UNIVERSITÄT
LEIPZIG

Institut für Informatik
Faulty of Mathematics and Computer Science
Abteilung Datenbanken

Investigating implementation details of reinforcement learning for self driving agents

Expose Masterarbeit

vorgelegt von:
Georg Schneeberger

Matrikelnummer:
3707914

Betreuer:
Dr. Thomas Burghardt

Abstract

In my master's thesis I will investigate using neural networks for self driving and the practicability of using reinforcement learning for the training process. The reinforcement learning agent's task is to complete a parcour in a simulated environment without collisions. The thesis will build upon previous student's work at the Scads.AI by changing implementation and training details. The goal is to improve the agent's performance and evaluate the changes. The work will be compared to the results of the previous thesis [1].

Georg Schneeberger
3707914

Inhaltsverzeichnis

1. Motivation

The increasing utilization of artificial intelligence in academia and industry have lead to massive efficiency improvements for all kinds of tasks. The development of autonomous vehicles promisses to greatly reduce the number of traffic accidents and transportation cost. As a result researchers and private enterprises from all over the globe are making progress towards fully autonomous driving agents and integrating them in commercial vehicles, the most well known of these being Tesla FullSelfDriving FSD [2]. Due to the recent developments in artificial intelligence and the very high complexity of the task of autonomous driving, artificial intelligence often plays a big role in these systems [3].

Predictions for the future of autonomous driving have been very optimistic and although huge progress has been made, the task of autonomous driving is still far from being solved [4]. This thesis aims to contribute to the research in this field by applying reinforcement learning to autonomous driving agents in a simulated environment. Different implementations will be tested and evaluated to investigate their contributions to the performance of the agent. This work builds upon the work of [1] and will use the same task and evaluation metrics.

2. Research Goals

The goal of this thesis is to contribute in the domain of autonomous driving by investigating the efficiency and contributions of different implementation details using reinforcement learning in a simulated environment. The thesis will build on the work of [1] and review the utilised algorithms and implementation details. Implementation changes will be proposed based on research in the field of reinforcement learning and autonomous driving. The contributions of these implementation details will be evaluated on their own and in combination with each other.

The self driving agent is trained and evaluated on a simulated parcour. Different parcours, lighting settings and motor-power settings are used to evaluate the agent's reliability and generalisation capabilities. The evaluation scenarios will consist of the ones employed by [1] and potentially additional ones based on related work and the investigated implementation changes.

The task of driving through the parcour is motivated in part by the Scads.AI research facility, it would be ideal to be able to transfer the agent from this thesis to a physical arena at the Scads.AI research facility. The Scads.AI premises are often used for exhibitions and events where a physical self driving agent could be used to raise interest in applications of AI and research.

Um ehrlich zu sein war es schwierig diese Paragraphen zu schreiben. Es war schwierig Research Goals zu formulieren, die unterschiedlich von den vorherigen sind.

3. Related Work

Since this thesis will employ reinforcement learning in the training of the autonomous driving agent it is important to review the current state of the art in reinforcement learning and autonomous driving. Some techniques of the researched works might not be applicable in this thesis due to the smaller scope and availible resources of this thesis, it is still important to review them to understand the current state of the art and to be able to utilise aspects of them. Simulation environments for reinforcement learning and self driving will also be reviewed, since they play a bit role in the state of the art of both fields and will be employed in this thesis. Finally some papers about the use of reinforcement learning for self driving will be reviewed.

3.1. Reinforcement Learning

Reinforcement learning algorithms have been around for a long time, but only recently have they been able to achieve superhuman performance in games and control tasks. Most reinforcement learning algorithm formalize the problem using a state space and an action space and a reward function associated with state transitions. The RL algorithms are built to select an action in a given state and try to maximize the cummulative reward along the state transitions. This process of selecting an action is called the policy. The classical version of the Q-learning RL algorithm keeps a table of state-action pairs and their associated quality values which are learned/computed during training. The amount of state-action pairs can be very large and not feasible to compute in many cases, especially for environments with continuous state and action spaces. An example of a continuous state space would be the position of an object in a continuous 2D space. To solve this problem neural networks have been used to approximate the Q-values, they take a representation of a state from the state space as input. This approach is called Deep Q-Learning and has been used to achieve superhuman performance in many games [5]. The initial success of Deep Q-Learning has lead to many improvements and variations of the algorithm, since then RL algorithms have proved to be very useful in a wide range of applications.

A mayor weakness of the Deep Q-learning algorithm was its stability, the parameter updates of the Q-learning agent's neural network can result in a collapse in performance during training. The Proximal Policy Optimization (PPO) and other algorithms have been developed to improve the stability of the training process. The PPO [6] algorithm restricts the size of policy changes caused by parameter updates, this ensures the policy cannot change drastically and improves stability. PPO is currently one of the most popular algorithms for reinforcement learning and has already been successfully used in the domain of autonomous driving [1].

Another major improvement in the domain of reinforcement learning was the combination of neural networks with traditional planning and search algorithms, the most famous example for this is AlphaGo [7]. AlphaGo and the many algorithms inspired by it use a search algorithm (typically Monte Carlo Tree Search) to evaluate the possible actions at a given state. Neural networks are used for the evaluation of states and actions in the search algorithms. AlphaGo was developed for a 2 player game with a discrete state and action space, but the combination of neural networks

and search algorithms has also been used in continuous state and action spaces with a single player [8].

3.2. Self Driving

As mentioned before there has been a lot of progress in the domain of self driving in recent years, often driven by private enterprises that might not publish their findings in academic journals. Sophisticated self driving algorithms often consist of many components to achive satisfying performance, Tesla FSD for example uses separate object detection, occupancy and planning components [3]. This thesis builds directly upon the work of [9], [10] and [1]. [9] built a self driving agent that was trained to avoid collisions in a simulated arena using an evolutionary approach to neural network training, the agent used visual inputs that were preprocessed before feeding it to the neural network. [10] investigated the feasibility of transfering the agent to the real world, this research showcased many difficulties, most notably the object recognition part of the agent. [1] investigated a different task than the two previous papers, the agent was trained to pass a parcour by driving through a sequence of goals. This task is identical to the one investigated here. [1] successfully used PPO to train the agent which was fed preprocessed data similar to [9].

Another interesting approach to building self driving agents is using imiatation learning [11], imitation learning agents learn to mimic the behaviour exhibited in the training set. The training set is usually generated by humans driving the car, this approach is rumored to be used by Tesla to train their self driving agents [3].

3.3. Simulation for Reinforcement Learning (and Self Driving)

Simulations play a huge role in reinforcement learning and the development of self driving agents. Simulations provide a number of benefits over real world experiments. They are much cheaper to run, they can be run in parallel and they can be run much faster than real world experiments. In additon the programmers have direct and perfect control over the environment. This allows for much faster experimentation and training of reinforcement learning agents. Simulations also allow for the creation of scenarios that are not possible in the real world, this is especially useful for reinforcement learning agents that are trained to avoid collisions. Simulations also allow for the creation of ground truths, which are often not available in the real world such as perfect sensor data and object bounding boxes. Ground truths are used to evaluate the performance of the agent and can be used to train the agent. In addition the physics of the simulation can be modified, for example it is possible to increase the simulation speed.

Simulated environments, especially games have served as baselines for reinforcement learning algorithms. The most famous baseline are the atari games [5]. The need for easy use of reinforcement learning algorithms has lead to the development and adoption of the Gynasium API [12], the Gymnasium API defines an interface that can be used to model tasks as reinforcement learning problems. This interface is called a Gym environment, an allows for easy testing of reinforcement

learning algorithms on a wide range of tasks. Furthermore the Gymnasium API allows for easy comparison of different algorithms on the same task. A wide range of reinforcement learning frameworks support the Gymnasium API, for example Google's dopamine [13] and OpenAI's baselines [14].

Wrappers for Gymnasium environments are often used to facilitate the use of more advanced simulations for environments, such as for example the game engines Unity and Unreal or the physics simulator MuJoCo [15]. The complexity and interest in self driving has also lead to the development of dedicated simulators, such as the Carla [16] and the AirSim [17] simulator. The Carla simulator provides researchers with useful features such as weather control and ground truths for object detection and segmentation.

There are also dedicated frameworks for reinforcement learning that directly integrate with game engines, such as the ML-Agents framework [18] for Unity.[1] used this framework to train the self driving agent.

3.4. Reinforcement Learning for Self Driving

[19] review the use of reinforcement learning for autonomous driving, they also describe many improvements for reinforcement learning algorithms that can improve the training stability and performance, such as for example reward shaping. In addition to published research papers there have been a lot of experiments, tutorials and demonstrations of self driving agents on YouTube and GitHub. The University of Tübingen published their full lecture series on Self-Driving Cars [20], the series also includes a section on reinforcement learning.

4. Methods

4.1. Task Description

In this section I will be describing the task and the simulation environment as these two aspects are the foundation of this thesis and will not change over the course of the project. The task is to develop and agent using reinforcement learning that is able to complete a parcour in a simulated environment without collisions. The agent has to traverse the parcour by passing through a number of goals indicated by pairs of either red or blue blocks without collisions, see figure 4.1. The agent starts at a random location and rotation on a line and the parcour is successfully completed when the agent reaches the final goal. The task and agent are simulated using the Unity game engine, the game engine handles the rendering of the environment, collisions, agent movement and everything else.

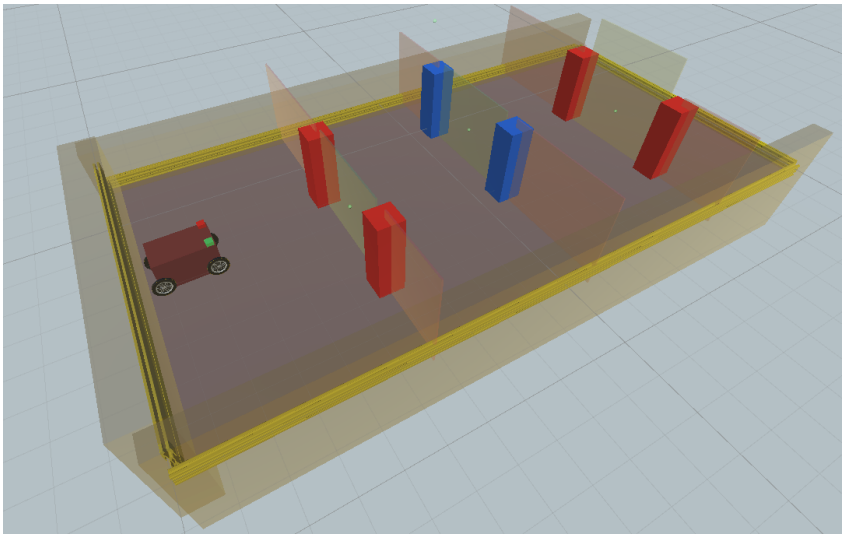

Abbildung 4.1.: Example image of the agent at the start of a parcour with 3 goals in Unity

4.2. Reinforcement learning algorithm and frameworks

I will then describe the reinforcement learning algorithm and possible frameworks that may be used to implement it. There are multiple approaches to training an agent in the Unity simulation environment as highlighted in the related works section, it is not yet decided which approach will be chosen, therefore I will quickly describe the possible scenarios and highlight their advantages and disavantages.

Unity and ML-Agents

Reinforcement learning agents can be trained directly in the Unity simulation environment using the ML-Agents framework [18]. This approach has the advantage of being very simple to implement

and use, since the agent and simulation are in the same framework. The biggest disadvantage of this approach is that it might be difficult to implement some of the implementation details investigated in this thesis. The PPO algorithm by [18] was used in [1] to successfully train the agent on the investigated task.

Unity and separate reinforcement learning frameworks

There are many reinforcement learning frameworks publically available that can be used to train agents in simulated environments, such as the OpenAI's baselines [14] and Google's dopamine [13]. These frameworks often support the Gymnasium API [12], wrapping the Unity simulation environment in a Gymnasium environment would allow for easy use of these frameworks. This approach has the advantage of being able to use the many features of these frameworks and makes it easy to change the training algorithms which might be necessary for the investigated implementation details. The disadvantage of this approach is that it might be difficult to wrap the Unity simulation environment in a Gymnasium environment, there already exist frameworks to help with this [21].

4.3. Investigated implementation details

As mentioned in the research goals section multiple different implementation details will be tested in this thesis and evaluated, a list of these implementation details will be given in this section, together with the reasoning behind using them.

4.3.1. Reward shaping

Reward shaping is the practise of providing reinforcement learning agents with frequent and accurate rewards. This helps the agent develop the desired behaviour quicker and more reliably since reward signals are less sparse with reward shaping [19]. This practice was already employed by [1] by providing the agent with a reward proportional to its speed, this encourages the agent to drive faster and thus hopefully complete the parcour quicker. This thesis will investigate the use of a reward proportional to the difference in distance to the next goal between timesteps, this should encourage the agent to drive towards the next goal and to drive faster in this direction.

It is also possible to let agents learn not only from the reward at the current timestep but also from the near future reward [22]. This also results in a less sparse reward signal and can improve the training stability and performance of the agent. This is called n-step reward and is proven to be useful in solving environments with delayed rewards (passing a goal + the associated reward may happen much later than a steering action that lead to this reward).

4.3.2. Convolutional Neural Networks

The works by [10] and [1] showed that the agent's performance greatly depended on the quality of the input preprocessing pipeline. This object detection pipeline had difficulties detecting objects under varying light settings. This thesis will investigate using convolutional neural networks instead of an object detection pipeline, this should make the agent more robust to varying light settings and improve the performance of the agent. Using convolutional networks to process images is a common practise in the field of reinforcement learning, due to the ability of these networks to adapt. The convolutional neural network could potentially learn to identify relevant information in images more reliably than the previously used image detection pipeline. The research by [10] showed that not all of the information provided by the object detection pipeline was considered to be relevant by the neural network, this could be mitigated by training the CNN end-to-end.

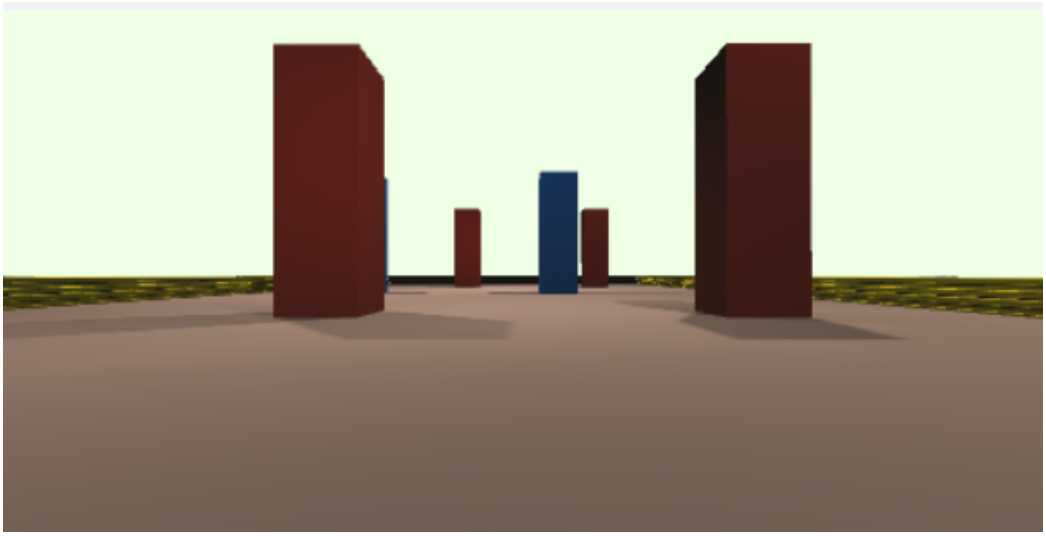

Abbildung 4.2.: Example image that will be processed by the agent's convolution neural network.

4.3.3. brightness rebalancing

The previous work this thesis builds upon used the HSV color space to extract the differently colored objects. Colors in this space consist of three values, one for the hue, saturation and brightness. The hue value was used for extracting the objects. In theory the utilization of this color space should make the object detection resilient to changes in brightness since this information does not affect the hue value. Convolutional neural network typically use the RGB color space or a greyscale colorspace. A brightness rebalance of the imput images could play a big role in making the agent more resilient to changes in illumination, with which the agent by [1] struggled with.

4.3.4. Time perception

Two configurations of the agent by [1] used a memory to enhance the agent's input, the memory consisted of the input of the last few steps of the agent. This can improve the performance and stability during training [23]. This exact technique is often used for discrete environments, such as

the boardgame Go [7]. However since the investigated environment does not have fixed timesteps this approach might not be as effective. The time elapsed between two steps can vary due to many factors such as varying compute resources. It could prove useful to provide the model with the input of the last few steps and in addition the elapsed time between these steps.

5. Experiments and Evaluation

5.1. Experiments

The proposed implementation details will be tested in the same scenarios as [1] and potentially additional ones. [1] included three parcours with different difficulty levels and conducted 3 different experiments with each trained agent. The first experiment was under optimal conditions with minimal changes from the simulation environment. The second experiment was conducted under different lighting settings. The third one changed the motor power of the agent's two front wheels. Using these settings allows for comparison of the agent developed in this thesis with the previous work. In addition, I will also test the agent with different motor power of the two front wheels, this is a more realistic scenario of varying motor power than the one used by [1].

5.2. Evaluation

During the training and the final experiments, the agents are evaluated using the success rate, the average time needed to complete the map and the collision rate. The success rate is the percentage of episodes in which the agent successfully completed the map. These metrics were already used by [1] and measure the most important properties of the agent's behaviour.

There will be additional metrics monitored during the training process to identify weak-points of the agent and erroneous behaviour. Surveilling the training process can provide insights into the agent's behaviour and help with choosing appropriate hyperparameters. These metrics could be the average cumulative reward, the average number of passed goals, the average distance travelled, the average amount of collisions, the average amount of collisions with certain objects and the average speed of the agent. The metrics help

6. Schedule

- 1.5 month - Literature research: Finding a suitable RL-training algorithm and training framework. Researching implementation details for the task of autonomous driving.
- 2 month - Implementation of the training algorithm. Including preliminary testing/evaluation.
- 1 month - Training and Evaluation
- 1.5 month - Writing the thesis

Literatur

[1] Maximilian Schaller. „Train an Agent to Drive a Vehicle in a Simulated Environment Using Reinforcement Learning“. Magisterarb. Universität Leipzig, 2023.

[2] x. *x*. TODO tesla autopilot is most famous self driving result. x.

[3] Think Autonomous. *Breakdown: How Tesla will transition from Modular to End-To-End Deep Learning*. 2023. URL: https://www.thinkautonomous.ai/blog/tesla-end-to-end-deep-learning/ (besucht am 15. 09. 2023).

[4] Collimator. *The State of Autonomous Vehicles: Seeking Mainstream Adoption*. 2023. URL: https://www.collimator.ai/post/the-state-of-autonomous-vehicles-in-2023 (besucht am 16. 02. 2023).

[5] Volodymyr Mnih u. a. *Playing Atari with Deep Reinforcement Learning*. 2013. arXiv: 1312.5602 [cs.LG].

[6] John Schulman u. a. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[7] David Silver u. a. „Mastering the game of Go with deep neural networks and tree search“. In: *Nature* 529 (Jan. 2016), S. 484–489. DOI: 10.1038/nature16961.

[8] TODO. *a paper that uses search and NN for single player continuous action games*. 2017. arXiv: TODO [cs.LG].

[9] TODO. *TODO*. 2017. arXiv: TODO [cs.LG].

[10] TODO. *TODO*. 2017. arXiv: TODO [cs.LG].

[11] Joonwoo Ahn, Minsoo Kim und Jaeheung Park. „Autonomous driving using imitation learning with look ahead point for semi structured environments“. In: *Scientific Reports* 12 (Dez. 2022), S. 21285. DOI: 10.1038/s41598-022-23546-6.

[12] Mark Towers u. a. *Gymnasium*. März 2023. DOI: 10.5281/zenodo.8127026. URL: https://zenodo.org/record/8127025 (besucht am 08. 07. 2023).

[13] Pablo Samuel Castro u. a. „Dopamine: A Research Framework for Deep Reinforcement Learning“. In: (2018). URL: http://arxiv.org/abs/1812.06110.

[14] Antonin Raffin u. a. „Stable-Baselines3: Reliable Reinforcement Learning Implementations“. In: *Journal of Machine Learning Research* 22.268 (2021), S. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

[15] Emanuel Todorov, Tom Erez und Yuval Tassa. „MuJoCo: A physics engine for model-based control“. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, S. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[16] Alexey Dosovitskiy u. a. „CARLA: An Open Urban Driving Simulator“. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, S. 1–16.

[17] Shital Shah u. a. „AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles“. In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065.

[18] Andrew Cohen u. a. „On the Use and Misuse of Absorbing States in Multi-agent Reinforcement Learning“. In: *RL in Games Workshop AAAI 2022* (2022). URL: http://aaai-rlg.mlanctot.info/papers/AAAI22-RLG_paper_32.pdf.

[19] B Ravi Kiran u. a. *Deep Reinforcement Learning for Autonomous Driving: A Survey*. 2021. arXiv: 2002.00444 [cs.LG].

[20] Prof. Andreas Geiger. *Self-Driving Cars*. 2022. URL: https://www.youtube.com/watch?v=GYnlqiSqZiU&list=PL05umP7R6ij321zzKXK6XCQXAaaYjQbzr&index=13 (besucht am 23. 10. 2023).

[21] Hugh Perkins. *Peaceful Pie, Connect Python with Unity for reinforcement learning!* 2023. URL: https://github.com/hughperkins/peaceful-pie (besucht am 23. 10. 2023).

[22] TODO for example the history stacking in alphago. „nstep reward more stability in training“. In: *TODO* (2022). URL: TODO.

[23] TODO for example the history stacking in alphago. „Some paper that shows that memory is important for RL, show memory helps with stability during training and performance“. In: *TODO* (2022). URL: TODO.