. We deemed a 1-channel grayscale image with size 64x64 large enough to contain key information for the algorithm (fig. x l.)

gt_combined.png

Fig. 1: left: Ground truth segmented bird's eye view image, 64x64 in grayscale, 9 classes right: Ground truth segmented front view image, 80x80 in grayscale, 9 classes

b) Ground truth segmented frontview: Much like the Ground truth segmented bird's eye view, the corresponding front view input representation assumes the availability of a perfect segmentation camera. Nonetheless, we increase realism with this representation as the camera is placed in front of the car. Again, we chose a 13-class, 1-channel grayscale image with a slightly increased size of 80x80 to account for a larger view angle of the camera.

c) Latent space generated from ground truth segmented images: In this section we will further discuss a modified version of the integrated encoder-decoder network that is based on dziubinski's [**?**] implementation. In its original architecture, the network comprises 5 input tensors and 7 output tensors with the inputs representing images of a front-, rear-, left-, right, and top-view camera. The model is built with the Keras functional API and generally serves two purposes. On the one hand, each input tensor is encoded into a 64-sized feature vector and decoded into it's original shape with a cross entropy loss with regard to the original input. This is achieved with separate branches of autoencoders with 3 convolutions respectively. On the other hand, a separate generative branch creates a bird's eye view reconstruction based on the concatenated feature vectors of the vehicle-based cameras. In addition to the cross entropy loss at the reconstructed bird's eye view output, a mean square error loss is calculated against the output of a subtract layer between the autoencoder- und reconstructed feature vector to support convergence.
In comparison to dziubinski's [**?**] vanilla version, we adjusted the architecture to fit the single camera bottlenecks into a length 64 vector and the reconstructed bird's eye view bottleneck into a length 128 vector. To reduce complexity, we condensed both input and target segmentation images to 3 classes and implemented a generalized weighted cross entropy[**?**] loss function to account for the unbalanced distribution of vehicles and obstacles.

d) Latent space generated from rgb images: To deal with the higher input complexity in RGB-images, we extended the encoder and decoder models to 5 convolutions.

rgb_birdseye.png

Fig. 2: left: rgb images from 4 vehicle-based cameras 2nd from right: ground truth segmented bird's eye view right: reconstructed bird's eye view

F. Autoencoder/Latent space

TODO: Encoder-decoder architecture bzw. Generator nennen

G. Training

TODO: Training beschreiben analog zur präsentation (vgl folien)

V. Evaluation

A. CarRacing-v0

B. CARLA

C. Results

D. Comparison algorithms/reward functions

VI. Conclusions

- Similar to the abstract but more detail

- Conclusion of the key points of each section
- Summary of main findings
- Important conclusions that can be drawn
- Discuss benefits and shortcomings of our approach
- Suggest future areas of research

References