Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Deep Reinforcement Learning for High-Level Behavior Decision Making

Florian Dittrich

Course of Study:	Informatik
Examiner:	Prof. Dr. rer. nat. Marc Toussaint
Supervisor:	Dr. Ing. Felix Schmitt (Bosch Center for Artificial Intelligence)
Commenced:	May 7, 2018
Completed:	November 7, 2018

Abstract

As the vision of fully autonomous vehicles potentially introduces significant benefits for our society, this work investigates approaches for sequential decision making for high-level actions in highway scenarios. These scenarios are modeled using an markov decision process (MDP) and consider deep reinforcement learning to solve it. Our approach, based on deep Q-networks (DQNs), is able to fully avoid collisions and learns a policy that results in comfortable trajectories compared to baseline policies we developed. One of the main challenges for reinforcement learning are sparse rewards, which we aim to overcome employing reward shaping. Additionally, the necessity of multiple layers of non-liniearities in the DQN algorithm is empirically evaluated using our scenarios. The results support the usage of multiple levels of non-linearities, as a linear variant of the DQN is not capable of learning effective policies in our experiments. Due to a weight initialization with behavioral cloning, an acceleration of the learning procedure is achieved.

Kurzfassung

Da die Vision des vollautonomen Fahrens potenziell signifikante Verbesserungen für unsere Gesellschaft bringt, untersuchen wir Ansätze für sequentielle Entscheidungsfindung für übergeordnete Aktionen in Autobahnszenarien. In dieser Arbeit modellieren wir diese Szenarien mit einem Markow-Entscheidungsproblem und betrachten *Deep Reinforcement Learning* um es zu lösen. Unser Ansatz, basierend auf *Deep Q-Networks (DQNs)*, ist in der Lage Kollisionen vollständig zu vermeiden und lernt eine Policy, die im Vergleich zu den von uns entwickelten Baseline Policies komfortable Trajektorien findet. Eine der größten Herausforderungen für *Reinforcement Learning* sind *sparse Rewards*, welche wir durch *Reward Shaping* versuchen zu bewältigen. Zusätzlich, evaluieren wir empirisch die Notwendigkeit von mehreren nicht-linearen Schichten im DQN. Unsere Ergebnisse unterstützen den Gebrauch von mehreren nicht-linearen Schichten, da eine lineare Variante der DQN in unseren Experimenten nicht in der Lage ist effektive Policies zu lernen. Auf Grund der Initialisierung der Gewichte durch *Behavioral Cloning* erreichen wir eine Beschleunigung des Lernprozesses.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

ACC autonomous cruise control. 45

Adam adaptive moment estimation. 26

BC behaivioral cloning. 18

DNN deep neural network. 17

DQN deep Q-network. 3

DQN_{BC} DQN initialized with behaivioral cloning (BC) using the TD baseline and trained with shaped rewards. 47

$\text{DQN}_{\text{linear}}$ Linear DQN trained with shaped rewards. 47

$\text{DQN}_{\text{shaped}}$ DQN trained with shaped rewards. 47

$\text{DQN}_{\text{sparse}}$ DQN trained with sparse rewards. 47

DRQN deep recurrent Q-network. 68

FSM finite-state machine. 19

HER hindsight experience replay. 22

IDM intelligent driver model. 20

IS importance sampling. 36

LTL linear temporal logic. 21

MCTS monte carlo tree search. 20

MDP markov decision process. 3

ML machine learning. 26

MSE mean-squared error. 32

NGSIM Next Generation Simulation. 20

NN neural network. 23

PD proportional derivative. 23

PER prioritized experience replay. 35

POMDP partially observable markov decision process. 68

ReLU rectified linear unit. 24

RL reinforcement learning. 17

SGD stochastic gradient descent. 26

SVM support vector machine. 20

tanh hyperbolic tangent. 24

TD temporal difference. 31

TD time-dependent. 40

TTC time-to-collision. 38

TTD time-to-destination. 43

TTH time-to-headway. 41

UCB1 upper confidence bound. 20

VTD Virtual Test Drive. 48

1 Introduction

Towards the vision of fully autonomous driving, current research still faces many difficult challenges. Yet, many leading technology companies heavily invest in the area as it introduces vast benefits for the general public. The idea of being able to focus on other things than the traffic and the street while driving already sounds very pleasant but the full potential of fully autonomous driving goes far beyond convenience aspects.

Some executives of leading technology companies share a common view on the topic as they say:

> *"If you recognize that self-driving cars are going to prevent car accidents, AI will be responsible for reducing one of the leading causes of death in the world."*
>
> – Mark Zuckerberg, CEO Facebook, Inc.

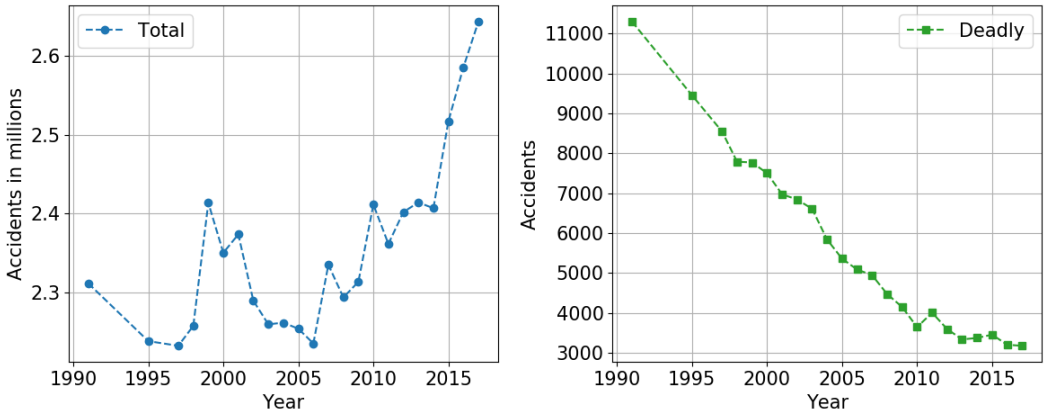
> *"Self-driving cars are the natural extension of active safety and obviously something we should do."*
>
> – Elon Musk, CEO Tesla, Inc.

> *"Think about how many people are under-served by transportation today, like those with disabilities, and how self driving cars will transform their lives. Or the wasted time you sit in your car every day commuting to and from work. Or the deaths and injuries that could be avoided."*
>
> – Larry Page, CEO Alphabet Inc.

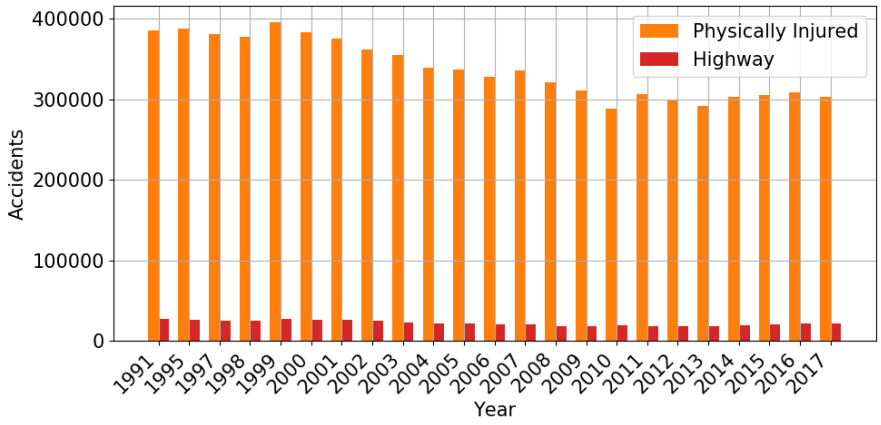
> *"This exciting and extensive study shows that driver-less cars could vastly improve the flow of traffic in our towns and cities."*
>
> – John Hayes, UK Transport Minister

> *"Automated driving can drastically reduce the number of accidents, and thus significantly increase road safety. […] Moreover, a better flow of traffic also reduces fuel consumption."*
>
> – Volkmar Denner, CEO Robert Bosch GmbH

These statements cover multiple aspects of autonomous driving, which can be summarized to safety, accessibility, efficiency and convenience. Especially in terms of safety, the improvement could be huge. In Germany alone, 2.4 million accidents were registered in 2017 [Des17b], which is an increase of 2.2 % compared to 2016. The progress of the total number of accidents registered in Germany since 1991 can be seen in Figure 1.1a, which shows a vast increase since 2006.



(a) Total number and number of deadly accidents registered in Germany per year



(b) Development of highway accidents compared to total number accidents with physical injuries

Figure 1.1: Statistics of accidents in Germany since 1991 according to Destatis [Des17b]

The number of fatal accidents in Germany already shows a decreasing trend, mostly because of legal regulations like e.g, speed limits or the duty to fasten seat belts, improvements in automotive engineering or medical first aid [Des17a]. Although this already looks promising, in 2017 still 302 656 people got injured in traffic accidents in Germany. Additionally, the number of highway accidents resulting in physical injuries do not show a negative trend since 1991 as displayed in Figure 1.1b. Introducing autonomous vehicles, which posses shorter reaction times and a wider overview over the situation, has the potential to drastically reduce these numbers. Autonomous vehicles do not suffer from fatigue, micro-sleep, hesitation or blind spots, which are among the main causes for human driving errors.

Apart from safety, this work emphasizes on comfort for the passengers during their ride in a self-driving vehicle. It has been shown that starting from a lateral acceleration threshold between 1.47 $\frac{\mathrm{m}}{\mathrm{s}^2}$ and 2.45 $\frac{\mathrm{m}}{\mathrm{s}^2}$, [KB12; XYSL15] strong discomfort for passengers is caused. Additionally, the

literature suggests that passengers perceive discomfort at lower rates of acceleration than drivers [FZB+07]. Therefore, it is important to achieve acceptance of autonomous driving by safety and comfort during their ride, especially in the early stages of autonomous vehicles.

In order to tackle these issues, this work investigates reinforcement learning (RL) for autonomous driving in highway scenarios. The task of an RL agent is depicted in Figure 1.2, where the controlled vehicle, which we will from now-on call ego vehicle, is supposed to overtake the truck without colliding into other vehicles.

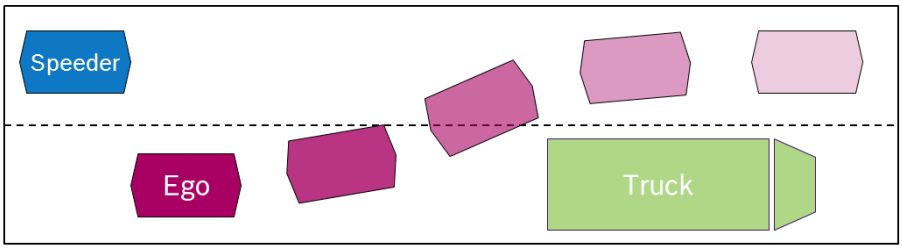


Figure 1.2: Illustration of a scenario used throughout this work. The ego vehicle in purple is supposed to overtake the slower truck in green while not colliding with the faster speeders in blue on the target lane.

The desired behavior of the ego vehicle in this task is defined as changing the lane to the target lane and passing the truck. Because the scenario is modeled based on a two-lane highway this can be achieved with a single lane change procedure. Other vehicles apart from the ego vehicle have varying locations and velocities based on a-priori defined sampling ranges. During a simulation, each vehicle will keep its lane and give way by decelerating as soon as other vehicles enter their lane in front of them.

In such scenarios, possibly many other vehicles participate and influence the decision making process of the ego vehicle by their behavior. This introduces a lot of uncertainties, which make it non-trivial to model the dynamics for each of the vehicles. Hence, reinforcement learning is a valid choice for scenarios of such kind as it solely learns based on the interaction with the environment. Reinforcement learning received a lot of attention recently and has been shown to be very strong in games like chess [BTW00] or go [SHM+16; SSS+17]. Combining RL with deep neural networks (DNNs), which serve as powerful, non-linear function approximators, researchers were able to train systems that beat the world-class human players of chess and go.

One of the main challenges for reinforcement learning in the scenarios developed in this work are sparse rewards, which are given only once at the end of a simulation. That is, the RL agent does not get any feedback about its behavior until the termination of the episode, when it gets either a very high or a very low reward, depending on the outcome. Hence, the agent has to learn the quality of its actions from sparse feedback. Similarly, in chess this would correspond in finding out which move was critical for winning or loosing after the game ended.

Another bottleneck of reinforcement learning approaches is data inefficiency, especially when it is combined with deep neural networks. In order to find a strong policy capable of making good decisions in complex scenarios, often a huge amount of interactions with the environment is necessary. As neural networks are usually randomly initialized based on some heuristics [GB10; HZRS15], the agent has to learn all the details of the environment from scratch.

In this work, DQN is used to control the ego vehicle, in a multi-agent highway scenario. The DQN algorithm is supposed to learn a policy that outputs high-level actions in two different highway scenarios that vary in difficulty and complexity. We employ reward shaping in combination with deep reinforcement learning in an overtaking task for autonomous driving. Reward shaping is used to tackle problem of sparse rewards. Minor rewards are calculated throughout the simulation, which eventually sum up to the same sparse reward but give immediate feedback to the RL agent. We aim to overcome the issue of data inefficiency by using behaivioral cloning (BC). The neural network policy is initialized using BC to exploit domain knowledge. In order to use BC, a designated dataset is constructed using one of our developed baselines. The idea is to provide the algorithm with key aspects of the scenario such as action repetition and the duration of a lane change.

The main question of this work is: Can deep reinforcement learning be applied to overtaking scenarios, where it has to learn a policy that decides *when* it is the right time to overtake in order to reach a certain goal, avoid collisions and reduce accelerations and jerk for the passengers? Therefore, the following hypotheses which we will thereafter evaluate according to empirical results, are proposed:

- The trained DQN policy is able to reach the goal in 90 % of the experiments
- The trained DQN policy is able to avoid collisions in 90 % of the experiments
- The trained DQN policy is able to find a comfortable trajectory with respect to acceleration and jerk measures
- Reward shaping accelerates the learning procedure significantly
- Deep Q-networks with multiple layers of non-linearities outperform purely linear Q-networks
- Behavior cloning as weight initialization of the neural network accelerates the learning procedure

By assessing these research hypotheses we aim to investigate the potential of DQN for solving real-world tasks in autonomous driving. A comparison with a purely linear DQN version is relevant, as there are recent doubts about the necessity of non-linearities in model-free RL [MGR18]. Additionally, a weight initialization exploiting domain knowledge is of particular relevance for RL with respect to data efficiency in such complex scenarios.

The remainder of this work is organized as follows. In Chapter 2 previous work from the literature relevant to our approach is presented. Next, in Chapter 3 relevant background related to this work, such as reinforcement learning, neural networks, etc is introduced. Chapter 4 asserts how the experiments are set up which scenarios are used and what are the developed baselines that are applied for evaluation. In Chapter 5 the results of the experiments are presented, which will be discussed in Chapter 6. Finally, Chapter 7 briefly summarizes our findings and gives some hints about possible future work.

2 Related Work

In this section literature closely related to this work is examined.The literature is briefly introduced and similarities as well as differences to this work are discussed.

So far, a wide range of approaches for autonomous driving have been published. The driving scenarios, most of the authors used, range from urban [UAB+08; UM13], to highway [HWL18; LKK16; SSS16] or even off-road environments [TMD+06]. Each of them introduces an individual set of challenges, whereas requirements like safety and reaching a certain goal are usually shared among them.

One challenge of autonomous driving is the multi-agent environment, which causes a lot of uncertainties due to the unknown behavior of other vehicles. As it is extremely difficult to model all these dynamics in a multi agent traffic scenario, planning for autonomous vehicles has been a very challenging task so far. One way to cope with these difficulties is to reduce the complexity by splitting the planning in a hierarchical fashion [HCR09]. This can be done by separating the decision on high-level actions from the actual maneuver execution. Hence, the algorithm first decides on whether to change the lane or not and then gives this information to the execution module, which performs the action accordingly.

Traditional approaches to make behavioral decisions in autonomous driving include finite-state machines (FSMs) with a fixed set of possible states a car can obtain. Montemerlo et al. [MBB+] proposed a behavior module using this approach. However, it lacks generalization to unseen events and handling inaccuracies as well as uncertainties in sensor inputs. Furthermore, the state machine cannot be easily extended or transferred to new traffic scenarios. Therefore, data-driven approaches are promising to improve generalization to uncertainties and multiple scenarios.

In the following, subsections different approaches to infer policies for autonomous driving based on data are examined. Supervised approaches based on expert data, online planning and reinforcement learning techniques are considered and assessed based on their applicability to the scenarios of this work.

2.1 Imitation Learning

In the following, approaches are introduced that incorporate expert data to learn policies. This supervised machine learning technique is also called BC.

Bi et al. [BMWD16] proposed a supervised data-driven approach, which consists of a decision-making module and an execution module. The decision-making module uses random forest, which not only chooses on whether to perform a lane-change or not, but also the gap between the vehicles on the target lane to merge into. Once a suitable gap is found, the execution model performs the

lane change into this previously selected gap by determining the low-level actions for the procedure. In order to train the random forest classifier, a dataset containing traffic trajectory data from the Federal Highway Administration's Next Generation Simulation (NGSIM) is used.

Vallon et al. [VECB17] treated the lane change decision as a classification problem as well given traffic data. They collected 25 lane-change procedures of two drivers in the real world using a test vehicle. The idea was to resemble the natural driving strategy of the particular driver. A support vector machine (SVM) that outputs a binary decision about the lane-change was trained using the collected dataset. The features used as an input for the SVM were relative distances and longitudinal velocities between the ego vehicle and each of the two target vehicles. In their scenario, one of the vehicles was located in front of the ego vehicle traveling slower, whereas the other vehicle was located on the target lane behind the ego vehicle.

The disadvantage of such policy generation approaches based on behavioral cloning is the lack of generalization out-side the expert data. Especially in automated driving, it is difficult to generalize from a given data set because each of the traffic participants can behave differently and a limited amount of data might not effectively model these uncertainties. Additionally, if a cloned policy is applied sequentially in an MDP, small training errors can accumulate over time, which can lead to weak performance. The approach used in this work mitigates these issues, as behavioral cloning is only used to initialize the neural network. The learning of the final policy is done by means of reinforcement learning.

2.2 Online Planning

In online planning techniques for autonomous driving, low-level policies are assumed to be available and solely sequential decisions on high-level actions are made. Here no learning is involved. Instead, planning techniques aim to find actions by online simulating all possible outcomes based on the current state and select the most promising action for success according to the current evaluation.

Lenz et al. [LKK16] use an online planning technique called monte carlo tree search (MCTS) for high-level maneuver decisions. They transformed the high-level decisions into low-level actions using previously hand-engineered lateral and longitudinal controllers. After selecting a node for expansion based on the upper confidence bound (UCB1) technique [BPW+12], the intelligent driver model (IDM) [THH00] is used as their default policy. Their scenario was modeled by a single lane change of the ego vehicle due to an ending of its lane. The used cost function incorporated cost terms for each of the relevant vehicles in the scenario, which modeled a cooperative strategy. The degree of cooperation can be tuned using a hyperparameter λ determining the influence of the cost functions of other vehicles into the cost function of the ego vehicle. One drawback of MCTS are the high execution costs because of the fast expansion of the search tree.

2.3 Reinforcement Learning for Sequential Decision Making

Because of the aforementioned shortcomings, applying reinforcement learning to autonomous driving use cases has been a promising research direction lately. In recent years deep reinforcement learning has become very popular, especially after the introduction of deep Q-networks, by Mnih

et al. [MKS+13] in 2013. Deep neural networks as non-linear function approximators were applied to the raw screen images of many Atari games to approximate Q-functions and in turn find policies to win the game. Up to now, many extensions for DQN have been published [HGS16; SQAS16; WSH+16], which showed great success and eventually reached super-human performance when combining many of them [HMH+17].

Similar to Lenz et al., Paxton et al. [PRHK17] used MCTS with UCB1. Instead of selecting high-level actions and leaving the low-level actions to predefined controllers, temporal abstraction with options was incorporated. First a set of low-level policies (options) and a policy over options were trained offline by reinforcement learning. Subsequently, those options were used online in combination with MCTS, which also checked constraints imposed by linear temporal logic (LTL).

A very recent publication of Hoel et al. [HWL18] used DQN to learn high-level actions for changing lanes to the right, to the left or to keep the current lane. They used the ego velocity and relative positions as well as velocities of all the surrounding vehicles as their features. Besides this, information whether the surrounding lanes are free and the relative distance to other vehicles measured in number of lanes were included. All the features per vehicle were stacked in a single vector and convolution was applied to it. The filter size and stride were specified to cover the data of a single vehicle, which aggregates the information on a vehicle basis. Another convolution layer was applied to the result of this to even further condense the information about each vehicle. Subsequently, max pooling and fully connected layers were used to find the high-level actions.

Mirchevska et al. [MPW+18] also used DQN in a similar scenario and state space to find high-level actions. Compared to other approaches it used an infinite scenario where no particular goal exists and the reward function solely emphasized on reaching a target velocity. Furthermore, they put a strong focus on safety as they introduced a separate module that verified the resulting Q-values given by the network. It checked whether the selected action given by the DQN is safe according to a minimal distance between other vehicles. They compared their approach against a rule-based system showing a higher average velocity over different scenarios.

Wang et al.[WCL18] also used a Q-learning approach, optimizing solely on lateral control. Instead of choosing high-level actions, their system directly output continuous yaw acceleration, which was then transformed to actual steering angles. They designed their reward function with a focus on safety, smoothness and efficiency. The respective terms in the reward function were based on the yaw acceleration, yaw rate and maneuver duration. The design of the Q-function is quadratic in action, given by $Q(s,a) = A(s) \cdot (B(s) - a)^2 + C(s)$, where A, B and C are coefficients modeled with neural networks. Optimal actions were output by B and the parameters for all networks were updated based on Q-learning with a target network.

Shalev-Shwartz et al. [SSS16] proposed a hierarchical method named option graph. A directed acyclic graph of semantic decisions is constructed, which are modeled as options. When traversing the graph, each node acts as a policy, picking one of its children. The output of the graph is a high-level action called “desire”, which is ultimately used to find actual trajectories. Each of the node policies of the option graph is represented by a neural network and initialized using imitation learning with data collected from human drivers. Afterwards, they are trained by means of RL. Because of the unknown intent of other drivers sequential decision making in autonomous driving does not fulfill the Markov property. Therefore, the authors applied policy gradient RL methods,

which can easily be applied in this setting. Policy gradient methods suffer from high variance of the gradient estimator when used with large time horizons. Therefore, baselines similar to REINFORCE and temporal abstractions with the options graph were used for variance reduction.

2.4 Reinforcement Learning for Sparse Rewards

Although, reinforcement learning has a big potential in autonomous driving, there are some challenges that need to be considered. One of these challenges are sparse or delayed rewards, which are present in the scenarios conducted throughout this work. Sparse rewards are very high or very low rewards given only once at the very end of an episode. Therefore, the reinforcement learning algorithm receives no feedback until the episode terminates. This makes learning very difficult because the algorithm internally needs to realize when it made a mistake during the episode, which lead to a negative reward after termination. Analogously, after a successful episode, it has to learn what action lead to a high reward.

In order to overcome this problem Ng et al. [NHR99] introduced reward shaping. A new MDP $M' = (\mathcal{S}, \mathcal{A}, P, R', \gamma)$ was defined, where $R' = R + F$ became the new transformed reward function. The authors proofed that without further knowledge about P and R, the only way to choose F that guarantees consistency with the optimal policy in M is a potential-based shaping function: $F(s,a,s') = \gamma\phi(s') - \phi(s)$.

Deisenroth et al. [DFR15] employed heuristic reward shaping to define their cost function. In the considered robotic scenario, they defined their cost function with respect to euclidean distances between the current state of the robot arm and the target state. The saturating cost is modeled with a gaussian function given in Equation (2.1). It gives a cost between zero and one, which is gradually decreasing the closer the current state gets to the target state.

$$c(x) = 1 - \exp\Big(-\frac{1}{2\sigma_c^2} d(x, x_{\text{target}})^2\Big) \in [0,1] \tag{2.1}$$

Using this cost function, immediate feedback is incorporated instead of giving a single high reward (negative cost) when reaching their target state. Additionally, the saturating cost function is considered as a natural exploration technique, as it favors uncertain states over certain ones.

In contrast to reward shaping, Andrychowicz et al. [AWR+17] proposed hindsight experience replay (HER) as a different approach for sparse or delayed rewards. The idea is to learn as much form examples resulting in undesired outcomes than from examples where the goal was reached. This is achieved by reexamining a trajectory and storing each transition twice or even more often in the replay memory: Once with the original goal g and additionally with the terminal state of the trajectory s_T. Therefore, it can still be learned how to reach the goal g but additionally the algorithm learns how to reach s_T. Thus, learning becomes much simpler, as at least half of the replayed trajectories contain non-sparse rewards. However, HER is restricted to off-policy RL algorithms.

Instead of using multiple goals, as seen above, Florensa et al. [FHW+17] limit the initial states to a subset of the uniform distribution over all possible states. This subset changes over the course of the training. Initially the starting states of the rollouts are selected nearby the goal state. Over time new start states are generated according to the learning progress by doing brownian motion starting from start states that were within a “good” goal reaching probability range in the last iteration.

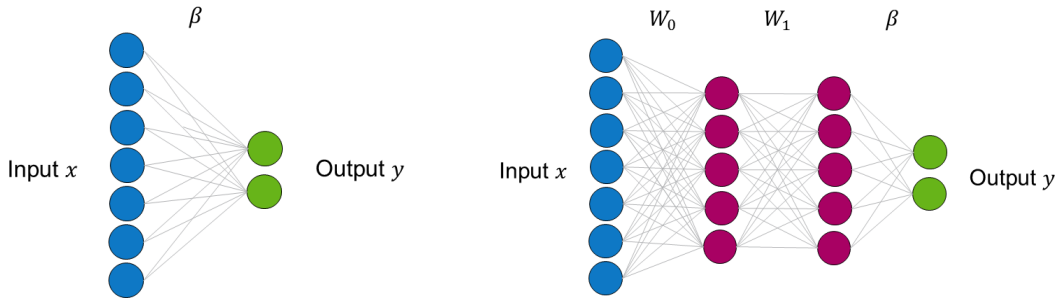
3 Background

In this section the standard formalisms and techniques that are relevant for this work and have shown significance in the machine learning community are presented. First, neural networks, which are non-linear function approximators, are briefly introduced. Next, an introduction to reinforcement learning as an approach to solve sequential decision problems is given. In Section 3.3 deep Q-networks are introduced, which combine the popular Q-learning approach with deep neural networks. Lastly proportional derivative (PD) controllers are reviewed, which are also relevant for the autonomous driving application in this work.

3.1 Neural Networks

This section introduces neural networks (NNs) based on the lecture slides of Toussaint and Nguyen-Tuong [TN18]. Neural networks are a class of models that can be trained to represent non-linear functions. They have recently experienced great success in many applications showing state-of-the-art performance. For instance in image classification [HZRS16; KSH12; ZL17] or automatic speech recognition [CSW+18; HDY+12] neural networks and especially deep neural networks with multiple hidden layers outperform most of the traditional approaches. A neural network consists of multiple layers, where the initial layer is called input- and the last layer is called output layer respectively. All other layers in between are called hidden layers.

The simplest example is a network with no hidden layers as shown in Figure 3.1a. With this architecture the network is represented by $f(x) = \beta^T x$ where β are the weights of the input vector x. The same model is also applied in linear regression.



(a) No hidden layer **(b)** Two hidden layers

Figure 3.1: Neural networks with different architectures. Figure 3.1a depicts a linear neural network, as it does not possess any hidden layer. On the contrary Figure 3.1b has two hidden layers.

Figure 3.1b depicts a neural network with two hidden layers, where the output of one layer is the input of the next. Activation functions, which are usually non-linear functions, are applied to each output of a layer. In general, the forward pass of a neural network is defined as follows:

$$\begin{aligned} z_l &= W_{l-1} x_{l-1} \qquad \forall l = 1, \dots L \\ x_l &= \sigma(z_l) \end{aligned} \tag{3.1}$$

where z_l is the output of the $l-1$-th layer, x_l is the input of the l-th layer, W_l is the weight matrix of the l-th layer and σ is the activation function. For the neural network shown in Figure 3.1b, this results in $y = f(x) = \beta^T \sigma(W_1 \sigma(W_0 x))$.

In order to learn a certain task, e.g. a classification or regression task, with a NN, first the task performance must be quantified by a suitable loss function. Exemplary loss functions for a classification task are cross entropy or negative log-likelihood, whereas in regression tasks usually the mean-squared-error or the hinge loss are used. The NN is trained to achieve the given task by optimization of the task loss with respect to the NN's weight matrices. This can be done by using gradient based optimization. To obtain the gradient, the backpropagation algorithm can be used. It takes the resulting loss from the forward pass and gradually calculates the gradients through the network. By using gradient descent, the weights are changed a little bit towards the direction of the gradient. More formally speaking, we assume there is a loss function $\mathcal{L}(f)$, and the gradient of the loss is given by $\delta_{L+1} = \frac{\partial \mathcal{L}}{\partial f}$. The gradient of each layer before that can be calculated recursively with respect to its input using the chain rule:

$$\begin{aligned} \forall l = L, \dots 1: \qquad \delta_l &= \frac{d\mathcal{L}}{dz_l} \\ &= \frac{d\mathcal{L}}{dz_{l+1}} \cdot \frac{\partial z_{l+1}}{\partial x_l} \cdot \frac{\partial x_l}{\partial z_l} && \text{//chain rule} \\ &= \left[\delta_{l+1} W_l\right] \circ \left[x_l \circ (1 - x_l)\right]^T && \text{//assumming } \sigma(z) = \text{sigmoid}(z) \end{aligned} \tag{3.2}$$

Here $\circ$ is an element-wise product. The gradients with respect to the weights can be calculated by:

$$\frac{d\mathcal{L}}{dW_l} = \delta_{l+1}^T \cdot x_l^T \tag{3.3}$$

The actual update of the weights can now for example be done by gradient descent:

$$W_l \leftarrow W_l - \alpha \frac{d\mathcal{L}}{dW_l} \tag{3.4}$$

3.1.1 Activation Functions

One important choice when designing the architecture of a neural network is the type of activation function. An activation function σ is usually a non-linear function that is applied to the output of each layer except the output layer. The activation functions mostly used are sigmoid, hyperbolic tangent (tanh) and rectified linear unit (ReLU), which can be seen in Figure 3.2.

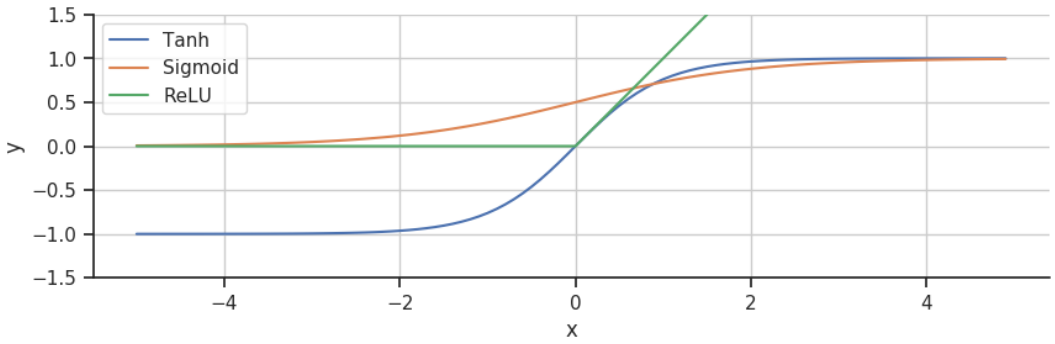


Figure 3.2: Sigmoid, hyperbolic tangent and rectified linear unit function

The sigmoid function is an S-shaped function given by Equation (3.5). It is a real-valued, non-negative function that is differentiable and bounded between 0 and 1. It is popular in the deep learning community because its derivative, given in Equation (3.6), can be represented with only sigmoidal and primitive operations. This is handy because it makes the backpropagation of the loss term and calculating the gradients easier.

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)} \tag{3.5}$$

$$\frac{\partial}{\partial x}\,\text{sigmoid}(x) = \text{sigmoid}(x) \cdot (1 - \text{sigmoid}(x)) \tag{3.6}$$

The hyperbolic tangent function is very similar to the sigmoid function but it is bounded between -1 and 1.

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{3.7}$$

$$\frac{\partial}{\partial x}\tanh(x) = 1 - \tanh^2(x) \tag{3.8}$$

One drawback of the sigmoid and the tanh function is that for extremely high or extremely low values of x the gradients become very small. This is known as the vanishing gradient problem [Hoc91]. It can lead to a very slow learning process or even divergence.

The ReLU function was introduced by Nair and Hinton [NH10] to solve this problem. For $x > 0$ its gradient is independent of the magnitude of x. Therefore, the gradient cannot vanish and accelerates the training.

$$\text{ReLU}(x) = \max(0, x) \tag{3.9}$$

However, ReLU can suffer from dead units, as for $x < 0$ there is no output and no gradient [MHN13].

3.1.2 Optimizer

In order to minimize the network loss function $\mathcal{L}$, gradient descent methods are in general a popular approach. The idea is to manipulate the set of parameters θ by gradually descending the gradient with respect to the parameters. Formally, this is defined by $\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}$. When using a small

subset of the whole dataset, so-called mini-batches, this procedure is called stochastic gradient descent (SGD). The learning rate α must be tuned depending on the task and usually has a strong influence on the learning process.

Recently there has been remarkable progress in optimization techniques and many extensions to the traditional SGD have been published. A good overview over the different optimizers is given in an early draft by Ruder [Rud16].

The adaptive moment estimation (Adam) optimizer was introduced in 2015 by Kingma and Ba [KB15]. It combines two of the previously published extensions: AdaGrad [DHS11] and RMSProb [HSS12]. Similarly to the aforementioned, Adam uses a different learning rate for every parameter θ_i at every time step t. Additionally, it stores an exponentially decaying average of past squared gradients v_t as well as an exponentially decaying average of past gradients m_t:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2. \end{aligned} \tag{3.10}$$

Because these estimates are initialized with zeros, there is a bias towards zero especially in the beginning of the optimization. Therefore, the authors propose a correction:

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t}. \end{aligned} \tag{3.11}$$

The final update rule is then given by:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t. \tag{3.12}$$

The hyperparameters β_1, β_2 and ϵ need to be set by the user. Default values of $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ are proposed by the authors. In their empirical results they show a good performance over a wide range of tasks compared to other adaptive learning methods.

3.2 Reinforcement Learning

In general, machine learning (ML) can be divided into three sub-disciplines: Supervised ML, unsupervised ML and reinforcement learning. Supervised ML techniques learn given a labeled dataset, whereas unsupervised ML techniques aim to cluster unlabeled datasets into groups. In this work we consider reinforcement learning to solve sequential decision making problems in unknown or uncertain environments. Most of the following content is based on the work of Sutton and Barto [SB98]. In general, in RL an agent interacts with an environment and learns a policy by receiving rewards and observations when executing actions on this particular environment. There are multiple ways to find a policy in reinforcement learning. In general, they can be divided into three areas depicted in Figure 3.3: Model-free RL, model-based RL and policy search.

Model-free methods estimate a value function that approximates the expected return of a state or a state and action pair. The value function is directly estimated by interacting with the environment. Subsequently, the value function is used for action selection. A problem with this approach often is

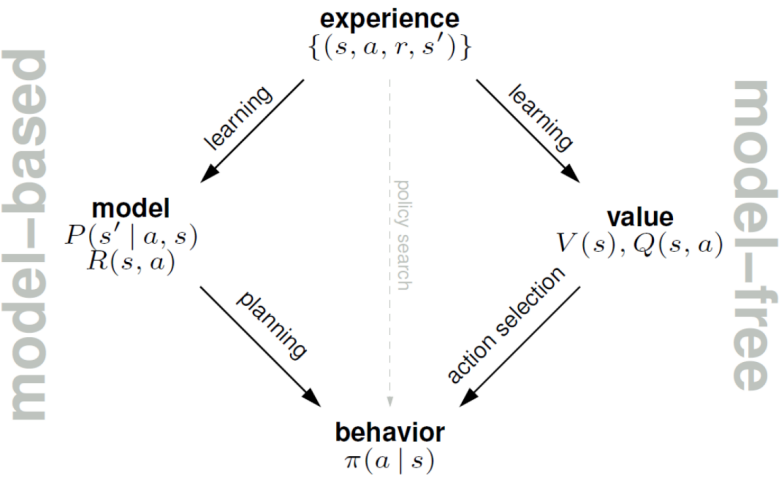


Figure 3.3: Reinforcement learning can be divided into three subgroups. Model-free RL, model-based RL and policy search [NN17]

that it is not very sample efficient and slow but applicable in a wide range of applications. With model-based techniques, first, a model of the reward and the transition function is learned using data from the interactions with the environment. Then traditional planning or search algorithms are be applied such as value or policy iteration. This is generally more efficient but not always applicable. Policy search focuses on finding the right policy directly by calculating gradients of the return. In general, policy search approaches are effective in high-dimensional or continuous action spaces but often converge to a local rather than the global optimum. There are also hybrid methods like actor-critic, which combine policy gradients with a value function.

In the following, we first introduce markov decision processes as a framework to model sequential decision making problems. Afterwards, a policy is defined and as a mapping from state to actions. In order to learn such policies, value functions based on the bellman equation are defined in Section 3.2.3. Approaches that estimate a value function, to learn a policy are so-called model-free RL approaches, which are introduced in Section 3.2.4. Finally, the trade-off between exploration and exploitation is examined in Section 3.2.5.

3.2.1 Markov Decision Process

The environment an RL algorithm is applied to is usually modeled by an MDP. An MDP is a framework for modeling sequential decision making problems. It is defined by a five tuple $M = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$, where:

- $\mathcal{S}$ is a set of states
- $\mathcal{A}$ is a set of actions
- $P = \Pr(s'|s,a)$ is a transition function, modeling the probability of being in state $s' \in \mathcal{S}$ after executing action $a \in \mathcal{A}$ in state $s \in \mathcal{S}$
- $R(s,a)$ is a reward function

- γ is a discount factor

The interaction between an agent and the environment is depicted in Figure 3.4. At any timestep t the agent is in a state $s_t \in \mathcal{S}$ and executes an action $a_t \in \mathcal{A}$. The action moves the agent to a successor state $s_{t+1} \in \mathcal{S}$ according to the transition function P. Additionally, a reward $r_t = R(s_t, a_t)$ is given to the agent by the reward function. This interaction process of an agent and the environment, can be of infinite duration or subject to terminal conditions. Often, the interaction ends as soon as the time t reaches a time horizon T. Furthermore, the interaction may end when the agent reaches a terminal state e. g. when the agent collides with another object or reaches a goal. A tuple of state, action, reward and subsequent state is called a transition and a sequence of transitions $\{(s_t, a_t, r_t, s_{t+1})\}_{t=0}^{T}$ is called an episode or rollout.

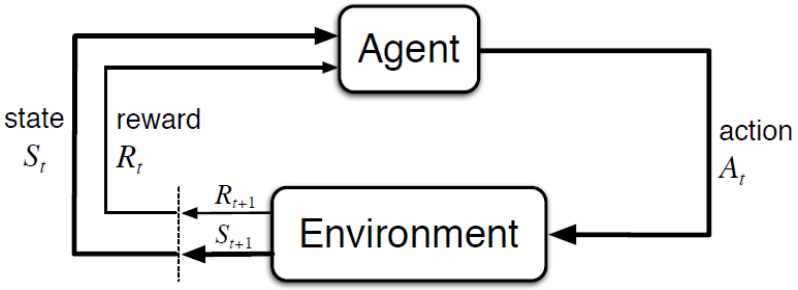


Figure 3.4: Interaction between an agent and an environment modeled by an MDP [SB98]

An example environment in a grid world is depicted in Figure 3.5. Here, the agent is located in an initial state in purple and has possible actions of going left, right or straight. The goal is to reach the terminal state in green, where the agent receives a reward of 50. In all other states there is no reward. Additionally, there is a wind in the center of grid world blowing from the bottom to top. In this region, a straight action of the agent possibly takes it one cell to the left because of the strong wind. This happens with a probability of $\frac{1}{3}$. The challenge in this environment is to learn the dynamics of the environment and to explore the state space properly to find the goal.

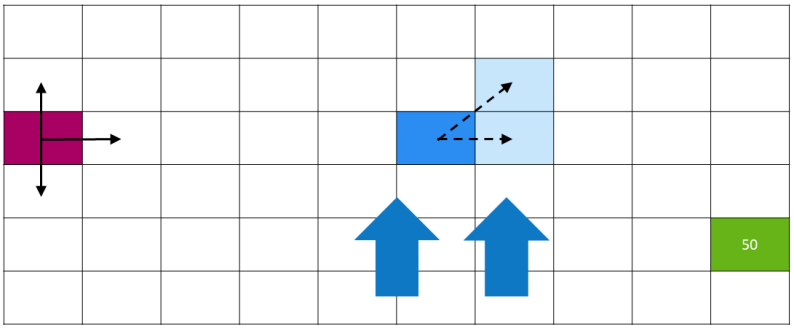


Figure 3.5: Grid world environment with single goal state and wind blowing from the bottom to the top.

In order to accurately define the dynamics of an environment one might assume that all previous states and actions have to be considered. This implies, the probability of reaching state s_{t+1} depends on the entire history of states and actions $Pr(s_{t+1}|s_t, a_t, \ldots, s_0, a_0)$. Instead, a Markov decision

processes fulfills the so-called *Markov property*. That is, a state contains all information about the previous interactions with the environment. The probability of the next state s_{t+1} then only depends on the current state s_t and current action a_t: $Pr(s_{t+1}|s_t, a_t, \ldots, s_0, a_0) = Pr(s_{t+1}|s_t, a_t)$.

3.2.2 Policy

The policy π is defined as a mapping from a state s to an action a: $\pi : \mathcal{S} \rightarrow \mathcal{A}, s \mapsto \pi(s)$ and therefore determines the behavior of the agent by selecting the action given its current state. The policy can be a deterministic function, meaning that the policy will always return the same action for a certain state. Policies can also be stochastic and be defined by a distribution $\pi : \mathcal{S} \times \mathcal{A} \mapsto [0,1]$ where a probability is assigned to each action a given a state s.

In order to assess a given policy in an environment a performance measure is needed. In reinforcement learning, the performance of the policy in an episode is quantified by means of its return R given in Equation (3.13). The return is defined as the sum over all rewards r_t received within an episode starting at $t = 0$ and terminating at $t = T$.

$$R = \sum_{t=0}^{T} r_t \tag{3.13}$$

In the finite horizon setting, the return is calculated from finitely many values and therefore bounded. However, for infinite horizons an evaluation of R is not possible anymore. Therefore, discounting has been introduced to cope with this problem. Here a discount factor $\gamma \in [0,1]$ is used to down-weight the contribution of distant future rewards:

$$R = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3.14}$$

This is called discounted return. The return can be used to calculate value functions, which is considered in the following section.

3.2.3 Value Function

Some of the RL algorithms maintain a so-called state value function or state-action value function. These functions indicate how beneficial it is to be in a certain state s or to perform a certain action a in a given state s. Formally, the value function is defined as the expected return when starting in state s and selecting actions according to the policy π:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\Big]. \tag{3.15}$$

Analogously, the state-action value function is given by

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\Big[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\Big]. \tag{3.16}$$

In case the reward function and the transition function are known, the value functions can be calculated recursively:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\Big[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots | s_0 = s\Big] \\ &= \mathbb{E}_{\pi}\Big[r_0 | s_0 = s\Big] + \gamma \mathbb{E}_{\pi}\Big[r_1 + \gamma r_2 + \ldots | s_0 = s\Big] \\ &= R(s,\pi(s)) + \gamma \sum_{s'} P(s'|\pi(s),s)\ \mathbb{E}\Big[r_1 + \gamma r_2 + \ldots | s_1 = s'\Big] \\ &= R(s,\pi(s)) + \gamma \sum_{s'} P(s'|\pi(s),s)\ V^{\pi}(s') \end{aligned} \tag{3.17}$$

Equation (3.17) is called the *Bellman equation*.

The Bellman equation for the state-action value function can be derived analogously:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}_{\pi}\Big[r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots | s_0 = s, a_0 = a\Big] \\ &= \mathbb{E}_{\pi}\Big[r_0 | s_0 = s, a_0 = a\Big] + \gamma \mathbb{E}_{\pi}\Big[r_1 + \gamma r_2 + \ldots | s_0 = s, a_0 = a\Big] \\ &= R(s,a) + \gamma \sum_{s'} P(s'|a,s)\ \mathbb{E}\Big[r_1 + \gamma r_2 + \ldots | s_1 = s'\Big] \\ &= R(s,a) + \gamma \sum_{s'} P(s'|a,s)\ Q^{\pi}(s',\pi(s')) \end{aligned} \tag{3.18}$$

Considering two policies π and π', π is better or equal than π' if its expected return is greater than or equal to that of π' for all states $s \in \mathcal{S}$. Therefore, $\pi \geq \pi'$ if and only if $V^{\pi}(s) \geq V^{\pi'}(s)$ for all $s \in \mathcal{S}$. Hence, we can assess policies given their value functions and we can find an optimal policy based on value functions. The value function that defines the optimal policy π^* is called the optimal state-value function $V^*(s)$ or the optimal state-action value function $Q^*(s,a)$:

$$\begin{aligned} V^*(s) &= \max_{\pi} V^{\pi}(s), \qquad \forall s \in \mathcal{S} \\ Q^*(s,a) &= \max_{\pi} Q^{\pi}(s,a), \qquad \forall s \in \mathcal{S},\ a \in \mathcal{A}. \end{aligned} \tag{3.19}$$

This leads to the Bellman optimality equations:

$$\begin{aligned} V^* &= \max_{a} \Big[R(s,a) + \gamma \sum_{s'} P(s'|a,s)\, V^*(s)\Big] \\ Q^*(s,a) &= R(s,a) + \gamma \sum_{s'} P(s'|a,s)\ \max_{a'} Q^*(s',a') \\ \pi^*(s) &= \operatorname*{argmax}_{a} \Big[R(s,a) + \gamma \sum_{s'} P(s'|a,s)\, V^*(s)\Big] \\ &= \operatorname*{argmax}_{a} Q^*(s,a) \end{aligned} \tag{3.20}$$

In this work, we consider model-free RL approaches as it is very complex to model an multi-agent environment with a lot of uncertainties. Additionally, they have shown good generalization on big or even infinite state spaces. Compared to policy search techniques, they are better applicable for discrete action spaces. Both of these properties are met in the scenarios evaluated in this work, which makes model-free RL a valid choice.

3.2.4 Model-free Reinforcement Learning

As long as the transition probabilities are given, the Bellman equation can be used to optimally determine the state and state-action value function. These methods are called dynamic programming. However, as soon as there is no model of the environment, i.e. transition probabilities and reward function are not known, some other approach needs to be used. This is usually the case for real-world problems, because it is often hard to model the dynamics in a complex scenario. One way to cope with this is estimating the value function by temporal differences.

In temporal difference (TD) learning we update the value function based on the experience of the next n timesteps. In the simplest case, only the next timestep is considered, which is called TD(0) or one-step TD. In Equation (3.21) one can see the TD update for the one-step TD learning. Here the current value $V(s_t)$ is updated based on the difference of $R(s_t, \pi(s_t)) + \gamma V(s_{t+1})$, which is called the TD target, to the current estimate of $V(s_t)$. This difference is called the TD error.

$$V(s_t) \leftarrow V(s_t) + \alpha \Big[\overbrace{R(s_t, \pi(s_t)) + \gamma V(s_{t+1})}^{\text{TD target}} - V(s_t) \Big]_{\text{TD error}} \tag{3.21}$$

In comparison to this, Monte-Carlo methods consider not only the next time step but the entire episode until reaching a terminal state. The TD target is then the return of the episode, as given in Equation (3.13).

Q-learning

One model-free RL method to learn the optimal state-action value function $Q(s,a)$ using temporal difference learning is Q-learning given in Algorithm 3.1. It was first introduced by Watkins and Dayan [WD92] in 1992. The update equation is analogous to the Equation (3.21) but using the Q-function instead of the V-function.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[R(s_t, \pi(s_t)) + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right] \tag{3.22}$$

It has to be noted that Q-learning is an off-policy RL algorithm, which means that behavior policy and target policy do not have to be the same. Instead of using the behavior policy to determine the next action a' like e.g. SARSA [SB98], a greedy policy based on the current estimate of the Q-function is used. As a result, Q-learning finds the optimal policy independent of the policy it follows while interacting with the environment.

3.2.5 Exploration vs. Exploitation

One of the major challenges in reinforcement learning is the trade-off between exploration and exploitation. At each timestep the agent has to decide whether to greedily exploit its knowledge encoded in the Q-function or to randomly explore the environment in order to possibly acquire new knowledge. Therefore, an agent has to *exploit* its current experience in order to maximize the

Algorithm 3.1 Q-learning: Off-policy TD-Learning

procedure Q-LEARNING(α)
  Initialize $Q(s,a)$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$, arbitrarily except that $Q(s_T,\cdot) = 0$
  for each episode **do**:
    Initialize $s_t = s_0$
    Choose a_t from s_t using policy derived from Q // e.g., ϵ-greedy
    while s_t is not terminal **do**:
      Take action a_t, observe r_t, s_{t+1}
      $Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \left[R(s_t,\pi(s_t)) + \gamma \ \max_{a'} Q(s_{t+1},a') - Q(s_t,a_t)\right]$
      $s_t \leftarrow s_{t+1}$
    end while
  end for
end procedure

reward but it also has to *explore* to be able to make better decisions in the future. This dilemma is exclusive to reinforcement learning and does generally not occur in supervised or unsupervised machine learning.

One popular way of tackling this problem is the epsilon-greedy approach. At each timestep the agent takes a random action with a probability of ϵ and with a probability of $1-\epsilon$ it exploits its current policy greedily. The parameter ϵ has to be chosen carefully and is usually application specific. One common practice is to decrease ϵ over time, so that in the beginning of the training, the agent explores the state-action space a lot until it has collected enough experience and then solely exploits its policy. This can for example be done by a linear decay of ϵ given in Equation (3.23) or with an exponential decay in Equation (3.24).

$$\epsilon_t \leftarrow \epsilon_{\text{init}} - \nu \cdot t \tag{3.23}$$

$$\epsilon_t \leftarrow \epsilon_{\text{init}} \cdot \exp(-\nu \cdot t) \tag{3.24}$$

The decay factor ν needs to be defined depending on the use case. Often this is combined with some minimum ϵ_{min} to keep a little chance to explore throughout the interaction.

3.3 Deep Q-Network

When using Q-learning in very large state spaces a tabular representation of the Q-function is not feasible anymore. Therefore, neural networks can be used to approximate the Q-function. In 2013, Mnih et al. first introduced deep Q-networks [MKS+13] as a non-linear function approximator for Q-functions. It was very successful in playing Atari games and achieved superhuman performance. In order to train the neural network mean-squared error (MSE) was used as the loss function:

$$L_j(\theta_j) = \frac{1}{n}\sum_{i=0}^{n}\left(r_i + \gamma \max_{a'} Q_{\bar{\theta}_j}(s_{i+1},a') - Q_{\theta_j}(s_i,a_i)\right)^2 \tag{3.25}$$

Here j indicates the j-th update iteration and n denotes the batch size. It uses two networks whose parameters are indicated by θ and $\bar{\theta}$. Q_θ is approximated by the policy network and $Q_{\bar{\theta}}$ by the so-called target network. The policy network gets updated in each iteration whereas the target

network is kept frozen for several iterations until its parameters are set to be equal to θ every k-th iteration. If the policy network is used for both evaluating the current Q-function and estimating the target, this will result in updating the network based on a moving target. This can lead to oscillatory training behavior, as the target moves when approaching it.

Additionally, an experience replay memory is introduced in which each transition (s_t, a_t, r_t, s_{t+1}) throughout an episode is stored. In order to train the neural network a batch is sampled from the memory. Using this batch Equation (3.25) can be evaluated and the parameters of the network can be optimized using backpropagation and SGD. There are multiple reasons for using an experience replay instead of on-line learning. First, by storing each transition in the memory a single transition can be used for training potentially multiple times. Therefore, it is not as sample inefficient as on-line Q-learning. Additionally, there are strong correlations between consecutive transitions. These correlations are broken by random sampling from the experience memory, which leads to more stable training.

3.3.1 Double Update

Using the loss function given in Equation (3.25), an overestimation bias is introduced because of the max term. This is particularly problematic in the beginning of the training, when the Q-values are very noisy. The maximum Q-value might not be properly estimated yet and therefore not leading to the optimal action. This makes learning difficult and causes a slow learning progress in the beginning of the training. Initially, this effect has been discovered by Thrun and Schwartz [TS93] in 1993. In order to overcome this problem, Hasselt et al. [HGS16] introduced double DQN, which uses a different loss function, given in Equation (3.26). The selection of the action is done using Q_θ approximated by the policy network and the evaluation of the Q-function is given by $Q_{\bar{\theta}}$ approximated by the target network:

$$L_j(\theta_j) = \frac{1}{n}\sum_{i=0}^{n}\left(r_i + \gamma Q_{\bar{\theta}_j}(s_{i+1}, \underset{a'}{\operatorname{argmax}}\, Q_{\theta_j}(s_{i+1}, a')) - Q_{\theta_j}(s_i, a_i)\right)^2 \tag{3.26}$$

The separation of the action selection and the Q-value evaluation helps to learn faster and increases the stability of the learning process of the neural network.

3.3.2 Duel Architecture

Depending on the environment there can be regions in the state space where the actual choice of the action is not very relevant, whereas in other regions the choice of a specific action is crucial for the agent to survive. An example of this can be seen in Figure 3.6 where in the top part of the figure the specific action for the next timestep is rather irrelevant, because there are no other vehicles, which cause a risk to collide. In the bottom figure, there are many surrounding vehicles, which makes the immediate action crucial for staying alive.

Motivated by this observation, Wang et al. [WSH+16] introduced the duel architecture, where the value of a state s, $V(s)$, and the advantage of each action a in a state s, $A(s,a) = V(s) - Q(s,a)$, are approximated independently. Afterwards, both of these approximations are aggregated to get

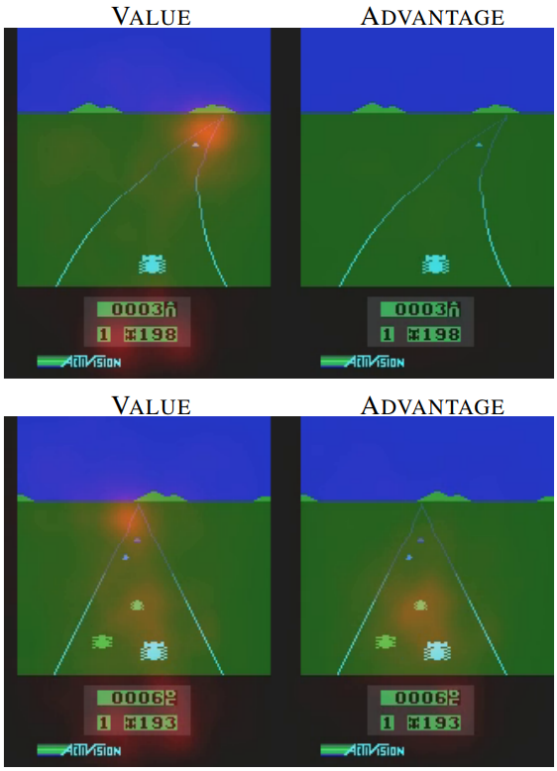


Figure 3.6: Illustration of the value and advantage streams in the duel architecture. In this example of the Atari game Enduro [WSH+16] the value stream at the top image, focuses on the horizon and the score, whereas the actual action given by the advantage stream is irrelevant. On the contrary, in the bottom image, the situation is critical. Here, the value focuses more on the nearby vehicles and the advantage stream tries to prevent collisions.

an estimate of $Q(s,a)$. These changes in the architecture of the neural network can be seen in Figure 3.7. The idea of this architecture is to more reliably estimate $V(s)$ without having to learn the effects for each of the possible actions a in state s.

One might assume that the aggregation is simply $Q(s,a) = V(s) + A(s,a)$ but this would not allow to recover V and A from a given Q. In order to solve this issue of identifiability the advantage estimator is forced to have zero advantage on the chosen action. The forward pass of the neural network then changes to:

$$Q_\theta(s,a) = v_\eta\big(f_\xi(s)\big) + a_\psi\big(f_\xi(s),a\big) - \frac{\sum_{a'} a_\psi\big(f_\xi(s),a'\big)\big)}{N_{actions}} \tag{3.27}$$

where $\theta = \{\xi,\eta,\psi\}$. f with parameters ξ is the network preceeding the last hidden layer. v with parameters η represent the value estimation with a single neuron, whereas a with ψ represent the layer for estimating the advantage.

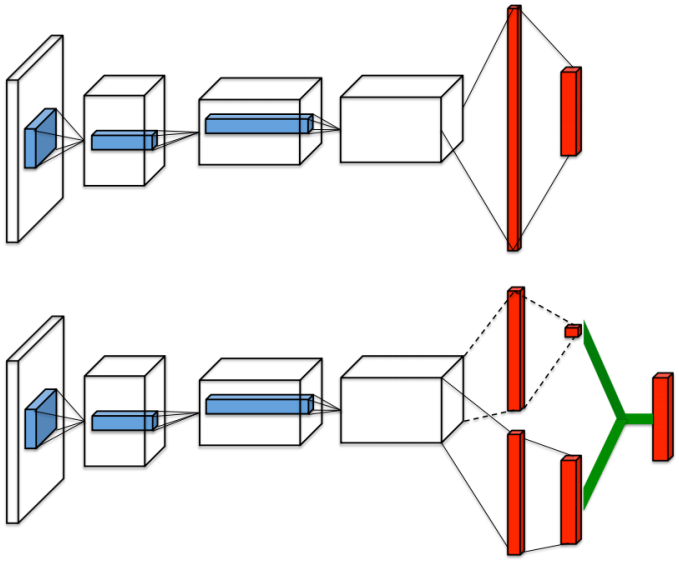


Figure 3.7: Duel Architecture [WSH+16] (bottom picture). The last hidden layer is separated into a single neuron for the value and an advantage layer with neurons for each action. Subsequently, the value and the advantage stream are aggregated in the output layer.

3.3.3 Prioritized Experience Replay

In 2016, Schaul et al. introduced prioritized experience replay (PER) [SQAS16]. It is based on the idea that there could be transitions in the replay buffer that contain more information than others but occur very rarely. In this case, uniform sampling from the replay buffer is information-inefficient. Therefore, it makes sense to prioritize the transitions in the replay buffer according to their information content. As a criterion whether a transition should be highly prioritized or not the TD-error is used. This is a valid choice because it measures how "surprised" the network is about this transition and how much it can learn from it.

It is computationally inefficient to recalculate the TD-errors after each parameter update of the neural network. Thereby, only the priorities of those transitions that are sampled from the replay buffer are updated. This implies that experiences, which initially have a very low TD-error, are never used for a parameter update. To cope with these problems a stochastic sampling, which interpolates between pure greedy prioritization and uniform random sampling, is needed.

Therefore, the probability of sampling a transition i is given by

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{3.28}$$

where p_i is the priority of transition i, p_k are the priority values for all other transitions in the replay buffer and α determines how much prioritization is used, with $\alpha = 0$ being uniform. One problem remaining is that when using stochastic gradient descent, the distribution of the data used for updating the network must match the distribution of the original data. This is not the case when prioritizing transitions based on the TD-error. As a consequence, a bias towards transitions with a

high priority can be observed. This bias can be corrected using importance sampling (IS), which introduces weights per transition:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)}\right)^{\beta}. \tag{3.29}$$

Here, N is the memory size and β controls how much IS is used, with $\beta = 1$ fully compensating for non-uniform $P(i)$. These weights are then multiplied with the corresponding TD-error in the Q-learning update in Equation (3.25).

3.4 Proportional Derivative Controller

As mentioned earlier, the high-level actions given by the policy trained by means of reinforcement learning need to be transformed into primitive low-level actions. In order to achieve this, PD controllers are used throughout this work. It is a closed-loop feedback control technique to control the behavior of a given system. The control signal $u(t)$ applied to the system is calculated using an error signal $e(t)$, which is the difference between the desired and the actual value of the quantity to be controlled.

The error signal and its derivative are weighted by the respective proportional and derivative contoller gains K_p and K_d. The control signal $u(t)$ is the sum of the weighted proportional and derivative error terms:

$$u(t) = K_p\ e(t) + K_d\ \frac{d\,e(t)}{dt}. \tag{3.30}$$

The controller gains K_p and K_d need to be derived depending on the system to be controlled.

4 Experimental Setup

In this section, the setup of the different experiments, which are conducted to empirically investigate our previously stated hypotheses, is explained. An overview over the individual components in the experimental setup is illustrated in Figure 4.1.

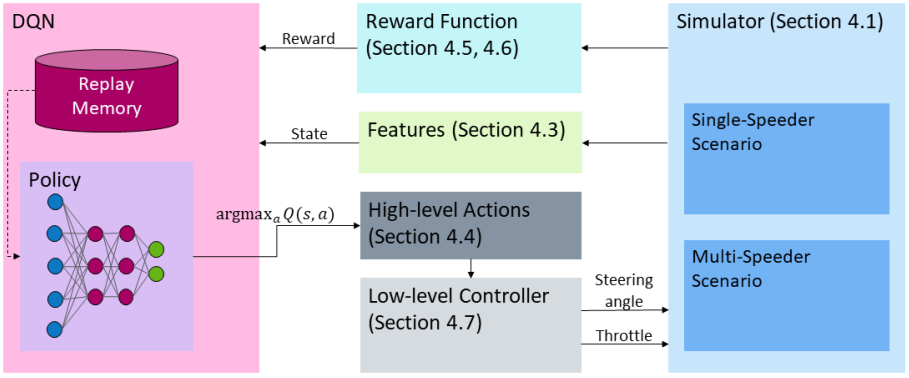


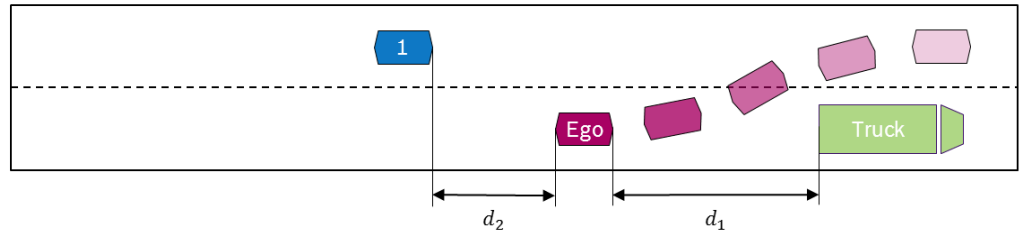
Figure 4.1: Illustration of the setup in the conducted experiments. The DQN algorithm trains a neural network policy. The training is performed by interacting with the simulator, which provides features and rewards. Two scenarios are designed within the simulator. The policy outputs high-level actions, which are transformed into low-level actions using low-level controllers.

First, the two different scenarios employed in the experiments are introduced and our design intent is explained. Afterwards, we describe three different baselines, which are used in Chapter 5 to evaluate the performance of the DQN algorithm. Next, the feature set is explained, which defines the state space of the MDP. The policy trained using DQN outputs high-level actions. In Section 4.4 the action space of the MDP is introduced. Subsequently, the reward function of the environment, which consists of multiple components, is presented. As there is a subset of sparse rewards among the reward function components such as rewards for reaching the goal or penalties for colliding into other vehicles as well as leaving the road, we use reward shaping to transform these sparse rewards into immediate rewards. The reward shaping algorithm used is explained in Section 4.6. Finally, we briefly introduce the controllers, which are used to transform the high-level actions output by the DQN policy into low-level actions such as steering angle and throttle.

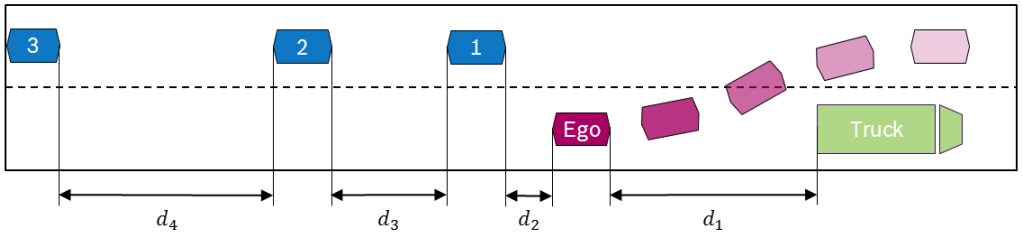
4.1 Scenarios

Two scenarios, given in Figure 4.2, were developed throughout this work. In order to assess the performance of the policy trained by means of reinforcement learning in different levels of complexity and difficulty, the scenarios are designed with varying numbers of traffic participants. Both of the scenarios are modeled as highway scenarios with two lanes and three different types of vehicles. Additionally, both of them are randomly initialized in terms of the positions and velocities of the vehicles.

The first scenario, illustrated in Figure 4.2a, consists of three vehicles including the ego vehicle. The ego vehicle is positioned behind a truck, which is slower than the ego vehicle. The goal of the



(a) Single-speeder Scenario



(b) Multi-speeder Scenario

Figure 4.2: Illustration of the two developed scenarios. The first scenario in Figure 4.2a contains just a single speeder with a distance d_2 between 5 m and 50 m. In Figure 4.2b we use three speeders where the distance d_2 is depending on the time-to-collision (TTC) between 0.0 s and 5.0 s, d_3 is between 0 m and 50 m to speeder 1 and d_4 is between 0 m and 100 m to speeder 2. In both scenarios d_1 is between 100 m and 200 m.

ego vehicle is to overtake the truck within a timeframe of 800 timesteps. On the target lane, another vehicle is positioned behind the ego vehicle, which is called speeder. The speeder is a potential risk for collisions as it has a higher initial velocity than the ego vehicle and aims to overtake both the ego vehicle and the truck. The sampling ranges for positions and velocities of all the traffic participants are given in Table 4.1.

In addition to the comparably easy first scenario, a second scenario with a higher difficulty, given in Figure 4.2b, was implemented. The increase of difficulty results from adding two more speeders. The position of the first speeder is determined based on the TTC to the ego vehicle, using the time-to-collision. The TTC with respect to a given vehicle is defined as the time until a collision with the ego vehicle occurs, given the velocities of both vehicles are fixed. The position of the first speeder is sampled such that the TTC is between 0.0 s and 5.0 s. The TTC is calculated based on the previously sampled initial velocities. As a result, the first speeder is located very close to the

Vehicle	Feature	Value
Ego	Velocity	$v_0 = 100\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -4$
	Lane-position	$s = 2\,600\,\text{m}$
Truck	Velocity	$v_0 \sim [70\,,\,90]\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -4$
	Lane-position	$s \sim [2\,700\,,\,2\,800]\,\text{m}$
Speeder	Velocity	$v_0 \sim [130\,,\,140]\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -3$
	Lane-position	$s \sim [2\,550\,,\,2\,595]\,\text{m}$

Table 4.1: Initial values and sampling ranges for each vehicle in the single-speeder scenario

ego vehicle independent of the initial velocities, so that it is difficult for the ego vehicle to start the lane change in the very beginning of the episode.

$$\text{TTC}_{\text{other}} = \frac{s_{\text{ego}} - s_{\text{other}}}{v_{\text{other}} - v_{\text{ego}} + \epsilon} \tag{4.1}$$

The position of the second speeder is sampled between 0 m to 50 m behind the first speeder. Finally, the position of the third speeder is sampled between 0 m to 100 m behind speeder 2. This results in four gaps between the ego vehicle and speeder 3. These gaps can be used for overtaking. The first gap, in front of speeder 1, is small and rather risky to take whereas the last one behind speeder 3 is large and safe. In this scenario, we expect more collisions to occur compared to the first scenario but in turn we expect that the algorithm learns to avoid collisions faster as it has more data to learn from.

In both scenarios, the desired ego vehicle behavior is defined as changing to the target lane and passing the truck. Hence, the overtaking procedure consists of just a single lane change without merging back in front of the truck. Additionally, we use action repetition in both of the scenarios, which means that we execute each selected action for two consecutive timesteps.

4.2 Baselines

As explained previously, we use two different scenarios to assess our reinforcement learning algorithms. In order to evaluate the performance of the policies obtained by reinforcement learning, we compare them to the performance of the baseline policies. These baselines range from random decisions to domain-specific heuristics hand-tuned for the considered scenarios. In the following, each of the baselines is explained and the differences among on the scenarios are outlined.

Vehicle	Feature	Value
Ego	Velocity	$v_0 = 100\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -4$
	Lane-position	$s = 2\,600\,\text{m}$
Truck	Velocity	$v_0 \sim [70\,,\,90]\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -4$
	Lane-position	$s \sim [2\,700\,,\,2\,800]\,\text{m}$
Speeder 1	Velocity	$v_0 \sim [130\,,\,140]\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -3$
	Lane-position	$s = s_{\text{ego}} - \left(v_{0_{\text{speeder1}}} - v_{0_{\text{ego}}} + 0.01\right) \cdot \text{TTC}, \quad \text{TTC} \sim [0.0\,,\,0.5]\,\text{s}$
Speeder 2	Velocity	$v_0 \sim [130\,,\,140]\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -3$
	Lane-position	$s \sim [\text{s}_{\text{speeder1}} - 50\,,\,\text{s}_{\text{speeder1}}]\,\text{m}$
Speeder 3	Velocity	$v_0 \sim [130\,,\,140]\,\frac{\text{km}}{\text{h}}$
	Lane	ID $= -3$
	Lane-position	$s \sim [\text{s}_{\text{speeder2}} - 100\,,\,\text{s}_{\text{speeder2}}]\,\text{m}$

Table 4.2: Initial values and sampling ranges for each vehicle in the multi-speeder scenario

4.2.1 Random Baseline Policy

The random baseline policy is a simple policy whereby actions are sampled randomly and kept for three consecutive time steps. This policy results in extremely bad driving behavior as it causes a lot of collisions, off-road driving, discomfort and it hardly reaches the goal.

4.2.2 Time-dependent Baseline Policy

In contrast to the random baseline policy, the time-dependent (TD) baseline policy uses more domain knowledge about the scenario. Initially one point in time is sampled. Then a lane change is executed for four seconds. After these four seconds of lane change, the policy outputs the lane keeping action until the termination of the episode. This baseline is optimal in terms of the execution of the lane change but not in terms of the timing. As the starting-time of the lane change is random, there is no active collision avoidance. Especially in scenarios where the target lane is occupied with many vehicles, it is likely that collisions occur. Off-road driving is prevented as the lane change is only performed for four seconds, which is exactly the time needed to fully complete the lane change.

4.2.3 Time-to-collision Baseline Policy

Different from the previously presented baseline policies, the TTC baseline policy includes information about the other vehicles in the scenario. Therefore, it is the only baseline policy that can successfully avoid collisions with other vehicles. The policy performs a lane change as long as the

TTC, given in Equation (4.1) is either greater than 5 s or less than −0.5 s for each of the speeders. One property of the TTC is that it approaches infinity when the velocities of the two vehicles are the same. Therefore, the time-to-headway (TTH), given in Equation (4.2), is further included into this policy. The TTH defines the time from now on until the other vehicle reaches the point where the ego is right now.

$$\mathrm{TTH}_{\mathrm{other}} = \frac{s_{\mathrm{ego}} - s_{\mathrm{other}}}{|v_{\mathrm{other}} + \epsilon|} \tag{4.2}$$

Here s_{ego} and s_{other} are the current longitudinal position on the lane of the ego vehicle and the other vehicle respectively and v_{other} is the velocity of the other vehicle in lane direction. Additionally, ϵ is set to 0.001 to avoid division by zero.

In addition to the TTC constraint, the TTH of the closest speeder needs to be less than -1 or greater than 1. More formally, the policy denotes as:

$$\pi(s) = \begin{cases} 1 & \forall sp \in SP: \ \mathrm{TTC}_{sp} \notin [-0.5\,,\,5.0]\,\mathrm{s} \wedge \mathrm{TTH}_{sp} \notin [-1.0\,,\,1.0]\,\mathrm{s} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

where SP is the set of speeders in the scenario.

In combination, these constraints ensure that there is either a sufficiently large gap to change the lane, or the speeders already passed and the lane is free. After successfully performing the lane change, the policy keeps the lane until the end of the episode.

In both of the scenarios, the TTC baseline is optimal in terms of avoiding collisions and reaching the goal. In terms of the time until reaching the goal, it is rather conservative, because 5 seconds to change the lane include some safety margin. Therefore, a reinforcement learning algorithm might beat this policy in terms of time needed to reach the goal, by driving more aggressively and changing the lane even with a TTC below 5 seconds.

4.3 State Space

The state space used in the experiments consist of a set of selected features. We chose a feature set covering global information about other vehicles, safety-critical information and information about the ego behavior. The complete list of features are listed in Table 4.3. They are composed of egocentric, safety and additional features.

The egocentric features are the heading, position, velocity, acceleration and Lane ID of all vehicles except the ego-vehicle, transformed into the ego coordinate system. These features cover overall data of the scenario and include all the basic information needed to navigate between the other vehicles. The features of the truck give the vehicle an indication of the goal location. In addition to these, safety features are calculated, including TTC and TTH for each of the speeders in the target lane. This additional information enables the algorithm to avoid collisions.

Besides this, additional features are included, which define the current state of the ego vehicle. We expect those features to be beneficial for the decision-making. First, the ego vehicle velocity is added, as this is not covered by the egocentric data. The algorithm should be aware of the current velocity, as a faster execution of the overtaking procedure is preferred. Moreover, constraints of traffic regulations regarding the speed could be included in the future. Next, the identifier of the

Type	Feature
Egocentric	Heading
	Position (Lateral, Longitudinal)
	Acceleration (Lateral, Longitudinal)
	Velocity (Lateral, Longitudinal)
	Lane ID
Safety	TTC
	TTH
Additional	Velocity (Ego)
	Lane ID (Ego)
	Lane position (Ego)
	Duration of lane change (in t)

Table 4.3: The feature set used for the experiments. It is split into three categories, namely egocentric, safety and additional. The egocentric features are calculated for each speeder and the truck, relative to the ego vehicle. The safety features are calculated for all speeders. The additional features are solely calculated for the ego vehicle.

ego vehicle's current lane indicates whether the lane change is performed or not. To avoid off-road driving we use the lateral position in the current lane of the ego vehicle, which is zero in the center of the lane and increases or decreases the more the ego vehicle moves to the left or the right respectively. Finally, the duration of the current lane change procedure is added in order to be able to learn action repetition. The duration of the current lane change is zero as long as lane keeping is performed and counts the time steps as soon as a lane change procedure has started.

4.4 Action Space

As mentioned above, we aim to make high-level behavior decisions in highway settings. The action space is discrete and consists of only two possible actions, namely keeping the lane or changing the lane to the left. The design intent of only two actions was regarding the simplicity of the experimental setup. In the future, a possible extension can be to add more actions such as changing the lane to the right or introducing different levels of acceleration to give the reinforcement learning algorithm more control over the behavior.

4.5 Reward Function

In the course of this investigation, the task of designing an appropriate reward function turned out to be unexpectedly challenging. We experienced that it is very important that each of the components are properly balanced, so that the algorithm does not focus on one particular aspect of the reward. When too much weight is given to collision avoidance or staying on the road, the policy found by reinforcement learning converged to lane keeping all the time. On the contrary, when too much

weight is assigned to goal reaching, the policy outputs mainly the lane changing action. Furthermore, the balance between immediate rewards like e.g. the comfort penalties and the sparse rewards turns out to be especially important. This is, because the immediate rewards are much easier to optimize for the RL algorithm, which often resulted in an over-focus. All the components of the reward function are listed in Table 4.4.

Component		**Parameter**
Terminal	Goal	5000
	Collision	-5000
	Off-Road	-5000
Comfort	$\text{Acceleration}_{\text{lat}}$	-1.690
	$\text{Acceleration}_{\text{long}}$	-0.130
	Jerk_{lat}	-0.014
	$\text{Jerk}_{\text{long}}$	-0.004
TTD		-1

Table 4.4: Components of the reward function

In order to calculate the reward in each timestep, we define three groups of rewards, namely terminal, comfort and time-to-destination (TTD) rewards. First the terminal rewards, are calculated using

$$r_{\text{terminal}}(t) = \sum_{c \in C_1} \vartheta_c \cdot \mathbb{I}\left[\text{pred}_{c,t}\right]. \tag{4.4}$$

ϑ_c indicates the parameters of each of the reward components $c \in C_1 := \{\text{goal}, \text{collision}, \text{off-road}\}$, shown in the right column of Table 4.4. Each reward component of the terminal rewards is given once a certain predicate $\text{pred}_{c,t}$ is fulfilled at a given timestep t. A predicate is for example defined as $s_t \in \mathcal{S}_{\text{goal}}$, meaning that the reward for reaching the goal is given as soon as the current state is a goal state. Analogously, penalties for colliding into other vehicles or leaving the road are given as soon as $s_t \in \mathcal{S}_{\text{collision}}$ or $s_t \in \mathcal{S}_{\text{off-road}}$ respectively.

As it is desired to find a policy that not only reaches the goal without colliding but also does this in a comfortable manner for the passengers, a comfort penalty is included into the reward function. Initially, the fluctuation of high-level actions was used for this purpose. Now, physical information obtained from the simulator is used, because it indicates more clearly how much discomfort a passenger is experiencing and is therefore easier to interpret. For both, acceleration and jerk, we use the squared lateral and longitudinal components, which has shown to be beneficial in motion planning tasks [GW06]. The comfort penalty for acceleration and jerk is

$$r_{\text{comfort}}(t) = \sum_{c \in C_2} \vartheta_{\text{acc}_c} \cdot a_c^2(t) + \vartheta_{\text{jerk}_c} \cdot j_c^2(t), \tag{4.5}$$

where ϑ_{acc_c} and $\vartheta_{\text{jerk}_c}$ denote the reward parameters for acceleration and jerk respectively with $c \in C_2 := \{\text{lateral}, \text{longitudinal}\}$. $a_c(t)$ and $j_c(t)$ are the acceleration and jerk measures respectively, obtained from the simulator.

Additionally, for each timestep we add a TTD penalty of -1. Finally the reward in each timestep is calculated using

$$r_t = r_{\text{terminal}}(t) + r_{\text{comfort}}(t) + r_{\text{ttd}} \tag{4.6}$$

Please note that the comfort penalties have some side effects. As soon as the ego vehicle reaches a minimal distance to the truck, the longitudinal controller of the car will decelerate, which causes a longitudinal penalty for deceleration and jerk. Similarly, the faster the ego vehicle is when starting the lane change, the higher the lateral acceleration will be in that point in time.

4.6 Reward Shaping

One of the main challenges in the examined scenarios are sparse rewards. A common technique to deal with sparse rewards is reward shaping, which was e.g. employed by Deisenroth et al. [DFR15]. If a binary reward is given when the agent's state is within a certain region of the state space, this reward can be approximated by a reward based on the distance to that region. In our scenario, we model this using Equation (4.7). The closer the ego vehicle gets to the goal, the more reward is given. Analogously, the closer the ego vehicle gets to other vehicles or the road boundaries the more negative reward it receives.

$$r_{t,c} = \xi_c \vartheta_c \exp\left(-\frac{1}{\eta_c} d_{t,c}^{\alpha}\right) \tag{4.7}$$

$c \in \{\text{goal}, \text{collision}, \text{off-road}\}$ refers to the reward component that is being shaped. $d_{t,c}$ is the distance from the ego vehicle to either the goal, the road boundary or closest vehicle, depending on c and time t. $d_{t,c}$ is always positive the absolute value is used when calculating distances. α determines the shape of the exponential function as depicted in Figure 4.3. $\alpha = 1$ was found to perform best in preliminary experiments. The width and slope of the exponential function is controlled by η_c, similarly to the standard deviation in a gaussian function. Similar to Equation (4.6), ϑ_c is the reward parameter defining the final reward given in the terminal state. I.e. ϑ_c is the magnitude of the exponential function. To scale the exponential function so that all these lower magnitude rewards sum up to 1, we use ξ_c. Empirically, we found that $\xi_{\text{goal}} = 0.4$, $\xi_{\text{collision}} = 0.33$ and $\xi_{\text{off-road}} = 0.4$ result in the best performance. In the future, one possible extension could be to learn these weights based on experience, as shown in an early draft by Jaderberg et al. [JCD+18].

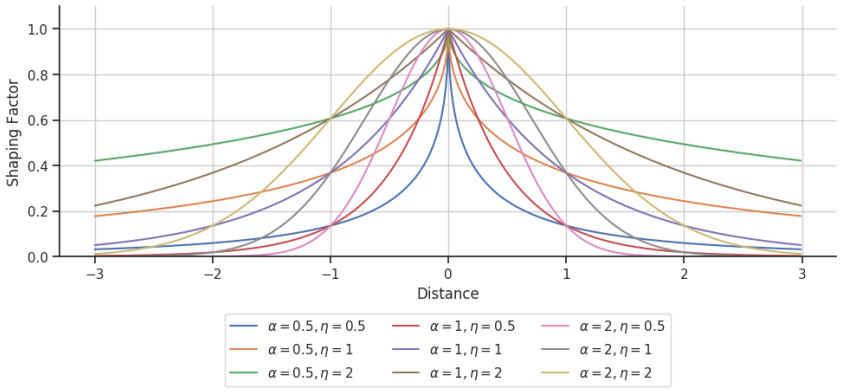


Figure 4.3: Illustration of reward shaping based on Equation (4.7) using multiple hyperparameters.

4.7 Low-level Controller

Our reinforcement learning algorithm learns a NN policy to make high-level decisions as explained in Section 4.4. However, the driving simulation, similarly to a real vehicle, requires physical inputs in the form of the desired steering angle and acceleration. Initially, the aim was to learn both a high-level policy, as well as low-level policies using the Option-Critic Architecture [BHP17]. However, this has shown to be very challenging as options often converged to the same policy. As a consequence, we decided to exploit domain knowledge and developed individual low-level controllers for each of the high level actions instead of learning low-level policies.

It is common practice to separate the controllers for complex vehicle behavior into distinguished lateral and longitudinal controllers [HCR09]. For our purpose, a longitudinal controller, based on Treiber et al. [THH00] is used, which is an autonomous cruise control (ACC) system. It calculates the acceleration to reach the given maximum velocity and reduces the acceleration in case other vehicles are present in the same lane in front of the ego vehicle. The control algorithm is given by Equation (4.8), where first the s^* measure is calculated and then used to determine the acceleration.

$$\begin{aligned} s^*(v, \Delta v) &= s_0 + Tv + \frac{v\Delta v}{2 \cdot \sqrt{a_{\max} a_{\min}}} \\ a_t &\leftarrow a_{t-1} - a_{\max} \cdot \left(\frac{s^*}{d_t}\right)^2 \end{aligned} \tag{4.8}$$

Here, v is the velocity of the ego vehicle, Δv is the approach rate towards the closest vehicle on the same lane in front of the ego vehicle and s_0 is the minimum distance to this vehicle. $a_{\max}$ and $a_{\min}$ are the possible maximum and minimum accelerations respectively and T is the reaction time. Lastly d_t denotes the current distance to the closest vehicle on the same lane in front of the ego vehicle.

Regarding the lateral control, two separate controllers are used, one for each of the possible actions. The controller for lane keeping aims to minimize the distance to the center of the road given by $e(t) = y_{\text{ego}}(t)$. Please note that $y_{\text{ego}}(t)$ is positive to the left and negative to the right of the center of the current lane. The lane change controller is illustrated in Figure 4.4. It is designed to perform a lane change within $T_{\text{lane change}} = 4\,\text{s}$. This is implemented by tracking a linear reference trajectory. Correspondingly, the PD controller for the lane change seeks to minimize the error

$$\begin{aligned} e(t) &= y_{\text{ego}}(t) - y_{\text{target}}(t) \\ y_{\text{target}}(t) &= \begin{cases} \frac{\text{timer}(t)}{T_{\text{lane change}}} \cdot d_{\text{lat}} & \text{if lane boundary not crossed} \\ \left(\frac{\text{timer}(t)}{T_{\text{lane change}}} - 1\right) \cdot d_{\text{lat}} & \text{if lane boundary crossed} \end{cases} \end{aligned} \tag{4.9}$$

between the reference trajectory and the actual position of the vehicle. $y_{\text{target}}(t)$ is the expected lane position at time t when following the linear reference trajectory of the controller. The calculation for $y_{\text{target}}(t)$ needs a distinction of cases, as the sign of $y_{\text{ego}}(t)$ changes, once the lane boundary is crossed. d_{lat} indicates the lateral distance the ego vehicle needs to travel from the current to the target lane in meters. timer(t) is counting the duration of the current lane change process in seconds.

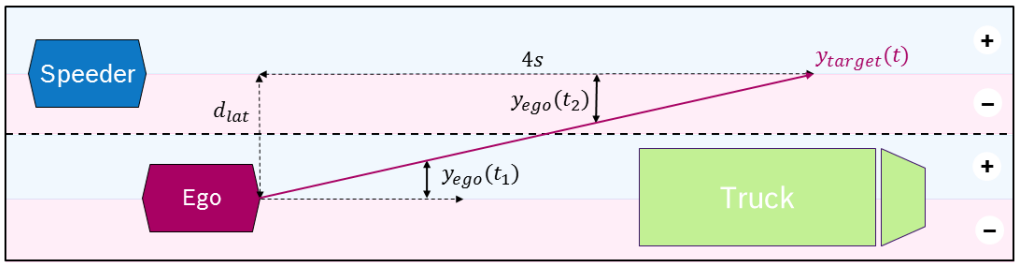


Figure 4.4: Illustration of the lateral controller. The linear reference trajectory $y_{\text{target}}(t_1)$ is depicted in purple. $e(t)$ is the lateral difference between the reference trajectory and the actual lateral position of the ego vehicle y_{ego}.

If the lane change is interrupted by choosing the lane keeping action in between the lane changing actions, the lane change is continued by resetting the timer to $\text{timer}(t) = \frac{y_{\text{ego}}(t)}{d_{\text{lat}}} \cdot T_{\text{lane change}}$. This ensures that the trajectory of the lane change looks always the same, independent of where the lane change starts and how random the policy is.

The parameters for the controllers introduced in Equation (3.30) were tuned by hand and are given by $K_p = -0.16649388$ and $K_d = -0.08438731$.

Even though the individual controllers result in stable lane changing and lane keeping maneuvers, a stable driving behavior is not guaranteed for any possible sequence of high-level maneuvers. In fact, we sometimes observed catastrophic oscillations when high-level decisions were switched in high frequency. This phenomenon is known in the literature [LA09] but cannot be fully avoided as the dynamics of the simulated vehicle are non-linear and unknown.

5 Evaluations

In this section, first the details of the experiments we conducted throughout this work are explained. Afterwards, the experimental results obtained from the experiments are presented.

In total, we conducted eight experiments, where we test the following four different variants of the DQN algorithm in two scenarios, to empirically validate our initially stated hypotheses.

- DQN trained with sparse rewards (DQN_{sparse})
- Linear DQN trained with shaped rewards (DQN_{linear})
- DQN trained with shaped rewards (DQN_{shaped})
- DQN initialized with behaivioral cloning (BC) using the TD baseline and trained with shaped rewards (DQN_{BC})

Each of the experiments is repeated five times with different random seeds to be able to properly evaluate the results. In all the experiments we use the DQN algorithm with double and duel extensions as well as prioritized experience replay. For exploration, epsilon greedy is used with a linear decay reaching its minimum at around half of the maximum number of environment steps.

In the first experiment, DQN_{sparse} is applied, where a NN with two hidden layers is used. The policy is trained solely given sparse rewards provided by the simulation environment. On the contrary, the policy of DQN_{linear} does not have any hidden layers but reward shaping is used as explained in Section 4.6. This yields a linear policy trained by means of DQN. We aim to see whether the deep neural architecture is necessary. This is relevant, because current research activities question the need of non-linearities in model-free RL [MGR18]. In the experiment with DQN_{shaped} the deep neural architecture with two hidden layers is combined with reward shaping. We expect this variant to outperform the two aforementioned because it receives immediate feedback and has the ability to learn non-linear functions. Finally, we conduct an experiment using DQN_{BC}, where weight initialization with behavioral cloning is applied. The TD policy introduced in Section 4.2.2 is used to collect 400 rollouts and supervised learning is applied to transfer the behavior into the initial DQN policy. We expect DQN_{BC} to learn faster than all the aforementioned algorithms because it already learned important aspects like action repetition and the duration of a lane change. Especially the overestimation problem, also addressed by the double update, should be vastly reduced with this initialization as the Q-function is not as noisy as the random initialization. However, DQN_{BC} has no accurate estimate of the Q-function as it is solely trained based on the cross entropy loss giving the states and actions of the TD baseline policy. We chose to use this baseline policy to initialize the network, so that no information about collision avoidance is given to the network prior to the simulation.

The DQN algorithm has a number of hyperparameters that need to be tuned depending on the task to be learned. A complete list of the parameters is provided in Table 5.1. The algorithms are executed for a total number of 5 million simulation steps using the Vires Virtual Test Drive (VTD)[1] simulator. Each simulation step corresponds to $\Delta t = 0.043\,\text{s}$. The default initialization for the neural networks, used in this work, is the Kaiming initialization [HZRS15] introduced by He et al. The NN is updated based on batches sampled from the prioritized experience replay with a batch size of 32 transitions. A network update is performed after each simulation step as soon as 200 steps have been performed to fill the replay buffer. The target network is updated every 2000 environment steps, which we found the most stable based on preliminary tests. As it is a finite environment we are using a discount factor of $\gamma = 1.0$. Even though the original paper of the DQN algorithm [MKS+13] mentions good results using the Huber Loss, we cannot confirm this in our preliminary tests, which is the reason we choose to use the MSE loss.

Hyperparameter	**Value**			
	$\text{DQN}_{\text{linear}}$	$\text{DQN}_{\text{sparse}}$	$\text{DQN}_{\text{shaped}}$	DQN_{BC}
Max. env. steps	$5 \cdot 10^6$	$5 \cdot 10^6$	$5 \cdot 10^6$	$5 \cdot 10^6$
Batch size	32	32	32	32
Loss	MSE	MSE	MSE	MSE
Learning rate	$9 \cdot 10^{-5}$	$9 \cdot 10^{-5}$	$9 \cdot 10^{-5}$	$9 \cdot 10^{-5}$
Hidden layer	**No**	64, 64	64, 64	64, 64
Weight init.	Kaiming	Kaiming	Kaiming	**TD Baseline**
Discount	$\gamma = 1.0$	$\gamma = 1.0$	$\gamma = 1.0$	$\gamma = 1.0$
Double update	True	True	True	True
Duel architecture	True	True	True	True
Initial ϵ	1.0	1.0	1.0	1.0
Minimum ϵ	0.01	0.01	0.01	0.01
Decay ϵ	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	$4 \cdot 10^{-7}$
PER capacity	$5 \cdot 10^5$	$5 \cdot 10^5$	$5 \cdot 10^5$	$5 \cdot 10^5$
Target update freq.	t=2000	t=2000	t=2000	t=2000
Gradient clipping	False	False	False	False
Optimizer	Adam	Adam	Adam	Adam
Start learning	t=200	t=200	t=200	t=200
Train freq.	t=1	t=1	t=1	t=1
Test freq.	ep=100	ep=100	ep=100	ep=100

Table 5.1: Hyperparameters of the DQN algorithm

The environment parameters are listed in Table 5.2. The function we used for reward shaping was introduced in Equation (4.7). Please note that we use action repetition, while interacting with the simulation environment. More precisely, the RL agent does not select a new action in every simulation step but every k steps. Here we use $k = 2$. This has already been used, e.g. by Mnih et al. [MKS+13], where every k-th frame of an Atari game is given to the RL agent. All the above parameters have been tuned on the single speeder scenario.

[1] https://vires.com/vtd-vires-virtual-test-drive/

Hyperparameter		Value			
		DQN_{linear}	DQN_{sparse}	DQN_{shaped}	DQN_{BC}
Action Repetition		2	2	2	2
Rand. Pos./Speed		True	True	True	True
Shaping	Goal	$\eta = 3.0, \alpha = 1$	**No**	$\eta = 3.0, \alpha = 1$	$\eta = 3.0, \alpha = 1$
	Collision	$\eta = 0.2, \alpha = 1$	**No**	$\eta = 0.2, \alpha = 1$	$\eta = 0.2, \alpha = 1$
	Off-Road	$\eta = 0.1, \alpha = 1$	**No**	$\eta = 0.2, \alpha = 1$	$\eta = 0.2, \alpha = 1$
Prevent Off-Road		True	True	True	True
Comfort		Physical	Physical	Physical	Physical

Table 5.2: Hyperparameters of the environment

5.1 Single-speeder Scenario

In this section the single-speeder scenario introduced in Section 4.1 is considered. First of all, the performance of the baselines introduced in Section 4.2 is presented, which we evaluated over 100 rollouts. The results are depicted in Figure 5.1.

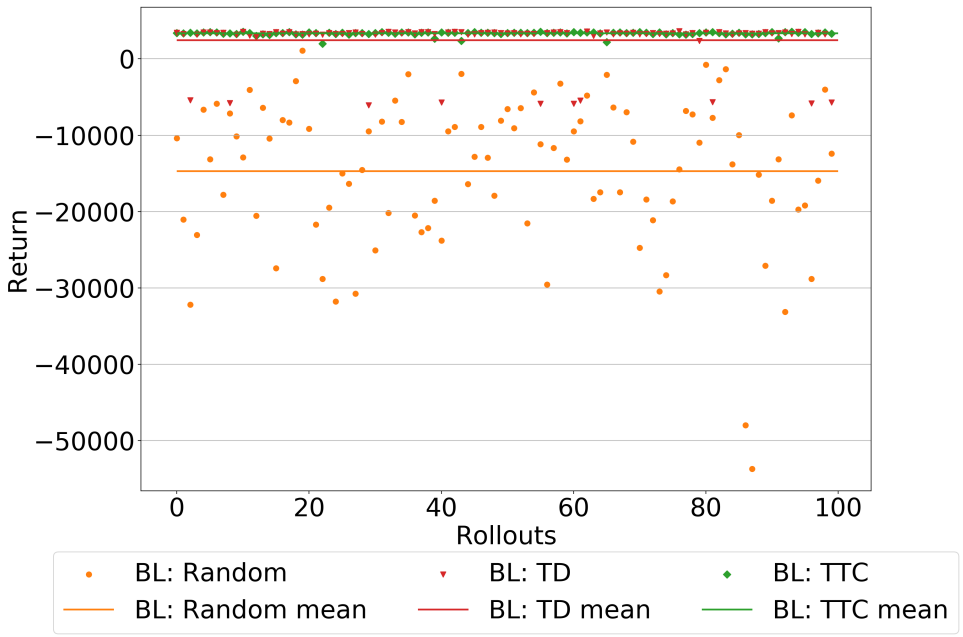


Figure 5.1: Result of baselines applied to single-speeder scenario

One can see that the assumptions about the baselines stated in Section 4.2 are met. The random baseline policy is the least effective, indicated by the low mean return. This is due to the high penalties for acceleration and jerk it receives by frequently changing the action. The TTC baseline policy performs best. It is very stable and consistently reaches the goal. In contrast to this, the TD baseline policy is slightly worse than the TTC baseline, and introduces more variance. This behavior is confirmed by Figure 5.2, which depicts the individual return components for each baseline. We

observe that the TTC baseline is able to fully prevent collisions, whereas this is not the case for the other two. Both the TD baseline policy and the TTC baseline policy perform optimal in terms of comfort as they show minimum acceleration and jerk.

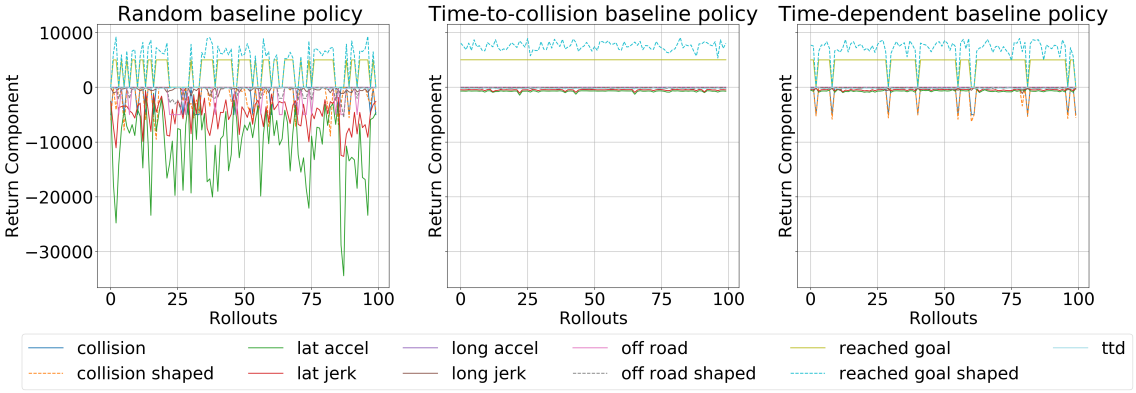


Figure 5.2: Resulting return components of the baseline policies applied to the single-speeder scenario

In Figure 5.3, the training process for each of the DQN variants can be seen. Each line depicts the mean over five random seeds, surrounded by twice the standard error, given by $\pm 2 \cdot \frac{\sigma}{\sqrt{N}}$, where σ is the standard deviation and $N = 5$ the number of random seeds. In this way, theoretically $\approx 95\,\%$ of the data is covered, assuming it is normal distributed.

The plotted data is collected by conducting a single test rollout after every 100 training rollouts. In these test rollouts the greedy policy is used, instead of the epsilon-greedy policy. As the test rollouts are conducted after every 100-th training rollout, the number of environment steps up to this point do not match for each of the different random seeds. Therefore, we split the x-axis into a regular grid with equidistant environment steps t, collect the nearest neighbors to the left for each of those timesteps per random seed and plot the mean and twice the standard error over the random seeds. This is the reason, why plateaus can be observed in the plot. Additionally, we plot the mean performance of the baselines with the error bar indicating twice the standard error over $N = 100$ rollouts.

In the upper left of Figure 5.3, the result for $\text{DQN}_{\text{shaped}}$ is depicted. It can be seen that initially the curve has a high variance but after around 2.5 million environment steps it converges to its maximum and mostly stays above the TTC baseline. We do not see a significant improvement of DQN_{BC} with weight initialization by behavioral cloning in the upper right plot. This is surprising, as the DQN_{BC} was expected to learn much faster as the previous variant. In fact, with the weight initialization the Q-function is not accurately approximated but we expected a lot less noise in the initial Q-function, which was assumed to result in more accurate estimations of $\text{argmax}_{a'} Q_\theta(s_{i+1}, a')$ in Equation (3.26). Even though the weight initialization achieved an accuracy of $\approx 97\,\%$ on the designated test set, the initialization seems not to be accurate enough and therefore not able to accelerate the learning process.

In comparison to those two variants, it can be seen that $\text{DQN}_{\text{sparse}}$ in the lower left, introduces a lot more variance and is eventually not able to reach the same level of performance as the aforementioned. This supports our hypothesis that reward shaping is crucial in this sparse reward scenario. Similarly,

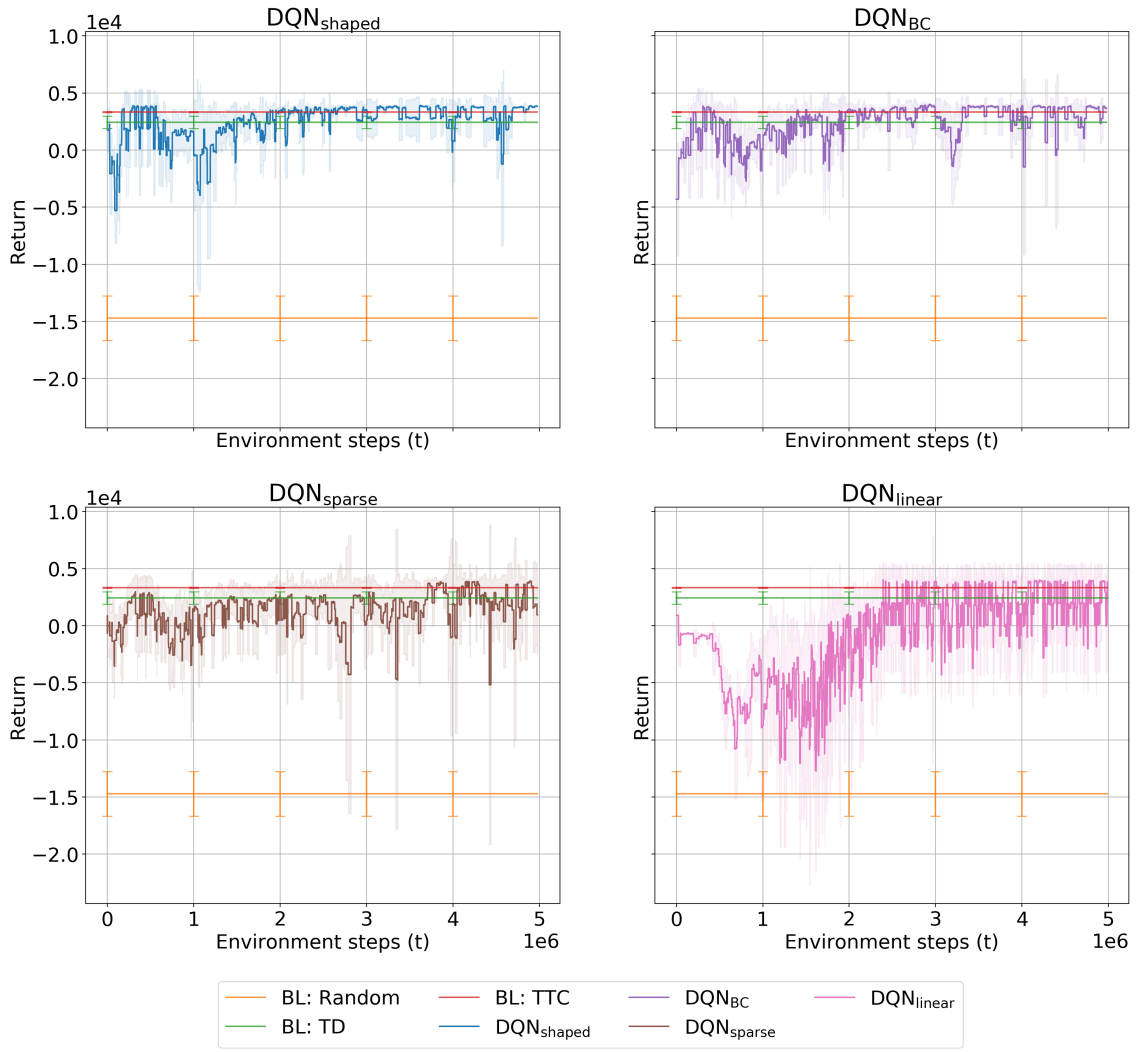


Figure 5.3: Results for the different DQN variants compared to the baselines

we see in the lower right the respective plot for DQN_{linear}. In this case, it takes around 3 million environment steps to converge, with a very high variance even in the final stage of the learning process. This supports the hypothesis of the necessity of non-linear function approximators in our usecases.

To further investigate the learning process, the return is split into its components and is plotted for each of the experiments in Figure 5.4. The dashed lines hereby depict the shaped rewards, which are only used during the training in the respective experiments. For DQN_{shaped} and DQN_{BC}, it can be seen that the major waste of rewards occur due to only occasionally reaching the goal. These policies can be considered a conservative, as they reduce the amount of collisions to zero but occasionally do not find a suitable moment to change the lane. The penalties for acceleration and jerk are successively decreased to a minimum. In contrary, the main penalty in the case of the DQN_{sparse} are high comfort penalties, which can not be reduced, even towards the end of the training. Regarding the DQN_{linear}, it can be observed that the main penalties are given due to collisions, which cannot be fully avoided.

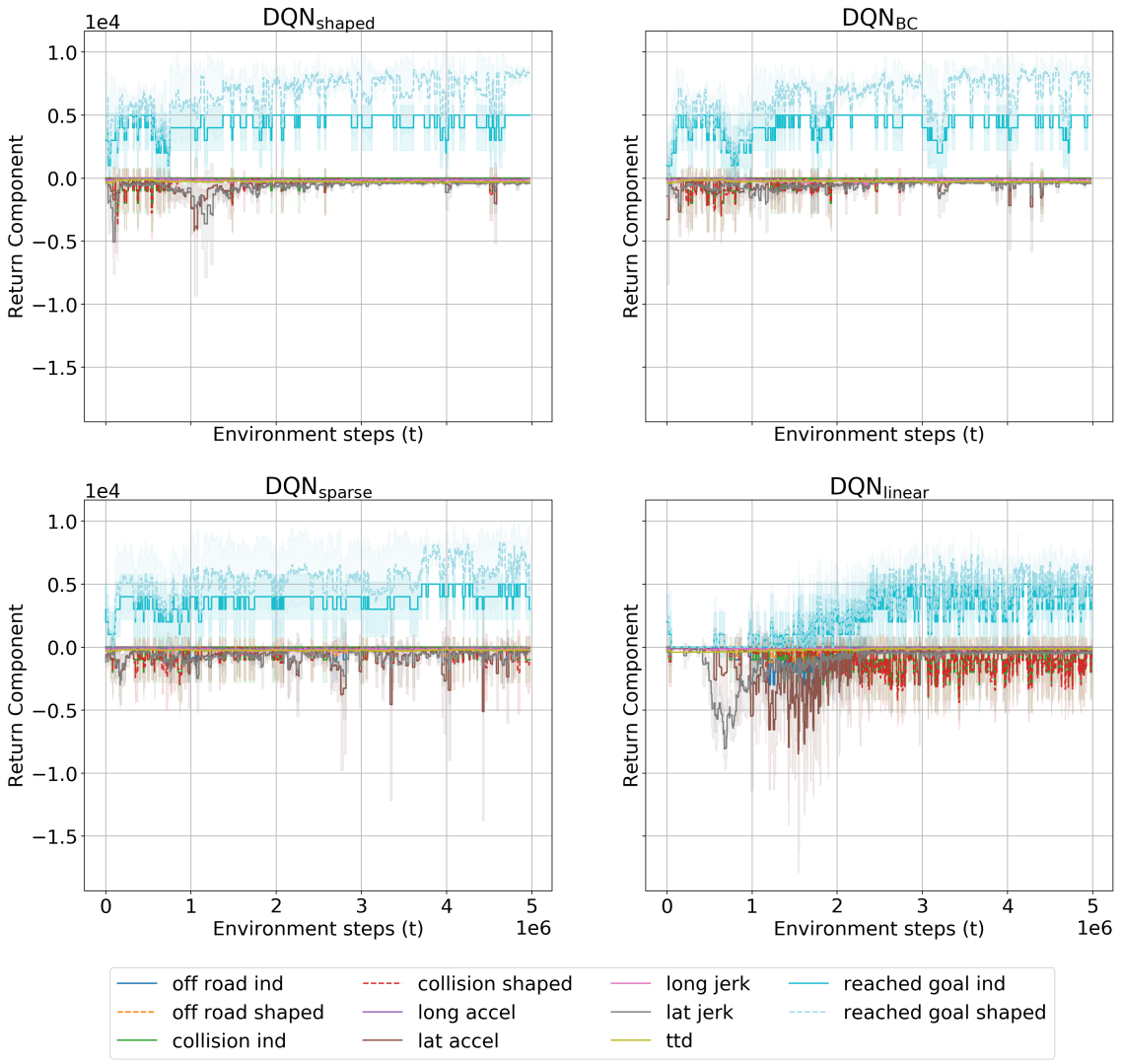


Figure 5.4: Return components for the different DQN variants.

At the end of the training process 20 test rollouts are conducted using the final greedy policy for every random seed. The bar chart in Figure 5.5 shows the average outcome per DQN variant over random seeds.

It can be seen that none of the DQN variants leaves the road, which might seem obvious as we prevent off-road driving but preliminary experiments showed that frequent action changes can result in oscillations. Therefore, the absence of off-road driving indicates that action repetition was learned. On average, the $\text{DQN}_{\text{shaped}}$ reaches the goal in 100 % of the test rollouts, supporting the hypothesis that DQN is able to avoid collisions and finally reaches the goal. The DQN_{BC} seems to find a slightly more conservative policy, which on average reaches the goal in 92 % and times out in 8 % of the simulations. Considering the high standard deviation, we assess DQN_{BC} to be comparable to $\text{DQN}_{\text{shaped}}$. The two remaining variants cannot fully avoid collisions with other vehicles. On average, in 85 % of the rollouts the $\text{DQN}_{\text{sparse}}$ reaches the goal, while in 14 % collisions occur and in 1 % of the rollouts the simulation is timed out. The standard deviation for collisions

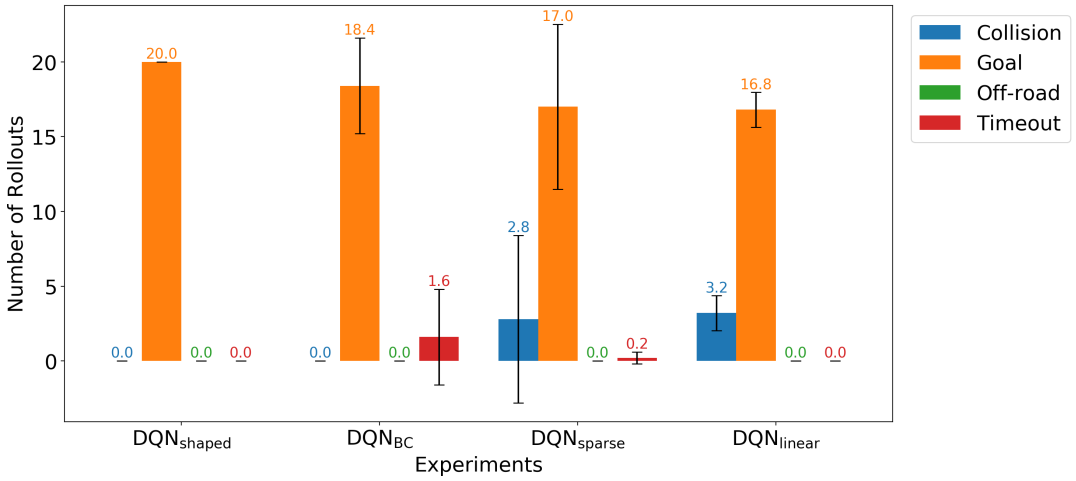


Figure 5.5: Outcomes across 20 test rollouts per DQN variant, averaged over random seeds

and goal are also high, suggesting that for only a few random seeds, there are rollouts resulting in collisions. On the contrary, the $\text{DQN}_{\text{linear}}$ version reaches the goal on average in 84 % but cannot avoid collisions in 16 % of the simulations. The bars for $\text{DQN}_{\text{linear}}$ show less standard deviation, which highlights the robustness of these results.

In order to further assess, the trained policies, we depict the average returns in Figure 5.6 using a box plot. Additionally, the mean and twice the standard error of the TD and TTC baseline policies are shown, by the dotted lines. We neglect the random baseline here, due to visibility reasons. Again, less variance can be observed for the $\text{DQN}_{\text{shaped}}$, than for any other DQN variant. This

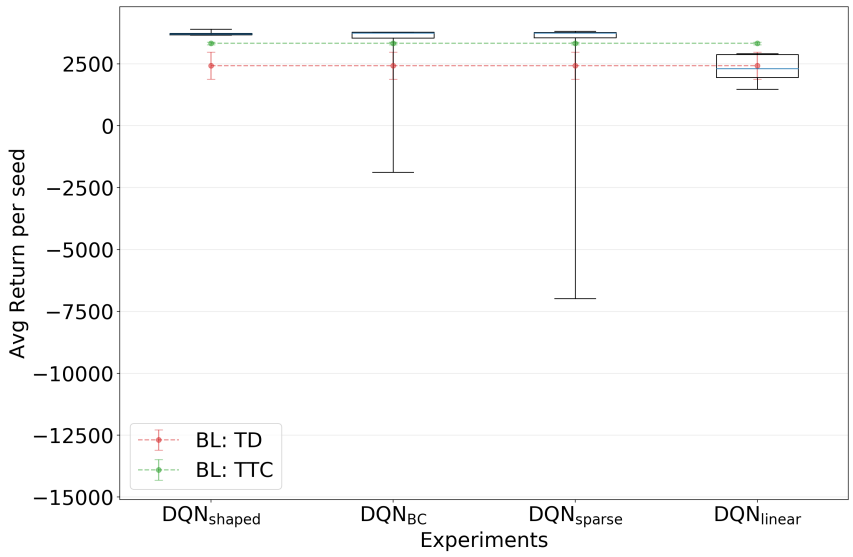


Figure 5.6: Box plot given by average returns per random seed per DQN variant. Each box ranges from the lower to the upper quartile with the median indicated in blue. The whiskers denote the range of the data points. Outliers are included in the plot.

indicates that the training is more robust over different random seeds for the $\text{DQN}_{\text{shaped}}$ algorithm.

Hence, when reward shaping is used in combination with the DQN algorithm the training is reliable in this scenario. The medians for DQN_{BC} and DQN_{sparse} are comparable to DQN_{shaped} but they show a considerably higher variance, due to outliers. Considering only the upper to lower quartile, DQN_{shaped}, DQN_{BC} and DQN_{sparse} show comparable results. They are all better than the TTC baseline, indicating optimal behavior in this scenario. DQN_{linear} achieves much lower returns, than all the others, which underlines the inability to avoid collisions already observed in Figure 5.5.

In order to investigate the final policies with respect to comfort, we depict the average return only considering the comfort penalties in Figure 5.7. Considering only the lower to the upper quartile,

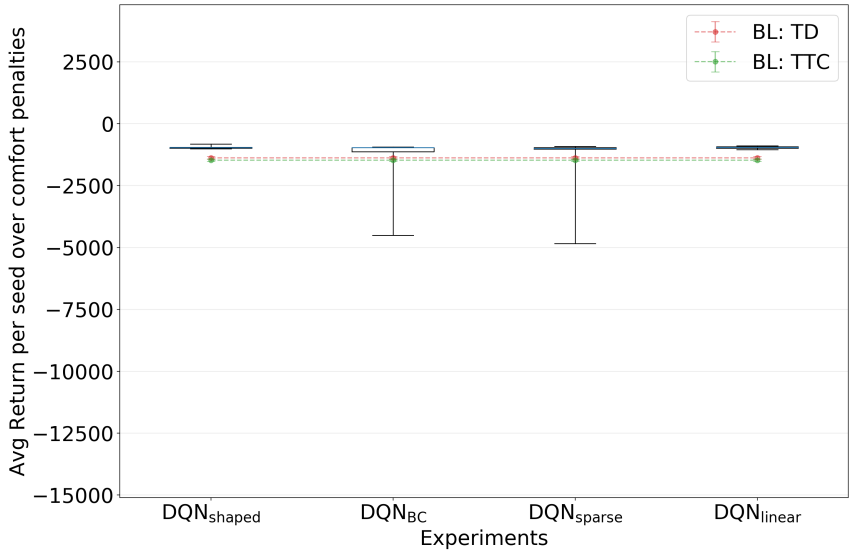


Figure 5.7: Boxplot given by average returns per random seed solely computed over comfort penalties per DQN variant. Each box ranges from the lower to the upper quartile with the median indicated in blue. The wiskers denote the range of the data points. Outliers are included in the plot.

all four DQN variants show comparable average returns. The plot does not show any outliers for DQN_{shaped} and DQN_{linear} and moreover small boxes indicate a stable policy in terms of comfort. On the contrary, the Figure suggests that the outliers in case of DQN_{BC} and DQN_{sparse}, already seen in Figure 5.6, are due to outliers of the comfort penalties. It seems that there is high action jitter in a few trajectories.

The action sequences of the final test rollouts for each of the DQN variants are presented in Figure 5.8. Each plot shows 100 action sequences, where the 20 test rollouts with the final greedy policy per random seed are concatenated. The y-axis indicates the time within a rollout in environment steps, while the x-axis covers the five random seeds with 20 test rollouts each. Light blue and green indicate the chosen actions, whereas the remainder in the top of the Figure is colored according to the outcome of the simulation. Please note that the displayed actions are the ones selected by the DQN policy. As off-road driving on the target lane is prevented, i.e. changing the lane for too long will be ignored, the plot does not necessarily show the actually executed actions, especially towards the end of an episode.

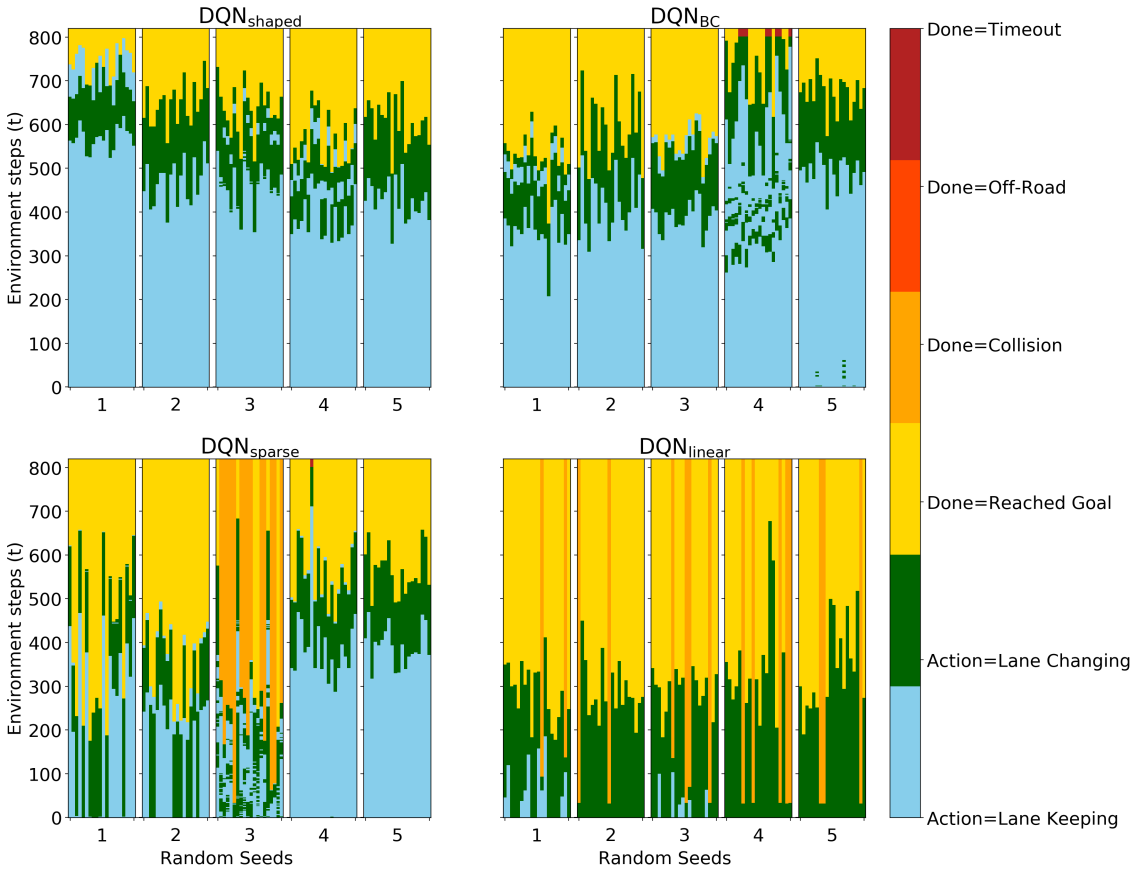


Figure 5.8: Actions given by final policies over 20 test rollouts and 5 random seeds applied to the single-speeder scenario. The x-axis shows the 5 random seeds with 20 test rollouts each, whereas the y-axis shows the environment steps per rollout. The colors indicate on the one hand the actions selected by the final policy and on the other hand the outcome after the rollout terminates.

The figure underlines the findings of the previous figures. One can observe a consistent behavior over multiple random seeds for $\text{DQN}_{\text{shaped}}$ in the upper left. It seems that the ego vehicle waits until the speeder passed and then changes the lane. DQN_{BC} shows a similar behavior but in one random seed it does not always reach the goal. Additionally, DQN_{BC} has more jitter in the actions, leading to higher comfort penalties, as previously seen in Figure 5.7. In the bottom left, the $\text{DQN}_{\text{sparse}}$ policy is depicted. Here, the policy shows a comparable behavior to the $\text{DQN}_{\text{shaped}}$ policy but the third random seed results in an extremely poor behavior. Compared to this, $\text{DQN}_{\text{linear}}$ does not seem to learn to avoid collisions at all. For most of the random seeds, it converges to a policy that chooses to change the lane from the start, which obviously leads to many collisions.

As the previous figures do not indicate how the ego vehicle behaves in relation to the other vehicles, we depict an example trajectory of $\text{DQN}_{\text{shaped}}$ for four different timestamps in Figure 5.9. Initially, the ego vehicle and the speeder are positioned very close to each other. Therefore, the ego vehicle waits and starts the lane change as soon as the target lane is clear. This can be considered as an example of optimal behavior, because it actively avoids a collision as it lets the speeder pass and reaches the goal as comfortable as possible.

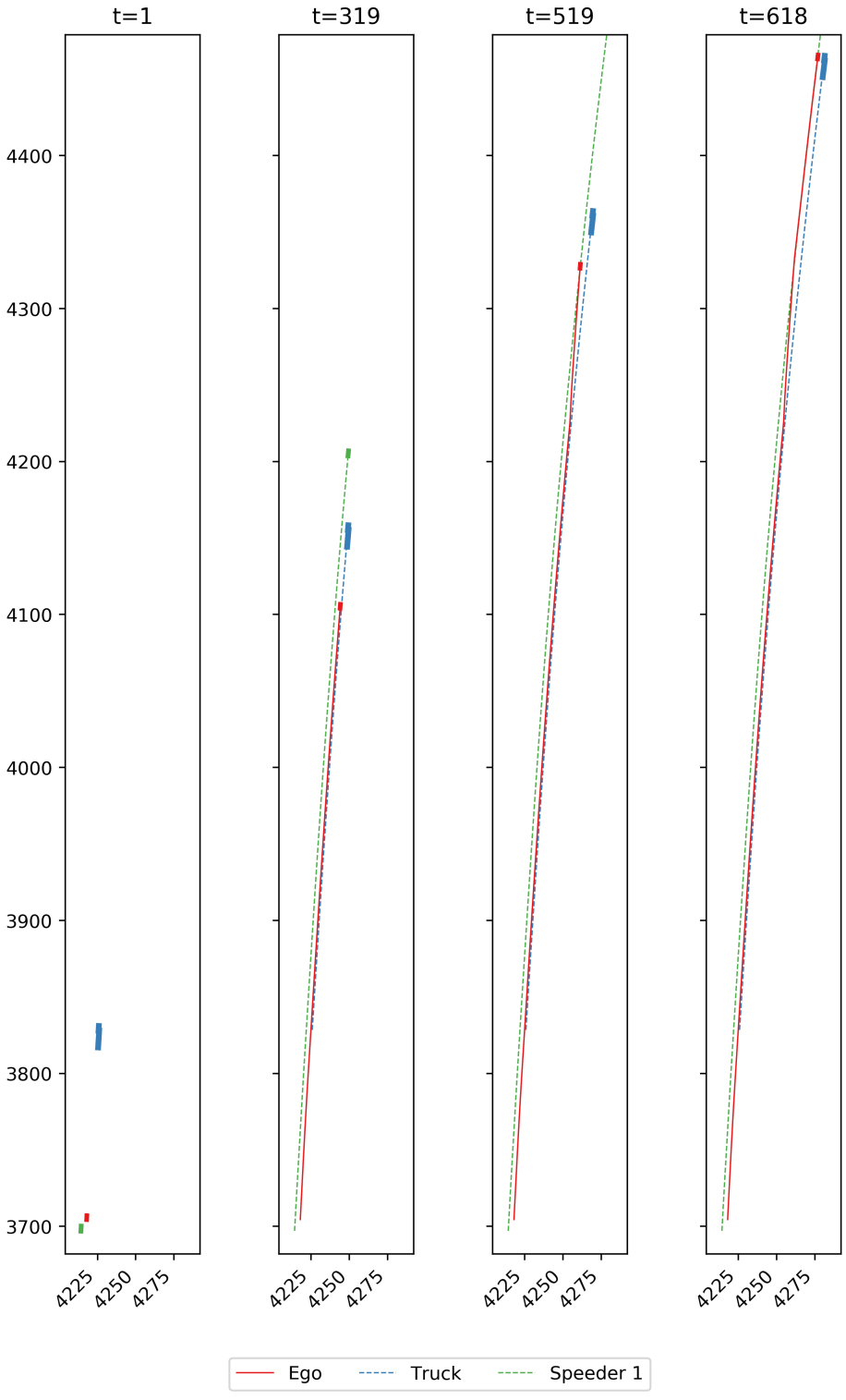


Figure 5.9: Example trajectory for a single rollout depicted for four different timesteps. In the left image, the initial sampling of the vehicles can be seen. The speeder is very close to the ego vehicle. In the central left image, the speeder just passed the ego vehicle and the truck. In the central right image, the lane change has just been completed. Finally, in the right image, the ego vehicle reaches the goal on the target lane next to the truck.

In summary, it can be confirmed that the policy trained using $\text{DQN}_{\text{shaped}}$ is capable of reaching the goal without colliding with other vehicles in more than 90 % of the final test runs in case of the the single-speeder scenario. We additionally showed that the $\text{DQN}_{\text{shaped}}$ algorithm also finds an optimal policy regarding the comfort, with respect to acceleration and jerk. Furthermore, our results show that deep non-linear policy architectures are of particular need for Q-networks in this scenario. Reward shaping does not only accelerate the training but also reduces the variance of the resulting policy. We cannot confirm that the weight initialization based on the TD baseline accelerates or improved the DQN algorithm, as there is no significant performance gain or speed-up.

5.2 Multiple Speeder Scenario

After reviewing the results for the single-speeder scenario, now the multi-speeder scenario is considered. In Figure 5.10, the results of the baseline policies are shown. The TTC baseline policy performs best, like in the single-speeder scenario. The difference between the TTC and the TD baseline is larger for this scenario compared to the single-speeder scenario, because the TD baseline collides more often with other vehicles. This can be seen in Figure 5.11, which displays

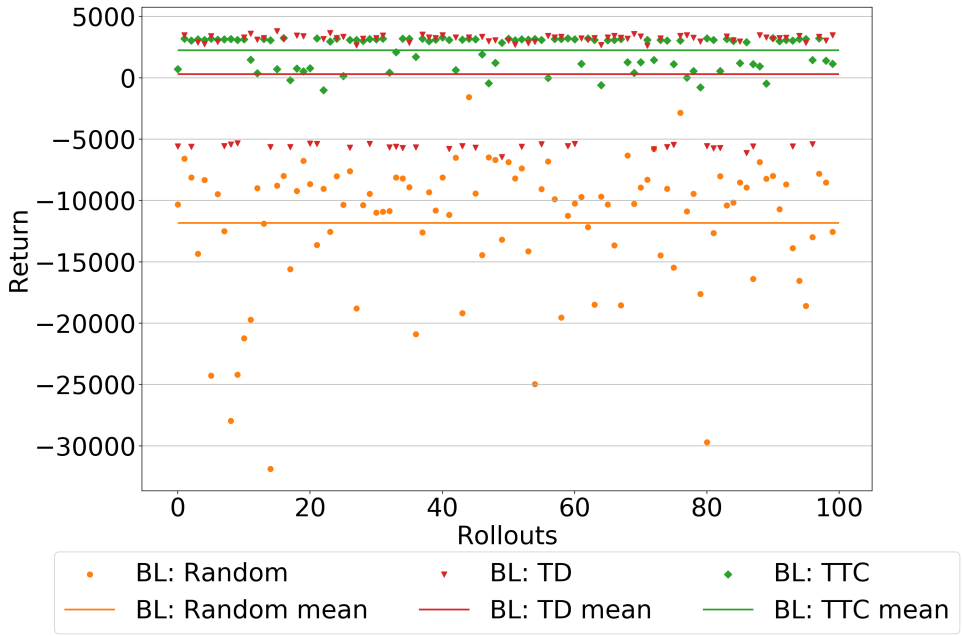


Figure 5.10: Results of baselines applied to multi-speeder scenario

the individual components of the return. Similarly to the single-speeder scenario, the TTC baseline fully avoids collisions. Although, the TTC is safe, it shows more acceleration and jerk compared to the single-speeder scenario, resulting in discomfort for the passengers. This is because it frequently attempts to change the lane but cannot always complete the lane change before the following speeder gets too close. As a result, the lane change is aborted and the ego vehicle returns to the initial lane. This is not the case for the TD baseline, as it never aborts lane changes. Analogously to the single-speeder scenario, the random policy only occasionally reaches the goal and receives high comfort penalties due to frequent action changes.

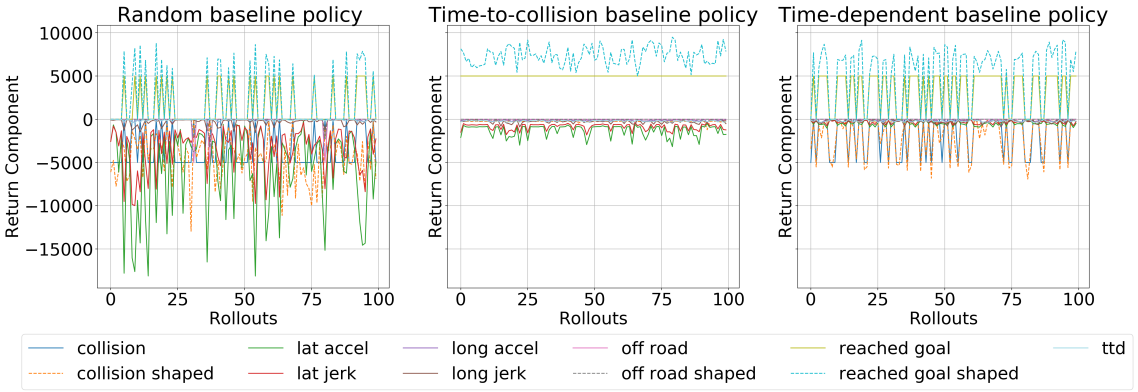


Figure 5.11: Resulting return components of the baseline policies applied to the multi-speeder scenario

In Figure 5.12, the results for each of the DQN variants applied to the multi-speeder scenario are depicted. Like in the single-speeder scenario, we conduct a test rollout after every 100-th training rollout and show the average return across five random seeds and twice the standard error. A significant drop can be observed in case of the $\text{DQN}_{\text{shaped}}$ towards the end of the training. It is difficult to interpret but one explanation for the instability in the training across different random seeds could be that the hyperparameters of the DQN training algorithm are not properly tuned for the multi-speeder scenario. In comparison, DQN_{BC} achieves more stable results. It exceeds the average TTC baseline policy within the first million timesteps. $\text{DQN}_{\text{sparse}}$ shows better results than in the single speeder scenario. One possible explanation for this could be that it gets more feedback in terms of collisions than in the single-speeder scenario. In the multi-speeder scenario there is a higher potential for collisions due to a higher number of vehicles on the target lane. Therefore, the $\text{DQN}_{\text{sparse}}$ algorithm receives potentially more data to learn collision avoidance. It reaches a superior performance than the TTC baseline between 2 and 3.5 million timesteps but towards the end of the training it shows a greater variance. In comparison to the three aforementioned DQN variants, $\text{DQN}_{\text{linear}}$ in the bottom right, learns much slower and exhibits very high variances. The $\text{DQN}_{\text{linear}}$ reaches the performance level of the TTC baseline very briefly after around 4.2 million timesteps but drops again towards the final training stage.

In Figure 5.13, the individual return components for each of the DQN variants are considered. One can observe that in case of $\text{DQN}_{\text{shaped}}$ the dominant negative return components towards the end of the training are lateral acceleration and lateral jerk. This indicates that the $\text{DQN}_{\text{shaped}}$ policy changes actions frequently and does not keep the action for changing the lane during the entire lane changing process. This is surprising, as we expected action repetition to be easy to learn, due to extensive exploration in the first stages of the training. This not only leads to strong discomfort but also causes the ego vehicle to not always reaching the goal and occasionally driving off-road. $\text{DQN}_{\text{sparse}}$ learns action repetition fairly quickly within 2 million environment steps. This is expected as the comfort penalties are immediate rewards, compared to the sparse rewards for the terminal outcomes. However, it experiences higher variances in the comfort terms and a drop in goal reaching towards the end of the training but does not show any collisions. This indicates that $\text{DQN}_{\text{sparse}}$ learns a conservative policy. In case of DQN_{BC}, no drop in the final training stages are observable. In general, DQN_{BC} is the most stable DQN variant across the entire training, as it has low variances in comfort penalties and mostly reaches the goal. Contrarily, in the case of $\text{DQN}_{\text{linear}}$ off-road

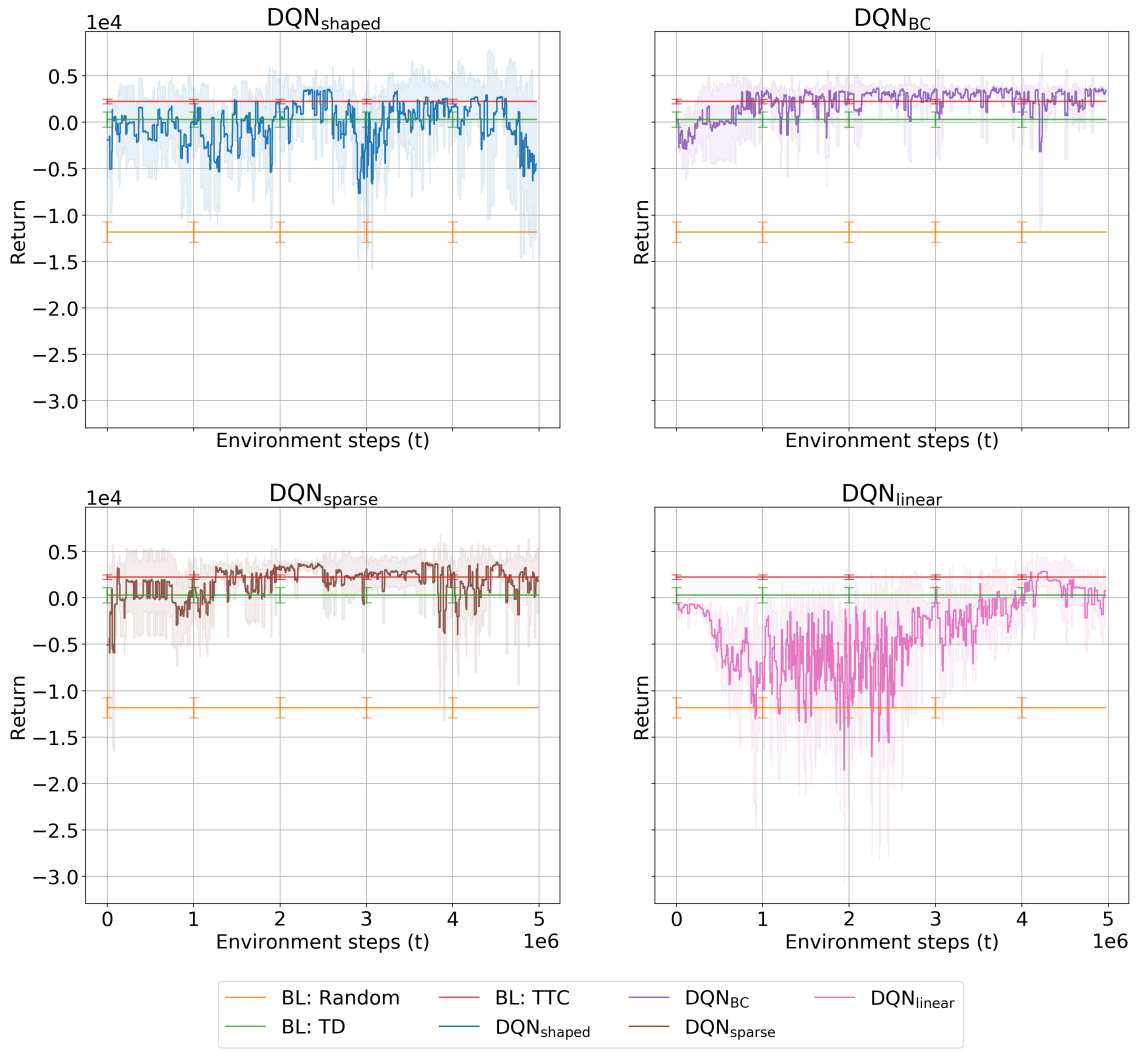


Figure 5.12: Results for the different DQN variants compared to the baselines

driving and high lateral acceleration and jerk frequently occur in the first 3 million environment steps of the training. Towards the end, the comfort penalties are reduced but the goal is only reached occasionally and collisions can still not be avoided.

Like in the single-speeder scenario we conducted 20 test rollouts with the final policy after the training for each of DQN variants using five different random seeds. In Figure 5.14, we show the outcomes of these final test rollouts using a bar chart. Compared to the single-speeder scenario, $\text{DQN}_{\text{shaped}}$ performs much worse in the multi-speeder scenario. It reaches the goal in only 64 % on average over the random seeds, while timing out in 27.9 %, driving off-road in 6 % and colliding in 2 % on average. In comparison, the DQN_{BC} outperforms all other DQN variants with 100 % success throughout the final test rollouts. $\text{DQN}_{\text{sparse}}$ reaches the goal in 82 % on average, while timing out in 16 % and colliding in 2 % on average. $\text{DQN}_{\text{linear}}$ reaches the goal in 60 %, while timing out in 39 % and colliding in 1 % on average.

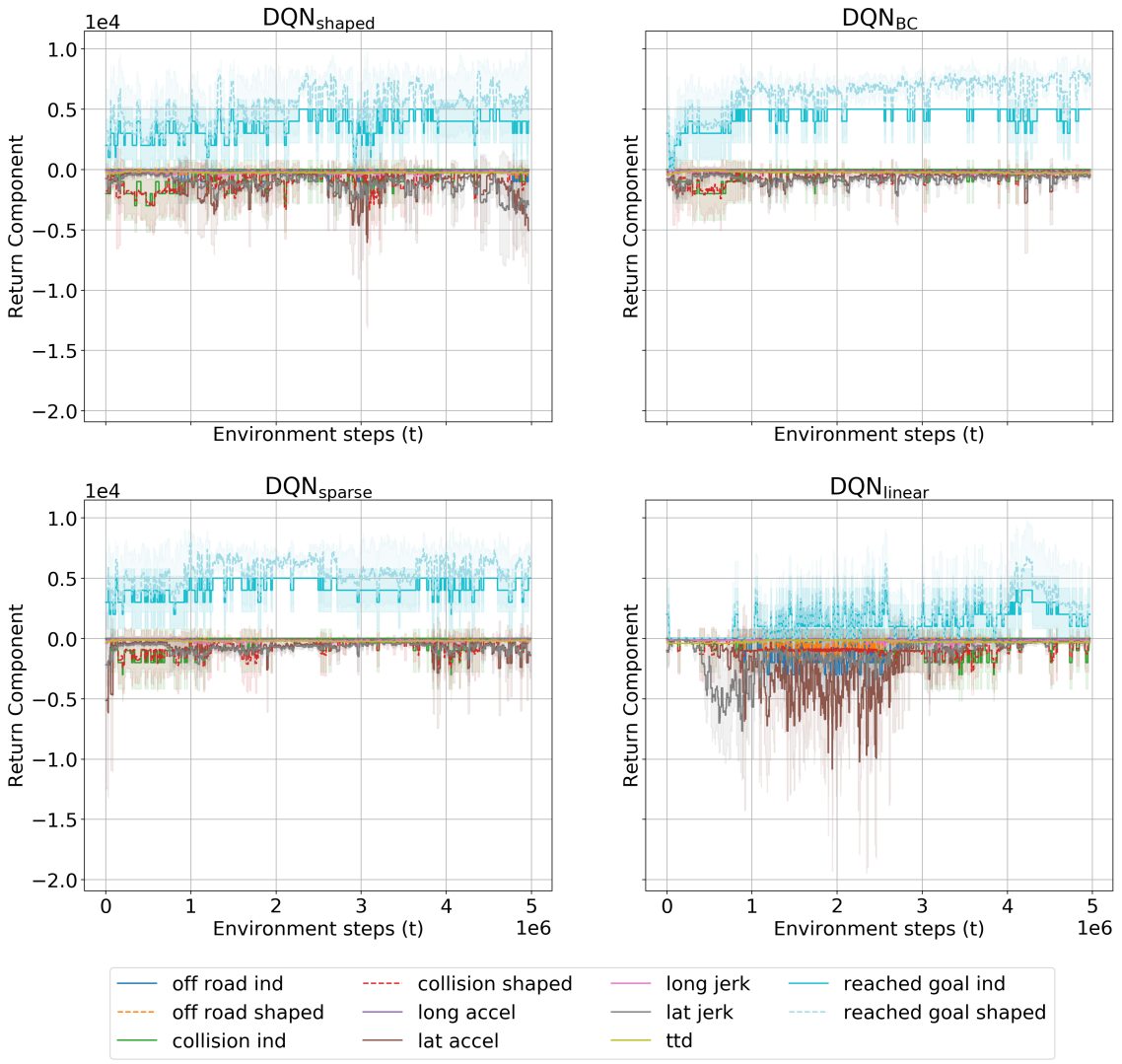


Figure 5.13: Return components for the different DQN variants.

In this scenario, the weight initialization with behavioral cloning proves to be beneficial for collision avoidance, as all other variants are not capable of fully preventing collisions. Surprisingly, $\text{DQN}_{\text{shaped}}$ reaches the goal less often than $\text{DQN}_{\text{sparse}}$. The shaped rewards for goal and collisions, which give immediate feedback to the RL algorithm, seem not to be beneficial in this scenario. $\text{DQN}_{\text{linear}}$ seems to converge to a conservative policy, as it shows a high proportion of timeouts but almost fully avoids collisions. In the multi-speeder scenario, almost all DQN variants show high standard deviations, indicating that more random seeds should be used.

In order to further investigate the performance of the final DQN policies, box plots of the averaged return are depicted in Figure 5.15. Each of the boxes ranges from the lower to the upper quartile of the averaged return per random seed. The dotted lines indicate the average return of the TD and the TTC baselines. We neglected the random baseline in this case, due to visibility reasons. It can be observed that the variance of $\text{DQN}_{\text{shaped}}$ compared to the other DQN variants and the baselines is extremly high, mainly due to high comfort penalties. In case of the DQN_{BC}, one can observe a

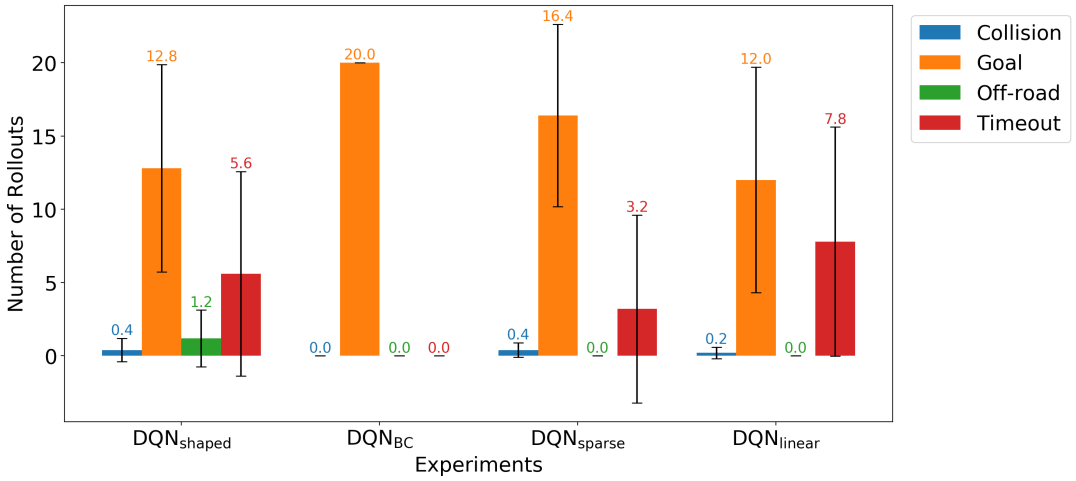


Figure 5.14: Outcomes across 20 test rollouts per random seed, per DQN variant

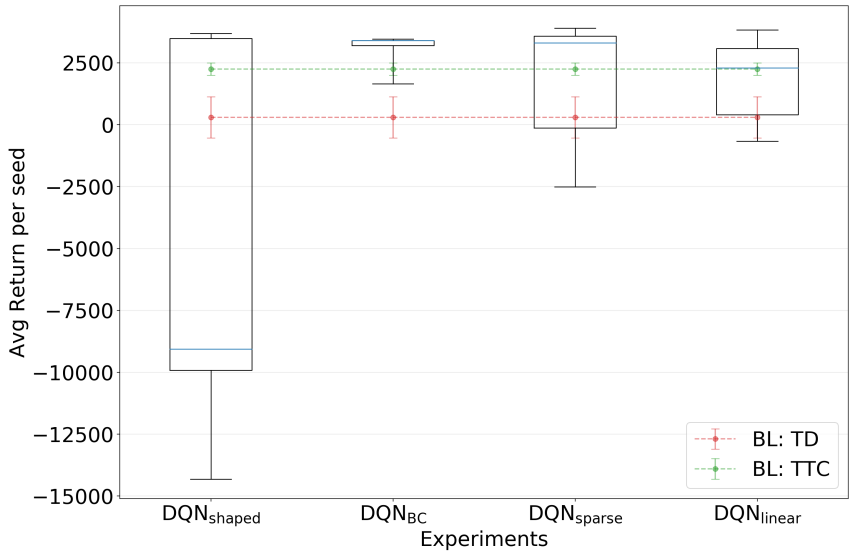


Figure 5.15: Box plot of average returns per random seed per DQN variant. Each box ranges from the lower to the upper quartile with the median indicated in blue. The whiskers denote the range of the data points. Outliers are included in the plot.

remarkable low variance, where even the lower quartile is above the TTC baseline. This strongly indicates a very stable policy and a reliable training across different random seeds. $\text{DQN}_{\text{sparse}}$ and $\text{DQN}_{\text{linear}}$ show a comparable variance but the median of $\text{DQN}_{\text{sparse}}$ is at the level of DQN_{BC}, while the median of $\text{DQN}_{\text{linear}}$ worse than all the other DQN variants.

Considering solely the comfort terms, one can observe that DQN_{BC}, $\text{DQN}_{\text{sparse}}$ and $\text{DQN}_{\text{linear}}$ achieve comparable results of the lower to upper quartile and the median. All of them seem to learn action repetition equally well. As shown in Figure 5.14, $\text{DQN}_{\text{linear}}$ often does not start a lane change at all, which does not cause any lateral acceleration and lateral jerk. Therefore, $\text{DQN}_{\text{linear}}$ achieves

the highest average return considering only the comfort return components. Moreover, the previous findings of the jerkily policy of DQN_{shaped} can be supported. DQN_{shaped} shows a considerably high variance and a bad median, compared to the other DQN variants and the baselines.

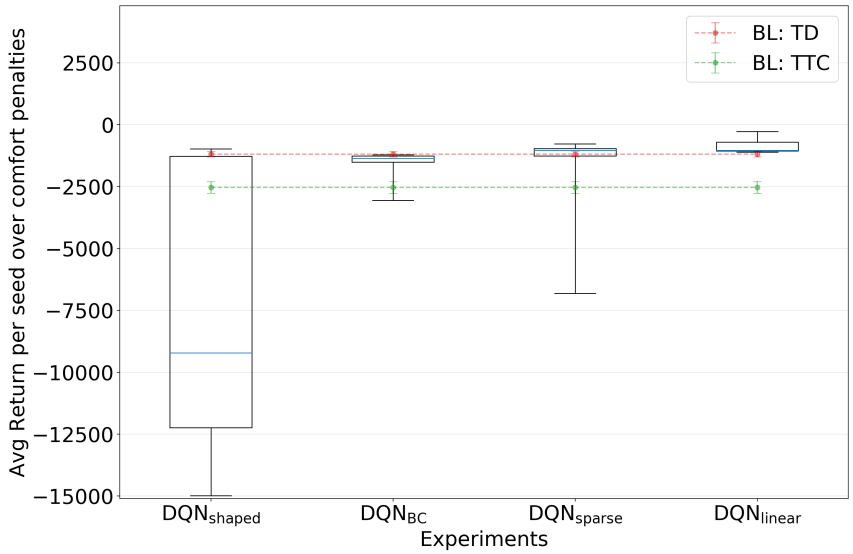


Figure 5.16: Box plot of average returns per random seed solely computed over comfort penalties per DQN variant. Each box ranges from the lower to the upper quartile with the median indicated in blue. The wiskers denote the range of the data points. Outliers are included in the plot.

In Figure 5.17, we highlight the differences in the action sequences for each of the DQN variants and their random seeds. The figure underlines our previous findings. In particular, it shows that action repetition seems to be more difficult to learn for almost all DQN variants in this scenario than the single-speeder scenario. DQN_{linear} exclusively shows consistent action repetition, whereas all the other show short periods of lane changing in between long lane keeping periods. This can be particularly seen in the first and third random seed of DQN_{shaped}. It can also be observed that rollouts containing a high share of action jitter, often result in off-road driving, due to oscillation effects.

Because some DQN variants experience a significant performance drop towards the end of the training, we do not only evaluate the final test rollouts but we additionally investigate the best policies of the DQN algorithms during the training. The results of these additional rollouts using the best policies per random seed are depicted in Figure 5.18. One can observe a significant performance gain in case of the DQN_{shaped}. In the test rollouts, DQN_{shaped} does not produce collisions anymore and reaches the goal invariably. This indicates that the current approach lacks in stability instead of quality. The results of DQN_{sparse} and DQN_{linear} are also slightly improved but still both can not fully prevent collisions. For further figures, regarding the best policies in the multi-speeder scenario, please see the supplementary material.

Now, the trajectories for the best policy in the multi-speeder scenario are examined. In the following, the behavior of the DQN_{BC} policy is examined using two examples. Figure 5.19 illustrates the first example trajectory of the DQN_{BC} policy. It depicts the initial situation in the very left image, where

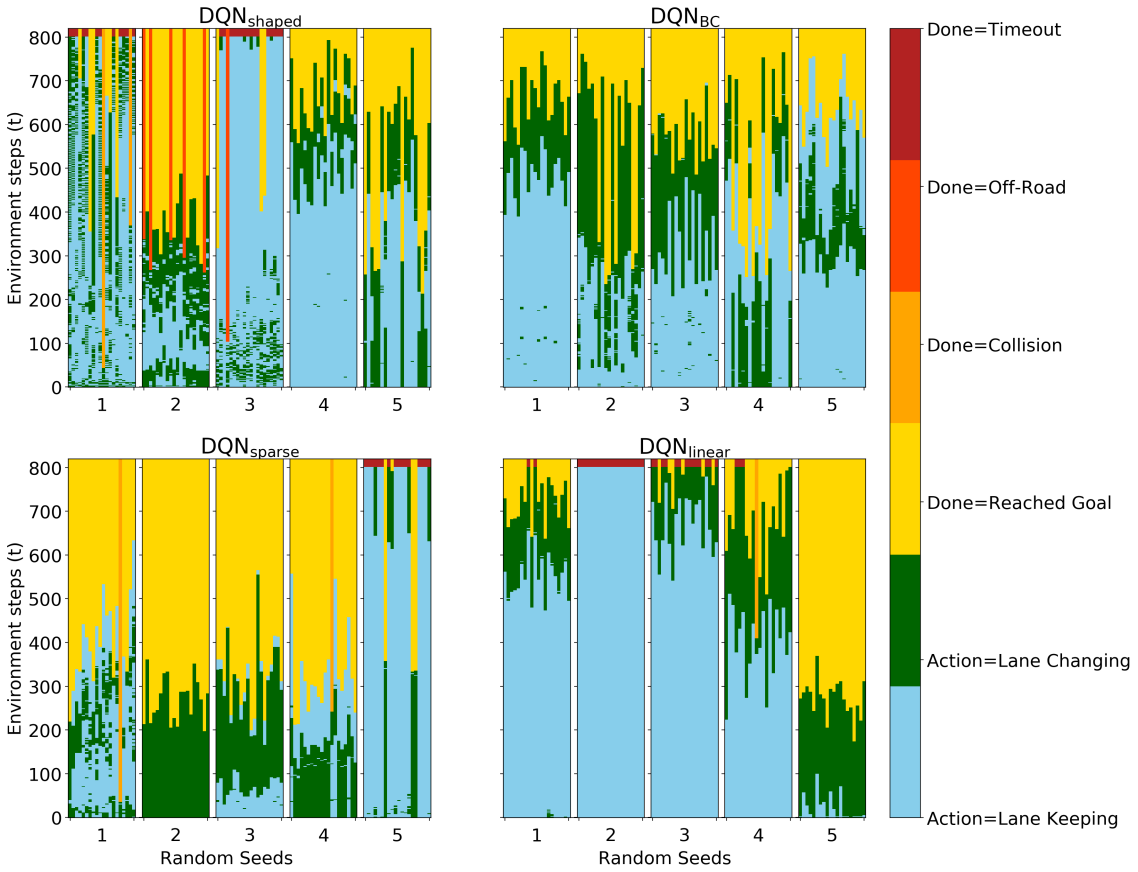


Figure 5.17: Actions given by final policies over 20 test rollouts and 5 random seeds applied to the multi-speeder scenario. The x-axis shows the 5 random seeds with 20 test rollouts each, whereas the y-axis shows the environment steps per rollout. The colors indicate on the one hand the actions selected by the final policy and on the other hand the outcome after the rollout terminates.

the ego vehicle is located very close to the first speeder on the target lane. The policy therefore waits until the first speeder passed. At the point in time the speeder is past the ego vehicle, the ego vehicle has to decelerate yet because it gets too close to the truck. Therefore, it is too slow to safely merge into one of the gaps between the speeders and in turn waits until all of them passed. The ego vehicle can then change the lane and reaches the goal safely. In this situation, this policy shows optimal behavior, as it reaches the goal as safely and as comfortable as possible.

In comparison, we show another situation in Figure 5.20, where the DQN_{BC} policy decides to change the lane from the very beginning. It realizes that the gap and the difference in velocities suggest for an early lane change. Once the ego vehicle passes the lane boundary, the first speeder decelerates and gives way for the ego vehicle. It is remarkable that the agent is able to distinguish between these two situations. The DQN_{BC} algorithm shows the ability to learn the difference in the two presented examples and is capable of learning that it is safe to overtake from the beginning in one but not in the other. For videos of these two example trajectories, please refer to the supplementary material.

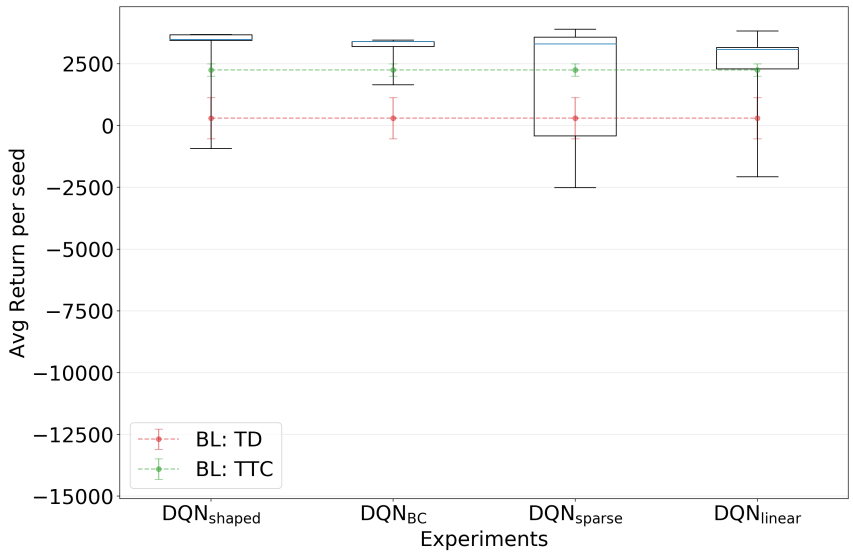


Figure 5.18: Box plot of average returns per random seed using the best policies per DQN variant. Each box ranges from the lower to the upper quartile with the median indicated in blue. The whiskers denote the range of the data points. Outliers are included in the plot.

To summarize the results of the multi-speeder scenario, a good performance of DQN_{BC} is observed, proving to learn collision avoidance even in the difficult multi-speeder scenario. DQN_{shaped} could not achieve comparable results, as in the single-speeder scenario, due to a catastrophic drop in the performance towards the end of the training. However, considering the best DQN_{shaped} policy throughout the training the results indicate comparable behavior to the DQN_{BC} policy in the multi-speeder scenario. DQN_{sparse} performed better, than in the single-speeder scenario but still shows some variance in the behavior and can neither always reach the goal nor avoid collisions. This can also be observed with DQN_{linear}, which learns a more conservative policy, than the others.

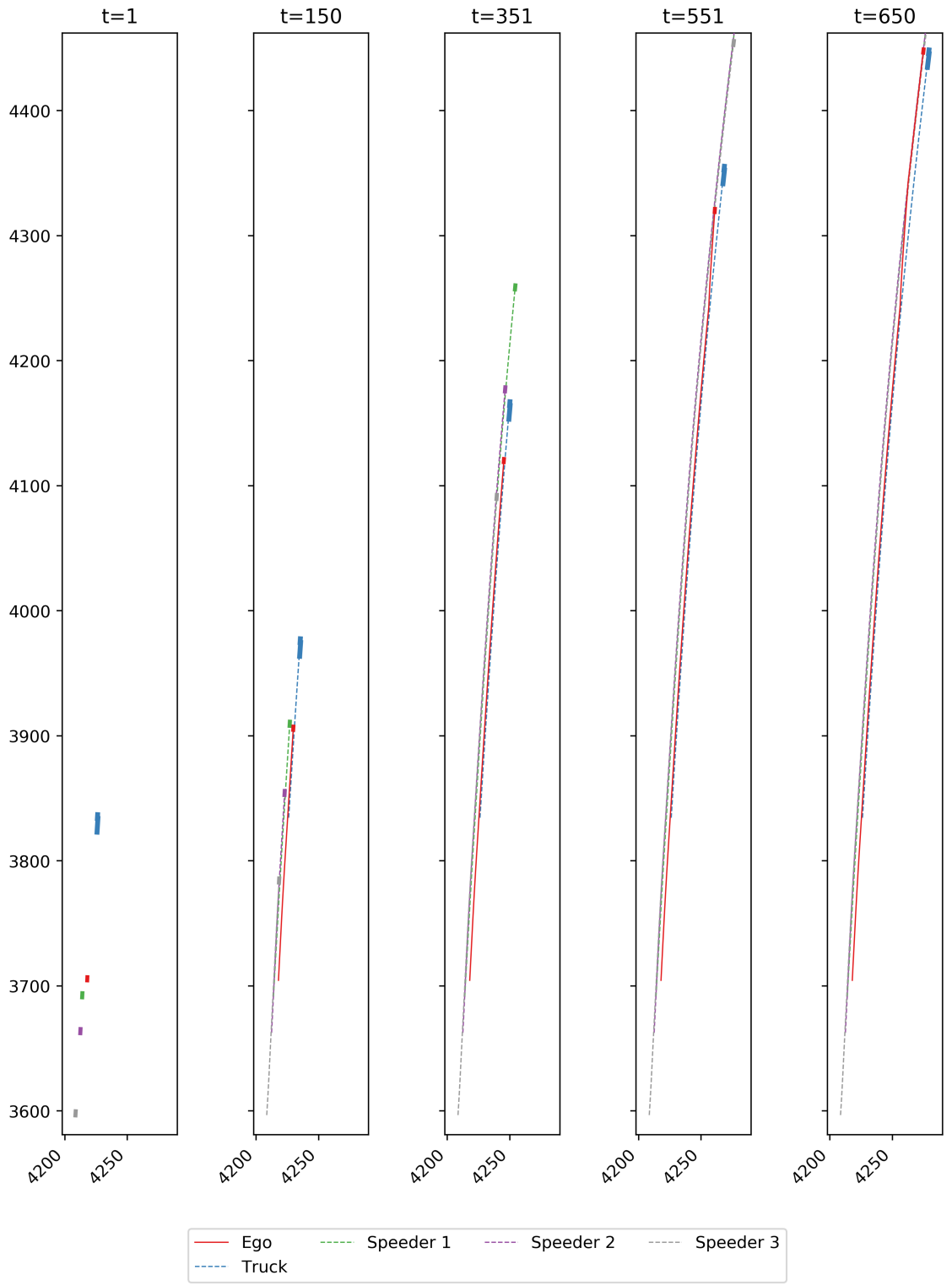


Figure 5.19: Example trajectory of a single rollout depicted for five different timesteps. The ego vehicle waits until all speeders on the target lane have passed and then changes the lane to reach the goal.

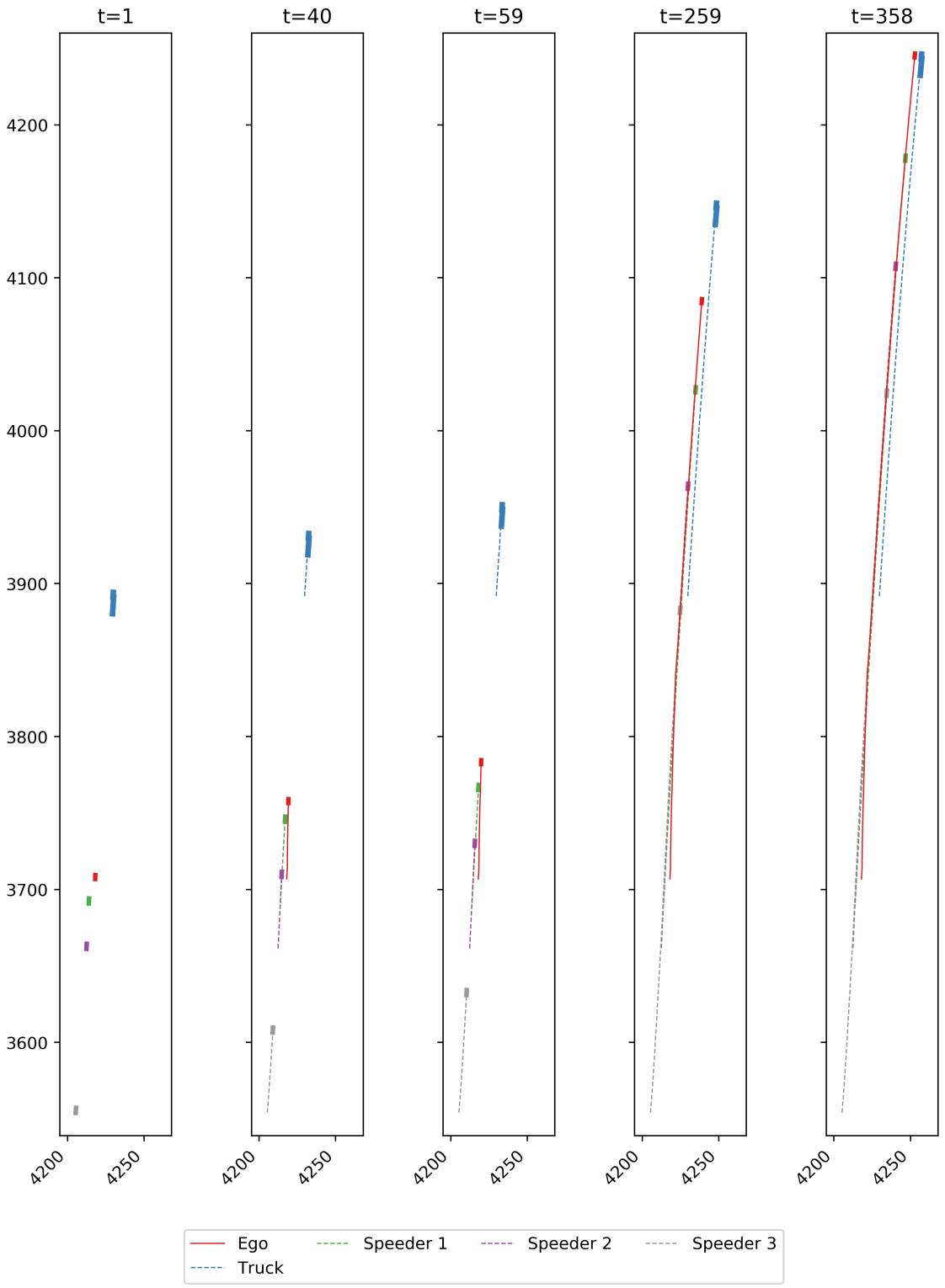


Figure 5.20: Example trajectory of a single rollout depicted for five different timesteps. The ego vehicle directly changes the lane to reach the target lane in front of the first speeder.

6 Discussions

In the following, we discuss our previously presented results and give possible explanations. Moreover, our approach is reviewed and possible alternative approaches to DQN, which was used in this work, are presented. Finally, the insights gained throughout this thesis are stated.

6.1 Results

In this work, it was hypothesized that the DQN algorithm is capable of finding a policy that reaches the goal in more than 90 % of the rollouts. Our results show that DQN_{BC} is capable of finding the goal in more than 90 % of the conducted experiments in both scenarios. Additionally, when using the best policy throughout the training instead of the final policy after the training, $\text{DQN}_{\text{shaped}}$ performed equally optimal in the test rollouts compared to DQN_{BC}.

Furthermore, we hypothesized that DQN avoids collisions in more than 90 % of the rollouts. In both scenarios, our experiments show that DQN_{BC} and $\text{DQN}_{\text{shaped}}$ prevent collisions in more than 90 % of the test rollouts.

Additionally, our hypotheses state that the trained DQN policy is capable of finding a comfortable trajectory for the passengers. We evaluated this using our developed baselines and compared the results. We found that DQN_{BC}, $\text{DQN}_{\text{sparse}}$ and $\text{DQN}_{\text{linear}}$ are capable of finding comparably good trajectories to the TD baseline in terms of the comfort return components. As the TD baseline policy is optimal regarding the comfort, we can confirm the hypothesis. Although, the results of the overall best performing DQN variant already show good behavioral decisions in such complex scenarios, there are still limitations of the approach. So far DQN_{BC} is not able to fully prevent action jitter, which can lead to unnatural experiences for human passengers.

Regarding the hypothesis that reward shaping, compared to sparse rewards, accelerate the training, we found that reward shaping does not necessarily result in an MDP, which is equivalent to the original MDP in terms of the optimal solutions. Hence, it is possible that the DQN is trained into the "wrong direction" as the MDP is possibly misspecified. This risk is increased by the fact that the reward is defined as a sum of several individual components. Correspondingly, the shaping can interact with the other components and their weighting. Furthermore, the reward shaping parameters must also be tuned and might be inappropriate.

Additionally, we empirically showed that a deep non-linear architecture of the Q-network is essential in our scenarios, as the $\text{DQN}_{\text{linear}}$ cannot achieve comparable results, to the DQN variants with a deep architecture. Even simple but successful policies such as the TTC baseline seem to be too complex to be expressed by a linear DQN policy.

All of our developed baselines are outperformed by DQN_{BC} in both scenarios. We assume that the weight initialization with behavioral cloning already incorporate aspects like action repetition into the policy. During the training DQN_{BC}, was able to focus on improving other components of the task, such as avoiding collisions. This accelerated the training during some of our experiments.

6.2 Alternatives

In retrospect, there are a couple of other approaches we could have followed instead of DQN. Policy gradient approaches could have been used as Shalev-Shwartz et al. [SSS16] indicate that those approaches are beneficial for multi agent environments, as they do not rely on a markovian setting. Additionally, evolutional strategies, show comparable performance to state-of-the-art RL algorithms in an early publication by Salimans et al. [SHCS17]. The authors claim benefits in parallelizibilty, data efficiency and robustness. Moreover, the DQN architecture could have been extended with MCTS, as in [PRHK17]. Instead of using DQN online, a high-level policy is trained offline, while online planning with MCTS is used to refine the high-level policy afterwards. In general, MCTS approaches seem to be a good choice as AlphaGo [SHM+16] and AlphaGo Zero [SSS+17] have shown remarkable success in generalizing in a huge state space.

Apart from the RL approach, there are a few shortcomings, which should be changed in the future. Currently, the developed lateral low-level controller uses a linear reference trajectory, resulting in very harsh steering motions in the beginning and in the end of the lane change. This results in high lateral jerk and increases with rising velocity of the ego vehicle. Therefore, an S-shaped reference trajectory should be used in the future to enable the algorithm to find smooth and comfortable trajectories for the passengers. In addition, the trajectory of a lane change controlled by the lateral controller is defined by a duration in seconds. In this work, the duration has been hard-coded to 4 seconds so that each lane change is performed consistently. Controlling this duration using RL would give the algorithm the opportunity to adapt the lane change trajectory according to the situation. In case, the target lane is free the lane change can be performed over a long period of time and therefore comfortable for the passengers. If there is a safety critical situation where the lane change is very urgent it could be performed very fast.

Since intentions of other traffic participants are non-observable and sensorial inaccuracies are present in real-world applications, one could consider using a partially observable markov decision process (POMDP) in the future. One could define a state s_t as the last k observations $(o_{t-k+1}, o_{t-k+2}, \ldots, o_t)$, in order to include temporal information into the state. In a subsequent step, convolution could be applied to the state to aggregate the information per vehicle. Another possible approach is to use a deep recurrent Q-network (DRQN) [HS15], which adds recurrence to the DQN by incorporating LSTM.

As we experienced difficulties to tune the hyperparameters to multiple scenarios, one could consider replacing reward shaping with other techniques to overcome the challenges of sparse and delayed rewards. One possibility could be hindsight experience replay [AWR+17], as it naturally integrates with DQN and shows good results in robotics tasks. Furthermore, reverse curriculum learning for RL [FHW+17] could be another approach to deal with sparse rewards.

6.3 Insights

Initially, we tried to use an end-to-end approach learning both the high-level decisions and the low-level controls using the option-critic architecture [BHP17]. This turned out to be very challenging as the algorithm does not only have to learn when to perform lane changing and lane keeping. Instead, the motions of lane changing and lane keeping need to be learned in addition to the behavior. The only information provided is the number of high-level actions, which is two in our case. Our experiments supported the literature [HBKP18] showing that end-to-end algorithms converge to either one option being used all the time or switching the option in each timestep. Both outcomes are generally sub-optimal as they do not exploit the potential of the hierarchical separation of the decision-making.

Therefore, we decided to exploit more domain-knowledge and hand-engineered controllers mentioned in Section 4.7, which determine the low-level controls. This resulted in less control of the RL algorithm, which is one aspect we want to change in the future. While keeping the DQN algorithm, we plan to allow the algorithm to change the lane to the right. Consequently, the algorithm can learn to abort lane changes and return to its previous lane once a safety-critical situation occurs. A separate module enforcing the traffic rules, e.g. the lane used for overtaking must be to the left of the vehicle to be overtaken, would be needed to guide the algorithm in learning a policy, which is consistent with the local traffic rules. Of course, this depends on additional factors like e.g., the traffic system and rules of the country the vehicle will be driving in.

We observed that it is very important to fine-tune the reward function as well as all the hyperparameters of the DQN. Especially, the balance between the individual reward components as well as the weighting of the shaped rewards took a lot of effort to accurately determine the parameters. In preliminary experiments, it often occurred that the RL agent did not learn what we intended it to learn, due to an ambiguous reward function. In general, reward hacking, as addressed in an early draft by Amodei et al. [AOS+16], is one of the major challenges for safe and reliable RL agents. There is current research focusing on how to reliably engineer a reward function that prevents reward hacking by the RL algorithm [Dew14; MPD+18]. An early draft of Jaderberg et al. [JCD+18] suggests that the weights of the shaped rewards can be learned from experience jointly with the policy. This is another potential direction for improvement, as it could help in extending the approach to a wider range of scenarios and usecases.

7 Conclusion

In this work we developed a DQN algorithm and applied it to an autonomous driving use case. We considered two highway driving scenarios with differing difficulty and complexity. The initially stated hypotheses were empirically evaluated by applying multiple variants of the DQN algorithm to the scenarios. The traditional DQN approach was improved by common extensions such as duel architecture, double update and prioritized experience replay. We found that the DQN is able to reach a goal state without colliding into other vehicles. Furthermore, the DQN algorithm learned a policy that results in comfortable trajectories with minimum acceleration and jerk for the passengers inside the vehicle. Our results state that a deep non-linear architecture and an initialization with behavioral cloning is particularly beneficial. The DQN variants were compared to three developed baselines, which range from random to heuristic based behavior. We showed that our approach outperforms the baselines in terms of the total return as well as the comfort return. Further evaluations with other algorithms and methods are left for future work. Possible candidates for a comparison are policy gradient approaches, tree-based planning techniques and evolutionary methods.

Bibliography

[AOS+16] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané. "Concrete Problems in AI Safety". In: *arXiv preprint arXiv:1606.06565* (2016). URL: https://arxiv.org/pdf/1606.06565.pdf (cit. on p. 69).

[AWR+17] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, W. Zaremba. "Hindsight Experience Replay". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett. Curran Associates, Inc., 2017, pp. 5048–5058. URL: http://papers.nips.cc/paper/7090-hindsight-experience-replay.pdf (cit. on pp. 22, 68).

[BHP17] P. Bacon, J. Harb, D. Precup. "The Option-Critic Architecture". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. Vol. abs/1609.05140. 2017 (cit. on pp. 45, 69).

[BMWD16] H. Bi, T. Mao, Z. Wang, Z. Deng. "A Data-driven Model for Lane-changing in Traffic Simulation". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. SCA '16. Zurich, Switzerland: Eurographics Association, 2016, pp. 149–158. ISBN: 978-3-905674-61-3. URL: http://dl.acm.org/citation.cfm?id=2982818.2982839 (cit. on p. 19).

[BPW+12] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton. "A Survey of Monte Carlo Tree Search Methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (Mar. 2012), pp. 1–43. ISSN: 1943-068X. DOI: 10.1109/TCIAIG.2012.2186810 (cit. on p. 20).

[BTW00] J. Baxter, A. Tridgell, L. Weaver. "Learning to Play Chess Using Temporal Differences". In: *Machine Learning* 40.3 (Sept. 2000), pp. 243–263. ISSN: 1573-0565. DOI: 10.1023/A:1007634325138. URL: https://doi.org/10.1023/A:1007634325138 (cit. on p. 17).

[CSW+18] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, M. Bacchiani. "State-of-the-Art Speech Recognition with Sequence-to-Sequence Models". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Apr. 2018, pp. 4774–4778. DOI: 10.1109/ICASSP.2018.8462105 (cit. on p. 23).

[Des17a] S. B. (Destatis). *Unfallentwicklung auf deutschen Straßen 2017. Begleitmaterial zur Pressekonferenz*. Tech. rep. Statistisches Bundesamt (Destatis), 2017. URL: https://www.destatis.de/DE/Publikationen/Thematisch/TransportVerkehr/Verkehrsunfaelle/PK_Unfallentwicklung.html (visited on 10/09/2018) (cit. on p. 16).

[Des17b] S. B. (Destatis). *Verkehrsunfälle - Zeitreihen*. 2017. URL: https://www.destatis.de/DE/Publikationen/Thematisch/TransportVerkehr/Verkehrsunfaelle/VerkehrsunfaelleZeitreihen.html (visited on 10/31/2018) (cit. on pp. 15, 16).

[Dew14] D. Dewey. "Reinforcement learning and the reward engineering principle". In: *2014 AAAI Spring Symposium Series*. 2014 (cit. on p. 69).

[DFR15] M. P. Deisenroth, D. Fox, C. E. Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.2 (Feb. 2015), pp. 408–423. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.218 (cit. on pp. 22, 44).

[DHS11] J. Duchi, E. Hazan, Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization". In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159 (cit. on p. 26).

[FHW+17] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, P. Abbeel. "Reverse Curriculum Generation for Reinforcement Learning". In: *Proceedings of the 1st Annual Conference on Robot Learning*. Ed. by S. Levine, V. Vanhoucke, K. Goldberg. Vol. 78. Proceedings of Machine Learning Research. PMLR, 13–15 Nov 2017, pp. 482–495. URL: http://proceedings.mlr.press/v78/florensa17a.html (cit. on pp. 22, 68).

[FZB+07] K. Fitzpatrick, K. Zimmerman, R. Bligh, S. Chrysler, B. Blaschke. *Criteria for High Design Speed Facilities*. Tech. rep. Texas Transportation Institute, The Texas A&M University System, 2007. URL: https://trid.trb.org/view/806966 (visited on 10/20/2018) (cit. on p. 17).

[GB10] X. Glorot, Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Y. W. Teh, M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html (cit. on p. 17).

[GW06] J. S. Gyorfi, C. H. Wu. "A Minimum-Jerk Speed-Planning Algorithm for Coordinated Planning and Control of Automated Assembly Manufacturing". In: *IEEE Transactions on Automation Science and Engineering* 3.4 (Oct. 2006), pp. 454–462. ISSN: 1545-5955. DOI: 10.1109/TASE.2005.860987 (cit. on p. 43).

[HBKP18] J. Harb, P. Bacon, M. Klissarov, D. Precup. "When Waiting is not an Option: Learning Options with a Deliberation Cost". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*. 2018 (cit. on p. 69).

[HCR09] M. L. Ho, P. T. Chan, A. B. Rad. "Lane change algorithm for autonomous vehicles via virtual curvature method". In: *Journal of Advanced Transportation* 43.1 (2009), pp. 47–70. DOI: 10.1002/atr.5670430104. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/atr.5670430104. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/atr.5670430104 (cit. on pp. 19, 45).

[HDY+12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, B. Kingsbury. "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups". In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), pp. 82–97. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2205597 (cit. on p. 23).

[HGS16] H. v. Hasselt, A. Guez, D. Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI'16. Phoenix, Arizona: AAAI Press, 2016, pp. 2094–2100. URL: http://dl.acm.org/citation.cfm?id=3016100.3016191 (cit. on pp. 21, 33).

[HMH+17] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, D. Silver. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *The Thirty-Second AAAI Conference on Artificial Intelligence*. Vol. abs/1710.02298. 2017 (cit. on p. 21).

[Hoc91] S. Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: *Diploma, Technische Universität München* 91.1 (1991) (cit. on p. 25).

[HS15] M. Hausknecht, P. Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". In: *2015 AAAI Fall Symposium Series*. 2015 (cit. on p. 68).

[HSS12] G. Hinton, N. Srivastava, K. Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent". 2012 (cit. on p. 26).

[HWL18] C. Hoel, K. Wolff, L. Laine. "Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning". In: *2018 21st IEEE International Conference on Intelligent Transportation Systems*. 2018 (cit. on pp. 19, 21).

[HZRS15] K. He, X. Zhang, S. Ren, J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. ICCV '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 1026–1034. ISBN: 978-1-4673-8391-2. DOI: 10.1109/ICCV.2015.123. URL: http://dx.doi.org/10.1109/ICCV.2015.123 (cit. on pp. 17, 48).

[HZRS16] K. He, X. Zhang, S. Ren, J. Sun. "Deep Residual Learning for Image Recognition". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2016 (cit. on p. 23).

[JCD+18] M. Jaderberg, W. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, T. Graepel. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning". In: *CoRR* abs/1807.01281 (2018) (cit. on pp. 44, 69).

[KB12] A. S. Kilinç, T. Baybura. "Determination of Minimum Horizontal Curve Radius Used in the Design of Transportation Structures, Depending on the Limit Value of Comfort Criterion Lateral Jerk". In: *In Proceedings of the Thirty-Fifth FIG Working Week*. 2012 (cit. on p. 16).

[KB15] D. P. Kingma, J. Ba. "Adam: A method for stochastic optimization". In: *Third International Conference on Learning Representations*. 2015 (cit. on p. 26).

[KSH12] A. Krizhevsky, I. Sutskever, G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cit. on p. 23).

[LA09] H. Lin, P. J. Antsaklis. “Stability and Stabilizability of Switched Linear Systems: A Survey of Recent Results”. In: *IEEE Transactions on Automatic Control* 54.2 (Feb. 2009), pp. 308–322. ISSN: 0018-9286. DOI: 10.1109/TAC.2008.2012009 (cit. on p. 46).

[LKK16] D. Lenz, T. Kessler, A. Knoll. “Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search”. In: *2016 IEEE Intelligent Vehicles Symposium (IV)*. June 2016, pp. 447–453. DOI: 10.1109/IVS.2016.7535424 (cit. on pp. 19–21).

[MBB+] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, S. Thrun. “Junior: The Stanford entry in the Urban Challenge”. In: *Journal of Field Robotics* 25.9 (), pp. 569–597. DOI: 10.1002/rob.20258. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20258. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20258 (cit. on p. 19).

[MGR18] H. Mania, A. Guy, B. Recht. “Simple random search provides a competitive approach to reinforcement learning”. In: *NIPS Workshop on Critiquing and Correcting Trends in Machine Learning* (2018) (cit. on pp. 18, 47).

[MHN13] A. L. Maas, A. Y. Hannun, A. Y. Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013 (cit. on p. 25).

[MKS+13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller. “Playing Atari With Deep Reinforcement Learning”. In: *NIPS Deep Learning Workshop*. 2013 (cit. on pp. 20, 21, 32, 48).

[MPD+18] P. Mallozzi, R. Pardo, V. Duplessis, P. Pelliccione, G. Schneider. “MoVEMo: A Structured Approach for Engineering Reward Functions”. In: *2018 Second IEEE International Conference on Robotic Computing (IRC)*. Jan. 2018, pp. 250–257. DOI: 10.1109/IRC.2018.00053 (cit. on p. 69).

[MPW+18] B. Mirchevska, C. Pek, M. Werling, M. Althoff, J. Boedecker. “High-Level Decision Making for Safe and Reasonable Autonomous Lane Changing Using Reinforcement Learning”. In: *2018 21st IEEE International Conference on Intelligent Transportation Systems*. Vol. abs/1803.10056. 2018 (cit. on p. 21).

[NH10] V. Nair, G. E. Hinton. “Rectified Linear Units Improve Restricted Boltzmann Machines”. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7. URL: http://dl.acm.org/citation.cfm?id=3104322.3104425 (cit. on p. 25).

[NHR99] A. Y. Ng, D. Harada, S. J. Russell. “Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping”. In: *Proceedings of the Sixteenth International Conference on Machine Learning*. ICML ’99. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, pp. 278–287. ISBN: 1-55860-612-2. URL: http://dl.acm.org/citation.cfm?id=645528.657613 (cit. on p. 22).

[NN17] H. Ngo, V. Ngo. *Reinforcement Learning Course. Temporal Difference Learning*. Summer Semester 2017. URL: https://ipvs.informatik.uni-stuttgart.de/mlr/wp-content/uploads/2016/06/03-TemporalDiffHung.pdf (visited on 11/02/2018) (cit. on p. 27).

[PRHK17] C. Paxton, V. Raman, G. D. Hager, M. Kobilarov. "Combining neural networks and tree search for task and motion planning in challenging environments". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6059–6066. DOI: 10.1109/IROS.2017.8206505 (cit. on pp. 21, 68).

[Rud16] S. Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016) (cit. on p. 26).

[SB98] R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. URL: http://www.cs.ualberta.ca/~sutton/book/the-book.html (cit. on pp. 26, 28, 31).

[SHCS17] T. Salimans, J. Ho, X. Chen, I. Sutskever. "Evolution Strategies as a Scalable Alternative to Reinforcement Learning". In: *CoRR* abs/1703.03864 (2017) (cit. on p. 68).

[SHM+16] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (Jan. 2016). Article, 484 EP -. URL: http://dx.doi.org/10.1038/nature16961 (cit. on pp. 17, 68).

[SQAS16] T. Schaul, J. Quan, I. Antonoglou, D. Silver. "Prioritized Experience Replay". In: *Proceedings of the 4th International Conference on Learning Representations*. 2016 (cit. on pp. 21, 35).

[SSS+17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis. "Mastering the game of Go without human knowledge". In: *Nature* 550 (Oct. 2017). Article, 354 EP -. URL: http://dx.doi.org/10.1038/nature24270 (cit. on pp. 17, 68).

[SSS16] S. Shalev-Shwartz, S. Shammah, A. Shashua. "Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving". In: *NIPS Workshop on Learning, Inference and Control of Multi-Agent Systems* (2016) (cit. on pp. 19, 21, 68).

[THH00] M. Treiber, A. Hennecke, D. Helbing. "Congested traffic states in empirical observations and microscopic simulations". In: *Phys. Rev. E* 62 (2 Aug. 2000), pp. 1805–1824. DOI: 10.1103/PhysRevE.62.1805. URL: https://link.aps.org/doi/10.1103/PhysRevE.62.1805 (cit. on pp. 20, 45).

[TMD+06] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, P. Mahoney. "Stanley: The Robot That Won the DARPA Grand

Challenge: Research Articles". In: *J. Robot. Syst.* 23.9 (Sept. 2006), pp. 661–692. ISSN: 0741-2223. DOI: 10.1002/rob.v23:9. URL: http://dx.doi.org/10.1002/rob.v23:9 (cit. on p. 19).

[TN18] M. Toussaint, D. NguyenTuong. *Machine Learning Course. Neural Networks & Deep Learning*. Summer Semester 2018. URL: https://ipvs.informatik.uni-stuttgart.de/mlr/wp-content/uploads/2018/06/ML_NeuralNetworks_SS18.pdf (visited on 10/04/2018) (cit. on p. 23).

[TS93] S. Thrun, A. Schwartz. "Issues in Using Function Approximation for Reinforcement Learning". In: *Proceedings of the 1993 Connectionist Models Summer School*. Ed. by M. Mozer, P. Smolensky, D. Touretzky, J. Elman, A. Weigend. Erlbaum Associates, Jan. 1993 (cit. on p. 33).

[UAB+08] C. Urmson, J. Anhalt, H. Bae, J. A. (Bagnell, C. R. Baker, R. E. Bittner, T. Brown, M. N. Clark, M. Darms, D. Demitrish, J. M. Dolan, D. Duggins, D. Ferguson, T. Galatali, C. M. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, S. Kolski, M. Likhachev, B. Litkouhi, A. Kelly, M. McNaughton, N. Miller, J. Nickolaou, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, V. Sadekar, B. Salesky, Y.-W. Seo, S. Singh, J. M. Snider, J. C. Struble, A. (Stentz, M. Taylor, W. (L. Whittaker, Z. Wolkowicki, W. Zhang, J. Ziglar. "Autonomous driving in urban environments: Boss and the Urban Challenge". In: *Journal of Field Robotics Special Issue on the 2007 DARPA Urban Challenge, Part I* 25.8 (June 2008), pp. 425–466 (cit. on p. 19).

[UM13] S. Ulbrich, M. Maurer. "Probabilistic online POMDP decision making for lane changes in fully automated driving". In: *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*. Oct. 2013, pp. 2063–2067. DOI: 10.1109/ITSC.2013.6728533 (cit. on p. 19).

[VECB17] C. Vallon, Z. Ercan, A. Carvalho, F. Borrelli. "A machine learning approach for personalized autonomous lane change initiation and control". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. June 2017, pp. 1590–1595. DOI: 10.1109/IVS.2017.7995936 (cit. on p. 20).

[WCL18] P. Wang, C. Chan, A. de La Fortelle. "A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers". In: *2018 29th IEEE Intelligent Vehicles Symposium*. Vol. abs/1804.07871. 2018 (cit. on p. 21).

[WD92] C. J. C. H. Watkins, P. Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698 (cit. on p. 31).

[WSH+16] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, N. De Freitas. "Dueling Network Architectures for Deep Reinforcement Learning". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 1995–2003. URL: http://dl.acm.org/citation.cfm?id=3045390.3045601 (cit. on pp. 21, 33–35).

[XYSL15] J. Xu, K. Yang, Y. Shao, G. Lu. "An Experimental Study on Lateral Acceleration of Cars in Different Environments in Sichuan, Southwest China". In: *Discrete Dynamics in Nature and Society* 2015 (2015), p. 16. URL: 10.1155/2015/494130 (cit. on p. 16).

[ZL17] B. Zoph, Q. V. Le. "Neural Architecture Search with Reinforcement Learning". In: *5th International Conference on Learning Representations*. 2017 (cit. on p. 23).

All links were last followed on November 5, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature