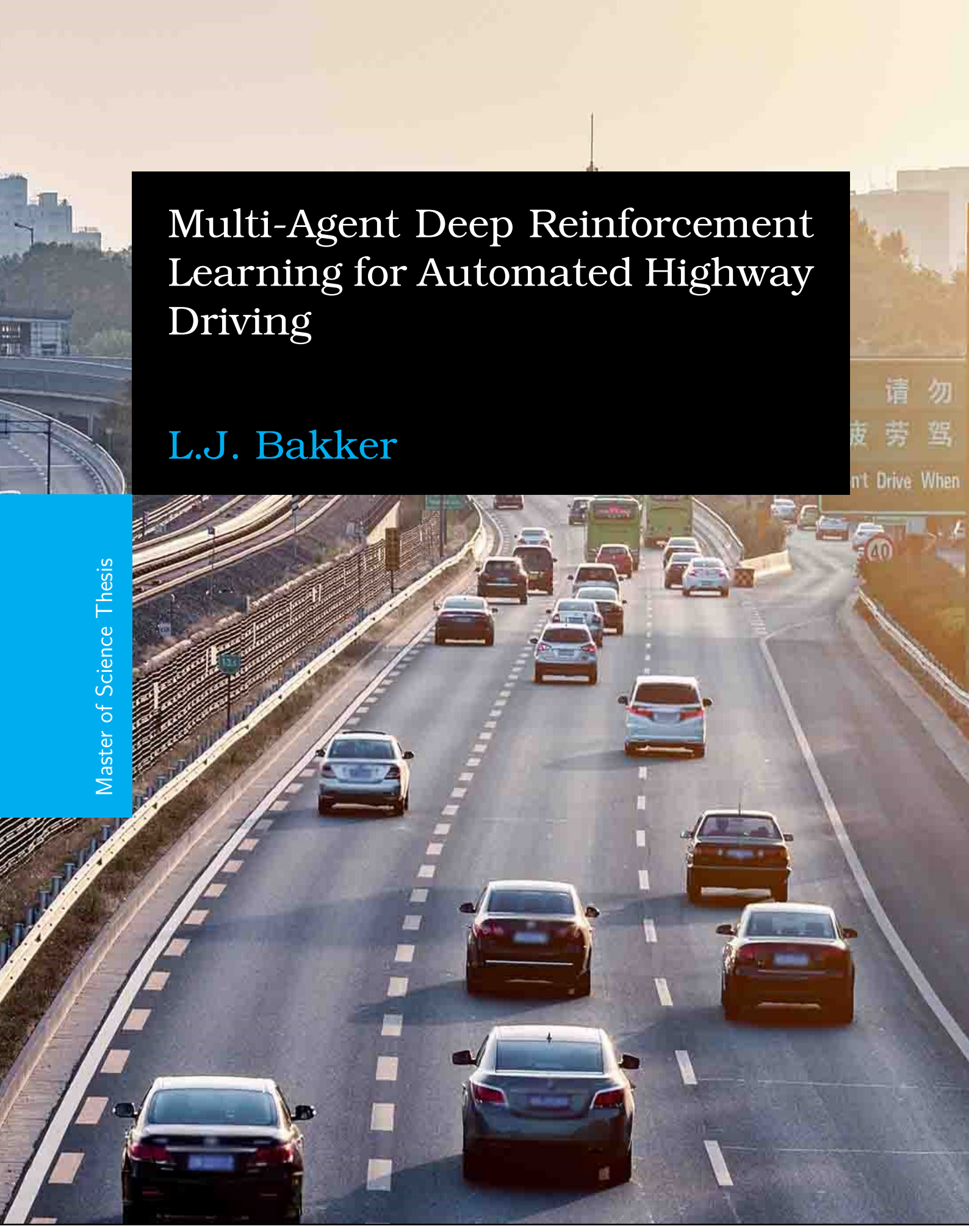
Multi-Agent Deep Reinforcement Learning for Automated Highway Driving

L.J. Bakker

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

Multi-Agent Deep Reinforcement Learning for Automated Highway Driving

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

L.J. Bakker

June 21, 2019

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

Recent advances in Deep Reinforcement Learning have sparked new interest in many different research topics, including Automated Highway Driving where agents model autonomous vehicles. The main advantage of Deep Reinforcement Learning is that the training algorithm is adaptable to its environment. In highway driving, researchers often simplify the framework of an agent by using lower level controllers and observers. However, agent observations do not yet include lane change intentions of surrounding vehicles. Resulting agents were able to drive on a maximum of three lanes and unfit to drive in lane changing environments. We aim to simplify the current state-of-the-art agent frameworks even further to improve performance. We also believe that observing other vehicles lane-change intent, or blinker status, is essential for collision avoidance in a highway driving environment. In this paper, we try to implement multi-agent Deep Reinforcement Learning on a six-lane highway, including lane changes. After training, agents are able to avoid collisions while reaching destination lanes. Moreover, a lane-selection strategy according to desired speed evolved from open freeway training.

Table of Contents

List of Figures

List of Tables

Preface

This manuscript is my Master of Science graduation thesis for the Delft Center for Systems and Control (DCSC) at the Delft University of Technology (TU Delft). Even though automated driving is an appealing subject to me, I arrived at this subject by coincidence. Now, it is my believe that within the near future vehicles will be fully automated on highways without human surveillance. Artificial Intelligence (AI) has also been a somehow forgone interest of mine (educationally speaking) and, to no surprise, AI techniques are often used in automated driving. I am happy to combine these two intriguing subjects and apply them in a, in my opinion, creative and useful way.

I want to thank my supervisor Sergio Grammatico, for guiding me throughout this project and for giving me the freedom to explore the topics most interesting to me.

Delft, University of Technology L.J. Bakker
June 21, 2019

Chapter 1

Introduction

While people have concerns about fully autonomous Self Driving Vehicles (faSDVs) [1], there is a high potential in safety and efficiency improvements [2]. More and more companies are trying to fully automate highway driving, including: Google, Tesla, Uber and TomTom.

While deterministic control is needed at a low-level to ensure safety, there is no deterministic algorithm providing optimal efficiency of a high-level driving strategy on highways. One of the main difficulties in deriving an efficient high-level decision making policy is that it is impossible to simulate the highway environment exactly as it is. Since Reinforcement Learning (RL) agents are adaptable to their environment, RL could be a useful tool for providing a policy of more efficient high-level driving decisions in the, unknown, real life, highway environment.

Recent improvements in Deep Reinforcement Learning (DRL) have proven a high potential in game-like settings. Achieving human-level control on Atari games [3] and super-human control in chess and game of Go [4]. It is even extended to multi-agent systems; Multi-Agent Deep Reinforcement Learning (MADRL) algorithms [5], [6], [7].

Even though DRL is a relatively new research topic, some researchers have already tried to implement DRL in Autonomous Driving (AD). The highway driving problem could be modeled as a game with four major goals, in order of importance:

1. Drive safe
2. Reach the destination
3. Drive fast
4. Drive comfortable

A reward function could describe these goals and DRL could be applied to form a policy for the control actions of an agent, where in this case the agent represents an autonomous vehicle. The ultimate goal is then to find a highway driving policy that performs optimal on the above

given goals, or at least better than human drivers. In this work, we train in a multi-agent setting, but the policy we will derive is still a single-agent policy. Our long term goal is to find a training algorithm or policy for individual vehicles that is theoretically implementable and would perform well in real life highway situations.

End-to-end DRL algorithms for AD [8] have a high potential. The input is raw sensor information, such as direct camera input. The action space is the steering angle and throttle-break actuation. Those type of algorithms are however, currently unfit for an environment with other vehicles because of problem complexity. Further improvements on Deep Reinforcement Learning are needed.

More successful attempts have been made in [9], a truck on a three lane highway, and [7], a multi-agent system of multiple agents on a highway. The actions and observations of the agents are simplified by using lower-level controllers and observers. Both publications are trained and simulated on a three lane highway. The system in [7] did not include lane changes of non-ego-vehicles, [9] did. In [7], the crash rate was significantly higher.

We believe that improvements can be made on the simplification of the highway problem. In this manuscript, we present a novel observation grid that will make the lane observations of surrounding vehicles obsolete. Moreover, we split lateral control (lane change maneuvers) and longitudinal control (Adaptive Cruise Control (ACC) controller) to simplify the agents decisions even further.

The main improvement of our agents is that they observe non-ego lane change intent. At the moment other agents decide to change lanes, this is observable to the surrounding agents. In combination with the observation grid, all collisions will be predictable and therefore preventable. We aim to achieve a zero collision rate after training.

We will present three classes of agents:

- Passive agent: Not able to change lanes.
- Basic agent: Only able to make lane change decision.
- Advanced agent: Able to change lanes and adjust its ACC controller.

The basic and advanced agent are trained in two multi-agent scenarios:

- Exit scenario: All agents want to reach their destination lane.
- Open freeway scenario: All agents drive on an open freeway, optimizing their speed while aiming to drive in accordance with European driving guidelines; drive on the right side and do not overtake on the right.

Similar to Radulescu in [7], we employ centralized training with decentralized execution. The multi-agent system is a semi-homogeneous multi-agent system. All agents share the same deep Q-Network (DQN). The policies could be slightly different, but the DQN determines most of the policy. First we train in a single agent environment where non-ego vehicles are passive agents. After training in this environment, we switch to multi-agent training. The

multi-agent initial policy is the train result of the single-agent setting. Initializing the multi-agent policy with single agent training speeds up the training significantly and improves the preformance [7].

In multi-agent training, all agents are of the same class (basic or advanced), and share the same DQN. We split the agents into model agents and one training. The training agent is used to update the DQN according to conventional DRL [3] and the model agents are used to simulate the environment. This is different to Radulescu's algorithm where all agents are training agents.

Since agents observes each-others lane change decisions, it is an implicitly communicating system where sequencing is important. By assigning the agent that decides its decisions last to be the training agent, we ensure that the training agent has the most information possible on the environment. We train agents to be able to react to other vehicles lane change decisions. We chose to employ an individual (selfish) reward since it is easier to implement, the design and investigation of a shared (cooperative) reward function is left for further research.

Training resulted in four different DQN agents, a basic and advanced agent in both the exit and open freeway scenario. In the exit scenario, agents are tested on their success rate of reaching their destination lane for multiple highway lengths. In the open freeway scenario, agents lane distributions are plotted on their corresponding maximum speed (intended speed of the ACC controller) and their average speed is compared.

We aim to to form, to the authors knowledge, the first non-colliding highway system including lane-changes optimized with RL techniques. Moreover, we want to expand the highway size of three lanes from previous literature to six lanes.

First, in chapter 2, we will present the training algorithm we employed, in both the single- and multi-agent setting. Then, in chapter 3, the framework of the agents are presented. I.e. the way the agent observes the highway, what actions it can choose from and the reward it receives after executing the chosen action. In chapter 4, we present the highway simulator and training algorithm. In chapter 5, we present some experimental results. And finally, in chapter 6, we state some conclusions and further research possibilities.

Chapter 2

Multi-Agent Deep Reinforcement Learning

In this chapter, we present the algorithm that is used to train an autonomous driving vehicle (agent) to drive in a single and multi-agent setting. For the single-agent setting, we use a basic Deep Reinforcement Learning (DRL) algorithm similar to [3]. For the multi-agent setting, we use the same single-agent algorithm to train a shared policy by employing *centralized training with decentralized execution*, much like Radulescu in [7]. The difference is that our algorithm should be suited to a system where agents observe each other's lane changing decisions; a communicating system where sequencing is important. In the first section, we present the single-agent learning algorithm. In the second section, we present the algorithm given by Radulescu and our modification to it.

2-1 Deep Reinforcement Learning Algorithm

In this section, we present the algorithm used for the initial single-agent training. First, we formalize the single-agent highway driving problem as a Partially Observable Markov Decision Process (POMDP). Then, we present the algorithm that is used to train in the single-agent case, by first giving an introduction to Q-Learning, and then explaining the deep Q-Network (DQN) learning algorithm [3], which is the algorithm we used.

2-1-1 Problem formalization

The Highway simulator (chapter 4) transforms the continuous highway driving problem into a discrete time planning problem. The agent chooses and executes a high-level discrete control action every two seconds. A commonly used formalization of discrete time planning tasks of a single-agent system is a Markov Decision Process (MDP). We can include partial observability of the state of the agent by reformulating the formalization as a POMDP.

In the single-agent case, where the non-ego vehicles are 'passive' agents (subsection 3-2-2), the problem can be formalized as a POMDP.

A POMDP is a tuple of: $\mathbb{M} = \{H, A, T, R, S, Z, \gamma\}$

- H is a set of states h of the environment (Highway)
- A is a finite set of actions a of the agent (Autonomous Vehicle actions)
- T is transition probability function to the next state $\mathbb{P}(h'|h, a)$ (The model)
- R is the immediate reward function $H \times A \to \mathbb{R}$ (Feedback to the agent)
- S is the set of (observed) states s of the agent
- Z is the observation probability function dependent on the highway state and previous action $\mathbb{P}(s'|h', a)$
- $\gamma \in \{0, 1\}$ is a discount factor (representing the optimization horizon of an agent)

The POMDP operates as follows. At each time-step, the highway is in some unobserved state h. The agent chooses an action a and the highway is simulated for two seconds. On the next time-step: The state of the highway is changed to the future state h' according to the highway simulation (formalized as transition probability function P), the agent observes its own state s' (formalized as observation probability function Z) and receives a reward r according to R. The mapping of the agent from observations to actions, i.e., the decision the agent makes given the current state, is called the policy π: $a = \pi(s)$.

Reinforcement Learning is based on the Reward hypothesis: "*All* goals can be described by the maximization of the expected cumulative reward" [10], where the reward is a scalar feedback value. It is common to use a discounted cumulative reward, the return G_t:

$$G_t = \sum_{k=1}^{\infty} \gamma^{k-1} r_{t+k} \tag{2-1}$$

Where t is the current time-step, r_{t+k} is the reward on future time-steps $t + k$ and $\gamma \in (0, 1)$ is the discount factor. The discount factor ensures that future rewards are discounted more.

The sampling time of our simulator is two seconds. At each time-step, the agent chooses an action according to the current observation s and policy π: $a = \pi(s)$. After two seconds it receives a new observation s' and reward r. Together this forms an experience, $e = \{s, a, r, s'\}$. In Reinforcement Learning (RL), the policy is updated iteratively using the experiences of the agent, to optimize the return.

2-1-2 Q-Learning

Q-Learning is a commonly used value-based RL algorithm [11]. It is the basis for DRL, and therefore, a brief introduction to Q-Learning follows.

The Q-value (Q), also known as action value, is an estimation of the return of the agent, given the current observed state s, taking action a and then following policy π.

$$Q(s, a) = \mathbb{E}[G_t|s_t = s, a_t = a, \pi] \tag{2-2}$$

Where policy $\pi(s)$ would determine all actions after the current one ($a_t = a$) is executed. $Q(s, a)$ is an estimation for multiple reasons: The current state of the highway h, the model $T = \mathbb{P}(h'|h, a)$ and the observation function $Z = \mathbb{P}(s'|h', a)$ are unknown to the agent. Moreover, T and Z are probability functions.

Let us define the optimal policy $\pi^*(s)$ as the policy that maximizes the return. The optimal Q-value can then be defined as $Q*(s, a) = \mathbb{E}[G_t|s_t = s, a_t = a, \pi^*]$. If the optimal Q-value of each action is known, it is best to pick the action with the highest return; follow a greedy-policy:

$$\pi(s) = \text{argmax}_a[Q(s, a)] \tag{2-3}$$

If a greedy policy is expected, we could estimate the return as follows:

$$G_t = r + \gamma \arg\max_{a'} Q(s', a'). \tag{2-4}$$

The stationary Bellman optimality equation holds for all actions and states if the optimal policy and Q-value are found [11]:

$$Q^*(s, a) = r + \gamma \max_{a'} Q^*(s', a') \tag{2-5}$$

In Q-learning the return is optimized by iteratively bootstrapping the temporal difference error (Equation 2-6). It is an update of the expected return using the error of the bellman optimality equation:

$$Q(s, a) = Q(s, a) + \alpha[G_t - Q(s, a)] \tag{2-6}$$

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)] \tag{2-7}$$

Where α is the learning rate. One experience $e = \{s, a, r, s'\}$ is sufficient information to update $Q(s, a)$ for one iteration (Equation 2-7).

If the agent follows a greedy policy, the algorithm is likely to end up in local optima. This can be avoided by incorporate some exploration of the agent into the training process. We use an ϵ-greedy policy during training:

$$\pi(s) = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \text{argmax}_a[Q(s, a)] & \text{with probability} 1 - \epsilon \end{cases} \tag{2-8}$$

Where $\epsilon \in (0, 1)$ is the exploration fraction. We anneal ϵ from 1 to 0.05 during single-agent training to start with more exploration and end with more exploitation.

2-1-3 Deep Q-Networks

In DRL, a DQN is used to estimate the Q-value [3]. A DQN is a deep learning network with weights and biases θ [12]. The input neurons are all the observations the agent makes on time-step t; the state (s_t) of the agent. The output neurons are the Q-values per individual discrete action choice. I.e., every output node corresponds to its own action choice and represents the expected return of that action, given the current state s_t.

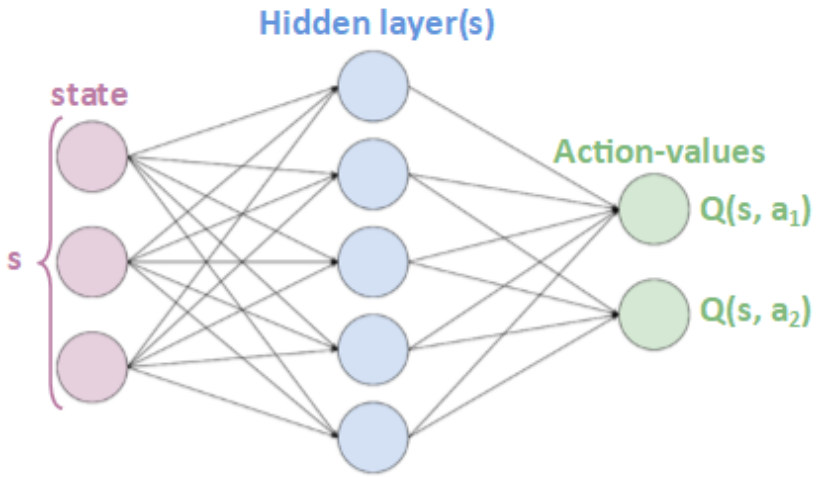


Figure 2-1: deep Q-Network

See Figure 2-1. In this case, the agent has three observation variables and two action choices. Input neurons s_t are the observed state of the agent. Output neurons $Q(s, a_1)$ and $Q(s, a_2)$ are the estimated return of the agent (Q-values) for observation s and action choice a_1 and a_2 respectively.

The Q-value (output of the DQN) is dependent on weights and biases θ. $Q(s, a)$ can be reformulated as: $Q(s, a, \theta)$. A key factor in DRL is the use of a target network with stored weights θ^-: $Q(s, a, \theta^-)$. Every k iterations, the target network is updated $\theta^- = \theta$. Essentially freezing the Q-Network in terms of estimation of the future return.

Another key factor of DRL is the use of an experience replay memory. At each time instance, experience $e_t = \{s, a, r, s'\}$ is stored in replay memory $D_t = \{e_1, e_2, \cdots, e_t\}$. A mini-batch $M \sim U(D_t)$ of batch-size m (a tuning hyper-parameter) is uniformly sampled from the replay memory. Let us define e_i^M as the i-th experience in mini-batch M. θ is trained by minimization of the error of the Bellman equation, done by stochastic gradient descent on a Mini-batch of experiences. The following Loss function (L) is minimized:

$$L(\theta) = \sum_{i=1}^{m} [(r + \gamma \max_{a'} Q(s', a', \theta^-) - Q(s, a, \theta))^2]_{(s,a,r,s')=e_i^M} \tag{2-9}$$

We use the Adam optimizer of Tensorflow [13] with a learning rate of $2.5e^{-5}$ to minimize the loss function over each batch. The target of the DQN: $r + \gamma \max_{a'} Q(s', a', \theta^-)$ is determined with a target network, reducing direct correlations of $Q(s, a, \theta)$.

2-2 Multi-Agent Algorithm

In this section, we present our modification of single-agent DRL towards multi-agent learning. Similar to Radulescu et al. [7]; Deep Reinforcement Learning in a multi-agent homogeneous open population, we employ *centralized training with decentralized execution.* First, we present the formal problem setting for our multi-agent system, then, the way Radulescu et al. tried to solve this for a multi-agent highway problem, and finally, our modification to it.

2-2-1 Decentralized Partially Observable Markov Decision Process

A POMDP describes a single-agent system, to include multi-agent decision making, we extended the formalization to a Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [14]; this is the formal problem setting of our multi-agent highway problem. In this work, we simulate the highway as a homogeneous open population, so the formalization is slightly compressed compared to a broader statement found in [14].

Our Dec-POMDP is a tuple of: $\mathscr{M} = \{D, H, \mathbb{A}, T, \mathbb{S}, O, R, \gamma\}$, where

- $D = \{1, \cdots, n\}$ is a finite set of agents
- H is a set of highway states h
- $\mathbb{A} = A_i$, $(i \in D)$ is a set of joint action spaces A of all agents
- T is transition probability function $\mathbb{P}(h'|h, a)$
- R is the immediate reward function $H \times A \to \mathbb{R}$ (feedback of an agent)
- $\mathbb{S} = S_i$, $(i \in D)$ is the set of joint observation state spaces S_i of all agents
- Z is the observation probability function $\mathbb{P}(s'|h', a)$
- $\gamma \in \{0, 1\}$ is a discount factor (representing the optimization horizon of the agents)

Our Dec-POMDP operates as follows. At each time-step, the highway is in some unobserved state h. All agents observe their state s_i and initialize their action a_i according to observation probability function Z and the *homogeneous* policy π respectively. After two seconds of execution, the highway will end up in state h'. Agents receive *their own* reward r_i according to R and observe their new state s_i according to Z.

The difference with the broad Dec-POMDP formulation in [14] is: In their formulation, agents can employ individual policies π and the reward function is a central feedback signal representing system-wide performance. In our case, all agents share the same homogeneous policy π and employ an individual (also homogeneous) selfish reward function R. While the goal is to form a policy that optimizes the sum of all agent rewards (broad definition of Dec-POMDP), we try to do it by maximizing the selfish reward of individual agents, designed to promote good system-wide driving behavior: Agents are non-cooperative, each agent receives its selfish reward feedback.

2-2-2 From single- to multi-agent training

As stated earlier, we use centralized training with decentralized execution on our homogeneous multi-agent system. The single-agent learning algorithm of section 2-1 is used to train the shared policy of all the agents. In our case, an individual agent stores its experience in the replay memory. The replay memory (of the individual agent) is used to update the DQN according to the loss function (Equation 2-9). The DQN is shared by all the agents in the system.

First, we train the policy in a single-agent setting, where the non-ego vehicles are passive agents. After completing the single-agent training process, we use the resulting policy as the starting policy of the multi-agent system. According to Radulescu, homogeneous multi-agent learning via policy reuse from a single agent training yields better results and is much faster than learning from scratch in this type of setting.

There is one major difference in our system compared to that of Radulescu. Agents observe and display each other's lane changing decisions. In our algorithm, we need to initialize agents sequentially. Meaning that at each time-step, agents will observe their environment s_i and initialize their action $a_i = \pi(s_i)$ one at a time. The current agent can observe all previously initialized agent's lane change decisions. Agents that still need to initialize their actions are observed as non-lane changing agents by the current agent.

To optimize the centralized training efficiency, we split the agents into one 'training' agent and other 'model' agents. We assign the agent that initializes its action last as the training agent. All previously initialized agents are model agents. This way, the training agent has full information of non-ego lane change decisions, and the homogeneous DQN is trained to adapt to non-ego lane change decisions. This DQN is shared by the training and model agents.

Let us give an example of action initialization during a time-step. The first agent considers all other agents as non-lane-changing during the time-step, and decide its action accordingly. The next agent knows the decision of the previous agent and considers all other agents to be non-lane-changing, and so on.

If following the optimal policy:

- **Safe actions will remain safe**
  Agents avoid collision with previously initialized agents. If a lane change seems safe in the current situation, it is safe after the initialization of other agents, even while incorrectly predicting other vehicles lane-change intent.

- **There is always a safe option**
  Since previous agents assume the current agent to keep its lane and avoid collisions accordingly, the safe option of not changing lanesis always available for the currently deciding agent, even if it is the last one.

An additional advantage of splitting the agents into model agents and one training agent, is that model agents do not need to explore the action space. Only the training agent needs to perform exploration. Model agents drive according to the greedy policy (Equation 2-3), whereas the training agent drives according to the ϵ-greedy policy (Equation 2-8). Because of this, model agents simulate the environment more accurately. Only the agent that updates the shared DQN needs to perform exploration. A disadvantage is that filling up the replay-memory takes longer. Only one experience is stored during a time-instance of the highway. Whereas when all agents are training agents, every agent stores its experience in the replay memory and multiple training iterations could be applied on one time-step of the environment.

Chapter 3

Observations, actions and reward function of the agents

In this chapter, we present the framework of the deep Q-Network (DQN) agents. In Deep Reinforcement Learning (DRL), an agent determines its action based on the observations s_t it makes at time instance t and its policy π: $a_t = \pi(s_t)$. π is optimized on the return. The state space is a vector of continuous variables, and the action space consists of discrete action choice options.

We designed three agent classes: A 'passive' agent, a 'basic' agent, and an 'advanced' one, where the difference lies mostly in the control actions. First, we present the state space (section 3-1), surrounding vehicles are observed in an observation grid yielding a couple of advantages. Then in section 3-2, an Adaptive Cruise Control (ACC) controller is presented to determine longitudinal acceleration with a sampling time of 0.1 seconds without the need for agent actions. In the following section, we present the three different action agent classes. A passive agent has no action choices, a basic agent can choose between lane change maneuvers, and an advanced agent includes additional action choices that adjust the ACC controller. In the final section of this chapter, we present the reward function that is used to give feedback on the desired driving policy of the agents.

The design goal of the state and action space was to simplify the highway driving problem as much as possible by reducing action and state space complexity and size without losing performance. The reward function is there to reinforce the four driving goals:

1. Drive safe
2. Reach the destination
3. Drive fast
4. Drive comfortable

Table 3-1: State space of the agent

	State	Description	
Ego states	s_1	Speed	v (m/s)
	s_2	Intended speed	v_{int} (m/s)
	$s_3 \cdots s_8$	Current lane	(six bools)
	$s_9 \cdots s_{14}$	Destination lane(s)	(six bools)
V_1	s_{15}	Distance (long.)	Δx
	s_{16}	Speed Difference	Δv
	s_{17}	Changing left	(bool)
	s_{18}	Changing right	(bool)
$V_2 \cdots V_{12}$	$s_{19} \cdots s_{62}$	$\vdots$	

3-1 Observed state

The observed state of an agent is a vectorization of some high-level data input, observed by lower-level observers. In this work, we assume perfect tracking for all states.

3-1-1 State vector

Let us divide the states of the agent into two groups: ego states and surrounding states. The ego states consist of the agent's current speed, intended speed of the ACC (subsection 3-2-1), the current lane, and the destination lane(s). The surrounding states consist of the speed difference, longitudinal location difference, and the lane change intention (blinker status) of the surrounding vehicles. Agents observe twelve surrounding vehicles via an observation grid, explained in subsection 3-1-2 (Observation grid).

The speed and location difference of surrounding vehicles could be measured with object detection algorithms, see [15], [16] and [17] for some examples. For some lane change detection algorithms refer to [18], [19] and [20], where prediction of future lane changes is accurate up to 93%. We leave modeling of the observation errors and researching the best object- and lane change intent- detection algorithm for further research.

The current and desired speed states are real values. The current lane and destination lane is a commonly used one-hot representation, a logical vector of zeros and ones. Since there is a maximum of 6 lanes, the state of the lane is a vector of six states where the state corresponding to the current lane is equal to one, and the other states are equal to zero. Same for the destination lane(s). Since there could be multiple destination lanes, the destination lane vector could consist of multiple ones.

See Table 3-1. V_i is observed vehicle i where i corresponds to its location in the Observation grid presented in the next section in Figure 3-1b.

3-1-2 Observation Grid

In [9], vehicles are observed by order of proximity, making the surrounding states subject to object interchangeability. A convolutional neural network was used to solve this problem. In

this work, we aim to reduce the state vector complexity by using the objects state vector indices to our advantage. By using an observation grid, the agents induce the lane of surrounding vehicles from the indices in the state vector, making lane observations of observed vehicles obsolete. Moreover, a small change in the location of surrounding vehicles leads to big changes in the state vector because of the strict sorting of vehicles.

See Figure 3-1a and Figure 3-1b, an example of a state of the highway and the state of the highway including the observations of the agent respectively. We divide the observed surroundings of an agent into ten areas, five lanes and two per lane, one in front and one behind the agent. Every region contains at least one observed vehicle, the two areas in front of- and on adjacent lanes of- the agent contain two observed vehicles each.

Per observation region, we observe the vehicle(s) closest to the agent. If there is no vehicle inside an observation area, the agent observes a 'phantom vehicle' on the saturation distance with a speed difference of zero. If a lane inside the observation grid is an off-road lane (e.g., if the agent is driving in the leftmost lane there are no lanes to the left), we fill the corresponding vehicle states with phantom vehicles.

The numbers in Figure 3-1b are there for explanation purposes and will from now on correspond to its location in the observation grid, e.g., vehicle 7 (V_7) is referred to as the observed vehicle directly in front of the agent A brief explanation of the observations per lane follows:

On two lanes to the left: The agent should observe one vehicle behind and one vehicle in front at all times. There is only one lane to the left of the agent. The observed lane is a phantom lane. The agent observes the vehicle behind it as phantom vehicle 1 (V_1), and in front as phantom vehicle 2 (V_2). Both phantom vehicles are driving on the saturation distance with a speed difference of zero and have no lane change intentions.

On the left adjacent lane: The nearest vehicle behind the agent is vehicle 3 (V_3). The agent observes two vehicles in front, on an adjacent lane. It observes the nearest vehicle V_4, and the second nearest is a phantom vehicle V_5.

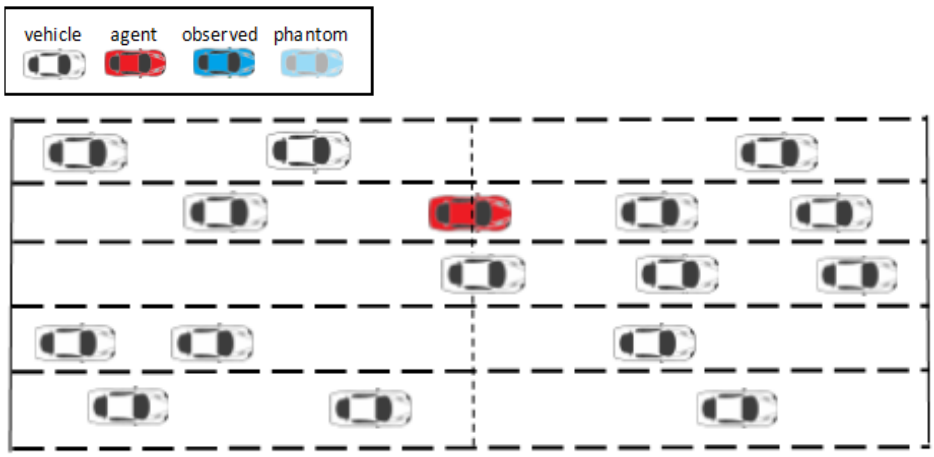
On the current lane: The agent observes the nearest vehicles behind and nearest in front as V_6 and V_7 respectively.

On the right adjacent lane: There is no vehicle behind the agent, it observes phantom vehicle V_8. There are three vehicles in front of the agent. The agent observes the two closest vehicles; V_9 and V_{10}. Vehicles are observed through their center-location in the observation grid, that is why V_9 is observed as a vehicle in front of the agent.

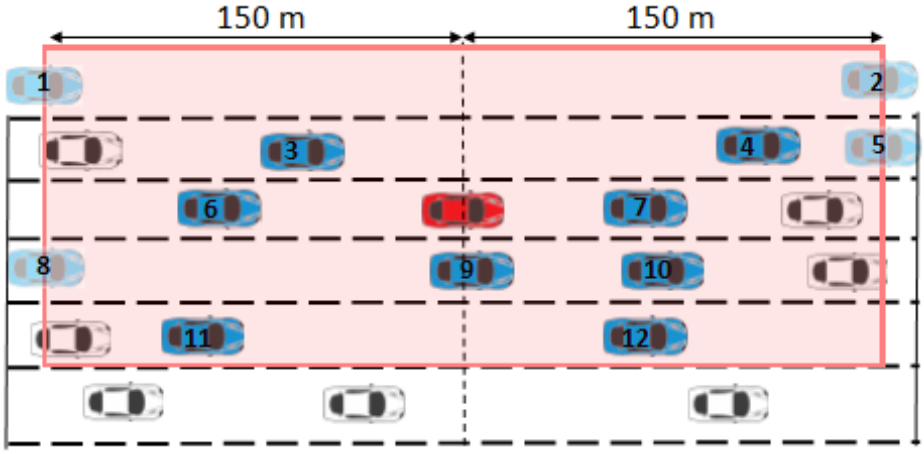
On two lanes to the right: The agent observes the closest vehicle behind the agent V_{11}, and the closest vehicle in front V_{12}.

Some advantages of the observation grid are:

1. *More compact observations.* State values have a corresponding location in the environment, making lane observations of surrounding vehicles obsolete. In this work, the maximum amount of lanes is six. Since the lane observations is a one-hot representation, this would result in six extra state indices per observed vehicles. By using an observation grid, we can reduce the state vector size from 136 to 62.



(a) Real highway state



(b) Observed highway state

Figure 3-1: Observation grid example

2. *More direct correlation with environment.* When using the order of proximity for the surrounding vehicles, a small change in the surroundings can lead to a big change in the state since the order of the observed vehicles could change.

3. *Implementable.* In [9], a fixed amount of vehicles are observed, but there is no use of phantom vehicles. There is no intuitive solution for an empty highway. Our phantom vehicles provide a way to fill the state space on an empty highway. Moreover, in the simulation, we only observe vehicles that are close to the agent. In [9], observed vehicles could be located too far away from the agent to observe them in real life.

3-2 Actions

The actions of the agent are high-level driving decisions, carried out by lower-level controllers. In this work, we assume perfect tracking of a reference signal created by the simulator presented in chapter 4. For some path following algorithm examples, refer to [21], [22]. We assume perfect tracking of the reference signal and leave errors and design of the tracking algorithm for further research.

We have split the dynamics in the simulation into longitudinal and lateral control systems. The lateral dynamics of the basic agent is determined with the lane change maneuver it chooses. The ACC controller we present in subsection 3-2-1 determines the longitudinal dynamics of the agents. In the final section of this chapter, we present the different action spaces for the three agent classes; passive, basic, and advanced agents.

All agents are unable to choose off-road lane changes. I.e., on the leftmost lane, the agent can not change lane to the left, on the rightmost lane, the agent can not change lanes to the right. The ϵ-greedy and greedy policy are both restricted to only possible actions. Moreover, the estimation of the target network: $r + \gamma \max_{a'} Q(s', a', \theta^-)$ only considers the possible actions of the next state s'. If on the next state s' the agent is on the leftmost lane, the action of changing left is not considered in $\max_{a'} Q(s', a', \theta^-)$. This way, the action space is restricted to only on-road actions during simulations and training.

3-2-1 Adaptive Cruise Controller

In this work, we separated lateral and longitudinal control, simplifying the action space of the agent. We simulate real-time longitudinal control of the agent with an ACC controller based on [23]. The ACC controller is a lower level controller carrying out higher level decisions of the agent. The controller uses a 'target' vehicle, usually the vehicle directly in front of the agent, to determine its acceleration. The target vehicle can be set to multiple vehicles, as explained in section 3-2. For explanation purposes, assume in this section, the target vehicle to be the vehicle directly in front. As a result of the ACC controller, the agent slows down when it approaches a target vehicle driving at a lower speed, and the agent increases its speed to the intended speed v_{int} when the target vehicle accelerates or disappears.

An important concept for this ACC controller is the Time In-between Vehicles (TIV), also known as inter-vehicular time. TIV is a common risk indicator. It is the time it takes for the agent to cover the distance towards the target vehicle; $\frac{\bar{x}-x}{v}$. Let us define the desired

TIV and reference distance as t_{ref} and $r_{\text{ref}} = v * t_{\text{ref}}$ respectively. The ACC controller uses a desired TIV of $t_{\text{ref}} = 2\,\text{s}$. If the agent drives at a distance of $x = \bar{x} - r_{\text{ref}}$, the TIV will be two seconds. The ACC controller uses r_{ref} as the target distance to control the agent towards a 'safe' driving distance, where TIV is equal to $t_{\text{ref}} = 2\,\text{s}$.

It is a feedback controller with as input: the intended speed of the ego vehicle (v_{int}), the acceleration of the target vehicle ($\bar{a}$) and the location and speed of the ego vehicle; x, v, and the target vehicle; $\bar{x}$, $\bar{v}$. The output is the acceleration of the ego vehicle $\dot{v}$. In this work, the ACC controller employs a sampling-time of 0.1 s.

Two accelerations are calculated and used to determine the acceleration of the ego vehicle.

Free acceleration $\dot{v}_{\text{free}}$: It is used to accelerate towards an intended speed of v_{int} if the target vehicle is far away.

$$\dot{v}_{\text{free}} = k(v_{\text{int}} - v) \tag{3-1}$$

with k as a constant factor.

Reference acceleration $\dot{v}_{\text{ref}}$: It is used to decelerate towards a safety distance of r_{ref} if the target vehicle is close by.

$$\dot{v}_{\text{ref}} = k_a\bar{a} + k_v(\bar{v} - v) + k_d(\bar{x} - x - r_{\text{ref}}) \tag{3-2}$$

with k_a, k_v and k_d as constant factors.

$\dot{v}$ is then determined by taking the minimum of $\dot{v}_{\text{ref}}$ and $\dot{v}_{\text{free}}$ and bounding it between $-3\,\text{m/s}^2$ and $2\,\text{m/s}^2$:

$$\dot{v} = \min[\dot{v}_{\text{ref}}, \dot{v}_{\text{free}}]; \quad -3 < a < 2 \tag{3-3}$$

We employed $k = 0.3$, $k_a = 1.0$, $k_v = 0.58$ and $k_d = 0.1$ in this work. According to [23]: This resulted in the most smooth and fast reaction of the controller without leading to unsafe situations compared to the other settings, while keeping $k = 0.3$ and $k_a = 1.0$ in accords with earlier microscopic-traffic simulation model (MIXIC) literature.

3-2-2 Passive agents

A passive agent is an agent that is unable to change lanes. It does not contain a state- or action space. It is purely controlled by the ACC controller presented in subsection 3-2-1.

3-2-3 Basic Agent

A basic agent's action space consists of three maneuver choices: change lane to the left, stay in the current lane, and change lane to the right. The target vehicle of the ACC controller is the vehicle directly in front, on the current lane (V_7) at all times. During a lane change, when the agent crosses the lane border, the vehicle directly in front and on the new current lane is used as the target vehicle (V_7 is another vehicle because of the lane change).

If the agent chooses to stay in the current lane, it maintains the center of the lane as the lateral location of the agent. If an agent decides to change lanes, the agent follows a trigonometric S-function from the center location of the current lane to the center location of the adjacent lane in two seconds (see section 4-2). Taking two seconds to change lanes is reasonably comfortable at all allowable speeds.

Table 3-2: Action Space of both agents

Passive Agent	
a_1	Stay in current lane
Basic Agent	
a_1	Change lane to the left
a_2	Stay in current lane
a_3	Change lane to the right
Advanced Agent	
a_1	Change lane to the left
a_2	Stay in current lane
a_3	Change lane to the right
a_4	Set V_3 as target vehicle
a_5	Set V_4 as target vehicle
a_6	Set V_8 as target vehicle
a_7	Set V_9 as target vehicle

Neural Network design

Both DQNs (basic and advanced) have two rectified linear units (ReLU) hidden layers with 256 neurons each. The input layer is the size of the observation state: 62. The output layer is the size of the number of actions: 3 for the basic agent.

3-2-4 Advanced agent

The advanced agent's action space includes additional options for choosing the target vehicle of the ACC controller from its surroundings. Instead of the vehicle directly in front (V_7), it could also choose the target vehicle to be a vehicle on adjacent lanes. Namely: V_3, V_4, V_8 and V_9 (see Figure 3-1b). Since we only use longitudinal measurements, the ACC controller works the same as in subsection 3-2-1.

For an overview of the action space of both agents see Table 3-2. a_i is action choice i. V_i is a vehicle in the observation grid where index i corresponds to its place in the observation grid, shown in Figure 3-1b.

The advanced agent can efficiently adjust its speed with a relatively long action sampling-time of two seconds. When the agent chooses a target vehicle, it remains the target vehicle for the action span of two seconds. If the agent selects a lane change maneuver, V_7 is set as the target vehicle. At the moment the agent crosses the lane border, the target vehicle switches automatically to the new vehicle directly in front of the agent.

The desired longitudinal TIV towards target vehicles on adjacent lanes (t_{ref}) is changed to one second, half the t_{ref} when the target vehicles is the vehicle directly in front (two seconds). This TIV ensures that when the agent chooses adjacent vehicles as its target vehicle, the agent drives longitudinally in the middle of the vehicles on adjacent lanes, allowing for better merging.

Moreover, the lowest reference acceleration $\dot{v}_{\text{ref}}$ between the chosen target vehicle and if the target vehicle is V_7 is used. This way, the agent does not collide with the vehicle in front of

it if it suddenly decides to slow down, even when an adjacent vehicle is used as the target vehicle.

Let us give an example of when the agent would like to change lanes behind V_3 (vehicle left behind): It would first choose a_4 (follow V_3) until V_3 passes and changes to V_4 (vehicle left in front). The action choice should now change to a_5 (follow vehicle left in front) until the agent drives at a TIV of $t_{\text{ref}} = 1\,\text{s}$. At that point, a lane change maneuver would be safe, and the agent should pick action a_1 (change lanes left).

Neural Network design

Both DQNs (basic and advanced) have two ReLU hidden layers with 256 neurons each. The input layer is the size of the observation state: 62. The output layer size is the number of actions: 7 for the advanced agent.

3-3 Reward function

The reward function is given in Equation 3-4. There is no shared reward between agents; agents are non-cooperative. The training agent tries to optimize its own (selfish) reward function. The Open Freeway Cooperation rewards are there to reinforce good system-wide driving performance when vehicles cannot pass on the right side, by using single-agent stimulation.

In this work, two scenarios are simulated with different reward functions: An exit scenario where all agents want to drive towards a particular target lane, and an open freeway scenario where all agents drive freely on the open highway. See section 4-3 for more information on the difference between these scenarios.

$$r = \underbrace{\omega_1 r_d + \omega_2 r_c}_{\text{Drive safe}} + \underbrace{\omega_3 r_r}_{\text{Reach destination}} + \underbrace{\omega_4 r_s}_{\text{Drive fast}} + \underbrace{\omega_5 r_l}_{\text{Drive comfortable}} + \underbrace{\omega_6 r_p + \omega_7 r_t}_{\text{Open Freeway Cooperation}} \tag{3-4}$$

Danger penalty r_d: 1 if the agent was in danger during the action, 0 if not. The agent is in danger if the TIV of the agent and a vehicle in the same lane (in front or behind) is below 0.5 s. A negative ω_1 will result in avoidance of dangerous driving behavior. We used, $\omega_1 = -5$ for both scenarios.

Collision penalty r_c: 1 if collided during the action, 0 if not. A negative ω_2 will result in a collision avoidance, we used $\omega_2 = -5$ for both scenarios. Note that when an agent collides, they will also be in danger; TIV = 0. So the reward will be $\omega_1 + \omega_2 = -10$

Target lane penalty r_t: 0 if the vehicle is on a destination lane, 1 if not. Negative ω_3 results in a faster reaching of destination lanes, we used $\omega_3 = -1$ for both scenarios.

Speed reward r_s: This is the speed of the vehicle normalized between the minimal and maximal speed:

$$r_s = \frac{v - v_{\min}}{v_{\max} - v_{\min}} \tag{3-5}$$

. A positive ω_4 reinforces faster driving strategies. We used $\omega_4 = 1$

Lane change penalty r_l: 1 if changing lane, 0 if not. Negative ω_5 results in less lane changing and more comfortable driving, we used $\omega_5 = -1$ for both agents.

keep Right penalty r_r: r_r linearly scales from 0 to 1 for all six lanes. 1 if on the leftmost lane, 0, if on rightmost lane. Because of 6 lanes, r_r scales with 0.2 per lane. Negative $\omega_6 > -\omega_1$ results in more driving on the right side of slow vehicles. We used $\omega_6 = 0$ in the exit scenario and $\omega_6 = -0.5$ in the open freeway scenario.

Overtake penalty r_o: 1 if agent overtook a vehicle on the right side during the action, 0 if not. Positive ω_7 results in less passing on the right, faster vehicles move to left lanes, and slower vehicles move to right lanes (because of r_r). We used $\omega_7 = 0$ in the exit scenario and $\omega_7 = -5$ in the open freeway scenario. The agent can only observe vehicles up to two lanes to the side, overtaking on the right is only penalized if the agent can observe the vehicle that is overtaken. Keeping true to the design-goal of excluding explicit communication

Since it is not possible for agents to choose off-road lane changes (section 3-2), we do not need to include penalties for driving off-road.

Chapter 4

Highway simulator

In this chapter, we present the 2-D highway environment created in Python. It is a loop of 500 m containing six lanes with lane width 3.6 m. All Vehicles are automated driving vehicles modeled as either: passive, basic, or advanced agents with a state and action space described in chapter 3. They are simulated as rectangles with vehicle length 3.2 m, vehicle width 1.8 m and center-location x and y.

We first present how we initialized the simulation on each episode. We use a 'passive agent' as model agents in single-agent simulation and basic or advanced agents for model agents in multi-agent simulation. After that, we present the reward function and its hyper-parameters, and finally, the two training scenarios and their corresponding weights in the reward function.

See Figure 4-1. This figure is a time-stamp of the highway simulation. We changed aspect ratios and vehicle lengths for display purposes. The red vehicle is the training agent, and the transparent red area is the observation area of the training agent. White vehicles are unobserved vehicles, blue vehicles are observed vehicles, and blue transparent vehicles are phantom vehicles. If the simulation is in a multi-agent setting, every vehicle is an agent with its observation area, observed vehicles, and phantom vehicle.

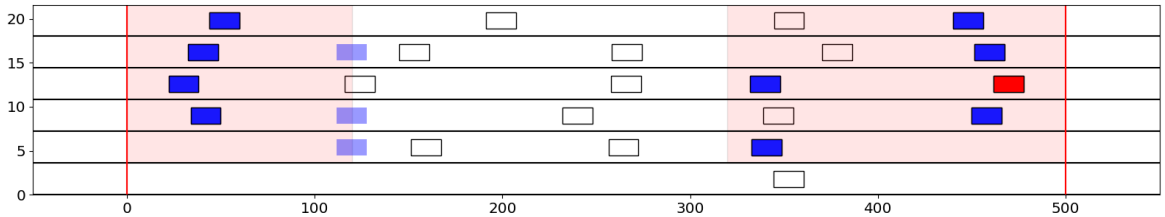


Figure 4-1: Highway Simulation time-stamp

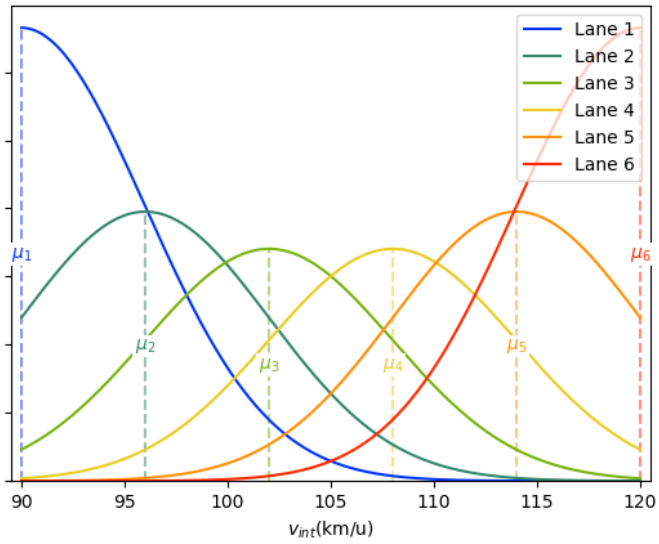


Figure 4-2: Intended speed distributions of passive agents per lane

4-1 Highway Reset

At the start of a simulation episode, we initialize the highway by removing all previously active agents and spawning new random agents. One training agent and either passive agents (single-agent system) or model agents (multi-agent system). Let us define this as 'reset' of the highway. The episode runs until the training agent reaches the end of the highway given by L_w. Then, we reset the highway and start simulating a new episode.

The amount of spawned vehicles are sampled linearly from $2 < n < 54$ (maximum of 9 per lane). They are distributed randomly over the highway loop. Their lane is randomized (max 9 per lane) and they are distributed longitudinally on each lane by using a random vector with a minimum difference and fixed sum representing each vehicles longitudinal location on the lane. The way their speed is sampled is stated in the next subsection.

4-1-1 Intended speed sampling

The intended speed of training and model agents are uniformly randomly sampled from $v_{\min} = 90\,\text{km/h}$ to $v_{\max} = 120\,\text{km/h}$. For passive agents we sample intended speeds to model the highway in a similar way as the traffic flow model in [24]. Intended speed sampling distributions per lane for passive agents are shown in Figure 4-2.

All speed distributions are normal distribution with an area of 1. Mean μ_i is linearly sampled from $v_{\min} = 90\,\text{km/h}$ to $v_{\max} = 120\,\text{km/h}$ for each lane i, and standard deviation $\sigma = 30/5 = 6$ is one fifth of the speed difference on the highway. These speed distributions result in a highway where vehicles are generally driving faster on the left lane and slower on the right. The starting speed of the vehicles is equal to the sampled intended speed but will change due to the Adaptive Cruise Control (ACC) controller during simulation.

Intended speed of advanced and basic agents are uniformly randomly sampled from $v_{\min} = 90\,\text{km/h}$ to $v_{\max} = 120\,\text{km/h}$.

4-2 Lane change maneuver

A trigonometric S-function is used to simulate lane change maneuvers. This function is used to simulate the lateral position of a vehicle during a lane change. The goal is for the vehicle to end up in the center of an adjacent lane along a smooth and realistic path after $t_{\text{change}} = 2$ seconds.

First let us define Δx and Δx_{end} as:

$$\Delta x = x - x_{\text{start}} \tag{4-1}$$

$$\Delta x_{\text{end}} = v * t_{\text{change}} \tag{4-2}$$

. Where x_{start} is the longitudinal position of the vehicle at the start of a lane change. In this work, a lane change maneuver will take $t_{\text{change}} = 2$ seconds. The maneuver should be completed when $\Delta x = \Delta x_{\text{end}}$.

The S-function used in this manuscript is given in Equation 4-3, resulting in Figure 4-3 if the agent would drive at a constant speed.

$$\begin{aligned} y &= y_{\text{start}} + \Delta y_{\text{left/right}} \\ \Delta y_{\text{left}}(\Delta x) &= -[\cos(\frac{\pi \Delta x}{\Delta x_{\text{end}}}) - 1] * 0.5 L_w \quad (0, \Delta x_{\text{end}}) \\ \Delta y_{\text{right}}(\Delta x) &= [\cos(\frac{\pi \Delta x}{\Delta x_{\text{end}}}) - 1] * 0.5 L_w \quad (0, \Delta x_{\text{end}}) \end{aligned} \tag{4-3}$$

y is the lateral location of the vehicle, y_{start} is the lateral position of the vehicle at the start of a lane change (middle of the current lane) and L_w is the lane width in meters.

Since the maneuver is dependent on the speed, all lane change maneuvers take a 'comfortable' two seconds.

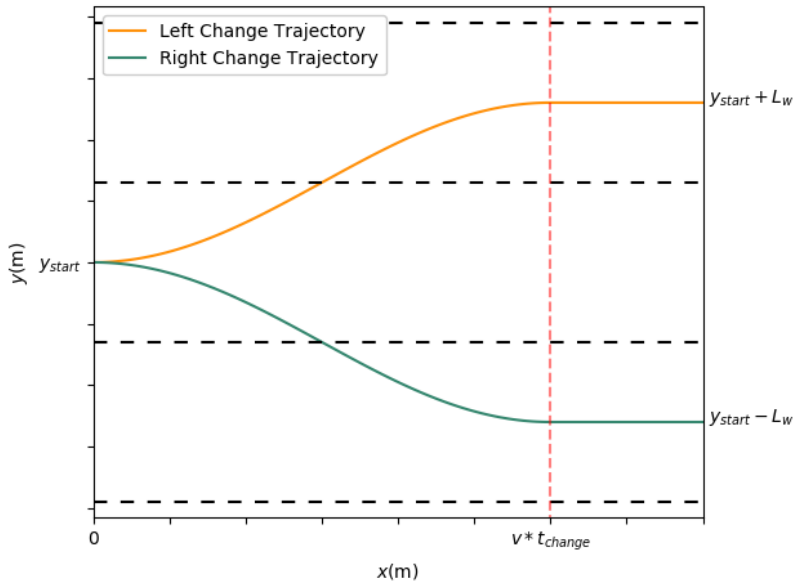


Figure 4-3: Lane Change Maneuver (Equation 4-3)

Table 4-1: Reward function hyper-parameters

Hyper Parameters	ω_1	ω_2	ω_3	ω_4	ω_5	ω_6	ω_7
Exit scenario	-5	-5	-1	1	-1	0	0
Freeway scenario	-5	-5	-1	1	-1	-0.5	-0.5

4-3 Exit- and Open freeway scenario

The highway driving problem is split up into two tasks. One is reaching a destination lane, and the other is driving on an open freeway as fast as possible without breaking European traffic regulations.

See the table of tuning choices ω in Table 4-1. $\omega_1 \cdots \omega_5$ are similar for both scenarios. Tuning choices of ω_6 and ω_7 will be explained below and is different for the different driving scenarios.

4-3-1 Exit scenario

The exit scenario simulates a part of the highway where there are multiple exits. All agents have their destination lane. They want to reach this lane while maintaining their intended speed. The destination lane is sampled during highway initialization and is a random lane that is not the current lane. This scenario involves a high amount of lane changing. Keeping right and overtaking rules are unimportant when trying to reach a destination lane; $\omega_6 = \omega_7 = 0$.

4-3-2 Open freeway scenario

The open freeway scenario simulates a part of the highway where there are no exits and where agents can drive freely on an open highway. Agents choose their preferred lanes according to their intended speed. The destination lane is set to be all lanes on the highway. The destination lane state is a vector of six ones.

In the open freeway scenario, the entire highway consists of destination lanes. We assumed vehicles are only allowed to overtake on one side, in our case, the left side. Hence $\omega_6 = -5$, resulting in vehicles with high intended speed driving on the left lanes. Driving on the left side is penalized by scaling the reward from 0 to $\omega_5 = -0.5$ making slower vehicles drive on the right side.

4-4 Training algorithm

We used the highway simulator presented in this chapter as the environment of the agents. During training, the training agent uses an ϵ-greedy policy to determine its action and explore the environment. During test simulations, the training agent uses the greedy policy to simulate full exploitation.

See the training algorithm in Algorithm 4.1. First, we initialize the Replay Memory by using random actions for the training agent until the replay memory has the same size as the batch size so that we can sample a batch.

Then every training step, we first anneal ϵ towards ϵ_{min}. This is done by linearly annealing from the start value ϵ_{start} to end value ϵ_{min} within the percentage of exploration iterations (half) of the total iteration ($10^6 * 0.5 = 5\text{e}^5$ iterations). After half of the total iterations, ϵ remains equal to $\epsilon = \epsilon_{\text{min}}$. ϵ_{start}, ϵ_{min} and the number of exploration iterations are tuning hyper-parameters of the training process, describing the amount of exploration of the training agent.

Every iteration, after ϵ annealing, the agents sequentially initialize their actions. When the system is a single agent system, there are no model agents; only the training agent needs to be initialized. Note that in the multi-agent setting, we initialize the training agent after the model agents.

After the initialization of the actions, we simulate the highway for two seconds and store the experiences. If the highway is in a terminal state, we reset the highway, and the experience is not stored.

After simulating and updating of the replay-memory, we sample a mini batch M and update the DQN weights and biases with Equation 2-9.

Then, if the current iteration is a multitude of the Test iteration (k_{test}), a tuning hyper-parameter), a test simulation is run for n_{test} iterations (another tuning hyper-parameter). We do this is to observe how well the system would perform if the training agent would not do any exploration. I.e., we stop training every k_{test} iterations and perform a test of n_{test} iterations to evaluate the state of the current deep Q-Network (DQN). If the mean reward of the test ($\bar{R}$) is the highest yet, we store the DQN as the best performing one.

Separately, if the current iteration is a multitude of the target network update (k_{update}), the target network's weights and biases are updated to be the same as the current Q-network. We freeze the Q-network that estimates the future rewards.

The hyper-parameters used for the training algorithm are different in exit- and open freeway scenario and single- and multi-agent training. We will present them in section 5-1

Algorithm 4.1 Training Loop

Reset Highway
Instantiate Replay Memory: D_0
while Iteration < Training Length **do**
  Anneal ϵ towards $\epsilon_{\min}$
  for All model agents **do**
    Observe model agent state: s
    Determine greedy action: $a = \pi(s)$ (Equation 2-3)
    If lane change, put on blinkers
  end for
  Observe training agent state: s_k
  Determine ϵ-greedy action: $a_k = \pi(s_k)$ (Equation 2-8)
  Simulate highway for 2 s
  if In terminal state **then**
    Reset Highway
  else
    Get Reward of training agent: r_k
  end if
  if Previous experience was not terminal **then**
    Store previous experience in Replay-Memory D_k += $\{s_{k-1}, a_{k-1}, r_{k-1}, s_k\}$
  end if
  Sample mini-batch from Replay-Memory: $M \sim U(D_k)$
  Update Q-Network with M (Equation 2-9)
  if Iteration is a multitude of Test Iteration (k_{test}) **then**
    Simulate highway for n_{test} iterations where training agent follows a greedy-policy
    Get mean Reward of the completed test: $\bar{R}$
    if $\bar{R}$ is highest yet **then**
      Store current (best performing) DQN
    end if
  end if
  if Iteration is multitude of Target Network update (k_{update}) **then**
    Update target network: $\theta^- = \theta$
  end if
  Iteration += 1
end while

Chapter 5

Numerical experiments

In this chapter, we present some numerical experiments we performed in the Highway simulator. First, we present the training results in the four different training settings, two scenarios with two agent classes each. We then present a multi-agent simulation experiment for exit scenario training and a multi-agent simulation experiment for open freeway scenario training where all agents drive according to full exploitation policies.

We trained four deep Q-Network (DQN) agents. The basic and advanced agent in an exit scenario and the basic and advanced agent in an open freeway scenario, first in a single-agent setting, and then used the single-agent policy as an initial policy for multi-agent training. After training, we tested the resulting agents in some evaluation experiments, where the training agent also followed the greedy policy (same as model agents) instead of ϵ-greedy policy during training. Exit scenario agents are evaluated on their percentage of success to reach their destination lane for differing highway lengths. Open freeway scenario agents are trained on their lane distribution corresponding to their intended speed. We only evaluate the performance of multi-agent systems since the single-agent system is not realistic.

Both the exit scenario and open freeway scenario experiments involved zero collisions.

Table 5-1: Hyper-parameters of training processes

Hyper-Parameters	Exit scenario		Open freeway scenario	
	Single	Multi	Single	Multi
Total iterations	10^6	10^6	10^6	10^6
Learning rate	$2.5e^{-5}$	$1e^{-5}$	$2.5e^{-5}$	$1e^{-5}$
Batch size	256	256	256	256
Test iteration (k_{test})	10^4	10^4	10^4	10^4
Test length (n_{test})	10^4	10^4	10^4	10^4
discount factor (γ)	0.85	0.85	0.95	0.95
Target network update (k_{update})	$5e^4$	1	$5e^4$	1
ϵ-start	1	0.2	1	0.2
ϵ-min	0.05	0.005	0.05	0.005
Exploration fraction	0.5	0.5	0.5	0.5

5-1 Training Process

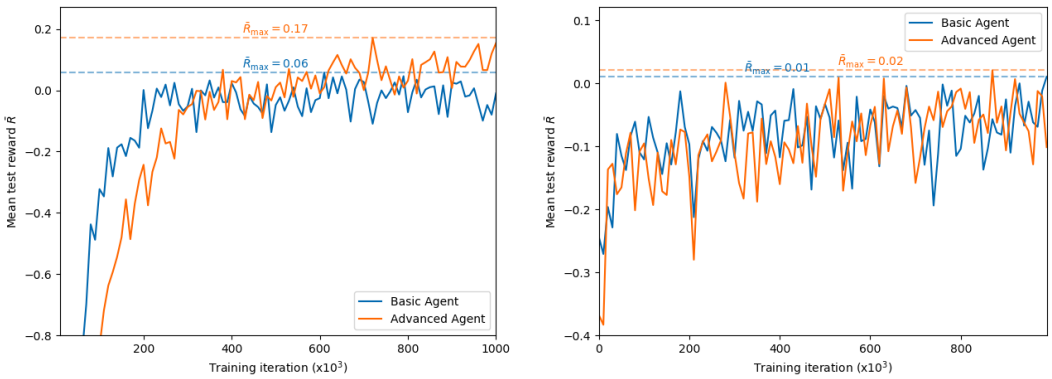
For training, we used the algorithm given in section 4-4 (Training algorithm). The hyper-parameters we used can be found in Table 5-1. Hyper-parameters were found be excessive testing and trial and error.

As stated in the previous chapter, we ran a performance simulation every $k_{\text{test}} = 10^4$ training iterations, and simulated the highway for $n_{\text{test}} = 10^4$ test iterations. In Figure 5-1 and 5-2, we plotted the mean rewards of the test simulations according to their current training iteration, in the exit and open freeway scenario, respectively. We stored the DQN that resulted in the maximal test reward ($\bar{R}_{\max}$) as the best performing one.

Although training shows an increasing performance trend, the mean rewards of performance simulations fluctuated a lot. Those large fluctuations are due to the randomness of simulations; the number of vehicles on the highway and the intended speed of the training agent are random. Even though there are many episodes in the 10^4 test iterations, it would be better to have a fixed amount of vehicles during testing, and a fixed intended speed in the exit scenario case. This way, the randomness would be lower and $\bar{R}_{\max}$ would correlate better with the best performing DQN. We did not implement this due to the time constraints of the project.

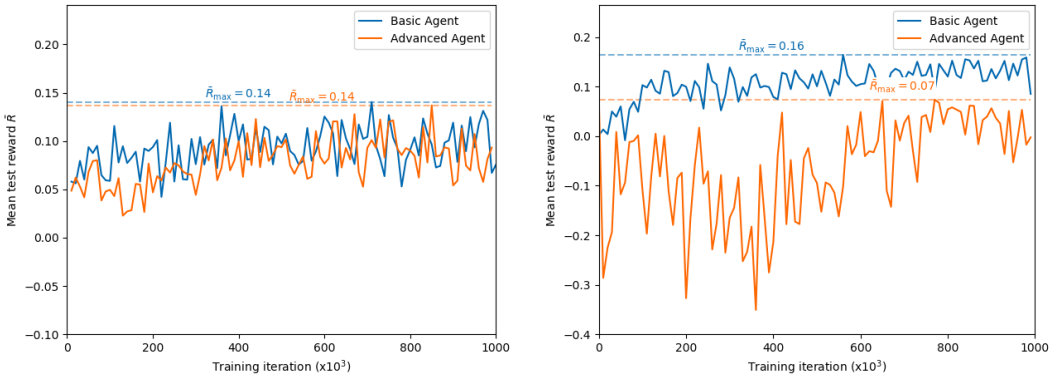
See Figure 5-1. In a single-agent exit scenario, advanced agents perform slightly better than basic agents. This result is intuitive since advanced agents can adjust their speed towards driving behind surrounding vehicles, allowing for better merging. However, the difference between performance in the multi-agent exit scenario is not significant. This result could be due to a multitude of reasons, one being that the problem is too difficult to solve with the presented Multi-Agent Deep Reinforcement Learning (MADRL) algorithm.

See Figure 5-2. In the open freeway scenario single-agent case, basic and advanced agents have similar performance. In the multi-agent case, basic agents perform better than advanced agents. One could expect a better performance of the advanced agent since it has all the options of the basic agent plus some additional ones. However, when the agents are selfish, additional actions could result in a worse performing multi-agent system. Basic agents have a more cooperative driving behavior; they do not slow down but wait till there is room to



(a) Single-agent training **(b)** Multi-agent training

Figure 5-1: Mean test rewards during training in exit scenario



(a) Single-agent training **(b)** Multi-agent training

Figure 5-2: Mean test rewards during training in open freeway scenario

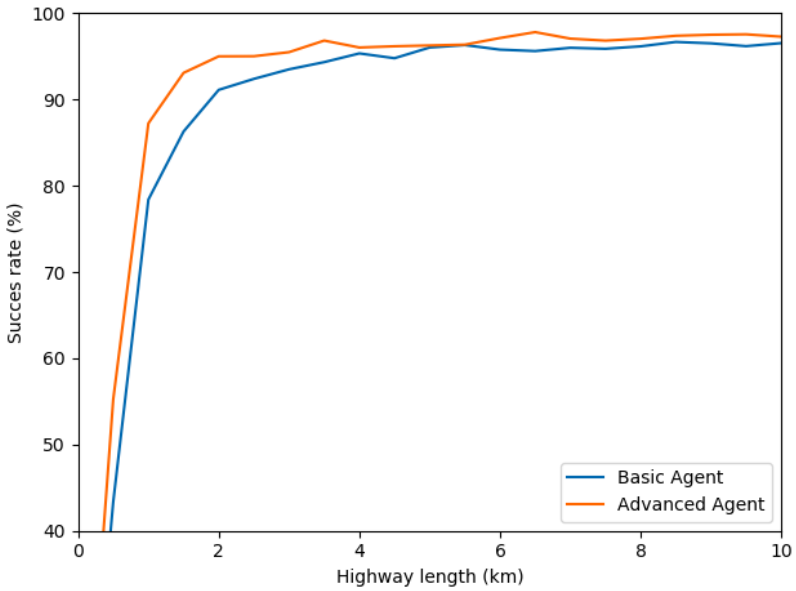
change lanes, whereas advanced agents slow down sometimes to not overtake on the right or change lanes faster, decreasing overall system performance.

5-2 Exit scenario

To evaluate the performance of the exit scenario agents, we tested the success rate of the exit scenario agents on different highway lengths. All agents are driving as model agents; they employ a greedy policy. A highway reset is done in the same way as described in section 4-1; with a different amount of agents each episode. Let us define success as an agent being on a destination lane at the moment the training agent reaches the end of the highway and failing as not reaching it or colliding. In this section, we tested the success rate (success to fail ratio) and the collision rate (collision to non-collision) over a multitude of highway lengths.

We simulated twenty-one different highway lengths. Lengths are linearly sampled from 0 to 10 km, with a difference of 500 m per sample. Each highway length was simulated for 100 episodes, i.e., the training agent reached the end of the highway 100 times for each highway

Figure 5-3: Exit scenario evaluation

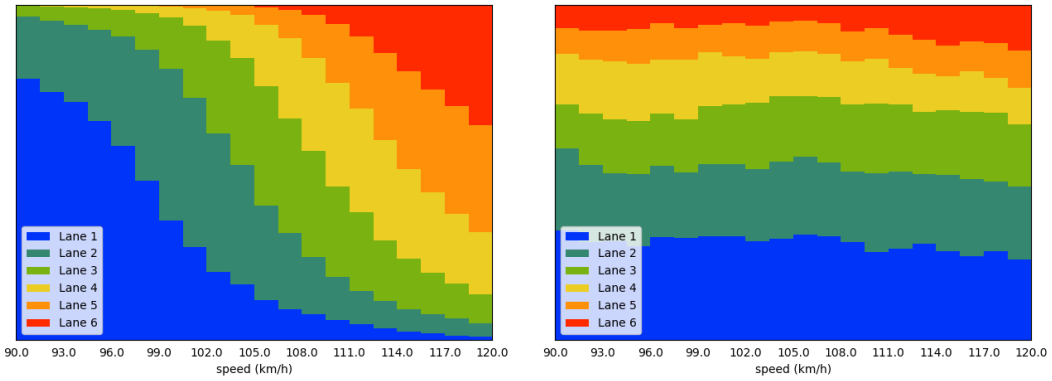


length. During each episode, we keep track of the collision amount, zero collisions occurred. At the instance the agent reaches the destination, all agents are tested if they are on one of their destination lane (success) or not (fail), resulting in a success rate per episode and an average success rate per highway length over 100 episodes, plotted in Figure 5-3.

Figure 5-3 shows a slightly higher success rate of the advanced agent compared to the basic agent for a short highway. We expected this result because of the ability of the advanced agent to adjust its speed. A notable outcome is that the advanced agent system does not converge to a 100% success rate. After ± 5 km, the advanced agent's success rate increases slower compared to the basic agent's success rate. In theory, the advanced agent should be able to achieve a 100% success rate. We can conclude that a sub-optimal solution is reached.

Nevertheless, the success rate is significantly higher than the success rate in previous literature' $\pm 80\%$, where the other 20% was collision rate [7]. They employed a similar setting but on a smaller (less complex) three-lane highway. In our simulation, no collisions occurred. However, the agent did not reach its destination within the highway length sometimes.

Since we included speed in the reward function, advanced agents might not choose to adjust their Adaptive Cruise Control (ACC) controller towards slower driving, despite it being beneficial for changing lanes earlier. A higher discount factor γ would result in a higher success rate since the agent receives a destination reward on every time-step, while safety rewards are occasional. A disadvantage of a higher γ is that safety rewards would be of less significance, and some collisions start to occur. Since safety is the number one priority, we chose to employ a slightly lower than usual discount factor $\gamma = 0.85$. Agents choose safe driving behavior above going to their destination lane and sometimes fail to reach their destination lane, even for a length of 10 km. If agents were constraint to choose only safe actions (see chapter 6), the discount factor could be much higher (e.g., $\gamma = 0.99$), and since there would not be a need for any disproportionately high collisions or danger reward in the reward function, the



(a) Basic Agent system (average speed 99.3 km/h)

(b) Advanced Agent system (average speed 95.4 km/h)

Figure 5-4: Lane Distributions of advanced agents

performance should be higher as well.

We failed to prove the benefit of using the advanced agent in an exit scenario. However, we still believe that they are better suited for this scenario compared to the basic agent. We believe that a 100% success rate is achievable when we would use another training algorithm, tuned the hyper-parameters better or trained for longer. Further research would be needed to prove this.

5-3 Open freeway scenario

For the open freeway scenario, we set the destination lanes state of the agents to all lanes on the highway. Agents are driving freely on the highway. During the evaluation, when the training agent reached the end of the highway, a highway reset was done in the same way as described in section 4-1; with a random amount of agents each episode. The highway length was constant; 10 km. Agents are grouped according to their intended speed, uniformly sampled from 90 to 120 km/h. We split the intended speeds into 20 groups of equal speed spacing. If an agent has a speed of 90-91.5 km/h it is part of group 1, and 91.5-93 km/h it is part of group 2 etc.

Each intended speed group contains multiple agents that ended up in a particular lane. At the end of each episode, we checked the intended speed of each agent and the lane it is in, and we updated the intended speed and lane distributions. After 5000 episodes, the lane amounts per group are normalized and plotted, resulting in Figure 5-4.

Faster vehicles are driving on the left side, and slower vehicles are driving on the right side. Training already showed the higher performance of basic agents. The average speed in the basic agent system is higher compared to the advanced agent system. Therefore the lane distributions of basic agent systems are more extreme.

Both lane distributions show an insightful behavior of the agents. The basic agent shows the best lane sorting behavior. Remember that the number of agents during simulation are randomized. On an empty highway, even the fastest vehicles are driving on the right side,

that is the reason some agents are ending up in lane 1 in the group with the highest intended speed. For slow vehicles, it is never optimal to drive on lane 6. It still occurred in the advanced multi-agent system.

The advanced agent has a less desirable speed distribution. This result is due to the ability of an advanced agent to slow down and not pass a slower vehicle on a left adjacent lane; there is no reason to go to the left lane. Two factors are keeping the low performance of the advanced agent in existence:

- Not sorting of the lanes by the intended speed is the reason the system has a low average speed.
- A low average speed makes the sorting of fast advanced agents by intended speed redundant since the agent can slow down to not overtake vehicles on the right side.

In the basic agent system, agents can not slow down. They need to change lanes to the left in order not to overtake. The need of changing lanes is the reason the system performance increases significantly for basic agent systems.

In the open freeway scenario, agents intended speed was sampled uniformly from 90 to 120 km/h. The maximal possible average speed of a system in the open freeway scenario would be $\frac{120-90}{2} = 105$ km/h, if all agents were driving on their maximal speed (intended speed) at all times. An average speed of 105 km/h is not achievable since vehicles on the same lane with different intended speeds would collide. The average speed of the basic agent system: 99.3 km/h, was a result above the author's expectation and could well be close to optimal. There is a clear benefit of a basic agent above an advanced agent in the open freeway scenario.

Chapter 6

Conclusion and outlook

In this chapter, we present some conclusions and possible further research examples.

6-1 Conclusions

We presented two novel deep Q-Network (DQN) agents that model autonomous highway driving vehicles, and employed non-cooperative homogeneous Multi-Agent Deep Reinforcement Learning (MADRL) to optimize their policy in two scenarios, one where agents tried to reach their destination lanes; exit scenario, and one where agents did not; open freeway scenario. Basic agents were limited to lane-changing maneuvers. Advanced agents were able to influence the Adaptive Cruise Control (ACC) controller as well, i.e., able to slow down if needed.

Agents can observe non-ego lane change intentions, and by observing vehicles on two lanes on each side, all collisions are observable and therefore preventable. Zero collisions occurred during experiments, which is the primary goal of the highway driving 'game'. We have proven the usefulness of observing lane change intent of surrounding vehicles. Moreover, agents incorporate an observation grid to observe surrounding vehicles, reducing the state size by making lane observations of the surrounding vehicles obsolete, and providing more structure to the state of the agent.

The agents' framework, i.e., the observations, actions, and reward, are (theoretically) implementable. They do not rely on explicit communication, observe all information within 150 m longitudinally and two lanes to the side laterally. Even the reward function is observable for the agent. It is only dependent on the observed states and the control actions of the agent. Finding the exact implementation methods is left for further research.

The resulting policies could be used as a policy for a vehicle that drives on real highways. Continuing the learning process on real highways will, however, lead to collisions. We have already constrained the agent not to drive off-road without use of the reward function. In the further research section (section 6-2), we propose to constrain the actions of vehicles to

only safe actions, even during exploration. Then, vehicles could employ the learning process presented in this work on real highways. A basic agent in an open freeway scenario shows the most potential in terms of efficiency improvements. In the exit scenario, deterministic control seems more fit compared to our presented learning algorithm.

6-1-1 Exit scenario

In the exit scenario, the system performed better in terms of safety and reaching the destination than the multi-agent system in [7]. The maximal success rate of the advanced agent on a six-lane highway was 98%, compared to the 80% success rate in [7] with a collision rate of 20%.

Even though we still believe that advanced agents are better suited in this scenario than a basic agent, we failed to prove this. No 100% success rate was reached, even for distances as long as 10 km. An optimal policy of the advanced agent should result in a 100% success rate.

6-1-2 Open freeway scneario

The basic agent trained in an open freeway scenario could be useful. Even though the agents are selfish, the vehicles showed a sound lane selection strategy according to their desired speeds, resulting in a well-performing six-lane multi-agent system. Lane distributions showed a similar driving behavior of the best performing three-lane system in [25]. The main advantage compared to other lane-selection literature is that this multi-agent learning algorithm is adaptable to any highway environment. Advanced agents performed worse in the open freeway scenario because they optimize a selfish reward in a multi-agent system. Slowing down sometimes has a positive effect on the individual return of the agent, but a negative effect on system-wide return.

6-2 Further research

A multitude of further research possibilities arose from this work. We present them in this section.

6-2-1 Other Deep Reinforcement Learning algorithms

The advanced agent in an exit scenario converged to a sub-optimal solution. Prioritized replay sampling [26] and dueling networks [27] could improve performance, but in a multi-agent setting where the policy entirely influences the environment, this is uncertain. We recommend to employ other, more state-of-the-art, Deep Reinforcement Learning (DRL) algorithms and investigate the performance of the resulting agents.

Multiple random seeds could also improve performance. The initialization of the DQNs weights and biases is random; the DQN is updated towards a random local optimum. This is a common problem in deep learning and solved by training multiple times and using the best performing one.

6-2-2 Cooperative reward

In this work, agents employ a selfish reward function. It would be interesting to investigate similar systems with a cooperative reward, e.g., a reward function including surrounding vehicle speeds. Moreover, when the reward is cooperative, we could include additional blinker actions to the action space of the agent. Agents currently activate their blinkers only when they execute a lane change. We propose to investigate the inclusion of separate blinker actions in a cooperative multi-agent setting. Agents can communicate their lane change intent before changing lanes. Assertiveness is then reinforced by the reward function, but only when it is also beneficial for the surrounding vehicles since the reward is cooperative.

6-2-3 Randomized spawn times

Some parts of the simulator are still unrealistic. In the presented multi-agent system, all agents initialize their actions at the same time instance. In real life, however, agents determine their actions on a random time instance. Further system simulations should be set up with random agent spawning times. Sequencing happens automatically, and it is possible to use more conventional homogeneous multi-agent deep reinforcement learning [7], without splitting model and training agents. Randomizing spawn times could even increase performance since assigning the last agent as a training agent could result in agents being too reactive. When driving in real life, assertiveness is needed. If other agents do not react to the training agent (because they already decided their current action), assertiveness is not reinforced.

6-2-4 Observation errors

Observations are currently assumed to be of zero error. In current state-of-the-art literature, lane change intentions are predicted correctly up to 93% of the time [28]. However, if the lane change is already activated, lane change intention could be observed with a higher success rate, but with a delay. In our system, lane changes are observed correctly 100% of the time, and at the moment of initialization.

Perfect tracking of the speed and location of surrounding vehicles are also unrealistic. We propose in further research, to investigate what happens when errors are included in the simulation when the agents are optimized in this setting. We recommend to use a recurrent neural network [29] to maintain high performance in simulations that include errors.

6-2-5 Safety constraints

We did not investigate deterministic safety constraints in this multi-agent highway environment. System-wide safety and liveness (the possibility to reach your destination while employing the safety constraints) could then be analyzed. In [30] for instance, safety and liveness are proven within a zero-error Model Predictive Control (MPC) setting. System-wide safety and liveness could even be investigated in a multi-agent environment that includes the above-suggested observation errors (subsection 6-2-4).

If the agents were constrained to choose only safe actions, performance could be significantly improved. If the agent could observe that a lane change is unsafe deterministically, it is not

necessary to train on these decisions. No (disproportionately high) safety penalty is needed in the reward function. Training would purely focus on optimizing the driving efficiency, whereas safety would then be 'handled' with deterministic control. Constraining actions to be safe in a deterministic way, and applying DRL to optimize efficiency in the safe action domain, is our primary recommendation for future applications of Deep Reinforcement Learning on Automated Highway Driving. Also, non-ego lane change intentions should be observed by the agent.

Bibliography

[1] B. Schoettle and M. Sivak, "A survey of public opinion about autonomous and self-driving vehicles in the us, the uk, and australia," 2014.

[2] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167 – 181, 2015.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[4] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[5] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.

[6] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[7] R. Radulescu, M. Legrand, K. Efthymiadis, D. M. Roijers, and A. Nowé, "Deep multi-agent reinforcement learning in a homogeneous open population," 2018.

[8] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[9] C.-J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," *arXiv preprint arXiv:1803.10056*, 2018.

[10] S. David, "Advanced topics: Reinforcement learning." University Lecture, 2015.

[11] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[12] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[14] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems, Springer, May 2016.

[15] B. Leibe, K. Schindler, N. Cornelis, and L. Van Gool, "Coupled object detection and tracking from static cameras and moving vehicles," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 10, pp. 1683–1698, 2008.

[16] C.-C. Wang, C. Thorpe, and A. Suppe, "Ladar-based detection and tracking of moving objects from a ground vehicle at high speeds," in *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No. 03TH8683)*, pp. 416–421, IEEE, 2003.

[17] R. W. Wolcott and R. M. Eustice, "Visual localization within lidar maps for automated urban driving," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 176–183, IEEE, 2014.

[18] J. Schlechtriemen, A. Wedel, J. Hillenbrand, G. Breuel, and K.-D. Kuhnert, "A lane change detection approach using feature ranking with maximized predictive power," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 108–114, IEEE, 2014.

[19] D. D. Salvucci, "Inferring driver intent: A case study in lane-change detection," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 48, pp. 2228–2231, SAGE Publications Sage CA: Los Angeles, CA, 2004.

[20] S. Patel, B. Griffin, K. Kusano, and J. J. Corso, "Predicting future lane changes of other highway vehicles using rnn-based deep models," *arXiv preprint arXiv:1801.04340*, 2018.

[21] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1094–1099, IEEE, 2015.

[22] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on control systems technology*, vol. 15, no. 3, pp. 566–580, 2007.

[23] B. Van Arem, C. J. Van Driel, and R. Visser, "The impact of cooperative adaptive cruise control on traffic-flow characteristics," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 4, pp. 429–436, 2006.

[24] T. Nagatani, "Kinetic segregation in a multilane highway traffic flow," *Physica A: Statistical Mechanics and its Applications*, vol. 237, no. 1-2, pp. 67–74, 1997.

[25] D. E. Moriarty and P. Langley, "Learning cooperative lane selection strategies for highways," *AAAI/IAAI*, vol. 1998, pp. 684–691, 1998.

[26] J. Zhai, Q. Liu, Z. Zhang, S. Zhong, H. Zhu, P. Zhang, and C. Sun, "Deep q-learning with prioritized sampling," in *International Conference on Neural Information Processing*, pp. 13–22, Springer, 2016.

[27] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[28] K. Li, X. Wang, Y. Xu, and J. Wang, "Lane changing intention recognition based on speech recognition models," *Transportation research part C: emerging technologies*, vol. 69, pp. 497–514, 2016.

[29] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.

[30] K.-D. Kim and P. R. Kumar, "An mpc-based approach to provable system-wide safety and liveness of autonomous ground traffic," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3341–3356, 2014.

Glossary

List of Acronyms

MIXIC	microscopic-traffic simulation model
DCSC	Delft Center for Systems and Control
TU Delft	Delft University of Technology
AI	Artificial Intelligence
RL	Reinforcement Learning
MPC	Model Predictive Control
TIV	Time In-between Vehicles
faSDVs	fully autonomous Self Driving Vehicles
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
DQN	deep Q-Network
DRL	Deep Reinforcement Learning
ReLU	rectified linear units
AD	Autonomous Driving
MADRL	Multi-Agent Deep Reinforcement Learning
ACC	Adaptive Cruise Control
Dec-POMDP	Decentralized Partially Observable Markov Decision Process

List of Symbols

α	Learning rate
ϵ	Exporation fraction
ϵ_{min}	Minimal (end value) exploration fraction
ϵ_{start}	Starting exploration fraction
γ	Discount factor
ω	Tuning paramter of reward function
π	Policy: mapping of state to action of the agent
θ	Weights and biases of Deep Q-Network
θ^-	Weights and biases of target network
$\bar{R}$	Mean performance simulation reward
$\bar{v}$	Speed of target vehicle
$\bar{x}$	Longitudinal location of target vehicle
$\mathbb{A}$	Set of joint action spaces
$\mathbb{M}$	Partially Observable Markov Decision Process
$\mathbb{S}$	Set of joint (observation) state spaces
$\mathscr{M}$	Decentralized Partially Observable Markov Decision Process
A	Finite set of actions of an agent
a	Action of an agent
D	Finite set of agents
D_t	Replay Memory: history of all experiences
e	Experience: tuple of state, action, reward and next state
G_t	Return: discounted expected cumulative future reward
H	A set of states of the highway
h	State of the highway
h'	Next state of the highway
k_{test}	Performance simulation iteration
k_{update}	Target network update iteration
L	Loss function
L_w	Lane width
M	Mini-batch of randomly sampled experiences from replay memory
n_{test}	Amount of iterations per performance simulation
Q	Q-value: expected return given the current state and action
R	Reward function
r	Feedback reward (scalar value)
S	Set of (observed) states of an agent
s	(Observed) state of an agent
s'	Next state of the agent
T	Transition probability function

v	Speed of a vehicle
V_i	Observed vehicle i in the observation grid
v_{int}	Intended speed of a vehicle
x	Longitudinal location of the agent
x_{start}	Longitudinal position at the start of a lane change
y	Lateral location of the agent
y_{start}	Lateral position at the start of a lane change
Z	Observation probability function