KU LEUVEN

Robust Deep Reinforcement Learning for Autonomous Vehicles in Dense Highway Traffic

Exploration of neural network architectures to deal with dynamically sized and temporal sensor-based input data

Victor Leon Van Wymeersch

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisor:
Prof. Dr. Johan Suykens

Assessor:
Prof. Dr. Chris Tampère
Edward De Brouwer

Mentor:
Bram De Cooman

Academic year 2020 – 2021

Preface

I would like to thank my thesis promotor Prof. Johan Suykens, for providing me with the opportunity to research this thesis topic. I would also like to thank Bram De Cooman for mentoring and guiding me throughout the year, all your technical support was invaluable. Thank you to my friends, family, and specifically Ines for your endless encouragement. Without it, I would have not been able to manage working and studying in the midst of Covid-19. Finally, a big thank you to my colleagues at Capgemini Engineering and Toyota Motors Europe for enabling me to take the time to do my Masters in Artificial Intelligence and as a result, this thesis.

Victor Leon Van Wymeersch

Contents

Abstract

Autonomous driving is a paramount application of artificial intelligence, and more specifically deep learning. Through various architectures, the push towards fully autonomous driving has been furthered, enabling autonomous vehicles to learn how to drive directly from sensor measurements alone.

However, despite numerous successes, the application of deep-learning techniques for autonomous driving applications still faces several challenges. The two most pertinent of these challenges include the vast amounts of both computing power and data that are required to train deep learning models, and the safety-concerns that arise from the black-box nature of deep learning's neural networks.

One somewhat unexplored deep learning technology with regards to autonomous driving is reinforcement learning. Reinforcement-learning addresses the problem of data requirements that are needed for training deep learning models, by learning through trial and error in a simulation setting. Despite several pitfalls of reinforcement learning, namely it's lack of generalisability to real-world settings and safety-concerns, reinforcement learning warrants further exploration in the context of autonomous driving.

This paper therefore seeks to further investigate the applicability of reinforcement learning techniques to learn high-level discrete driving policies for autonomous driving applications in highway scenarios with dense traffic, using several neural network architectures from outside the domain of autonomous driving. These architectures include deep sets, and Convolutional Neural Networks, with and without pooling functions and LSTMs. In theory, the added benefit of several of these architectures includes both permutation invariance and ability to capture temporal dependencies between environment states.

This work shows that these networks enable better generalisability, robustness to sensor-based input noise, and ability to learn in dynamic environments that more closely match that of real-world autonomous driving applications. These contributions serve to help bridge the gap between simulation and reality, bringing reinforcement learning closer to being practically applicable.

List of Figures

List of Tables

List of Abbreviations

Abbreviations

USD	United States Dollar
ADAS	Advanced Driver-Assistance Systems
ML	Machine Learning
NN	Neural Network
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
ReLU	Rectified Linear Unit
Tanh	Hyperbolic Tangent
RL	Reinforcement Learning
MDP	Markov Decision Process
POMDP	Partially Observable Markov Decision Process
DDPG	Deep Deterministic Policy Gradient
SAC	Soft Actor-Critic
A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
TRPO	Trust Region Policy Optimisation
DQN	Deep Q-Network
RNN	Recurrent Neural Networks
IDM	Intelligent Driver Model
MOBIL	Minimizing Overall Braking Induced by Lane Changes
KBM	Kinematic Bicycle Model
ODE	Ordinary Differential Equation
PID	Proportional Integral Derivative
FOV	Field Of View
DDQN	Double Deep Q-Network
D3QN	Duelling Double Deep Q-Network
PER	Prioritised Experience Replay
TD	Temporal Difference

Chapter 1

Introduction

Autonomous driving is one of biggest challenges of the 21st Century. Successfully implementing autonomous technologies at scale is of utmost importance because it has the potential to drastically reduce the death tolls associated with driving. 1.35 million people are killed in car accidents every year around the world, being the leading cause for death for young people between 5-29 years of age [1]. Automobile accidents are estimated to cost the world economy approximately $1.8 trillion (in 2010 USD) between the years 2015 and 2030 [2]. The deaths caused by speeding, distracted driving, and from drivers falling asleep at the wheel, amongst other factors, can all be mitigated when the capability and quality of driving by autonomous driving technologies supersedes that of humans. However, before the benefits of autonomous driving can be realised, ethical, legal, security and safety concerns need to be addressed, and the technology needs to be improved.

Serious research efforts have been made in the field, which has led to remarkable achievements by various companies and research institutions. While cars capable of semi-autonomous driving are already available to be bought, these vehicles are still far from being fully autonomous. The adoption of autonomous vehicle technology and its commercialisation has been impeded by various factors such as cost, available sensor technologies, safety concerns, legislation, or general trust.

Artificial Intelligence, or more specifically, *deep learning* [3] has been one of the most promising technologies for the development of fully autonomous vehicles, enabling them to learn how to drive directly from sensor measurements. Similar ground breaking results have been achieved across various domains such as natural language processing [4, 5], machine translation [6, 7], and image classification [8, 9]. Despite the progress made already, deep learning techniques for autonomous driving still face several challenges. The training of these neural network models is computationally intensive and requires immense amounts of real-world driving data. Perhaps the most pertinent concern limiting the adoption of the technology is safety-related [10]. Deep learning's neural networks are uninterpretable *black-box* models, where it is impossible to trace back exactly why a decision was made under certain circumstances. This hinders the evaluation of the intrinsic safety of these systems, especially in autonomous driving where unsafe decisions may cost lives.

Similar to deep learning technologies, *Reinforcement Learning* (RL) [11] has in recent years demonstrated unprecedented results in a wide variety of challenging tasks and various fields of research. RL has exceeded expert-level human performance across numerous video

games, from arcade games such as those from Atari [12, 13], to fast-paced first-person shooters like Doom [14] directly from pixel values. Furthermore, RL has demonstrated the ability to learn cooperative team play and defeat the world champions in Dota2 [15]. It has even been able to beat the world's best Go players by training a model entirely through self-play [16]. Perhaps more impressively, RL has achieved these feats without the need for large amounts of real world training data, by instead learning through trial and error in a simulation setting where data is more readily available and easily collected. These achievements and the advantages RL has over deep learning encourages the investigation into how it may benefit the current state-of-the-art in autonomous driving.

Unfortunately, RL also has disadvantages impeding its applicability in autonomous driving related to safety (the same as deep learning methods), and the lack of generalisability of these models to real-world settings. Various research efforts have been made into creating safe RL algorithms specifically tailored to autonomous driving [17], but the lack of generalisability is often ignored. Simulation environments are often not realistic enough to train robust RL driving policies, resulting in driving policies that perform well during training and fail to generalise to real-world applications.

Most real-world environments where RL-based control systems could make a difference, like autonomous driving, are partially observable systems [18]. In these systems, measurements almost always contain noise resulting in observations which do not truly reflect the true state of the system. States may be non-stationary, reflecting the degradation of electro-mechanical components found in autonomous vehicles, or contain naturally occurring elements of stochasticity. Current literature often ignores these complexities in simulation environments and fail to explicitly test the generalisability and robustness of their methods to noise, and complex dynamic environments with elements of stochasticity. Simple neural network architectures are shown to perform well in simple simulation settings, uncharacteristic of real world driving settings.

Several neural network architectures from outside the field of autonomous driving have the ability to approximate real world systems better than those commonly employed in literature. Neural network architectures exist that can process dynamically sized input spaces. These include *Deep Sets* [19], *attention-based Long Short Term Memory* (LSTM) networks [20], and *Convolutional Neural Networks* (CNN) with pooling functions [21], which have the ability to learn from a permutation invariant input space. This permutation invariance has been shown to enable the networks to generalise to better new scenarios than otherwise possible by architectures dealing with fixed input sizes with fully-connected networks.

Additionally, neural network architectures such as LSTMs and CNNs, which are able to capture the temporal dependencies between environment states and learn in more complex, partially observable environments, are often neglected. CNNs capture temporal dependencies between states by performing convolutions across the time domain while LSTM networks encode the evolution of the states in a form of internal memory and can use this memory to make long-term planning decisions, vital to dynamic real-world autonomous driving settings. Despite the benefits recurrent and convolutional neural networks may provide to learn in realistic, dynamic, complex, and noisy environments, and enable long-term planning, they have not been investigated for the learning of highway driving scenarios among other vehicles.

These aforementioned neural network architectures, in conjunction with RL, could

help to address several of the shortcomings associated with current autonomous driving technologies, and therefore are the focus of this thesis.

The remainder of the introduction is organised as follows. Section 1.1 provides a high level overview of the prerequisite knowledge for the rest of the paper. It firstly covers the field of autonomous driving in Section 1.1.1. Thereafter, the basics and terminology of deep learning and reinforcement learning are provided in Sections 1.1.2 and 1.1.3, respectively. Section 1.1.4 covers the specific literature related to deep RL for autonomous driving, and offers a more detailed overview the neural network architectures that are investigated in this paper. Section 1.2 concretely outlines the research objectives of this work along with the main contributions made to RL in autonomous driving. It also covers the concrete contributions made through this thesis.

1.1 Background

This section further elaborates on the points outlined in the introduction and the benefits that deep reinforcement learning, with powerful neural network architectures, could have on autonomous driving. It covers the relevant literature related to RL and autonomous driving and introduces the prerequisite knowledge for the remainder of this thesis.

1.1.1 Autonomous driving

SAE International, a global leader in the mobility industry, recognises six different levels of driving autonomy in its 2014 standard (*J3016*) on the classification of autonomous driving systems. The categorisations range from level 0, where the vehicle provides the driver with simple warnings, to level 5, where the vehicle is fully autonomous and the driver is unable to take manual control of the vehicle even if desired. The first three levels fall into the category of Advanced Driver-Assistance Systems (ADAS). In these levels the systems are semi-autonomous and have the ability to take full control of the vehicle in certain scenarios, but the driver can always intervene and is mostly in control. The last three levels are where the majority of driving is done by the vehicle's autonomous systems. Current regulations permit ADAS-enabled vehicles that can achieve level 2 autonomy, and are commonly found in high-end vehicles at most major manufacturers such as Toyota, BMW, Volvo etc.

From a purely technological point of view, however, numerous companies (Tesla, Waymo, Zoox, etc.) have achieved higher levels of autonomy than presently permitted by regulation. Tesla's Autopilot for example has demonstrated the ability to travel long distances without any driver assistance required, and companies like Waymo have already deployed small-scale fully-autonomous taxi services [22]. These types of autonomous vehicles utilise various different technologies, from traditional control systems, to more complex Machine Learning (ML) methods. The most promising technique for *end-to-end autonomous driving*, that is taking raw sensory data as input and outputting driving actions, has so far been deep learning. Deep learning is a supervised ML technique that learns to take optimal driving actions based on various sensor inputs, e.g. measured by the vehicle's cameras, lidar, sonar, etc.

Deep learning for autonomous driving faces several challenges. Firstly, the supervised learning setting requires large amounts of real-world driving data from people in a variety

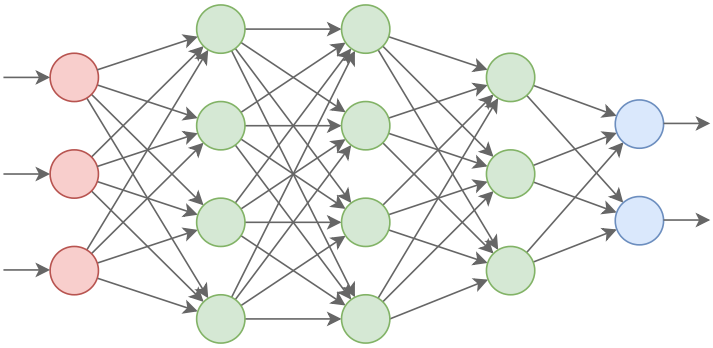


Figure 1.1: Deep neural network consisting of an an input layer (red), several hidden fully-connected layers (green), and an output layer (blue).

of different scenarios. Moreover, in order to capture all this data and train the models, enormous amounts of computational power are needed. Finally, despite the promising results of deep learning, adoption is still limited by safety-related regulations. Deep learning models are black-box models that suffer from a lack of interpretability. In other words, it is impossible to trace back exactly why a decision was made under the presented circumstances. This makes the evaluation of the intrinsic safety of the learned models difficult, especially in driving scenarios where unsafe decisions may cost lives.

1.1.2 Deep learning

Deep learning makes use of artificial neural networks that have been around for decades [23]. These networks consist of multiple successive layers connected to each other, thus making it "deep". The network depth allows them to learn complex feature representations and handle instabilities during training due to covariate shifts [24], vanishing gradients [25, 26], and difficulties during optimisation [27]. Since then, several of the initial problems have been solved and novel network architectures have emerged like *long short-term memory* (LSTM), [25] and *convolutional neural networks* (CNNs) [28]. These architectures help advance the field of deep learning by allowing to capture the temporal information needed to make long-term planning decisions, and deal with higher dimensional input spaces.

Neural network models usually consist of an input layer, a number of hidden layers, and an output layer as shown in Figure 1.1. Fully-connected layers of neurons consists of N number of input units that are all connected to M number of output units:

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}), \tag{1.1}$$

where $\boldsymbol{W} \in \mathbb{R}^{M \times N}$ and $\boldsymbol{b} \in \mathbb{R}^{M}$ denoting the weights and biases of the network. The input and output vectors are denoted as $\boldsymbol{x} \in \mathbb{R}^{N}$ and $\boldsymbol{h} \in \mathbb{R}^{M}$, respectively. The equation (1.1) essentially describes the transformation input and output of the network, where $\boldsymbol{W}$ and $\boldsymbol{b}$ are the learnable parameters responsible for affine transformation.

To increase the models expressiveness, a non-linear differentiable function $f(\cdot)$ is introduced. This function, or *activation function*, can have many forms, of which the most common are the rectified linear unit (relu), softmax, and the hyperbolic tangent (tanh), as

shown below:

$$f(x_i) = \begin{cases} x_i & \text{if } x_i \geq 0, \\ 0 & \text{otherwise} \end{cases}, \quad f(\boldsymbol{x}) = \frac{\exp \boldsymbol{x}}{\Sigma_i \exp\ x_i}, \quad f(x_i) = \tanh(x_i). \tag{1.2}$$

Training of these networks is achieved by the minimisation of a *loss* or *objective function*. Loss functions are chosen based on the learning problem. Regression-based problems frequently employ a mean-squared error loss function between the ground truth training data ($\boldsymbol{y}$) and the model's prediction ($\hat{\boldsymbol{y}}$):

$$\mathcal{L} = (\boldsymbol{y} - \hat{\boldsymbol{y}})^2. \tag{1.3}$$

This minimisation is generally done by differentiating the loss function with respect to the learnable parameters of the model $\boldsymbol{\theta}$, to obtain the gradient of the network $\boldsymbol{\nabla_\theta \mathcal{L}}$. The gradient points in the direction of the largest increase in the loss function, so during optimisation steps are taken in the opposite direction to minimise the loss function, or the error on the network's prediction:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla_{\boldsymbol{\theta}_k} \mathcal{L}, \tag{1.4}$$

where α denotes the learning rate, and $\boldsymbol{\theta_k}$ the network's parameters at the k^{th} iteration during optimisation. Computation of the gradient is done using reverse mode algorithmic differentiation, also known as back-propagation [29].

Practically speaking, in current state-of-the-art autonomous driving deep learning models the data used for training corresponds to: the ground truth actions taken by human driver on the road ($\boldsymbol{y}$) (e.g. steering angle or pedal position), and the input measurement data ($\boldsymbol{x}$) (e.g. camera images). By collecting enough of this data a large deep neural network can be trained using the algorithm described above to learn how to independently control the vehicle from the measurement data.

1.1.3 Deep reinforcement learning

The success of RL algorithms outlined in the introduction is partially thanks to the aforementioned deep neural networks. These networks have the ability to learn over different environment, enabling RL to go beyond simple game settings to complex control problems that are a closer reflection of those in autonomous driving. RL policies have managed complex continuous control problems involving locomotion and manipulation [30, 31, 32], and have even scaled to robots in complex real-world manipulation tasks like solving Rubik's cubes, putting clothes on hangers, opening bottles etc. [33, 34, 35]. These remarkable achievements of RL encourage further investigation of these algorithms into how they could benefit the current state-of-the-art in autonomous vehicles.

In contrast to the supervised deep learning strategies employed in autonomous vehicles, RL alleviates the need for massive amounts of labelled (human) data by instead learning through trial and error. Fundamentally RL consists of two main entities: an agent and environment with which the agent interacts. Based on the actions of the agent, the environment rewards or punishes the agent for the effect it has on the environment's state. The agent's ultimate goal is to learn what actions maximise its long-term cumulative reward.

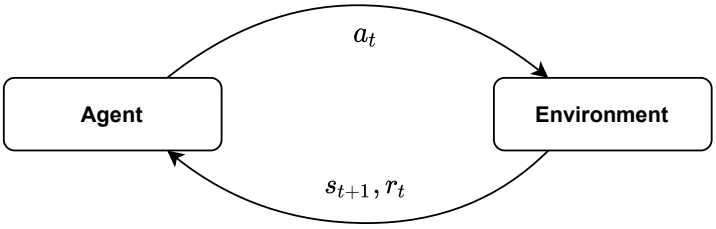


Figure 1.2: The fundamental interaction loop in reinforcement learning between the agent and the environment. The agent decides how to interact with the environment by choosing an action $\boldsymbol{a}_t$. In turn, the environment exhibits a new state $\boldsymbol{s}_{t+1}$ and provides a reward to the agent for taking the action $r_t := r(\boldsymbol{s}_t, \boldsymbol{a}_t)$, given the initial state $\boldsymbol{s}_t$. This interaction loop continues for all time steps during training.

Within autonomous driving the agent is also commonly referred to as the ego-vehicle, i.e. the vehicle tasked with learning how to drive autonomously. This process is called the fundamental interaction loop in RL as is depicted in Figure 1.2

Mathematically this interaction between the agent and the environment is described as a Markov Decision Process (MDP) which is defined as $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho_0, r)$ with $\mathcal{S}$ being the state space and $\mathcal{A}$ the action space. The reward at time step t is described by the reward function $r_t := r(\boldsymbol{s}_t, \boldsymbol{a}_t)$ given the current state $\boldsymbol{s}_t$ and action choice $\boldsymbol{a}_t$ pair. The probability distribution over the initial states is given by $\rho_0 : \mathcal{S} \times \mathcal{A} \in [0, 1]$. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \in [0, 1]$ denotes the state's transitional probability distribution, that is, given that we are in state $\boldsymbol{s}_t$ and take action $\boldsymbol{a}_t$, how likely we are to end up at state $\boldsymbol{s}_{t+1}$. Here, the Markov property holds that the next state depends only on the current state and action, and not any of the previous states.

Such Markov processes can either be fully or partially observable. When environmental states are not fully observable or do not accurately encode the information of previous states, then the MDP is transformed into a Partially Observable MDP (POMDP). This occurs when for example, a state contains measurement noise and the observed state does not completely and accurately reflect the true state of the environment.

The general goal of the agent is to find a policy π (represented by a deep neural network's learnable parameters) that maximises the expected discounted return:

$$J(\pi) = \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{1.5}$$

The expected discounted return $J(\pi)$ is the summation of the rewards collected over a trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \cdots)$, where $\boldsymbol{s}_0 \sim \rho_0$, $\boldsymbol{a}_t \sim \pi(\cdot|\boldsymbol{s}_t)$, and $\boldsymbol{s}_{t+1} \sim \mathcal{P}(\cdot|\boldsymbol{s}_t, \boldsymbol{a}_t)$. The discount factor $\gamma \in (0, 1)$ is responsible for ensuring that the summation converges to a finite value, and mathematically models the idea that rewards obtained recently are more valuable than rewards obtained in the distant future. From equation 1.5 it follows then that the optimal policy π^* is given by:

$$\pi^* = \arg \max_{\pi} J(\pi). \tag{1.6}$$

Exploration versus exploitation

The concept of exploration versus exploitation is essentially about the balance between how long agents should spend exploring their environments before they have gained sufficient knowledge to determine how to act to maximise their overall reward. Since agents initially do not have any information on the probability distributions $\rho_0(\boldsymbol{s}_0)$ and $\mathcal{P}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, they are required to explore the environment by taking different actions to find those that possibly lead to rewards. Once the environment has been sufficiently explored, the agent needs to exploit the environment and take actions it knows leads to the highest rewards. This process is analogous to nature's learning process, where animals typically explore their environments during adolescence, to learn the most appropriate actions given certain scenarios to maximise their rewards or overall success at survival. Efficient exploration of an environment's state space can drastically increase an agent's overall performance, but because environment state spaces may be extremely large it is often not possible to fully explore them and thus, a fine balance should be struck between exploration and exploitation for RL agents to learn effectively. Depending on whether agents select actions deterministically or stochastically, two main methods exist for balancing exploration and exploitation:

- *ϵ-greedy deterministic policies:* Agents select a random action with a probability of ϵ and otherwise select a greedy action. Concretely, ϵ-greedy exploration is formulated as follows:

$$\pi(a_i|\boldsymbol{s}) = \begin{cases} 1-\epsilon & \text{if } a_i = \pi(s), \\ \epsilon & \text{otherwise.} \end{cases} \tag{1.7}$$

  The optimal, or greedy, action $a_i = \pi(\boldsymbol{s})$ according to the policy π is selected with the probability $1-\epsilon$. By varying the ϵ throughout training from $\epsilon = 1$, (at the beginning of training) to $\epsilon = 0$ (at the end of training) by some function $\epsilon = f(k)$, depending on the time step (k) during training, we can manage this exploration versus exploitation balance. ϵ-greedy exploration is commonly used for deterministic policies.

- *Stochastic policies*: Stochastic policies on the other hand do not require such techniques because actions are selected from a probability distribution which in itself manages the exploration exploitation balance.

Model-free versus model-based algorithms

An important distinction in the types of RL algorithms is whether they are model-free or model-based. Within model-free algorithms additional distinctions can be made between policy optimisation methods, Q-learning methods, and actor-critic methods that combine the two.

- *Model-based* algorithms have access to or directly learn the mathematical model of their environments. This task is inherently difficult and complex as a ground-truth model of the environment is usually not available to the agent, but has the benefit that once a model has been learned an agent is able to plan actions by thinking

ahead. Although many model-based techniques do exist, model-free RL algorithms are more commonly utilised at the time of writing [36, 37, 38, 39, 40].

- *Model-free* algorithms neither have access to, nor do they attempt to learn a model of the environment. Instead, their objectives are either to learn policies (stochastic or deterministic) in the case of *policy optimisation* algorithms, or the values of actions given certain states (Q-values) for Q-learning methods. While this is the main classification of model-free algorithms, it should be noted that several algorithms exist that blur the line between model-based and model-free algorithms by combining different parts of each method, such as *Deep Deterministic Policy Gradient* (DDPG) [30], and *Soft Actor-Critic* (SAC) [41] methods.

  Policy optimisation methods aim to either learn a policy by directly optimising their parameters π by gradient ascent on the objective function $J(\pi_\theta)$, or indirectly by maximising local approximations of $J(\pi_\theta)$. This optimisation is normally done **on-policy**, meaning that network updates are done using experiences collected while acting according to the current network parameters. Policy optimisation algorithms tend to be stable and reliable because they directly optimise the policy while learning. Common methods include *REINFORCE* [42], *Advantage Actor-Critic methods* (A2C and A3C) [43], *Trust Region Policy Optimisation* (TRPO) [44] and *Proximal Policy Optimisation* (PPO) [31].

  Q-learning methods, on the other hand, instead try to learn an approximation $Q_\theta(s, a)$ for the optimal action-value function, $Q^*_\theta(s, a)$. Typically, Q-learning methods use the Bellman equation as an objective function, of which more details are provided in Chapter 3. Optimisations are usually performed **off-policy**, meaning that updates to the network can be made using any of the trajectories time stamps during training, regardless of whether or not a training sample was collected at a time when the network was in a different state. Actions are taken by sampling those that yield the highest Q-values:

  $$a(s) = \arg\max_a Q_\theta(s, a). \tag{1.8}$$

  Q-learning algorithms only indirectly optimise an agent's performance by finding appropriate Q-values for state action pairs, and have many failure modes often making them suffer from training instabilities. These instabilities arise when nonlinear function approximators, like deep neural networks, are used to represent the action-value (Q) functions [45]. The causes of these instabilities and their remedies are covered in more detail in Section 3.1. Common value-based methods include the famous Deep Q-network (DQN) algorithm [13], amongst others, which have made improvements to the original DQN method [46, 47, 48]. Some of these improvements are further investigated in Chapter 3 and used as a comparative baseline for all experiments in this thesis.

1.1.4 Deep reinforcement learning applied to autonomous driving

Now that the foundations of the two main elements of deep RL have been laid out, the various different approaches that have been taken with respect to applying RL to autonomous driving are now introduced. Each of these approaches vary by the driving

scenario being trained under, the types of neural networks used, how measurements are presented to the agent for learning i.e. the states, the RL algorithm used, etc.

The formulation of the action space of the network is one of the distinctions for which RL algorithm is appropriate for the training setting. The most common setting is learning high-level discrete actions to control the vehicle, for which value-based techniques like DQN are most commonly used [49, 50, 51, 52, 53]. The algorithms used for learning continuous action spaces are gradient-based methods like DDPG [54, 55, 56], TD3, SAC [57], and PPO [58]. However, these lines are not always clear cut, as different scenarios often influence the action space, depending on whether the agent is driving in highway traffic [59], performing on-ramp lane merging [60], racing [61], etc.

The representation of the environment's state space is the final differentiation in approaches taken to learn autonomous driving policies. End-to-end methods are trained directly from high-dimensional sensor data, like pixel values [61, 62], however, they suffer from the curse of dimensionality, making training very difficult. These difficulties can be alleviated by using low-dimensional state representations. Here, reduced state spaces are created using affordance indicators [63] or occupancy grids [64, 65]. Reduced state spaces more compactly present the environment around the ego vehicle. Agents are given measurements from a single time instance in the form distances and relative velocities to other surrounding vehicles. The representation simplifies the learning process and has been the most popular approach in RL for autonomous driving [66, 67, 68].

Efficiently learning from dynamically sized state spaces

Affordance indicator state representations are powerful, but since neural network input sizes are always constant they may carry some inefficiencies. The number of vehicles around the ego-vehicle is dynamic throughout training resulting in a variable number of measurement available as inputs to the network. A simple remedy is to pad the missing measurements with zeros to maintain the input's size. At present, the majority of research for RL in autonomous driving ignores the inefficiencies associated with learning from this state representation when using fully connected neural networks. Network architectures, like CNNs and LSTMs from the fields of image and language processing are often ignored in autonomous driving despite the benefits they provide.

Deep Sets [19] and LSTMs with attention mechanisms [20] on the other hand are able handle variably sized network inputs. In other words, they enable the input to be permutation invariant with respect to its number of elements. Deep sets and these LSTMs achieve their permutation invariance through pooling operations. Pooling functions can be simple summations like for Deep Sets, or in the case of LSTMs are performed through the aggregation of the network's hidden states (the network's memory). Both methods essentially create new internal representations of the inputs, invariant to the number of elements the state may contain. The decisions made in autonomous driving come as a direct result of the surrounding vehicles, and effectively process the state in a permutation invariant manner, meaning that decisions are made only on the relevant components in the state matrix, and not on missing measurement data.

Instead of providing a dynamically sized input containing only non-zero (non-padded) measurement information, an alternative approach is to retain a fixed input size with padded measurements, and employ network architectures like CNNs. CNNs are specifically

designed to extract the valuable features from large, sparse input spaces. They achieve this by performing convolutions over sections of the input state to recreate it with a different dimensionality, containing only the state's important features. In this way, even when there are few surrounding vehicles, the network will focus on the relevant measurements, and not process every element of the input equally as would fully-connected networks. CNNs additionally have the possibility to incorporate pooling operations within the network to obtain permutation invariance with respect to the ordering and number of vehicles in the states.

Deep sets have been shown to outperform both Set2set LSTM networks with attention mechanisms and 2D CNNs, and provide better generalisation to new scenarios [69]. While the CNN architecture performing 2D convolutions over sets of vehicle measurements performed comparably to the Deep Sets in training, it failed to generalise well. The aforementioned research paper made a comparison to a 2D all-convolutional network [70]. However, by using 1D convolutions over each individual vehicle's measurements it is possible to create a network that mathematically works in the same way as for Deep Sets, and incorporate permutation invariance into the network with a pooling layer. While the commonly used max pooling layers of CNN architectures used in vision applications would cause information loss in such a compact state representation, average, or summation pooling operations would have the effect of rendering the input permutation invariant [21]. Incorporating this permutation invariance within a CNN architecture may result in improved generalisability equivalent to that of Deep Sets. It is also uncertain as to whether performing convolutions over different dimensions in the state yields better overall driving performance. The majority of other research involving CNNs has been aimed at learning end-to-end driving policies directly from camera inputs [71, 62, 50]. Chapter 4 continues the comparison between Deep Sets and various forms of CNNs, to evaluate the effectiveness of these architectures at learning robust, generalisable autonomous driving policies.

Learning temporal relationships between measured states

States used for learning autonomous driving policies normally come from single measurement instances. They do not contain any historical information about how the state may have evolved over time. The vast majority of research in RL for autonomous driving using affordance indicators as cited above, represent vehicles as a single set of measurements containing their current relative positions and velocities. By only using states from single time instances, it is implicitly assumed that only the current measurements are important for making driving decisions as they fully describe the environment under the Markovian assumption. In other words, temporal dependencies that may exist between states, such as the acceleration and jerk (the rate of change of acceleration) profiles of other vehicles, do not contribute to the learning process. In quasi-static driving scenarios, where vehicle dynamics are negligible, and in simulation environments with perfect noiseless state representations it is fair to assume that the environment is a fully observable MDP, even when states contain only the relative positions and velocities of other vehicles. Otherwise, such as in real-world driving scenarios, the learning process is better approximated through a POMDP. POMDP RL environments benefit most from neural networks that capture the temporal dependencies between states. Even if the true state is only partially observable, for example due to sensor noise, the network can learn to approximate it from the evolution

of previous states.

Recurrent neural networks have the ability to capture temporal dependencies by encoding the evolution of the states in a form of internal memory. By capturing these dependencies in memory recurrent networks, they are able to learn how to make long-term planning decisions, an important element to autonomous driving. Recurrent neural networks are notoriously difficult to train because of vanishing and exploding gradients [72], which as mentioned in Section 1.1.2 cause training instabilities. Some recurrent networks like *Gated Recurrent Units* (GRU) [73] and LSTMs are more resilient to vanishing gradients and have demonstrated remarkable successes in various fields [74, 14].

CNNs also have the ability to capture temporal dependencies between states by performing convolutions across the time domain, using the same mechanisms as described above in Section 1.1.4. CNNs lack an internal memory component and therefore do not have the same long-term planning capabilities of recurrent networks. Despite this, they are frequently used in RL environments outside of autonomous driving [13] for encoding temporal variations between states. CNNs have shown to perform equivalently to LSTM networks in fields like language translation without the problems of vanishing or exploding gradients, while training at nine times the speed of recurrent networks [75].

Despite the benefits recurrent and convolutional neural networks may provide, in terms of robustness to state noise and capturing temporal dependencies between measurements, they have not been investigated for the learning of highway driving scenarios among other vehicles. LSTMs have been used for end-to-end learning of driving policies but only in driving scenarios without other vehicles [61, 51]. Mukadam *et al.* learned tactical decision making for highway driving using a 2D CNN that performed convolutions over the time domain [64]. However, their state space representation was based on occupancy grids that do not provide the relative distance and velocities to surrounding vehicles. By using the CNN architecture they were able to infer the velocities and acceleration of other vehicles from the evolution of their positions across time.

From previous literature it is not clear if network architectures that capture relationships between affordance indicator-based states containing positions and velocities is useful for autonomous highway driving. And if it would be useful, whether the ability of recurrent networks to use memory and plan ahead would provide any benefit over CNNs which are faster and easier to train remains questionable. Moreover, these architectures have not been explored for their ability to deal with sensor noise or to generalise to new scenarios. All of these aspects are important in autonomous driving applications and have the potential to help bridge the gap between simulation and reality. LSTM and CNNs are further discussed and evaluated in Chapter 5.

Limitations to the real-world applicability of RL for autonomous driving

The applicability of reinforcement learning in real-world autonomous driving is constrained by several problems. General challenges that face other autonomous driving technologies also apply to RL, for example: the challenges of predicting the intentions of pedestrians [76], navigating unseen traffic scenarios, dealing with the computational requirements of autonomous driving technologies, and of course safety. However, RL additionally has the challenge of bridging the gap between simulation and reality and enabling safe during training.

- **The gap between simulation and reality** Simulated environments provide the benefit that data is cheap to collect and can then be used to train driving policies from, but simulated data often does not have the same distribution and characteristics as data collected from the real-world. As a result, driving policies often fail to generalise to real-world applications and have inspired research from domain adaption, to augmenting simulation environments for better generalisability [77, 78].

  The gap between simulation and reality is often not addressed in RL for autonomous driving research, and does not include explicit testing of the learned policy's ability to generalise to new scenarios, handle measurement noise, and perform in more difficult dynamic and uncertain driving environments. As highlighted in Section 1.1.4, neural network architectures exist that have demonstrated an improved ability to generalise to new scenarios without the need for directly augmenting the environment. However, these approaches are not common simply because explicit testing for generalisability is not widely adopted.

  Furthermore, often basic and unrealistic simulation environments are used to train RL driving policies, which causes a further gap between simulation and reality. For example, simple elements such as noise on measurements are not modelled, and they fail to capture the dynamics and stochasticity of real-world settings. Some efforts have been made to test trained RL driving policies against the presence of sensor noise [57, 69, 52], but the noise levels are typically small and modelled as simple Gaussian distributions, uncharacteristic of the common uniformly distributed white noise present in many sensor technologies. By modelling simple simulation environments, the needs for complex models like the LSTMs and CNNs covered in Section 1.1.4 do not arise. This is in spite of the fact that these models have the potential to bridge the gap between simulation and reality, and bring RL learning closer to real-world applicability.

- **Safety in reinforcement learning** Since RL policies need to explore their environment to learn, they may often take unsafe actions during training. As mentioned before, this behaviour cannot be tolerated in autonomous vehicles where safety is a top priority. For this reason, this thesis implements the concept of *safe reinforcement learning* [79].

  Safe reinforcement learning aims to ensure that during training and deployment, RL policies avoid (or completely disallow) unsafe actions. Several techniques exist for ensuring safety in reinforcement learning, two of which two are listed below:

  – *Punishment during training*: The most common approach for teaching agents to avoid unsafe actions is to punish them for doing so. In this case reward functions are shaped to negatively reward agents when taking actions that lead to an unsafe environmental state, like maintaining unsafe highway speeds. Moreover, large negative punishments can be given when actions are taken that lead to a simulation reset during training, like a crash. Over time, agents can then learn to avoid such actions all-together and ideally never perform them during the deployment of trained policy. Nevertheless, during training agents may still take unsafe actions and safety guarantees using reward shaping alone are intractable and cannot be tolerated in autonomous vehicle applications.

– *External safety checks*: Safety controls outside of the scope of the RL algorithm provide the strongest guarantees that safety is maintained during training and deployment. It involves shifting safety to the simulation environment to check the action selections of the agent before they are executed in the simulation environment. For example, should an agent select an action that leads to a crash, the simulation environment simply ignores the action and instead choose another safe action instead. External safety checks have the benefit that there are fewer simulator resets during exploration, leading to more efficient and faster training [80]. Additionally, because agents are unable to explore unsafe environmental states, they indirectly learn to avoid unsafe actions.

Additional RL safety mechanisms are not the focus of this paper, but it is a very active research area [80, 81, 82], especially in relation to autonomous driving [83, 55, 84]

1.2 Research objectives, contributions and thesis structure

The overarching aim of this thesis is to look at different neural network architectures that may aid in bridging the gap between simulation and reality, such that RL may be one step closer to being applicable for the learning of real-world driving policies. The following three main points are metrics considered in the evaluation of the network architectures: their ability to generalise, deal with sensor-based noise on the state inputs, and perform in dynamic driving environments, closer reflecting those of real-world systems.

To this end several architectures are investigated including deep sets, all-convolutional CNNs, CNNs that incorporate permutation invariance into the network and LSTM networks.

Chapter 3 investigates the commonly used DQN RL algorithm using a simple fully-connected neural network. Several improvements to the algorithm, like DDQN, Prioritised Experience Replays (PER) and Duelling networks are investigated in an attempt to create a strong baseline for comparison to the other network architectures, because the improvements have demonstrated promising results in other environments. A strong baseline for comparison is established and a unique improvement is made to the original DQN method providing better stability during training.

Chapter 4 expands on recent research (within autonomous driving) on the neural networks that are able to more efficiently deal with varying input sizes through permutation invariance. These permutation invariant pooling functions render the number and ordering of vehicles in the state irrelevant. Deep sets are exhaustively compared to different types of CNN architectures with and without permutation invariance to investigate their promise of improved generalisability to environments with varying numbers of vehicles. Prior research found Deep Sets to out perform CNNs, and standard fully connected networks, however failed to investigate CNNs with permutation invariant pooling functions, as this permutation invariance is what enables the generalisability of Deep Set networks. It shows the benefit that CNNs can provide in enabling generalisability and that through the incorporation of pooling functions, CNN networks operating similarly to deep sets can provide even better generalisability.

Chapter 5 explores network architectures that have not yet been employed in RL for autonomous driving in scenarios where other traffic is present. LSTMs are investigated first

because of the benefit memory may provide in enabling long term planning into decision making especially in dynamic environments containing stochasticity. 2D CNNs performing convolutions over the time domain are also covered. While CNNs do not have a component of memory and thus cannot make long term planning decisions, they have the benefit of being easier to train and, like LSTMs, provide the opportunity to deal with partially observable (realistic) state spaces i.e. that contain sensor noise. Networks that capture the temporal evolution of the state space typically only add value in environments where there is partial observability of the state space, where the current state does not capture all the information necessary to take an optimal decisions. The chapter demonstrates the applicability of LSTM networks in enabling the training of real-world autonomous driving policies.

Chapter 2

Simulation Environment

The fundamental reinforcement learning interaction loop consists of two main entities: an agent and an environment (see Figure 1.2). The environment is an integral part of learning and responsible for providing the agent with information on the evolution of states and the associated rewards for taking actions in the environment. This chapter is dedicated to defining the autonomous driving learning environment used throughout this thesis, including the safety checks and reward function design that enable realistic driving policies.

Section 2.1 describes the fundamental building blocks and structure of the driving simulation environment. Details are provided on the exact highway scenario used for the training of the autonomous driving RL policies, along with the behaviour and control of other vehicles on the road. While the base simulator and low-level vehicle control algorithms are not in the scope of this thesis' contributions, they are nevertheless explained because successfully trained policies capture these vehicle behaviours when learning how to act in the simulation environment. Section 2.2, provides more details on the simulator's state space, and the format of the affordance-based state input vector used by the networks during learning. While the affordance-based state representation offers low-dimensionality, it still carries some inefficiencies that are difficult to learn from. It offers the motivation for the investigated Deep Set and CNN neural network architectures of Chapter 4, that remedy some of these state inefficiencies. Finally, Section 2.3, goes through the design of the reward functions of the environment and how they are shaped so that agents learn safe and realistic driving policies.

2.1 Simulation environment

Several simulation environments exist for the training of autonomous driving policies [85, 86, 87, 88]. However, a custom highway simulation environment developed by PhD student Bram De Cooman for the *Ford Alliance Project* is used in this thesis. It allows maximal flexibility in adapting the simulator to meet the goals of this thesis. The graphical display of the simulator is shown in Figure 2.1.

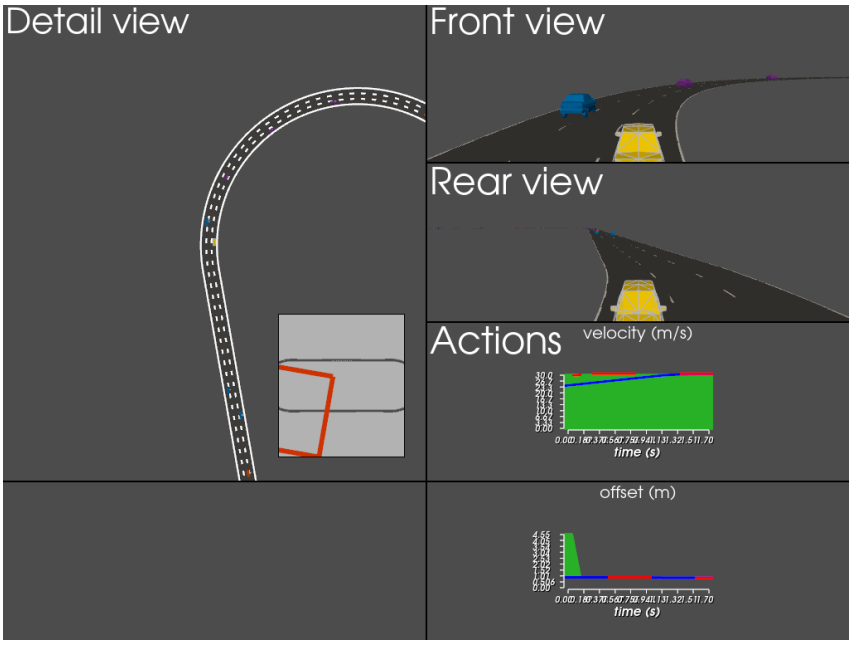


Figure 2.1: Simulator's display of the agent's vehicle. Three views of the vehicle are presented, the top view in "detail view", the "front view" and the "rear view". Velocity and offset (steering) actions are presented with red lines denoting action set-points, blue lines denoting vehicle measurements, and green areas the safety bounds for each action.

2.1.1 Scenario

The highway simulator comes pre-configured with different scenarios that can be used for the training of RL policies, but for the purpose of this thesis a simple three lane oval circuit is used. Upon instantiation, the circuit is populated with a number of independently controlled vehicles with different driving characteristics. High-level driver characteristics are generally based on their average speeds, ranging from slow, to medium, to fast, and are covered in more detail in the next section. The number of vehicles depends on what is being investigated in the subsequent chapters, but enough vehicles are always present to realistically reflect a highway with dense traffic. The amount of vehicles, and their varying characteristics, provides sufficient difficulty for learning in this scenario to be non-trivial. The circuit is shown in Figure 2.2.

2.1.2 Vehicle driving behaviours and dynamics

Vehicles in the simulation environment are controlled on multiple levels. High-level controllers are responsible for discrete action selections such as lane changes, while low-level controllers are responsible for continuous action selections such as reaching a velocity target. The two different levels of control abstraction are covered hereafter.

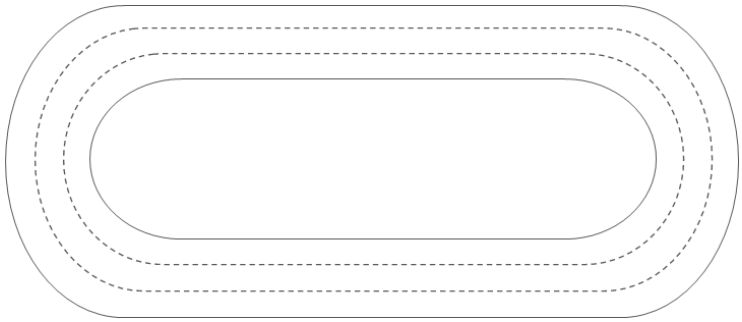


Figure 2.2: Oval highway track used as the driving scenario for learning.

High-level behavioural control

Vehicles in the simulator can be configured to have different driving styles beyond those that are mentioned above. For example, commonly used driving policies (amongst others) include IDM (Intelligent Driver Model) and MOBIL (Minimising Overall Braking Induced by Lane Changes) [89, 90]. IDM is a longtitudinal continuous-time driving behaviour model which tries to emulate human-like vehicle following, i.e. the accelerations and braking decelerations of drivers. MOBIL is a lateral driving model which uses relative velocity differences between vehicles to signal lane changing behaviours. Together IDM and MOBIL constitute a complete driving policy capable of realistic vehicle following and lane changes. However, its behaviour can be conservative, making learning in an environment populated with such vehicles relatively easy.

Instead of IDM and MOBIL, the custom rule-based driver models mentioned above are used for all vehicles besides that of the RL agent's vehicle. These rule-based models can be configured to have fast, medium, or slow average velocities. They tend to be more aggressive than the IDM and MOBIL models, resulting in more lane changes. Some rule-based vehicles additionally carry an element of stochasticity to their driving behaviours, further increasing learning difficulty in the simulation environment. This causes vehicles to randomly drift from the centre of their lanes, brake without apparent reason, and vary their speeds. The stochasticity in certain drivers' actions and having drivers with completely different driving behaviours within a single simulation makes learning safe policies more difficult. But it has the benefit that the resulting policies are more robust than their deterministic counterparts.

As mentioned in Chapter 1, this thesis aims to learn high-level decision making through reinforcement learning. As a result, the learned driving policies also fall under this classification of behavioural control. As with the rule-based controllers of the other vehicles in the highway scenario, the goal of the policy is to make discrete driving decisions, such has when to change lanes to maximise overall speed.

Low-level control of vehicle dynamics

The dynamics of vehicles are modelled using the Kinematic Bicycle Model (KBM) [91]. In the model the two sets of wheels of the vehicle are lumped together to form single wheels, located at each respective axle. The control inputs are the vehicles speed u_1 and the front

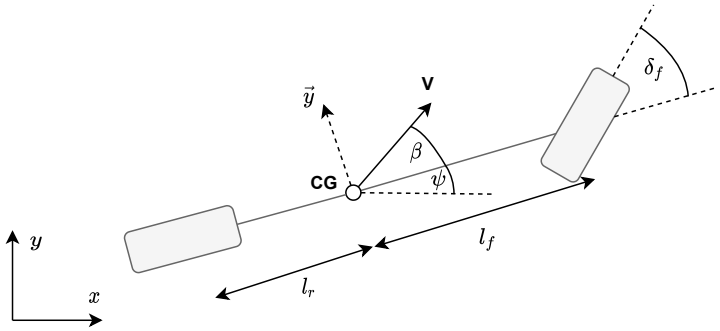


Figure 2.3: The kinematic bicycle model, used to mathematically approximate simple vehicle dynamics. Symbols denote: δ_f the steering angle, β the side-slip angle, V the velocity of the vehicle's CG, ϕ the vehicle heading, and l_r, l_f the lengths of the rear and front wheel bases, respectively.

wheel's steering angle u_2. The governing equations of the model can be written as:

$$\dot{x} = V\ \cos(\beta(u_2) + \psi) \tag{2.1}$$

$$\dot{y} = V\ \sin(\beta(u_2) + \psi) \tag{2.2}$$

$$\dot{\psi} = \frac{V}{l_r}\sin(\beta(u_2)) \tag{2.3}$$

where $\beta(u_2)$ is the side-slip angle at the centre of gravity (CG),

$$\beta(u_2) = \arctan\left(\tan(u_2)\frac{l_r}{l_f + l_r}\right). \tag{2.4}$$

The model is graphically illustrated in Figure 2.3, where the vehicle's velocity at the CG equates to V, and the front wheel's steering angle equates to δ_f.

At each time step in the simulator, the ordinary differential equations (2.1-2.4) are integrated using the Runge-Kutta fourth order integrator. The bicycle model is only valid under certain assumptions such as no tire slip, however, it suffices for our application where vehicles can accurately be approximated as quasi-static systems. Because of its simplicity it has the benefit of being easy to compute, which helps improve the training times of the RL models. This is especially important since the simulator runs at a frequency of 10Hz and needs to calculate the dynamics of each vehicle present in the simulation at each time step. The control of each vehicle's bicycle model is at first performed by the high-level rule based controllers. After the high-level controller selects an action like a lane change or a change in speed, the actions are sent to PID controllers. The low-level PID controllers ensure that measurements from the KBM after an integration step are properly driven to the control desired control set-points that would complete a manoeuvre.

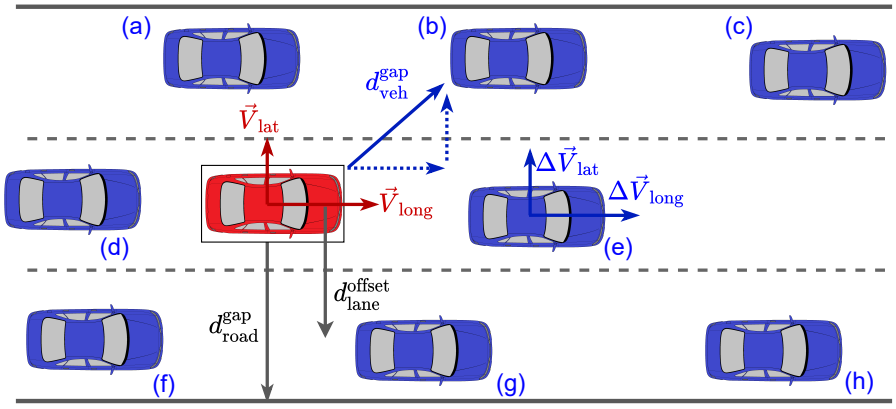


Figure 2.4: Illustration of all the measurements made by the agent's vehicle (red) during simulation relative to other vehicles (blue), other lanes, and the road edge (grey).

2.1.3 Simulator safety

Safety is built directly into the simulator and ensures that actions taken by the agent during the learning process do not result in accidents. At each time step the agent's chosen actions are compared to the computed safety bounds and if they fall outside of these bounds, the actions are discarded and instead replaced with a safe action. The new safe action is chosen to be as similar as possible to the originally chosen action.

2.2 Measured state

The standard state representation includes all the measurements available to the agent at any given time step. An illustration containing an example of each of the types of measurements contained within the state vector ($\boldsymbol{s}$) available to the agent is given in Figure 2.4. The types of measurements include:

- the ego-vehicle's velocity $\vec{\boldsymbol{V}} = (\vec{V}_{\text{long}}, \vec{V}_{\text{lat}})$,
- the distance to either side's road edge $\boldsymbol{d}^{\text{gap}}_{\text{road}}$,
- the offset distance to lane centres $\boldsymbol{d}^{\text{offset}}_{\text{lane}}$,
- the longitudinal and lateral distance to each measured vehicle $\boldsymbol{d}^{\text{gap}}_{\text{veh}}$,
- and the velocities relative to other vehicles $\Delta\vec{\boldsymbol{V}} = (\Delta\vec{V}_{\text{long}}, \Delta\vec{V}_{\text{lat}})$.

The total number of measurements depends on the size of the agent's *Field Of View* (FOV). The FOV is configurable, but is set to a longitudinal viewing distance of 150m, and 3m laterally observable lanes. This configuration is used for all simulations because of the similarity to the perception ranges of human drivers. Within this FOV, the total number of measurements depends on the number of perceivable vehicles. This parameter is set to two vehicles in front, and behind the agent's vehicle for each lane in the FOV. When the FOV is maximally populated with perceivable vehicles, the state vector is $\boldsymbol{s} \in \mathbb{R}^{53}$. In other

words, the environment provides the agent with a representation of its state consisting of 53 measurements.

2.2.1 Problems with the state representation

There are unfortunately a number of different problems that arise from this state vector representation. Each of these pose challenges for learning in such an environment. The two main causes come from the fact that the state vector has to be a fixed size, and that locations of vehicles in the state vector can vary.

Size of the state vector:

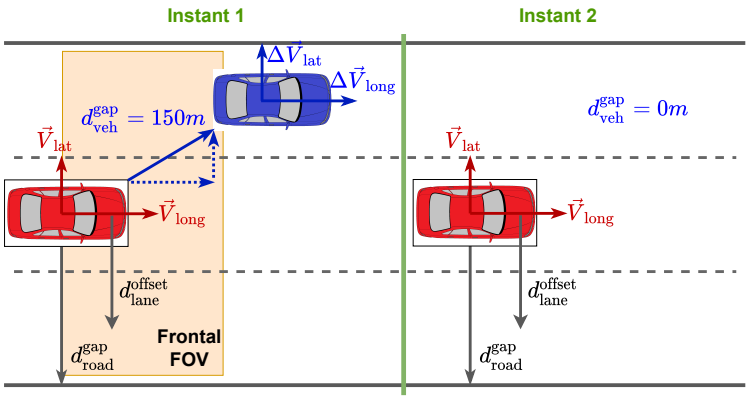


Figure 2.5: Demonstration of a discontinuous jump in the measured distance to another vehicle as it leaves the agents FOV and is no longer perceivable.

When training deep RL policies, the size of the state vector passed to the neural network has to be fixed throughout training because the network's size is fixed upon instantiation. However, a full set of 53 measurements may not be available because of the agent's perceivable FOV. If the agent is driving on a section of the 3 lane circuit that does not contain other vehicles in its FOV, the state vector should only contain 5 elements: the vehicles absolute velocity, and the distances to the road edges and lane centres. Such a scenario is generally easily remedied through *padding*, which refers to replacing missing entries in the state vector with zeros so that the state size is fixed to $\boldsymbol{s} \in \mathbb{R}^{55}$. However, this causes some measurements to experience discontinuous jumps in their values, and introduces sparsity into the state vector. Figure 2.5 below illustrates an example of such a discontinuous jump. The moment a vehicle leave the agents FOV and is no longer perceivable, the distance measurement to this vehicle jumps from a value of 150 to 0 in an instant. Such a jump is difficult to interpret during learning, and the sparsity introduced by padding adds additional complexity.

Location of vehicle measurements in the state vector:

Each vehicle's measurements are positioned in the state vector depending on their locations relative to the agent's vehicle. Vehicles in front of the agent's vehicle in the left-most lane

will be contained in a specific area of the state vector, different to that of vehicles behind the agent's vehicle, or in the other lanes. This in itself does not seem like a cause for concern, but when vehicles transition past these separation boundaries (e.g. behind to in-front or, left lane to centre lane), a vehicle's measurements discontinuously jumps from one location to another in the state vector. As before, such jumps are mathematically difficult to approximate and complicate learning. The crossing of a vehicle over the separation boundaries is shown in Figure 2.6.

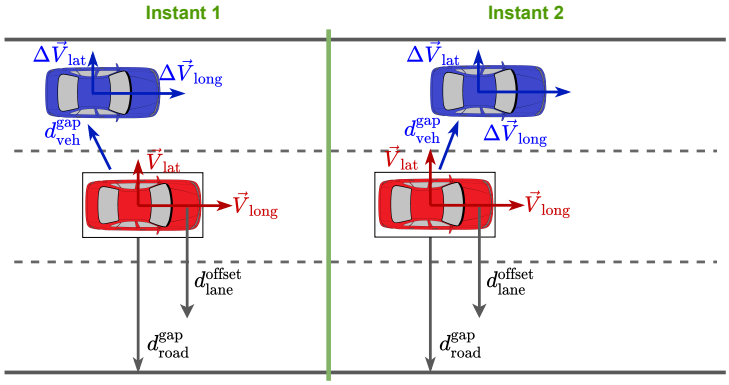


Figure 2.6: Demonstration of a discontinuous jump of the position of a vehicle's measurement in the state vector when moving from a location behind the agent to a position in front of the agent.

When the two problems are combined with the standard state vector padding method, our assumptions of the agent's FOV and maximum number of perceivable vehicles, more complex irregularities can arise. Two examples are explained hereafter, but many more exist. Firstly, let's consider the case in Figure 2.4 where a vehicle a increases its speed. Initially the state vector at time step t may contain these measurements to other vehicles:

$$\begin{aligned}\boldsymbol{s}_t = [&(0 \in \mathbb{R}^4, \boldsymbol{d}^{\text{gap}}_{\text{a}}, \Delta\vec{\boldsymbol{V}}^{\text{gap}}_{\text{a}})^{\text{left}}_{\text{rear}},\\ &(\boldsymbol{d}^{\text{gap}}_{\text{b}}, \Delta\vec{\boldsymbol{V}}^{\text{gap}}_{\text{b}}, \boldsymbol{d}^{\text{gap}}_{\text{c}}, \Delta\vec{\boldsymbol{V}}^{\text{gap}}_{\text{c}})^{\text{left}}_{\text{front}}, \ \dots],\end{aligned}$$

but one simulation time step later the vector will instead be:

$$\begin{aligned}\boldsymbol{s}_{t+1} = [&(0 \in \mathbb{R}^4, 0 \in \mathbb{R}^4)^{\text{left}}_{\text{rear}},\\ &(\boldsymbol{d}^{\text{gap}}_{\text{a}}, \Delta\vec{\boldsymbol{V}}^{\text{gap}}_{\text{a}}, \boldsymbol{d}^{\text{gap}}_{\text{b}}, \Delta\vec{\boldsymbol{V}}^{\text{gap}}_{\text{b}})^{\text{left}}_{\text{front}}, \ \dots].\end{aligned}$$

When it passes the classification boundary from being behind the agent's vehicle to in front, vehicle (c)'s measurements will be completely removed from the state vector and be replaced vehicle (b)'s measurements. Vehicle (b)'s measurements will similarly be replaced by vehicle (a)'s measurements.

Learning from state representations with such irregularities, like sparsity and discontinuous jumps, can be challenging, but definitely not impossible. Alternatively, all of the measurement information from all of the vehicles within the simulation could be incorporated into a large state vector of a fixed size. This, however, is not realistic in real-world driving scenarios because distant vehicles generally do not influence driving behaviour.

Moreover, it would drastically increase the dimensionality of the learning problem and thus the learning difficulty. In Chapters 4 and 5, we dive deeper into different types of neural networks, that more efficiently deal with the chosen state representation and remedy some of the highlighted problems through different mechanisms.

2.3 Reward function shaping

As mentioned, the environment is responsible for providing two pieces of crucial information to the agent during the learning process: the environment's state $\boldsymbol{s}$ and scalar rewards R. Higher rewards are provided to the agent for good actions, and lower ones for bad actions. By appropriately creating a reward function with the correct elements and tuning the importance of each of element correctly, drastically different driving policies can be obtained.

In accordance to European driving conventions, the chosen reward function consists of four terms:

$$R = w_{\text{velocity}} r_{\text{velocity}} + w_{\text{offset}} r_{\text{offset}} + w_{\text{follow}} r_{\text{follow}} + w_{\text{lane}} r_{\text{lane}}, \tag{2.5}$$

where r_{velocity} denotes the reward for the vehicle's velocity, r_{offset} the reward for the vehicle's distance from its current lane's centre, r_{follow} for maintaining a safe following distance, and r_{lane} for keeping in the right-most lane. The relative importance between each element in the reward function is controlled through the scalar weight variables w in front of each respective term. Together each of these components contribute to creating policies that: maintain highway speeds of $120 km/h$, avoid unnecessarily frequent lane changes, keep safe following distances, and drive in the right-most lane, when possible.

The reward functions for $r_{\text{velocity}}, r_{\text{offset}},$ and r_{follow} are all exponential functions defined as:

$$r_{\text{velocity}} = \frac{e^{-(v_{\max} - v_{\text{current}})^2}}{a}, \tag{2.6}$$

$$r_{\text{offset}} = \frac{e^{-(d_{\text{lane}}^{\text{offset}})^2}}{b}, \tag{2.7}$$

$$r_{\text{follow}} = -\frac{-e^{-(d_{\text{limit}} - d_{\text{veh}}^{\text{gap}})^2}}{c}. \tag{2.8}$$

While r_{lane} is a linear function:

$$r_{\text{lane}} = \left(\frac{1}{-W}\right) d_{\text{road}}^{gap}, \tag{2.9}$$

with W denoting the the road width, and d_{road}^{gap} the distance to the right road edge. Each reward function is graphically represented in Figure 2.7.

Reward functions are chosen to be smooth and continuous such that the evolution of the overall scalar reward mapping is also smooth. Weights in Equation (2.5) and constants in Equations (2.6 - 2.9) are tuned experimentally by observing the resulting driving policy behaviour after learning. The tuned constants are summarised in the table below:

Figure 2.7 shows the ego vehicle is maximally rewarded for speeds close 120hm/h, when it is in the right-most lane. A reward term for keeping a safe following distance is also

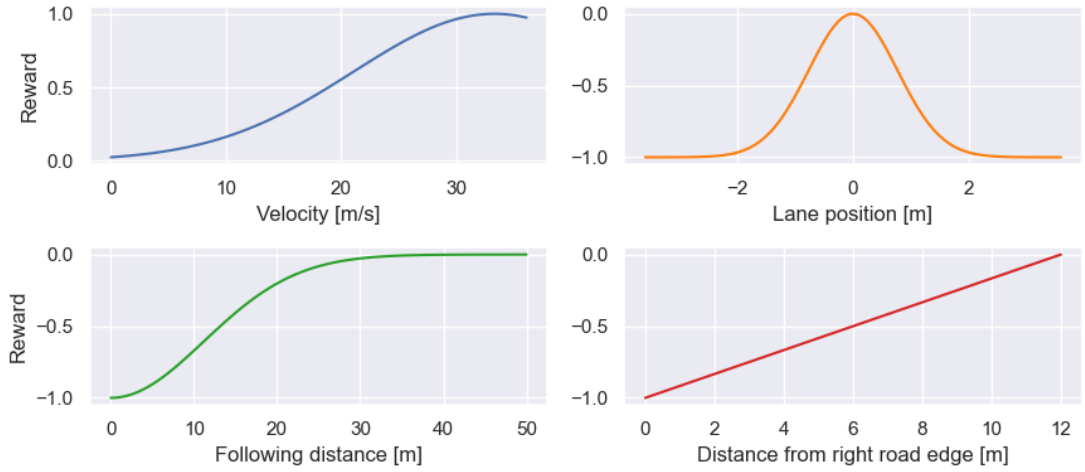


Figure 2.7: The reward function components that together form the scalar reward provided to the agent during learning.

Reward function parameters			
Velocity	$w_{\text{vel}} = 1.25$	$a = 300$	$v_{\max} = 33m/s$
Lane change	$w_{\text{offset}} = 0.15$	$b = 1.18$	
Following distance	$w_{\text{follow}} = 0.8$	$c = 250$	$d_{\text{limit}} = 0m$
Lane position	$w_{\text{lane}} = 0.35$		

Table 2.1: Values of the tuned reward function used in the simulation environment.

added, because although the environment's safety mechanisms control rear-end collisions, it encourages the agent to learn over its entire action distribution. If excluded, agents tend to exploit acceleration and tend not to learn to take other actions to increase their speed.

2.4 Conclusions

The low-level vehicle dynamics and the other rule-based vehicles in the simulation environment do not form part of the contributions of this work, but they are highlighted because these characteristics are all implicitly captured when training a RL autonomous driving policy in this environment. The affordance-based state representation offers a beneficially low-dimensional state space to learn from, however it still carries some inefficiencies that make learning difficult. The first inefficiency is from the unnecessary sparsity introduced when states are a fixed size and no vehicles are perceivable by the ego-vehicle. The second is due to the ordering of the vehicles in the state matrix as they move around the ego-vehicle. Both of these inefficiencies result in discontinuities in the state space that make learning more challenging. Chapter 4 investigates network architectures that remedy these inefficiencies through mechanisms like permutation invariance, which are able to process variable sized inputs and render the ordering of the vehicles in the state invariant.

Chapter 3

Baseline Autonomous Driving Agent Performance

Following the definition of the simulation environment, this chapter describes the Q-learning algorithms used throughout the rest of the thesis. Since the focus of this research is on investigating different neural network architectures, it is important to establish a strong baseline for comparison. To this end, four different variants of the original DQN method proposed by Mnih *et al.* [12] are considered: the *Double Deep Q-network* (DDQN), the *Double Deep Duelling Q-network* (D3QN), a meta algorithm for improved experience sampling, and the *Prioritized Experience Replay* (PER). Each of these have shown considerable performance increases across a number of different baselines [48], so they are re-evaluated to establish if they are benefitial to learning autonomous driving policies.

The chapter is organised as follows: Section 3.1 explains the inner-workings of deep Q-learning methods and the three improvements that have been implemented with respect to the original DQN algorithm. Section 3.2.1 defines all the simulation environment's parameters used for training, and implemented throughout the remainder of the thesis. Section 3.2.2 describes the configuration space used for the tuning of each algorithm's hyper-parameters. Section 3.2.3 illustrated the relative training performance between the RL algorithms considered and the absolute performance of the chosen DDQN baseline against two rule-based driving policies. The evaluation of the baseline with respect to generalisability and robustness are covered in Chapter 4 and 5 respectively. Section 3.3 describes preliminary results for a novel contribution made to improve the original DQN algorithm's stability during training. Finally, Section 3.4 provides conclusions for this chapter.

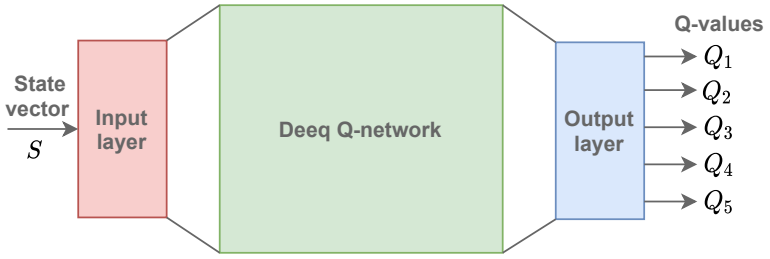


Figure 3.1: The implemented DQN architecture with five Q-value outputs. Each Q-value corresponds to the discrete action values: increase speed, maintain speed, decrease speed, change to left lane, or change to right lane.

3.1 Q-Learning methods

3.1.1 Deep Q-Networks (DQN)

DQN substantially contributed to the advancements in the field of deep reinforcement learning [12, 13]. DQN's learn the Q-function, defined as follows:

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \mathbb{E}_{\tau^{\pi}}\left[\sum_{i=1}^{T-t} \gamma^{t+i} r(\boldsymbol{s}_{t+i}, \boldsymbol{a}_{t+i})\right]. \tag{3.1}$$

The Q-function approximates the expected discounted reward when taking action $\boldsymbol{a}_t$ at state $\boldsymbol{s}_t$ and following the policy π thereafter. Through the *Bellman equation* it is possible to express this recursively as [11]:

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\tau^{\pi}}[Q^{\pi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})]. \tag{3.2}$$

Since DQNs consider only deterministic action choices we chose the optimal action as:

$$\pi(\boldsymbol{s}) = \arg\max_{\boldsymbol{a}} Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}). \tag{3.3}$$

The policy π is then implicitly defined by Q. Because of the large state space, the Q-function is represented using a deep neural network as illustrated in Figure 3.1, which can be trained by minimising the following loss function:

$$\mathcal{L} = \left(r + \gamma \max_{\boldsymbol{a}'} Q_{\theta}(\boldsymbol{s}', \boldsymbol{a}') - Q_{\theta}(\boldsymbol{s}, \boldsymbol{a})\right)^2, \tag{3.4}$$

where $(\boldsymbol{s}, \boldsymbol{a})$ denote the current state and action, and $(\boldsymbol{s}', \boldsymbol{a}')$ the next state and action pair. To compute this loss function minimisation, trajectories are stored as transition tuples $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', r)$ in a replay buffer, for use in an off-policy manner. The entire training process is described in algorithm 1 [12] .

When using nonlinear neural networks as Q-value function approximators several instabilities arise. The instabilities have several causes [13], namely:

- Correlations in the observations, for example, between successive observations.

Algorithm 1: Deep Q-learning with Experience Replay

Initialise replay memory $\mathcal{D}$ to capacity N
Initialise action-value function Q_π with random weights
for *episode* $= 1, ..., M$ **do**
  Receive initial state $\boldsymbol{s}_0$
  for $t = 0, ..., T-1$ **do**
    With probability ϵ select a random action $\boldsymbol{a}_t$ otherwise select $\boldsymbol{a}_t = \max_a Q_\theta(\boldsymbol{s}_t, \boldsymbol{a})$
    Execute action $\boldsymbol{a}_t$ and observe reward r_t and next state $\boldsymbol{s}_{t+1}$
    Store transition $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', r)$ in $\mathcal{D}$
    Uniformly sample a random mini-batch of transitions $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', r)$ from $\mathcal{D}$
    Set $y_i = \begin{cases} r_i & \text{for terminal } \boldsymbol{s}_{i+1} \\ r_i + \gamma \max_{\boldsymbol{a}'} Q_\theta(\boldsymbol{s}_{i+1}, \boldsymbol{a}') & \text{for non-terminal } \boldsymbol{s}_{i+1}. \end{cases}$
    Perform a gradient descent step on $(y_i - Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i))^2$
  end
end

- Small updates in the Q-function may result in large changes in the policy and hence the data distribution.
- Correlations between the action-values (Q) and the target values ($r + \gamma \max_{a'} Q(s', a')$) because they are both approximated using the same neural network.

The *experience replay buffer* is a key element in the DQN algorithm, and other off-policy RL methods originating in biology inspired mechanisms for learning [92, 93, 94]. By sampling uniformly from the buffer, the network is able to learn, and re-learn, from past experiences. This is important because as the policy is being trained, over time agents may "forget" about actions that previously led to high rewards and need to repeat the same behaviour. The ability to re-sample and reinforce past experiences is one of the reasons why DQN learning is considered sample efficient, and remedies some of the instabilities during training that arise when using neural networks to represent the Q-functions. The use of the experience replay buffer deals with the correlations between observations and thus smooths changes in observational distributions. Another feature of the algorithm is that updates to the target Q-values are not updated at every time step, but instead periodically in order to remove correlations between the target value and the action-values.

3.1.2 Double Deep Q-Learning (DDQN)

Despite the periodic update of the target value in the DQN algorithm, correlations between the target values action-values still exist because they are approximated by the same neural network. The target value, $y = r(s, a, s') + \gamma \max_{a'} Q_{\theta'}(s', a')$ frequently over-estimates the true return ($Q_{\theta'}(s', a') >$ true Q value) because of the max operator in the equation [95]. By introducing a second neural network for target value calculations, the action selection and action evaluation steps are decoupled. Thus decoupling of the target- and

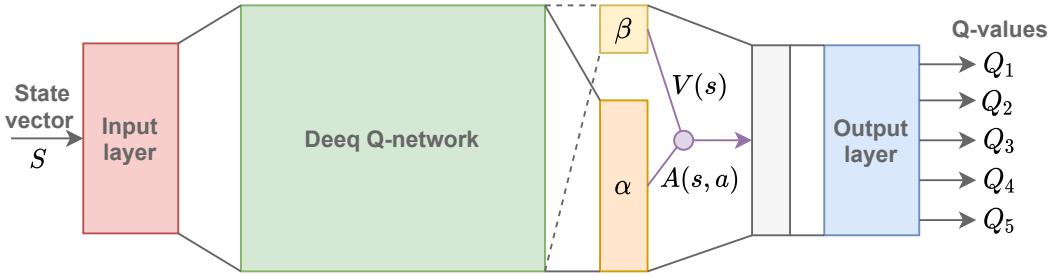


Figure 3.2: A duelling DQN architecture, which decomposes the Q-values into the value of a given state $V(s)$, and the advantage of an action $A(s, a)$.

action-values results in a reduction in overestimation biases when an agent is trying to predict its success. The technique is referred to as *double Q-learning* [95, 96].

Algorithmically, there is not much difference to the first algorithm except that the computation of the target value becomes:

$$y_i = r_i + \gamma \max_{\boldsymbol{a}'} Q_{\theta'}(\boldsymbol{s}_{i+1}, \operatorname{argmax}_{\boldsymbol{a}'} Q_\theta(\boldsymbol{s}_{i+1}, \boldsymbol{a}')), \tag{3.5}$$

where $Q_{\theta'}$ now represents the new target value, and $\operatorname{argmax}_{\boldsymbol{a}'} Q_\theta(\boldsymbol{s}_{i+1}, \boldsymbol{a}')$, the original action-value.

3.1.3 Duelling Double Deep Q-Leaning (D3QN)

The value of an action in the DQN algorithm ($Q_\theta(\boldsymbol{s}, \boldsymbol{a})$) depends on the inherent value of a state $V^\pi(\boldsymbol{s})$ and the value of the different actions given the current state $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$. Certain states are worse than others, independent of the actions taken, and some actions noticeable are worse than others, no matter the state. Mathematically this is represented by the *advantage* $A^\pi(\boldsymbol{s}, \boldsymbol{a})$ of an action:

$$A^\pi(\boldsymbol{s}, \boldsymbol{a}) = Q^\pi(\boldsymbol{s}, \boldsymbol{a}) - V^\pi(\boldsymbol{s}). \tag{3.6}$$

The advantage of taking an optimal action given the state $\boldsymbol{s}$ is zero because the optimal policy will always choose action $\boldsymbol{a}$ in state $\boldsymbol{s}$. All other actions will have a negative advantage because they will by definition return less reward than taking the optimal action.

Reformulating equation 3.6 above allows us to represent the Q-value of an action $Q^\pi(\boldsymbol{s}, \boldsymbol{a})$ as the the the advantage of an action given the state $A^\pi(\boldsymbol{s}, \boldsymbol{a})$, and value of being in that state $V^\pi(\boldsymbol{s})$:

$$Q^\pi(\boldsymbol{s}, \boldsymbol{a}) = V^\pi(\boldsymbol{s}) + A^\pi(\boldsymbol{s}, \boldsymbol{a}), \tag{3.7}$$

has been shown to advantageous in learning [97] because the advantage function contains less variance than the Q-values. This decomposition enables the logic that when the value of a state is known to be low, additional exploration of the values of other actions in that state would not result in much benefit.

This leads to the duelling DQN architecture, which when combined with the improvement of the previous section forms the *duelling double deep Q-network* or D3QN [47].

Algorithmically, D3QN is very similar to DDQN (and DQN), except that the Q-value now becomes:

$$Q_{\theta,\alpha,\beta}(\boldsymbol{s},\boldsymbol{a}) = V_{\theta,\beta}(\boldsymbol{s}) + (A_{\theta,\alpha}(\boldsymbol{s},\boldsymbol{a}) - \max_{\boldsymbol{a}} A_{\theta,\alpha}(\boldsymbol{s},\boldsymbol{a})). \tag{3.8}$$

The third term of the equation is present to counteract the addition of the constant $V_{\theta,\beta}(\boldsymbol{s})$ to the Q-value computation. Parameters α and β represent the network weights in the decomposed Q-network architecture as illustrated in Figure 3.2.

3.1.4 Prioritised Experience Replay

The final enhancement made to the DQN algorithm comes from an improved experience replay buffer, the *prioritised experience replay* (PER) buffer [48]. In the standard experience replay buffer, transitions are sampled uniformly. By sampling uniformly it is assumed that all transitions are equally important for learning but in fact, certain transitions should hold more weight. PER changes the sampling distribution to leverage the idea that transitions that do not fit well to the current Q function estimate are more important than ones that do. This is derived from biology, where generally more importance is placed on situations that differ from set expectations, so that these inputs can be used to more effectively learn from them.

The error between the current Q-value estimate $Q_\theta(\boldsymbol{s},\boldsymbol{a})$ and the target value $Q_{\theta'}(\boldsymbol{s},\boldsymbol{a})$ is commonly defined as:

$$\text{error} = |Q_\theta(\boldsymbol{s},\boldsymbol{a}) - Q_{\theta'}(\boldsymbol{s},\boldsymbol{a})|. \tag{3.9}$$

This Temporal Difference error (TD-error) is stored in the PER buffer and used to update the sampling distribution during training.

Sampling distribution updates are made according to *proportional prioritisation*. The error is first converted into a priority using:

$$p = (\text{error} + \epsilon)^\alpha, \tag{3.10}$$

where ϵ denotes a small positive constant to ensure that no transition has a priority equivalent to zero. $\alpha \in [0,1]$ denotes a hyperparameter controlling the amount of prioritisation to be used. When $\alpha = 0$ sampling is done uniformly from all transitions.

Priorities ($\boldsymbol{p}$) are translated to sampling probabilities ($\boldsymbol{P}$) through:

$$P_i = \frac{p_i}{\sum_k p_k}, \tag{3.11}$$

where the sample's index is denoted by i.

The PER buffer uses a different data structure than the standard experience replay buffer. The PER buffer uses a *sum-tree* structure, which is a binary tree where each parents value is the sum of its children. Samples themselves are stored in the leaf nodes and value updates to the leaf nodes propagate upwards through the entire tree. The reason for not using a simple sorted array is that keeping track of all the priorities adds complexity to the replay buffer, counteracting the benefit of more efficient sampling.

The final part of the PER buffer is to anneal the bias that is introduced on the data when changing the sampling distribution. Bias correction is done through the introduction

Simulation and training parameters	
Simulation steps per episode	5000
Number of training episodes	800
Simulator frequency	1 Hz
Ego-vehicle decision frequency	8 Hz
Number of slow vehicles	5
Number of medium vehicles	20
Number of fast vehicles	5

Table 3.1: Default configuration of the simulation environment and training parameters.

of *importance-sampling* weights:

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P(i)} \right)^{\beta}, \tag{3.12}$$

that completely compensate for the non-uniform sampling probabilities $P(i)$, even if $\beta = 1$. The β exponent is another hyperparameter that is introduced, however, it is not fixed (like α) but rather annealed linearly, reaching 1 only near the end of training. Importance sampling weights are always normalised by factor $1/\max_i w_i$ for stability reasons, and are easily folded into the Q-learning update by using $w_i \delta_i$ instead of δ_i.

3.2 Baseline Q-network performance

This section outlines the comparison between the different algorithms highlighted before: DQN, DDQN, D3QN, and both experience replay (ER) and PER buffers. Hyper-parameter sweeps are done on all of the networks and algorithms and the best performing algorithm during training is used as a comparative baseline for all of the investigated neural networks of the subsequent chapters.

3.2.1 Default simulator and training configuration

The default simulation environment and training parameters are configured as described in Table 3.1.

The simulation environment variables are all established from experimentation. Since the goal is to learn a high-level driving policy with discrete action choices, the ego-vehicle runs at a frequency almost a tenth slower than the simulator's. The ego-vehicles decision frequency is tuned to 8Hz so that after a lane change decision has been made, it is fully completed before the next decision instance. If we recall all the networks in this paper have five discrete outputs: increase speed, maintain speed, decrease speed, switch to left lane, and switch to right lane. Velocity increases and decreases have been tuned to increase of decrease the target velocity by $3m/s$.

Algorithm	Parameter	Final value
Experience replay	Buffer size	300000
Prioritized experience replay	Buffer size	300000
	Prioritisation factor (β)	0.25
	Rate of linear β annealing to 1	2e-6 per decision
	Priority scale (α)	0.75
DQN	Batch size	32
	ϵ_0 (start of training)	1
	ϵ_1 (end of training)	0.1
	ϵ exponential decay rate	0.999976
	Network update rate	8Hz
	Optimizer	Adam
	Loss function	Huber
	Discount factor (γ)	0.95
	Gradient clipping	False
DDQN	Target update rate	1e-4
Q-network	Size	(32,32)
	Hidden layer activation function	Relu
	Output layer activation function	Linear

Table 3.2: Final training hyper-parameters selected for DQN, DDQN, ER, and PER algorithms after several parameter sweeps.

3.2.2 Baseline Q-network parameter sweeps

Several hyper-parameters exist for each of the methods and networks considered. They have all been tuned by running successive parameter sweeps. Due to computational limitations it was not possible to try every parameter combination; instead, parameter sweeps are done on a single variable at a time while keeping the others fixed. This process is repeated until the best training performance is observed for each of the algorithms. The same approach is taken for all the parameter sweeps that are done for each of the methods and networks throughout the paper. Only the final results of the parameter sweeps are included in the main body of the text.

A few trade-offs are made when selecting between different parameters. Larger network sizes are only chosen when they offer a large increase in training performance, otherwise smaller networks are selected. Larger neural networks can be prone to over-fitting and result in a decrease in generalisability and are computationally more demanding during training. All parameters are swept over three different seeds, and from that each parameter is also considered because of the stability it provides during training. For instance, a relatively large buffer size of 300,000 experiences is chosen at the expense of training speed, because of the stability it provides in avoiding that agents "forget" past experiences that led to increases in rewards. Table 3.2 summarises the chosen hyper-parameters that offered the best overall training performance.

Exploration and exploitation are balanced through an exponential decay of ϵ from a value of 1 at the start of training, to 0.1 at the end of training. The rate of β annealing is linearly varied from a value of 0.25 to 1.0 throughout training. The evolution of these to

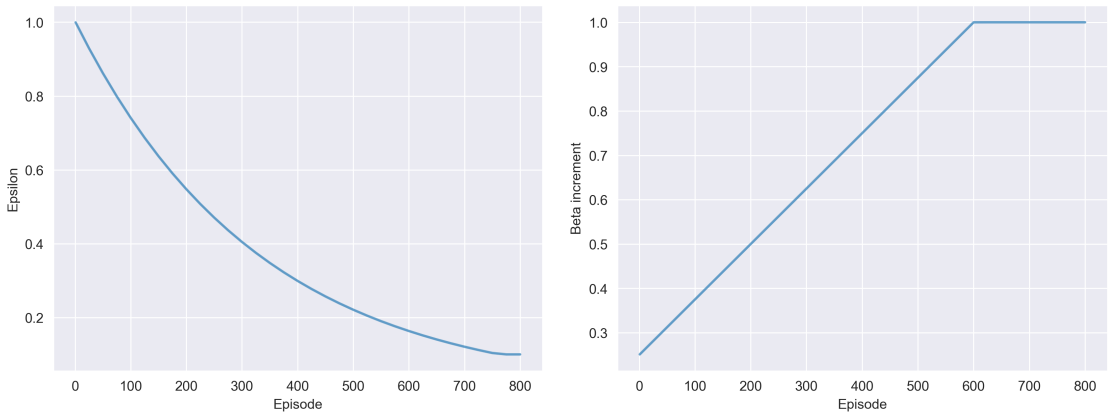


Figure 3.3: The evolution of ϵ and β throughout training. Responsible for managing exploration and exploitation, and the relative sampling importance from the PER buffer, respectively.

parameters throughout training is visually depicted in Figure 3.3.

3.2.3 Baseline Q-learning method results comparison

The comparison between the different Q-learning methods and meta algorithms is the first contribution to this thesis. The focus of this research is on investigating different neural network architectures. The generated results are important to establish a strong baseline for relevant comparison to other network architectures. These improvements to the DQN method have all shown promising increases to training performance on other baselines. However, the comparative benefit that they would have on the learning of autonomous driving policies has not been explicitly demonstrated in literature. Figure 3.4 shows the training performance of these different algorithms and their combinations over five random training seeds. The quantitative difference between each of these methods is summarised in Table 3.3.

The first interesting observation is that the standard ER buffer, which samples experiences uniformly, out-performs the PER buffer on all of the algorithms. On all the algorithms, the use of the PER buffer results in a higher standard deviation when compared to the ER buffer, but has a less pronounced negative effect on the rewards over training. This increase in standard deviation may indicate that the bias introduced when sampling experiences non-uniformly was not fully corrected for by the importance sampling weights as explained in Section 3.1.4 above. Another possibility is that the PER is simply not suitable for this driving scenario in particular, as it has been shown that the PER buffer does not result in an increase in performance on all training environments [48].

Overall the commonly used DDQN method with uniform experience sampling performs the best, with a +1.72% increase on the normalised reward averaged over the five training seeds. The D3QN algorithm performs comparable to the DDQN method in terms of reward maximisation. However, as explained in Section 3.1.3, the explicit decomposition of the Q-values $Q^{\pi}(s,a)$ into the advantage $A^{\pi}(s,a)$ and value of a state $V^{\pi}(s)$ results in

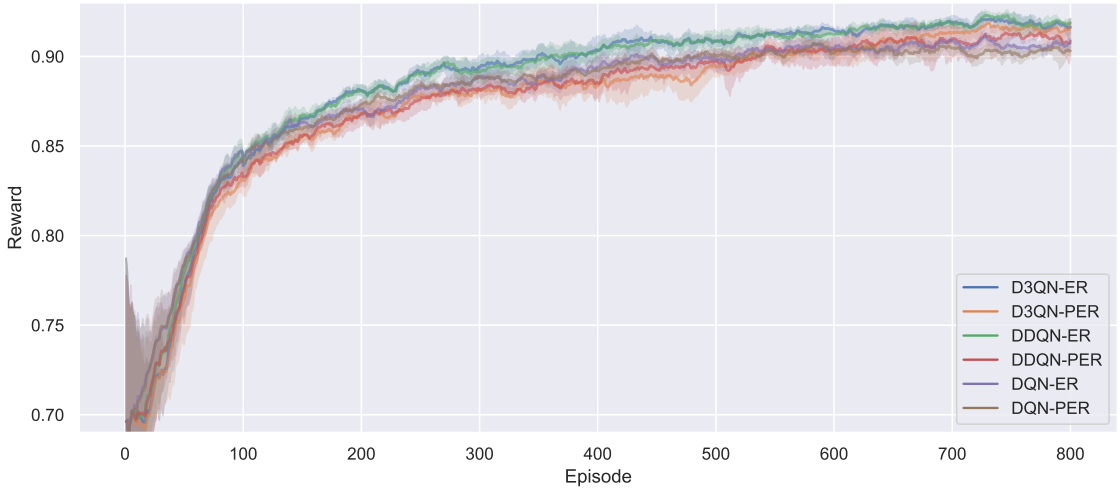


Figure 3.4: Overall training reward for the different Q-learning algorithms and experience replay combinations implemented for use as a baseline. Results are smoothed with an exponential moving average at a weight of 0.9 and presented as a distribution over five random seeds.

a decrease in the Q-value's variance. This decrease is rather small when compared to the DDQN method, and does not motivate the additional network architecture complexity. For these reasons, the DDQN method with a replay buffer that uniformly samples experiences is used as the comparative baseline for the remainder of the results of the thesis.

The introduction of an explicit target network in DDQN is specifically for the improvement of training stability over the original DQN method. However, from these results, the DQN algorithm had the lowest variance during training. The DQN algorithm presented here has a unique modification when compared to the original DQN algorithm presented by Mnih *et al.*. This improvement to the DQN algorithm is covered in more detail in Section 3.3 that follows.

	Average reward	**Peak reward**	**Final reward**	**Standard deviation**
D3QN ER	+1,26 %	+1,49 %	+1,47 %	+6,56 %
D3QN PER	0	+1,26 %	+1,42 %	+7,99 %
DDQN ER	**+1,33** %	**+1,73** %	**+1,72** %	+8,28 %
DDQN PER	+0,10 %	+0,67 %	+0,61 %	+50,53 %
DQN ER	+0,49 %	+0,47 %	+0,46 %	**0**
DQN PER	0,40 %	0	0	+0,45%

Table 3.3: Quantitative training performance differences between the baseline Q-learning methods. All values are displayed as percentage differences, relative to the minimum respective metric of all the algorithms. The best performing algorithm on each of the metrics is shown in bold face.

Only the relative training performance of the chosen DDQN method has been covered

Figure 3.5: Comparison between the average episodic vehicle speed for the trained DDQN baseline, IBM & MOBIL, and a ruled-based driving policy. Data is distributed over five random seeds, different to the seeds used for training of the DDQN model, and smoothed with an exponential moving average at a weight of 0.3.

and its absolute performance when compared to well known rule-based driving policies like IBM & MOBIL has not. In line with autonomous driving literature, Figure 3.5 shows that the DDQN method outperforms both the popular IBM & MOBIL driving policy and that of the less conservative rule-based driving policy. The figure shows the average vehicle speed per episode over 50 episodes, run over five seeds different to that used for the training of DDQN. DDQN has an overall higher speed than both rule-based driving policies, and a smaller variance in velocity over all the seeds. A 6% increase in overall speed may not seem significant, but the RL-based driving policy provides additional benefits over its rule-based counterparts. While rule-based driving policies suffer when exposed to the ambiguity associated with sensor noise, Section 5 demonstates that RL-based driving policies can learn even in very noisy environments.

3.3 Improving DQN training through target Q-network normalisation

DQN often suffers from training instabilities as covered in Section 3.1.2. By using the same neural network for the computation of the action-value and estimation of the target-value, the two values become correlated, resulting in an overestimation of the true return. Intuitively, it can be derived that the stability of the original DQN method improves when the target-value's estimation of the discounted return is improved.

This idea is the starting point for the contribution of a novel improvement to the DQN algorithm made in this thesis. A common practice in policy gradient methods is to standardise the discounted return during training before the associated network gradients are calculated. Mathematically this has the effect of controlling the variance of the policy's

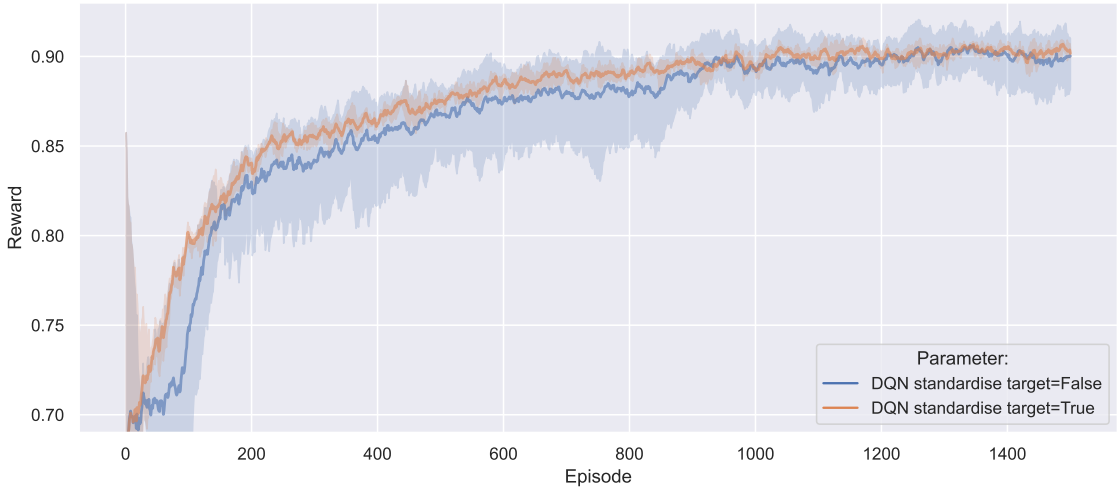


Figure 3.6: Comparison between the original DQN algorithm and the improved DQN algorithm through target Q value standardisation. Data is distributed over five random seeds, different to the seeds used for training of the DDQN model, and smoothed with an exponential moving average at a weight of 0.9.

gradient estimation and thus, improving training performance. The improvement made to the DQN method follows the same logic by standardising the target-values. Since the target-value (Q_T) is an approximation of the discounted return through

$$Q_{\mathrm{T}} = r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{a'} Q^*(\boldsymbol{s}', \boldsymbol{a}'), \tag{3.13}$$

by standardising the target Q value, the expected discounted return is indirectly standardised.

The training performance increase, especially in terms of stability, through the standardisation of the target Q value versus the original DQN algorithm, is illustrated in Figure 3.6 An important point to consider is that policy gradient methods are on-policy, and DQN operates off-policy. This means the standardisation of the target Q values is done on a uniformly sampled batch of past experiences. Although sampling is done uniformly, the drawn samples may not be representative of true distribution of past experiences. Batch sizes used were limited to 32 samples, but it could be that by increasing the batch size, even better performance could be observed in the improved DQN.

The overall reward maximisation of the improved DQN method is lower, but comparable to the DDQN algorithm. However, the true benefit needs further investigation on other common baseline environments. Unfortunately, there seems to be a decrease in training performance in terms of average vehicle speed per episode. This could be due to the fact that by standardising the target Q value on each batch, the estimation of the reward function's distribution is changed, resulting in a comparable overall reward while sacrificing performance with respect to certain components in the function itself. The additional standardisation at every training instance tends to slightly slow down training, but the improved DQN method has the benefit of increased stability without the need to instantiate another target network in memory as is needed for the DDQN algorithm. When working

on low-dimensional input spaces in small networks, the reduction in memory usage is negligible. However, for the training of very large neural networks, a reduction in memory usage may be very useful. Since this improvement to the DQN method is not the focus of this research it is not further investigated hereafter.

3.4 Conclusions

The goal of this chapter was to establish which of the Q-learning methods would provide the best performing baseline for comparison for the subsequent chapters. To this end, three improvements to DQN were investigated: DDQN, D3QN, and the PER buffer for better experience sampling. Each of these methods have shown promising increases in the performance on other baselines, but a comparison between these improvements when applied for the learning of high-level autonomous driving policies could not be found in literature. Each of the methods' hyper-parameters were tuned batch runs to ensure that the baselines would have the best performance possible. The parameter sweeps on the neural network's size showed the advantage of using an affordance-based state representation. From the low-dimensional state representation autonomous driving policies could be learned using small neural networks consisting of 2 fully-connected layers of 32 neurons. Results show that the commonly used DDQN method, with standard ER buffer, performs better than the other methods and is therefore used as the baseline for comparison.

When comparing the DDQN method to the well-known IBM & MOBIL policy and another rule-based driving policy, the DDQN RL controller proved to outperform both its rule-based counterparts in terms of the average speed achieved over various instantiations of the simulation environment. Finally, preliminary results for a novel improvement made to the DQN method through the standardisation of the target Q value was demonstrated. Through the standardisation of the target Q-value, the improved DQN method demonstrated lowest variance during training when compared to all the other methods. Further investigations of this improvement are not the focus of this thesis, but should be demonstrated on other baselines and further examined in the future.

The next section investigates neural network architectures that process the inputs to the networks different to that of the DDQN baselines. These include deep sets, which are able to learning from dynamically sized inputs and CNNs which can deal with the sparsity that exists in our state representation. These networks are all specifically investigated for their ability generalise well to new driving scenarios.

Chapter 4

Managing Measured State Complexities Through Permutation Invariance

The low dimensional state space, provided by affordance indicator state representations, offers numerous neural network training benefits. Nevertheless, they still contain some inefficiencies in the way they are represented, as described in more detail in Section 2.2. These inefficiencies generally arise because standard fully-connected neural networks are unable to process variably sized inputs. This chapter covers two main types of neural network architectures, namely Deep Sets and CNNs, that remedy some of these inefficiencies and have the potential to provide better generalisability of autonomous driving policies.

Deep sets have been shown to outperform fully-connected neural networks operating on a fixed input size, CNNs, and permutation invariant LSTMs also operating on variably sized inputs [69]. Performance was superior to all the methods in the maximisation of training rewards and in the ability to generalise to scenarios with different numbers of vehicles. It is not exactly certain whether this performance increase comes from the networks ability to process inputs of a dynamic size, because of its permutation invariant pooling function, or the combination of both. Nevertheless, incorporating permutation invariance into a network renders the ordering and number of vehicles in the state irrelevant, which can be helpful in learning.

CNNs can easily integrate pooling layers into the architecture to also render the state permutation invariant [21]. However, the aforementioned Deep Set research used an all-convolutional 2D CNN operating on an occupancy grid state representation, making it unclear whether adding pooling functionality to a CNN would assist in learning robust, generalisable autonomous driving policies. While CNNs still lack the ability to process dynamically sized inputs, they can learn efficiently despite any sparsity that may exist in the state matrix, resulting in performance increases over architectures like the DDQN baseline. Moreover, the shape and size of the convolutions significantly impact a network's ability to learn, resulting in benefits that Deep Sets might not be able to provide.

For these reasons, this chapter investigates Deep Set networks, several types of CNNs with and without permutation invariance, and compares their overall training and generalisability performance to the DDQN baseline of the previous section. The chapter is outlined

as follows. The fundamental basics of Deep Set and CNN architectures are covered in Sections 4.1 and 4.2, respectfully. These sections explain how these networks operate and why they promise improved generalisability. Section 4.2 specifically covers the integration of permutation invariance into a CNN architecture and the several variants tested in this chapter. Section 4.3 covers the chosen hyper-parameters of the networks and presents the training and generalisability performance of the methods when compared to the baseline DDQN algorithm with a standard fully-connected network processing inputs of a fixed size.

4.1 Deep Set network architectures

Deep set neural networks are able to process inputs of dynamic sizes, eliminating the need for the padding of the state when vehicles are outside the perceivable range of the ego-vehicle. They also incorporate a pooling function into the network, rendering the state input permutation invariant. Through these two mechanisms, the network's overall training performance and ability to generalise to new scenarios is improved. Figure 4.1 shows the form of a Deep Set network.

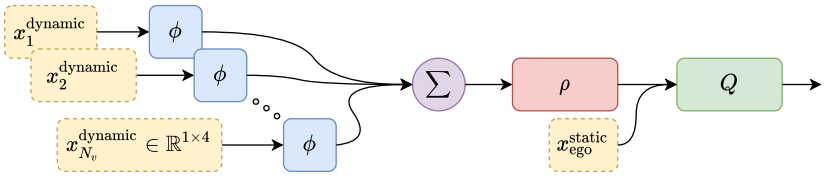


Figure 4.1: The implemented Deep Set network architecture, which is able to learn from variably sized inputs. It consists of ϕ, ρ, and Q fully-connected network components and a permutation invariant pooling function over the vehicle axis ($\sum$), which renders the order and number of vehicles in the state irrelevant.

The first difference between the Deep Set architecture and that of the DDQN baseline is that the input is split into dynamic $x_{N_v}^{\text{dyn}}$, and static components $x_{\text{ego}}^{\text{static}}$. The full dynamic state contains the lateral and longitudinal positions and velocities of vehicles relative to the ego-vehicle. It has a maximum size of $x_{N_v}^{\text{dyn}} \in \mathbb{R}^{N_v \times m}$, where the total number of vehicles $N_v = 12$ and the number of measurements per vehicle $m = 4$. The second difference is the ϕ network, which processes each of the measurements within the dynamic state matrix vehicle by vehicle, if measurements exist. Thereafter, the contributions of each vehicle passed through the ϕ network are summed by the pooling function over the vehicle axis. If no vehicles are within sensor range, the forward pass through the ϕ network is omitted, and a set of zeros is fed into the pooling function, such that there is no contribution from the dynamic state matrix. The dimensions at the output of the pooling function are then $(B \times 1 \times m \times \phi_f)$, where B is the batch size, and ϕ_f the number of features of the ϕ network. The permutation invariant pooling function used in this research is a simple summation, but maximum or average pooling functions can also be used. The summation pooling function is chosen because when working with low-dimensional inputs, it avoids any information loss that may occur when using functions like maximum pooling. The static state contains only measurement relative to the environment, such as to the road edge and is an array of the size $x_{\text{ego}}^{\text{static}} \in \mathbb{R}^{1\times 7}$.

4.2 Convolutional network architectures and the incorporation of permutation invariance

CNNs are able to efficiently learn from states that include sparsity by convolving over specific dimensions of the state space and aggregating their contributions using shared weights. These convolutions have the effect of smoothing the inputs, resulting in networks that are less prone to over-fitting than fully-connected networks.

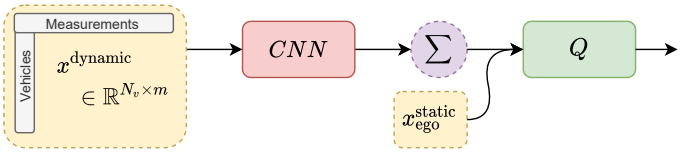


Figure 4.2: Generic depiction of the various 1D and 2D CNN architectures implemented. It consists of CNN and Q-network components, each with two hidden layers. Depending on the configuration, the CNN either performs 1D convolutions over the measurements or vehicles axis of the state, or 2D convolutions over both axes. Optionally included is a permutation invariant pooling function always over the vehicle axis ($\sum$), rendering the state's order and the number of vehicles irrelevant.

Five different CNN variations are implemented in this section, all of which have the basic form as depicted in Figure 4.2. The same decomposed state space is used as that of Deep Sets, described in Section 4.1 above. The only difference is that the state space is maintained at a fixed size and provided as is, with missing measurement data that may exist, to the CNN. Depending on the configuration, the outputs of the CNN are then passed through the summation pooling function over the vehicle axis of the feature space. The type of pooling operation is chosen for the same reasons as described for the Deep Set network. The specifics of each CNN and why they are implemented are listed below.

- *1D CNN convolving over the vehicle axis.* This network is chosen as a reference because it more matches the work of [69], which demonstrated the inability of CNNs to generalise as well as Deep Set networks. It performs convolutions over the entire vehicle axis, striding over the measurement axis. The transformation through the network is:

$$\text{input}(B \times 12 \times 4) \rightarrow \text{output}_{\text{CNN}}(B \times 1 \times 4 \times \text{features})$$

- *1D CNN convolving over the measurement axis.* This network transforms each vehicle's measurements into a higher-dimensional domain, striding over each of the vehicles one by one. It is implemented to establish if the addition of a pooling layer results in better generalisability. The transformation through the network is:

$$\text{input}(B \times 12 \times 4) \rightarrow \text{output}_{\text{CNN}}(B \times 12 \times 1 \times \text{features})$$

- *1D CNN convolving over the measurement axis with pooling.* The operations are the same as state above, but with the incorporation of the pooling function. Thus, it

renders the ordering and number of vehicles in the feature space irrelevant. This CNN is significant because, mathematically, it is the most similar to the Deep Set network and provides the closest comparable benchmark between Deep Set and permutation invariant convolutional networks.

$$\text{in}(B \times 12 \times 4) \rightarrow \text{out}_{\text{CNN}}(B \times 12 \times 1 \times \text{feat.}) \rightarrow \text{out}_{\text{SUM}}(B \times 1 \times 1 \times \text{feat.})$$

- *2D CNN convolving over the measurements and vehicle axis.* The perceptual field of the ego-vehicle can be divided into six sections depending on whether vehicles are located in front or behind, and by the lane they are in. The six sections are then, front left-lane, front current-lane, front right-lane, rear left-lane, etc. This 2D CNN is implemented with the intuition that even though two vehicles can be sensed in each of these sections, driving decisions are not necessarily made from the measurement of both of these vehicles. By performing 2D convolutions over measurement sets of two vehicles, and striding over each of the perceptual zones, the network can represent each zone as a separate feature space on which to make decisions. The transformation through the network is as follows:

  $$\text{in}(B \times 12 \times 4) \rightarrow \text{out}^{\text{layer 1}}_{\text{CNN}(2\times1)}(B \times 6 \times 4 \times \text{feat.}) \rightarrow \text{out}^{\text{layer 1}}_{\text{CNN}(1\times2)}(B \times 6 \times 2 \times \text{feat.})$$

- *2D CNN convolving over the measurements and vehicle axis with pooling.* The final CNN operates as above, but additionally incorporates a pooling function which renders the perceptual zones permutation invariant. Following on from the transformations above:

  $$\cdots \quad \text{out}^{\text{layer 1}}_{\text{CNN}(1\times2)}(B \times 6 \times 2 \times \text{feat.}) \rightarrow \text{out}_{\text{SUM}}(B \times 1 \times 2 \times \text{feat.})$$

4.3 Training and generalisability performance of permutation invariant architectures

The investigated networks are evaluated on their overall training performance and generalisability. However, they have only been tuned to maximise their training reward similarly as described for the DDQN baseline. The Q-networks are kept constant throughout the different architectures having the same (32×32) fully-connected structure as DDQN. The configuration space for the Deep Set network included: Relu, Elu, Tanh activation functions and network's of two and three layers of fully connected units with sizes between 16 and 64 neurons for the ϕ and ρ modules. Additionally, batch normalisation was tested at the output of the ρ network to re-scale the features between such that their magnitudes are comparable to the static inputs $x^{\text{static}}_{\text{ego}}$. The best performing configuration were ϕ and ρ network sizes of (32×64) and (64×32) with Relu activation functions, respectively. The 1D CNNs are configured similarly, resulting in two-layer CNNs with (32×64) numbers of filters. However, for the 2D convolutional network, the configuration space is established purely based on the intuition of the perceivable zones of the ego-vehicle. Dozens of other configurations are possible based on the kernel sizes, strides, and numbers of filters. However, the combinatorial tuning of these parameters was excluded because of the computation effort required to do so.

In order to make a fair comparison between Deep Set networks, CNNs and the fixed size DDQN, two different sets of training and generalisability are presented hereafter. The first results are when training is done in a scenario with 30 vehicles and the second with 70 vehicles. The testing of the networks' ability to generalise to unseen scenarios is performed by evaluating each of the networks on scenarios with a different number of vehicles than that used for training. The distribution between slow, medium and fast rule-based vehicles is kept constant, and only the total number of vehicles varies. Generalisability performance is by quantifying the average reward accumulated over 50 simulation episodes for each of the networks. These tests are distributed over 15 random seeds, different to those used for training. Testing on both training scenarios with few vehicles and with many vehicles ensures that the fixed-size fully-connected DDQN has some advantage over the Deep Set and CNNs, which are able to deal with missing measurement information more efficiently. However, it should be noted that the training scenario of 30 vehicles more closely matches that of realistic highway driving scenarios. The training scenario of 70 vehicles represents a densely packed highway moving at medium speed.

4.3.1 Training performance

The training performance of the 1D and 2D CNNs, Deep Set, and DDQN network architectures are displayed in Figure 4.3.

The training results between the architectures in scenarios with 30 vehicles are incredibly similar, with all of the networks obtaining a final reward, averaged over the five training seeds, of 0.92. The differences between the neural networks are less than 1% of their overall average reward throughout training, peak reward, and final rewards. From the graph, however, it can be noticed that intuitive 2D convolutions over different perceptual zones of the ego-vehicle can improve training performance. The only noticeable difference in the results is in terms of reward variance during training. The Deep Set network architecture had the lowest reward variance during training, and DDQN had the worst, 20% more than the Deep Set. CNN architectures have variances around 5% of that of the Deep Set network.

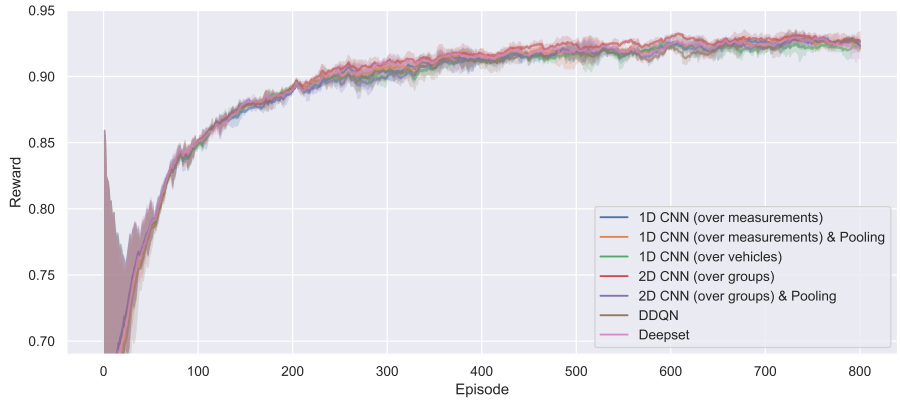
In training scenarios containing 70 vehicles, a more noticeable difference in training performance is noticed. Most noticeable is the decreased performance for the Deep Set and 2D CNN convolving over groups of vehicles when pooling is incorporated. Speculatively, the decrease in training performance of the 2D CNN with pooling and the Deep Set could be due to a bottleneck embedding caused by the pooling functions. However, the same phenomenon is not seen when pooling over the measurement axis of the state.

4.3.2 Generalisability performance

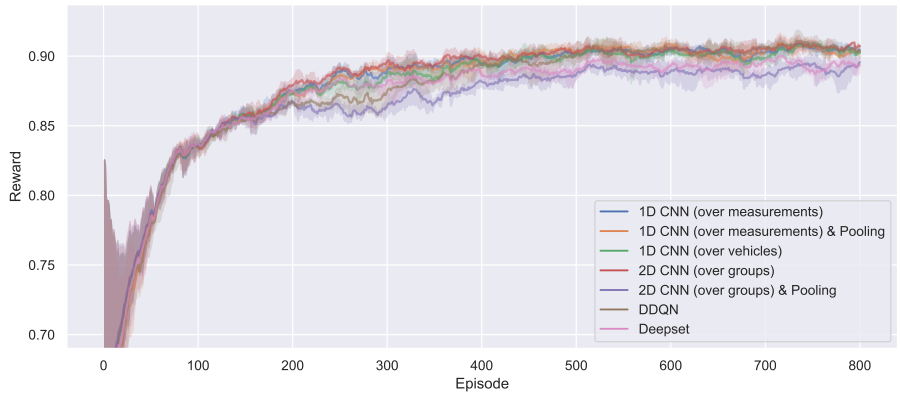
While training performance may be important, this chapter focuses on investigating these networks' ability to generalise to unseen scenarios. The comparative results are for both sets of training scenarios are presented in Figure 4.4.

The main discussion points relating to generalisability to scenarios with more and fewer vehicles are provided in lists to aid readability.

Generalising to scenarios with more vehicles than that of training (Subfigure 4.4a):



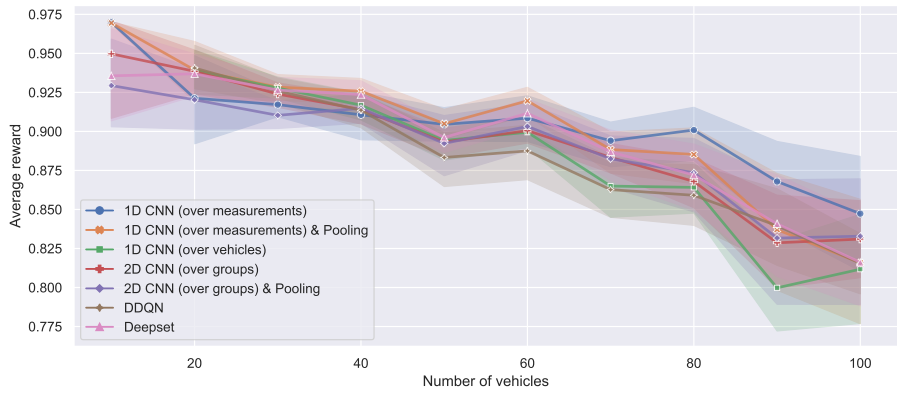
(a) Training scenario with 30 vehicles.



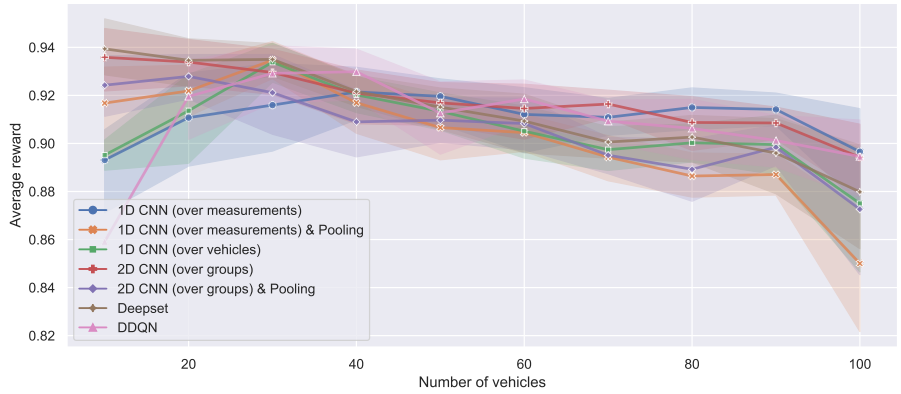
(b) Training on a scenario with 70 vehicles.

Figure 4.3: Overall training rewards for the baseline DDQN, Deepset, all-convolutional CNN, and permutation-invariant CNN neural network architectures on scenarios with 30 (above) and 70 (below) vehicles. Results are smoothed with an exponential moving average of 0.9, and distributed over 5 training seeds.

- Adding a pooling function into a 1D CNN convolving over the measurement axis out-performs the mathematically similar Deep Set architecture over the entire range of tested vehicles. The performance increase could arise from the CNN's smoothing effect to the measurements, which the fully connected ϕ layer of the Deep Set does not provide.
- In line with prior research, the Deep Set outperforms the CNN architecture convolving over the vehicle axis of the state [69], and this CNN does not generalise to scenarios with an increased number of vehicles.
- In general, adding a pooling layer into the CNNs results in an increased ability to generalise; however, this seems to depend on the axis of convolution and the direction to which generalisation is made. For fewer vehicles than that of training, pooling aids in generalisation for the 1D CNN, but it decreases performance for more vehicles. The opposite is true for the 2D CNN.



(a) Trained on a scenario with 30 vehicles.



(b) Trained on a scenario with 70 vehicles.

Figure 4.4: Generalisability of the Deep Set network, CNNs, and the DDQN baseline, in scenarios with a different amount of vehicles than the 30 (above) and 70 (below) vehicles used in training. The metric is the average reward per episode distributed over 15 seeds, different to that of training, presented without any smoothing.

- Across the board, the Deep Set and CNN's convolving over sections of the measurement axis outperform the fixed size fully connected DDQN because of their improved ability to handle the inefficiencies associated when learning from sparse input spaces.

Generalising to scenarios with more vehicles than that of training (Subfigure 4.4b):

- Almost all of the investigated networks perform worse than the DDQN baseline when generalising to scenarios with fewer vehicles than training. When training on a state input that does not frequently have missing measurement data, it makes sense to process it as a fixed-size with a fully connected network. In this scenario, the ability of CNNs to handle sparsity and of Deep Set networks to handle inputs of a variable size do not contribute to performance increases.
- When training on scenarios with many vehicles, adding pooling operations to CNNs almost always results in worse performance than all-convolutional networks. It is

hypothesised that the compression of the latent space by the pooling operation increases the training difficulty when inputs always contain measurements and results in policies that cannot generalise well.

- The 2D all-convolutional networks demonstrate the ability to generalise better to scenarios than the DDQN architecture. Up-scaling the low-dimensional affordance indicator representation seems to positively impact generalisability, especially when generalising to scenarios with more vehicles than training. This is especially the case for the 1D CNN convolving over the measurement axis of the state.

4.4 Conclusions

The results of this chapter demonstrated the ability of CNN, and Deep Set networks to generalise better than the fully connected DDQN baseline operating on fixed input sizes when trained on scenarios comparable to those present in normal highway situations. This ability to generalise is enabled by these networks' ability to train efficiently in environments where the state space contains sparsity. Deep Sets deal with this sparsity by processing variably sized inputs, while CNNs can filter over spare elements in the state to extract useful information for learning. Moreover, it has been shown that when generalising to scenarios with more vehicles, the addition of a pooling layer into a 1D CNN convolving over the measurement axis results in a network that can generalise better than the mathematically similar Deep Set network presented by Huegle [69].

When training on scenarios containing many vehicles, these networks do not benefit much over the DDQN baseline. However, in these cases, the measurement inputs rarely contain sparsity, and then processing inputs of a fixed size makes more sense than otherwise. The literature review of Chapter 1.1.4 highlights that most CNNs have only been investigated for occupancy grid state representations where the state often contains sparsity. While CNN's do not provide as much benefit against sparsity in low-dimensional affordance indicator state spaces, the results show that all-convolutional networks that upscale the input space provide good generalisability properties, even when using affordance indicator state representations. This is especially true when scaling to scenarios with more vehicles than that of training.

From these results, it is clear that the directionality of the convolutions and kernel sizes has a significant impact on the performance of CNNs. The absolute performance of CNNs' ability to generalise may be better quantified if all the hyper-parameters of these networks are tuned in a full combinatorial fashion. However, to make fair comparisons to prior research in the domain of Deep Set networks, only a few specific architectures are considered.

In the future, autonomous vehicles will potentially incorporate more and more information into their state space. These measurements may include distances and velocities of pedestrians and cyclists, states of traffic lights, among others. To cope with the growing dimensionality of future state inputs, neural networks will also need to be scalable. CNNs have already demonstrated immense scalability in computer vision and could potentially benefit future RL autonomous driving policies. This research showed the potential of Deep Set networks to enable generalisability over fixed-input DDQN methods; however, as state spaces grow, they may suffer from scalability problems [98].

Nevertheless, all the results presented here suggest that Deep Set and CNN architectures have the ability to generalise better than the commonly employed fully connected neural networks that process inputs of a fixed size. This is especially true when trained scenarios with reasonable amounts of vehicles, where the state input is not completely filled with measurement data. These networks have the potential to bring RL for autonomous driving closer to reality through an improved ability to generalise.

Chapter 5

Capturing Temporal State Information

The majority of research in the RL of autonomous driving policies focus on learning from single measurement instances containing the relative positions and velocities of surrounding vehicles. In doing so, the implicit assumption is made that there is no historical information in past states that may be beneficial to taking a optimal actions. This idea is essentially captured by the Markov property, that the current state perfectly captures all past information about how states have evolved over time. By excluding temporal information on the evolution of states in the position and velocity state space, agents are unable to capture information about the acceleration and jerk (the rate of change of acceleration) profiles of other vehicles. When training autonomous driving policies, it makes sense to exclude this information from the state input if it does not provide value to learning; thereby, we can keep the dimensionality of our state representation low and simplify the learning process. However, in research on the learning of autonomous highway driving policies, whether or not it is useful to capture this information is often not even considered.

RL has the potential to impact the progression of autonomous vehicle technology positively. Nevertheless, as highlighted in the introduction, it often struggles to bridge the gap between the simple simulation environments in which agents are trained in reality. Autonomous vehicles are complex mechatronic systems that do not fall perfectly into the Markovian assumption that is commonly made in RL. These systems exhibit properties of stochasticity in the way they behave to inputs and have elements of non-stationarity, where system dynamics may change over time due to the degradation of physical components. Moreover, the state information they provide in the form of measurements is often plagued with noise, resulting in a system more accurately described as a partially observable Markov process.

This chapter is dedicated to investigating these two ideas. Firstly, whether capturing the temporal dependencies between states using our position and velocity state representation provides any benefit to the learning of autonomous driving policies. Secondly, to see if these neural networks enable the learning of driving policies that are robust against the actual dynamics of real-world autonomous vehicles. To this end, two neural network architectures are explored, namely, LSTM networks and CNNs that convolve over the time domain, to investigate if they may aid in bridging the gap between simulation and reality and bring

RL closer to being practically implementable.

These networks are chosen because they can capture the temporal dependencies between states but through different mechanisms. LSTM networks are considered because of their ability to make long term planning decisions and cope in partially observable environments containing noise, both of which are valuable traits for learning autonomous driving policies. While CNNs cannot do any long-term planning, they do have the benefit of not suffering from training difficulties like the vanishing gradients of LSTM networks and have been shown to train much faster than LSTMs because of their relative simplicity. Traits and properties of the networks are described in more detail in Section 1.1.4 and have been omitted here for the sake of brevity.

The chapter is organised as follows. Section 5.1 highlights the changes made to the experience replay buffer, such that the networks can capture the temporal information between states. Section 5.2 covers how LSTM recurrent networks operate and how incorporating feedback into the network enables the benefits mentioned above. Section 5.3 explains the CNN architecture used in this research. Finally, results of the investigations into the ability of the networks to capture temporal information and train effectively in the presence of sensor noise are covered in Sections 5.3.1 and 5.3.2, respectfully.

5.1 Modification of experience replay buffer

Capturing the temporal evolution between successive states requires a modification to the experience replay buffer of the DDQN baseline. The replay buffer generally operates the same as the DDQN baseline for saving experiences and uniformly sampling from past experiences. Except when an experience gets sampled, we also aggregate the three prior experiences into a single sequence of states. When experiences are sampled at the beginning of an episode, and prior experiences are not available, the sampled experience is simply duplicated.

Several considerations are made into the number of past experiences sampled and the temporal spacing between the experiences. When sampling fewer states, the training time is slightly reduced because long sequences of experiences do not need to be constructed, resulting in lower-dimensional state spaces. However, a sampling sequence choice of four experiences is made. With four states containing only measurements on the relative positions of surrounding vehicles, it is possible to estimate their velocities, accelerations and jerk profiles as these are all derivatives of position in the time domain.

Samples are chosen at a temporal spacing equal to that of the decision frequency of the ego-vehicle, running a tenth slower than the simulation environment. The decision of the temporal spacing of states tends to be a difficult trade-off in autonomous driving scenarios. With smaller spacing between successive states, networks can better capture the dynamics of other vehicles and filter out noise that may exist in the state. However, when states are spaced very close together, the networks' ability to infer vehicles' overall behaviour and intention is reduced.

Successive states should ideally have a small temporal spacing, and the number of successive states should just be increased. However, this results in a larger state space, a possible increase in the network sizes needed to capture all this additional information, and additional computational requirements that are out of the scope of this work.

5.2 Recurrent LSTM networks

LSTMs are recurrent neural networks that overcome some of the training difficulties associated with recurrent networks operating on simple feedback loops. Each LSTM cell consists of: an input gate, a forget gate and an output gate, each with its own activation function. Each of the gates controls the formation, forgetting and retrieval of the cells hidden state, respectively. The hidden state essentially represents the memory of a single unit in the LSTM network.

This memory allows LSTMs to perform particularly well in partially observable environments. For example, when the true state of a measurement is corrupted by noise, the memory allows the network to infer the true state value through its memory component by looking at how the state has evolved over time. As mentioned, the memory also allows the network to capture temporal dependencies between states. It thus enables the network to learn about past information influenced current states, thereby enabling a sort of long-term planning. These capabilities are vital components associated with real-world driving scenarios and motivate further exploration of the network's benefits to learning of autonomous highway driving policies.

Figure 5.1 shows the basic structure of the LSTM network employed in this chapter. Its architecture consists of simply replacing the first hidden layer of the Q-network with a layer of LSTM units. The LSTM layer operates in a *many-to-one* configuration, where multiple past states are used as a warm-up only to return the last hidden state of each cell as a feature vector to the Q-network.

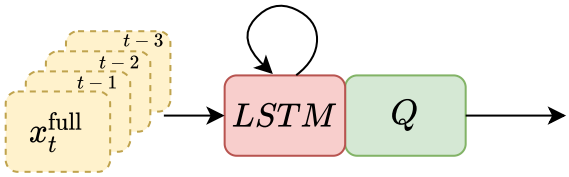


Figure 5.1: The LSTM network which processes inputs of four successive states x_t through x_{t-3}. It consists of the standard Q-network, with the first hidden layer replaced by a layer of LSTM units.

5.3 Convolutional neural networks that capture temporal information

CNNs that are able to capture temporal information operate in the same way as explained in the previous chapter. The only difference is that the networks perform convolutions over the time axis of the state. Different types of CNNs can be used to process inputs of this shape. However, a simple architecture performing 2D convolutions over each state is implemented because of the difficulties associated with tuning all the network hyper-parameters of 3D CNNs in a combinatorial fashion. The inputs of this 2D CNN have the dimensions of $(B \times N_v \times m \times T)$, where B is the batch size, N_v the number of vehicles, m the number of measurements, and T the sequences of past experiences. The last convolutional axis is normally represented by the RGB channels of pixels in vision

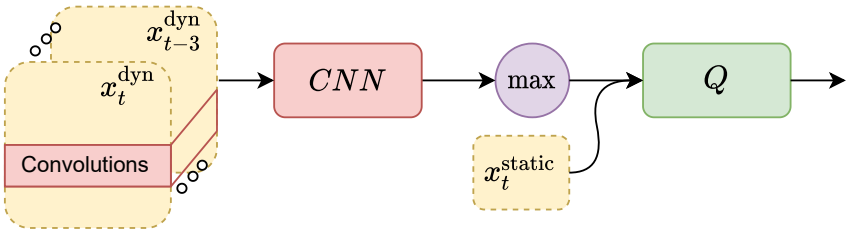


Figure 5.2: Depiction of the 2D temporal CNN convolving over the time domain. The network takes as an input a decomposed state, where measurements of other vehicles are presented as a 3D matrix of inputs x_t^{dyn} through x_{t-3}^{dyn}, and the measurements relating to the ego-vehicle are presented without any past historical data. The network incorporates a max-pooling function to reduce the dimensionality of the feature space.

applications but is now used as a temporal axis. The result is that a single feature of the 2D CNN shares weights along with all the other experiences in the time axis. The reduction of an axis dimension in the 2D CNN makes its tuning much simpler than the 3D CNN, reducing the latent space size that needs to be eventually flattened before being passed to the Q-network.

While CNNs do not have any memory like LSTMs, they have several other benefits in terms of training speed, flexibility in the way they can be configured, and they are relatively easy to train compared to LSTMs. Their ability to capture temporal information is not as well demonstrated in literature as that of LSTMs. However, their configurational flexibility is one of the main drivers for why they are chosen for investigation in this thesis.

As discussed in Chapter 4, they have the ability to incorporate permutation invariant pooling functions and tend to perform well at generalising to new scenarios. Moreover, they can efficiently deal with sparsity that may exist in the state space. These items are not further investigated in this chapter but may provide a valuable base for future work extensions.

5.3.1 Results on the usefulness of capturing temporal information between states

Networks that have the ability to capture evolutions in the time domain only provide real benefit when the Markov property is not a valid assumption. In other words, when the relative positions provide enough information for learning driving policies in highway scenarios. As described in Section 1.1.4, the majority of research that implements LSTMs and temporal CNNs involve end-to-end learning in scenarios very different to that of highway driving. For this reason, the first investigation is to determine if the addition of these networks improves training performance, knowing that standard highway driving scenarios, especially in simulation, have relatively low dynamics and can be approximated as quasi-static systems.

The training performance of the LSTM network, 2D CNN, and DDQN when learning

from the standard state representation of the baseline, containing the relative positions and velocities of other vehicles, is presented in Figure 5.3. The training scenario is the same as for the baseline DDQN method with 35 rule-based vehicles.

Figure 5.3: Training performance of LSTM, DDQN, and CNN architectures when including historic state information into the state. Results are smoothed with an exponential moving average and distributed over 5 random seeds.

The results demonstrate that simply replacing the first layer of the Q-network with a layer of LSTM units yields an increase in the training performance of the network over the DDQN baseline. Furthermore, it has the highest average reward over all the episodes and the lowest variance over the different training seeds. The 2D CNN performed poorly compared to the DDQN and LSTM networks. It is hypothesised that the is potentially due to the configuration space of the CNN. As discussed in the previous chapter, tuning on the 2D CNN by varying one hyper-parameter at a time does not result in an optimal configuration because of the dependencies that exist between each parameter. Moreover, a full combinatorial parameter sweep proved to be too computationally intensive to perform for this investigation. For comparative purposes, an addition is made to the standard DDQN baseline, such that it also processes past measurement information into its decision process. It demonstrates the efficiency of the LSTM to process and capture the dependencies between successive states. For the fully-connected DDQN, the additional temporal information only increases the state space size, thus decreasing overall performance.

From this set of results, there is only a marginal increase in performance between the LSTM utilising past measurement information and the standard DDQN baseline processing only single inputs. Therefore, an additional simulation is performed to establish any training performance degradation when all velocities of other vehicles are removed from the state. The results are presented in 5.4. Because the CNN did not perform better than the standard DDQN of the previous experiment, it is omitted from the figure to aid in readability.

It can be seen that even with state velocity components removed, the DDQN is slightly

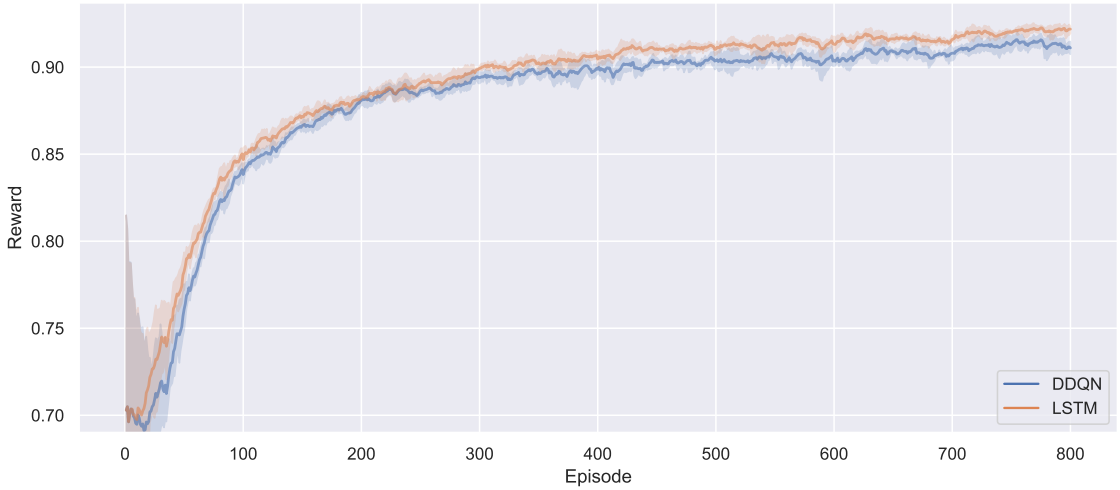


Figure 5.4: Training performance of LSTM network, and DDQN baseline, when velocity-based measurements are removed from the state input. Results are smoothed with an exponential moving average and distributed over 3 random seeds.

worse than it does when having velocity information available. However, the removal of the state velocity components does not result in an inability to learn. This suggests that the standard highway scenarios of the simulation environment are accurately approximated as quasi-static systems. Thus, the Markov assumption holds reasonably well, even with only relative positions to other vehicles in the state space. Additionally, without state velocity components, the performance increase of the LSTM network relative to the DDQN baseline is exaggerated. The LSTMs overall reward is extremely close to the previous set of results, demonstrating the ability of the network to capture the temporal dependencies between states and infer the velocities of other vehicles.

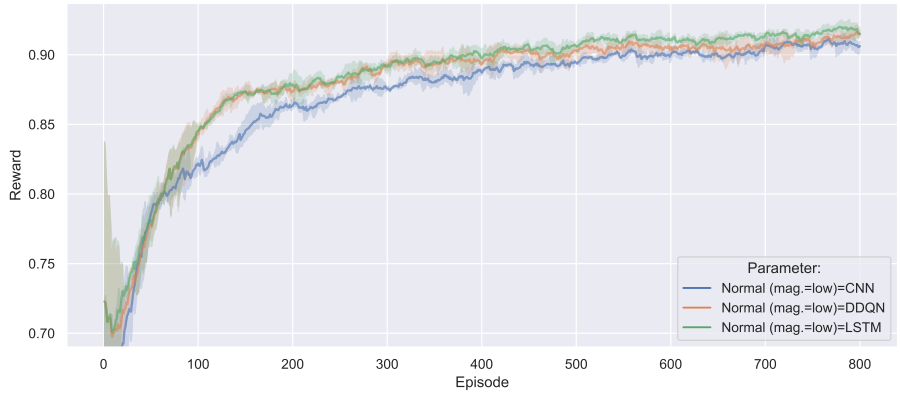
5.3.2 Results on the robustness to noise

Measurement noise is one of the factors that currently limits the practical applicability of RL in autonomous driving. Simulation environments are generally not complex enough to match that of real-world systems. As highlighted in Section 1.1.4, there have been some research efforts to try and quantify the robustness of trained policies against the presence of sensor-based noise. However, these investigations often model the noise as simple normally distributed functions. This is uncharacteristic of real-world noise sources, whose distributions may vary based on sensor type, present their distributions as multi-modal, or as a superposition of several distributions. An overlooked distribution of noise is uniformly-distributed white noise, which is present in many physical systems. Alone this noise is easily compensated for by simple averaging filters, but its filtering becomes more complex when superimposed with other distributions. LSTMs and CNNs have the potential to handle noisy environments better than the commonly-employed, fully-connected network architectures, where the MDP is more accurately represented as a POMDP.

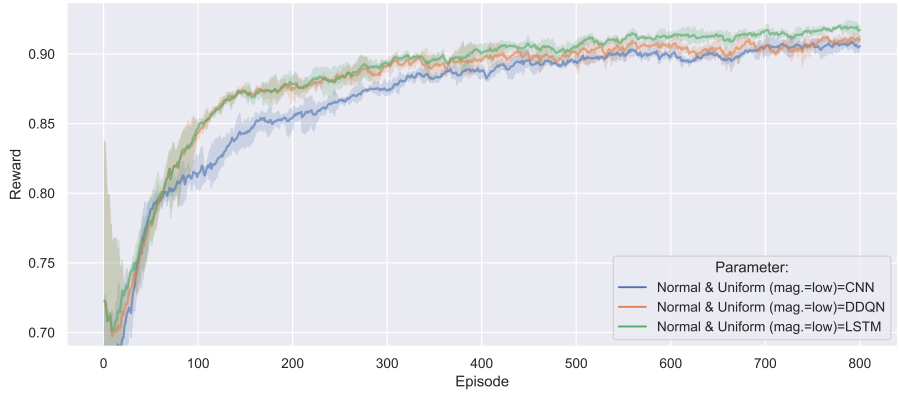
Two simulations are performed to highlight the degradation of the performance of the standard DDQN baseline, especially when faced with noise sources presented as the

superposition of normally and uniformly distributed noise sources. In the simulations, noise is added only to non-zero measurements in the state to accurately reflect a real-world perception module that can already distinguish true measurements from noise sources. For the sake of demonstrating the problems of prior research, the normally distributed noise is centred at a mean of $\mu = 0$ and has a standard deviation of $\sigma = 1/\sqrt{2}$. The uniformly distributed noise is limited to the range of $[-0.3, 0.3]$. Noise is added to the normalised measurement inputs such that when the uniform and normal noise sources are superimposed, they maximally distort the observed sensor value by 10% of its true value (on the edges of the normal distribution).

Figure 5.5a shows the training performance of the different networks in the presence of only normally distributed noise, and Figure 5.5b that of superimposed uniform and normal noise sources.



(a) Normally distributed noise.



(b) Superposition of uniformly and normally distributed noise.

Figure 5.5: Training performance between LSTM, 2D temporal CNN and DDQN network architectures in the presence of uniformly and normally distributed measurement noise.

From the graphs, it can be seen that the simple addition of an element of uniform noise to the normal noise, despite its minor contribution on the extreme ends of the resulting distribution, has a significant impact on the final reward reached. The CNN performs poorly in both cases, but it is suspected this is due to the max-pooling operating

in the architecture, which serves only to further capture noise on the extreme ends of the distribution. An average pooling function would most probably demonstrate an improved ability to train in the presence of sensor noise. Unfortunately, additional tuning of the CNN was not possible given the time restrictions of the thesis, but generally, CNN architectures would perform well in the presence of sensor noise because of the smoothing effect convolutional filters provide. The LSTM network, on the other hand, exhibits no change in its ability to train the presence of any of the noise sources, demonstrating the robustness of the network to deal with a partially observable environment like those found in real-world autonomous driving applications.

5.4 Conclusions

The purpose of the chapter was to provide concrete motivations for using neural networks that perform well in environments more closely matching those of real-world autonomous driving applications. Unfortunately, these considerations are commonly overlooked, and as discussed in Section 1.1.4 very few research efforts have been made into investigating their usefulness in learning autonomous driving policies containing traffic. Two neural networks were investigated, namely LSTM and CNNs and compared to the baseline DDQN architecture.

Results showed that simple highway simulation scenarios are not dynamic and can be accurately modelled as quasi-static environments. Because of this, even state representations containing only the relative positions of other vehicles are enough to render the training paradigm Markovian and not degrade the performance of the DDQN baseline by any significant margin.

Nevertheless, simply adding a memory component to the network through an LSTM layer increased overall training performance even when states contained both positions and velocities. The ability to make long term planning decisions enables better decision making than the DDQN baseline. LSTM networks also demonstrated the ability to cope in environments with partially observable states. This robustness to noise and ability to outperform the DDQN baseline despite the relatively low dynamics of the highway scenario of this thesis. In more dynamic and realistic driving scenarios, it can be argued that LSTMs would demonstrate an even more pronounced improvement to the simple scenario considered here. These points encourage the use of these networks in RL for autonomous driving in bridging the gap between simulation and reality.

CNNs proved not to provide many benefits in the performed tests. However, it is believed that this is primarily due to their impracticability. Unlike LSTM networks, they have many interdependent hyper-parameters that require significant effort to tune. Each of these parameters needs to be tuned in a full combinatorial fashion, each time requiring multiple training runs over different seeds. However, its believed that further exploration of these architectures is needed. They have the potential to handle noise effectively through the smoothing property of convolutional filters and capture the temporal information between states. They are trained quickly and do not suffer from some of the training difficulties of LSTMs, like vanishing gradients. They have the architectural flexibility to easily incorporate elements like pooling layers and have been shown in Chapter 4 to generalise well.

Chapter 6

Conclusion

Reinforcement learning has the potential to positively impact the field of autonomous driving because it enables learning in safe simulation environments where training data is easy to collect. However, its practical applicability is limited by a few factors. One of the largest gaps impeding its use is the gap between simulation environments and reality. RL policies trained in simulation settings are often unable to scale to real-world applications because they cannot take the same complexities as those present in real-world systems into account. Autonomous vehicles are complex mechatronic systems which have elements of stochasticity in the way they behave to inputs, exhibit properties of non-stationarity reflecting the degradation of physical components over time, and return partially observable states because of the presence of sensor-based noise. Moreover, real-world driving scenarios are dynamic and complex, and cannot always be accurately represented in a simulation environment.

These points are often neglected in literature. For these reasons several neural network architectures are investigated that have the potential to bridge the gap between simulation and real-world autonomous driving settings. Each of these potentially provide the ability to either improve the generalisability of learned policies or learn in the presence of the complex characteristics of real world systems.

6.1 Summary

6.1.1 Simulation environment

This thesis firstly covered the learning setting used for the learning of high-level autonomous driving policies contained in Section 2. The established driving scenario consists of a three lane highway, containing several other vehicles with rule-based driving policies. The state space of the learning environment is represented through affordance-based indicators provided by a perception module. The perception module provides lateral and longitudinal positions and velocities relative to the surrounding vehicles. The state of the environment provided by the perception module has a low dimensionality which makes learning from this representation very beneficial. However, it still carries some inefficiencies. The commonly used fully-connected networks in autonomous driving literature can only operate fixed size inputs, and thus when measurements are outside the perceivable range of the ego-vehicle, the measurements are padded with zeros. Additionally, when vehicles cross lane markings

or leave the perceivable zone, this padding causes discontinuities in the state space which can unnecessarily increase learning complexity. The last topic covered is the shaping of the reward function according to European driving conversions. The scalar learning reward is represented with four terms to ensure that learned policies maintain a good highway speed, do not make unnecessary lane changes, keep in the right lane if possible, and maintain safe following distances.

6.1.2 Establishing a Q-network baseline for comparison

Establishing a strong baseline for comparison to the implemented neural networks is a top priority to ensure that any results obtained are indeed representative. The DQN algorithm is investigated in Chapter 3 along with several improvements that have shown benefit in other learning environments. DDQN, D3QN and the improved experience sampling meta-algorithm PER, are all rigorously tuned and evaluated relative to each other.

Results show that despite the potential of these methods, the commonly used DDQN algorithm with standard ER buffer performs the best in the scenario considered in this thesis. In terms of absolute performance to well established metrics, the DDQN method is evaluated against the IBM & MOBIL and other rule-based policies in the simulation environment and manages to maintain higher average velocities than the other driving policies. The final investigation highlights some preliminary results in improving the standard DQN method through the standardisation of target-Q values. Through this standardisation the DQN method's stability during training is shown to surpass that of the DDQN algorithm.

6.1.3 Investigation of Deep Set and convolutional neural networks for improved generalisability

Chapter 4 looks at Deep Sets and several variants of CNN architectures and evaluates them on their ability to generalise to new scenarios. The chapter focused on building on prior research [69], which showed that a Deep Set architecture enables the learning from a variably sized input space and though pooling, rendering the state permutation invariant with respect to the number of and order of vehicles. The research showed that Deep Set networks outperform fixed-fully connected networks, all-convolutional neural networks, and permutation invariant LSTMs. However, they neglected the fact that by adding pooling functions to the CNN the state can also be rendered permutation invariant.

Results illustrate that by appropriately setting up a all-convolutional CNN it is possible to create a network, mathematically similar to that of Deep Sets with even better generalisability. The results also suggest that all-convolutional CNNs also show an interesting ability to generalise, however CNNs are rarely investigated when affordance indicator state representations because of its already low dimensionality. The type of convolution (size and direction) seems to have a large impact on the generalisability but the best configuration space of the networks hyper-parameters made optimal tuning of the networks outside the scope of this thesis. Almost all considered CNNs with and without permutation invariant pooling functions and Deep Set networks performed better than the standard DDQN baseline, providing promising results for future work.

6.1.4 Networks that can capture temporal state information

Networks that capture temporal information have the potential to deal better with training scenarios closer reflecting the true nature of real world systems. Chapter 5 investigates LSTM networks and temporal CNNs convolving over the time domain in their ability to better deal with partially observable states that contain measurement noise and make decisions based how states evolve over time.

However, capturing temporal information on the evolution of states only provides benefit if the state provided to the agent does not fully represent all the information needed for learning. In other words, when adding information about how the relative positions and velocities of other vehicles does not contribute to learning.

The first set of results presented demonstrates that the standard highway scenario used in this thesis is accurately described as a quasi-static system because of the relatively low dynamics involved. Even when removing velocities from the state vector the standard DDQN method is able to learn autonomous driving policies. However, despite the fact that state representations containing positions and velocities of other vehicles provide sufficient information for the Markov assumption to hold, incorporating a single LSTM layer into Q-network increases overall training performance. LSTM networks enable long-term planning decisions and general inference on the intent of other vehicles during simulation. Both of which are valuable in the learning of autonomous driving policies.

A final investigation is presented demonstrating that commonly performed tests made on the robustness of policies to normally distributed noise, are not sufficiently difficult or representative of noise profiles present in real sensor measurements. While standard DDQN architectures composed of fully-connected layers demonstrate the ability to learn in the presence of sensor noise, their training performance degrades when noise profiles are made more realistic. This is not the case for the LSTM network, further motivating the use of LSTM networks, which are practically implementable and easy to tune unlike temporal CNN architectures.

6.2 Future work

The most important items for future work are:

- Further investigation into the effect of standardisation of the target Q-value in the original DQN algorithm could be made. Future research should also look into different batch sizes, the performance of the improvements on other benchmarks, and whether it would be extendable to the DDQN algorithm.

- More broadly, different neural networks outside of autonomous driving like transformers and encoder-decoder networks could be investigated and implemented to see if they could have a positive impact on the learning of autonomous driving policies. These networks would also have the ability to process inputs of a variable sizes and may prove to enable better generalisation of driving policies.

- An evaluation of the true benefits of capturing the temporal information between states should be made, especially on more dynamic driving scenarios that are more

representative of real-world driving scenarios. An example would be scenarios where agents contain elements of stochasticity in their behaviours.

- The preliminary results of this thesis are promising, and further investigations should be made on combining the methods presented in this thesis. For example combining Deep Set and LSTM networks or CNN and LSTM networks. These networks may serve in providing robustness to noise while simultaneously enabling generalisability and helping to make long-term planning decisions.

Bibliography

[1] Geneva: World Health Organization, "Global status report on road safety 2018," tech. rep., 2018.

[2] S. Chen, M. Kuhn, K. Prettner, and D. E. Bloom, "The global macroeconomic burden of road injuries: estimates and projections for 166 countries," *The Lancet Planetary Health*, vol. 3, no. 9, pp. e390–e398, 2019.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[4] G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *IEEE Signal Processing Magazine*, vol. 29, pp. 16–17, 11 2012.

[5] A.-r. M. Alex Graves and G. Hinton, "Speech Recognition with Deep Recurrent Neural Networks , Department of Computer Science, University of Toronto," *Department of Computer Science, University of Toronto*, vol. 3, no. 3, pp. 45–49, 2013.

[6] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, vol. 4, no. January, pp. 3104–3112, 2014.

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–15, 9 2014.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks Alex*, vol. 25. Chapman and Hall/CRC, 5 2012.

[9] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning," *AAAI Conference on Aritifical Intelligence*, 2017.

[10] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, "Concrete Problems in AI Safety," pp. 1–29, 6 2016.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning - An Introduction*. Cambridge, Massachusetts: The MIT Press, second edi ed., 2018.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," pp. 1–9, 2013.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[14] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, no. 2015, pp. 2140–2146, 2017.

[15] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with Large Scale Deep Reinforcement Learning," 12 2019.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. A. Nature, and U. 2017, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2016.

[17] J. García and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, pp. 1437–1480, 2015.

[18] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of Real-World Reinforcement Learning," 4 2019.

[19] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," *Advances in Neural Information Processing Systems*, vol. 2017-Decem, no. ii, pp. 3392–3402, 2017.

[20] O. Vinyals, S. Bengio, and M. Kudlur, "Order matters: Sequence to sequence for sets," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–11, 2016.

[21] C. J. Hoel, K. Driggs-Campbell, K. Wolff, L. Laine, and M. J. Kochenderfer, "Combining Planning and Deep Reinforcement Learning in Tactical Decision Making for Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 5, no. 2, pp. 294–305, 2020.

[22] J. C. Krafcik, "Waymo is opening its fully driverless service to the general public in Phoenix," 2020.

[23] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[24] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," *International conference on machine learning*, 8 2015.

[25] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, pp. 1735–1780, 11 1997.

[26] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, vol. 28, 2013.

[27] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," *30th International Conference on Machine Learning, ICML 2013*, no. 3, pp. 2176–2184, 2013.

[28] P. Sermanet, S. Chintala, and Y. LeCun, "Convolutional Neural Networks Applied to House Numbers Digit Classification," *Proceedings - International Conference on Pattern Recognition*, pp. 3288–3291, 4 2012.

[29] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[30] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *International Conference on Learning Representations*, 9 2015.

[31] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust Region Policy Optimization," *32nd International Conference on Machine Learning, ICML 2015*, vol. 3, pp. 1889–1897, 2 2015.

[32] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," *33rd International Conference on Machine Learning, ICML 2016*, vol. 6, pp. 4135–4148, 3 2016.

[33] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang, "Solving Rubik's Cube with a Robot Hand," pp. 1–51, 10 2019.

[34] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-End Training of Deep Visuomotor Policies," *Journal of Machine Learning Research*, vol. 17, pp. 1–40, 4 2015.

[35] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection," *Springer Proceedings in Advanced Robotics*, vol. 1, pp. 173–184, 3 2016.

[36] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm," *arXiv*, pp. 1–19, 12 2017.

[37] Z. Xu, J. Chen, and M. Tomizuka, "Guided Policy Search Model-based Reinforcement Learning for Urban Autonomous Driving," vol. 28, 5 2020.

[38] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 7579–7586, 2018.

[39] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-Based Reinforcement Learning via Meta-Policy Optimization," no. CoRL, 2018.

[40] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-Ensemble Trust-Region Policy Optimization," pp. 1–15, 2 2018.

[41] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," *35th International Conference on Machine Learning, ICML 2018*, vol. 5, pp. 2976–2989, 1 2018.

[42] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine Learning*, vol. 8, no. 3, pp. 229–256, 1992.

[43] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2 2016.

[44] Y. Wang, H. He, X. Tan, and Y. Gan, "Trust Region-Guided Proximal Policy Optimization," *Advances in Neural Information Processing Systems*, vol. 32, 1 2019.

[45] J. N. Tsitsiklis and B. Van Roy, "An Analysis of Temporal-Difference Learning with Function Approximation," *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, vol. 42, no. 5, pp. 674–690, 1997.

[46] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 3215–3222, 2018.

[47] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Frcitas, "Dueling Network Architectures for Deep Reinforcement Learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, no. 9, pp. 2939–2947, 2016.

[48] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–21, 2016.

[49] S. Nageshrao, H. E. Tseng, and D. Filev, "Autonomous highway driving using deep reinforcement learning," *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, vol. 2019-Octob, no. March, pp. 2326–2331, 2019.

[50] Z. Bai, B. Cai, W. Shangguan, and L. Chai, "Deep Reinforcement Learning Based High-level Driving Behavior Decision-making Model in Heterogeneous Traffic," *Chinese Control Conference, CCC*, vol. 2019-July, pp. 8600–8605, 2 2019.

[51] M. Vitelli, "CARMA : A Deep Reinforcement Learning Approach to Autonomous Driving," *Tech. rep. Stanford University*, pp. 1–8, 2016.

[52] A. Alizadeh, M. Moghadam, Y. Bicer, N. K. Ure, U. Yavas, and C. Kurtulus, "Automated Lane Change Decision Making using Deep Reinforcement Learning in Dynamic and Uncertain Highway Environment," *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pp. 1399–1404, 2019.

[53] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2156–2162, IEEE, 11 2018.

[54] S. Wang, D. Jia, and X. Weng, "Deep Reinforcement Learning for Autonomous Driving," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12109 LNAI, pp. 67–78, 11 2018.

[55] X. Xiong, J. Wang, F. Zhang, and K. Li, "Combining Deep Reinforcement Learning and Safety Based Control for Autonomous Driving," *arXiv*, pp. 1–10, 12 2016.

[56] M. Kaushik, V. Prasad, K. M. Krishna, and B. Ravindran, "Overtaking Maneuvers in Simulated Highway Driving using Deep Reinforcement Learning," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2018-June, pp. 1885–1890, 2018.

[57] J. Chen, B. Yuan, and M. Tomizuka, "Model-free Deep Reinforcement Learning for Urban Autonomous Driving," *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pp. 2765–2771, 4 2019.

[58] D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev, "Driving in Dense Traffic with Model-Free Reinforcement Learning," pp. 5385–5392, 9 2019.

[59] B. Mirchevska, M. Hügle, G. Kalweit, M. Werling, and J. Boedecker, "Amortized Q-learning with model-based action proposals for autonomous driving on highways," *arXiv*, 2020.

[60] P. Wang and C.-Y. Chan, "Formulation of Deep Reinforcement Learning Architecture Toward Autonomous Driving for On-Ramp Merge," *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-March, pp. 1–6, 9 2017.

[61] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-End Race Driving with Deep Reinforcement Learning," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2070–2075, 2018.

[62] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "End-to-End Deep Reinforcement Learning for Lane Keeping Assist," *Machine Learning for Intelligent Transportation Systems Workshop*, pp. 1–9, 12 2016.

[63] C. Chen, A. Seff, A. Kornhauser, and J. Xiao, "DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, pp. 2722–2730, 5 2015.

[64] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical Decision Making for Lane Changing with Deep Reinforcement Learning," *Neural Information Processing Systems (NIPS)*, no. Nips, pp. 1–10, 2017.

[65] L. Fridman, J. Terwilliger, and B. Jenik, "DeepTraffic: Crowdsourced Hyperparameter Tuning of Deep Reinforcement Learning Systems for Multi-Agent Dense Traffic Navigation," *arXiv*, 1 2018.

[66] P. Wolf, K. Kurzer, T. Wingert, F. Kuhnt, and J. M. Zöllner, "Adaptive Behavior Generation for Autonomous Driving using Deep Reinforcement Learning with Compact Semantic States," *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2018-June, no. June, pp. 993–1000, 2018.

[67] B. Mirchevska, M. Blum, L. Louis, J. Boedecker, and M. Werling, "Reinforcement learning for autonomous maneuvering in highway scenarios.," *Workshop for Driving Assistance Systems and Autonomous Driving*, pp. 32–41, 2017.

[68] A. Author and A. Address, "Towards Practical Hierarchical Reinforcement Learning for Multi-lane Autonomous Driving," no. Nips, pp. 1–7, 2018.

[69] M. Huegle, G. Kalweit, B. Mirchevska, M. Werling, and J. Boedecker, "Dynamic Input for Deep Reinforcement Learning in Autonomous Driving," *IEEE International Conference on Intelligent Robots and Systems*, pp. 7566–7573, 2019.

[70] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for Simplicity: The All Convolutional Net," *3rd International Conference on Learning Representations, ICLR 2015 - Workshop Track Proceedings*, pp. 1–14, 12 2014.

[71] P. Wolf, C. Hubschneider, M. Weber, A. Bauer, J. Hartl, F. Durr, and J. M. Zollner, "Learning how to drive in a real world simulation with deep Q-Networks," *IEEE Intelligent Vehicles Symposium, Proceedings*, no. Iv, pp. 244–250, 2017.

[72] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," *30th International Conference on Machine Learning, ICML 2013*, pp. 2347–2355, 11 2012.

[73] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pp. 1724–1734, 6 2014.

[74] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, . Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," pp. 1–23, 2016.

[75] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional Sequence to Sequence Learning," *34th International Conference on Machine Learning, ICML 2017*, vol. 3, pp. 2029–2042, 5 2017.

[76] K. M. Kitani, D.-A. Huang, and W.-C. Ma, "Activity Forecasting," in *Group and Crowd Behavior for Computer Vision*, pp. 273–294, Elsevier, 2017.

[77] X. Pan, Y. You, Z. Wang, and C. Lu, "Virtual to real reinforcement learning for autonomous driving," *British Machine Vision Conference 2017, BMVC 2017*, 2017.

[78] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3803–3810, IEEE, 5 2018.

[79] A. Ray, J. Achiam, and D. Amodei, "Benchmarking Safe Exploration in Deep Reinforcement Learning," *arXiv preprint*, pp. 1–6, 2019.

[80] B. Eysenbach, S. Gu, J. Ibarz, and S. Levine, "Leave no Trace: Learning to Reset for Safe and Autonomous Reinforcement Learning," *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 11 2017.

[81] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained Policy Optimization," *34th International Conference on Machine Learning, ICML 2017*, vol. 1, pp. 30–47, 5 2017.

[82] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, "Safe Exploration in Continuous Action Spaces," 1 2018.

[83] A. Baheri, S. Nageshrao, H. E. Tseng, I. Kolmanovsky, A. Girard, and D. Filev, "Deep Reinforcement Learning with Enhanced Safety for Autonomous Highway Driving," *IEEE Intelligent Vehicles Symposium, Proceedings*, pp. 1550–1555, 10 2019.

[84] S. Nageshrao, H. E. Tseng, and D. Filev, "Autonomous highway driving using deep reinforcement learning," *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, vol. 2019-Octob, pp. 2326–2331, 2019.

[85] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An Open Urban Driving Simulator," pp. 1–16, 11 2017.

[86] A. Santara, S. Rudra, S. A. Buridi, M. Kaushik, A. Naik, B. Kaul, and B. Ravindran, "MADRaS : Multi Agent Driving Simulator," *Journal of Artificial Intelligence Research*, vol. 70, pp. 1517–1555, 10 2020.

[87] P. A. Lopez, E. Wiessner, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flotterod, R. Hilbrich, L. Lucken, J. Rummel, and P. Wagner, "Microscopic Traffic Simulation using SUMO," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, vol. 2018-Novem, pp. 2575–2582, IEEE, 11 2018.

[88] E. Leurent, "An Environment for Autonomous Driving Decision-Making," 2018.

[89] M. Treiber, A. Hennecke, and D. Helbing, "Congested Traffic States in Empirical Observations and Microscopic Simulations," *Physical Review E - Statistical Physics, Plasmas, Fluids, and Related Interdisciplinary Topics*, vol. 62, pp. 1805–1824, 2 2000.

[90] A. Kesting, M. Treiber, and D. Helbing, "MOBIL : General Lane-Changing Model for Car-Following Models," *Disponvel Acesso Dezembro*, 2016.

[91] P. Polack, F. Altche, B. D'Andrea-Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, no. April 2018, pp. 812–818, IEEE, 6 2017.

[92] J. L. McClelland, B. L. McNaughton, and R. C. O'Reilly, "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory.," *Psychological review*, vol. 102, pp. 419–457, 7 1995.

[93] J. ONeill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari, "Play it again: reactivation of waking experience and memory," *Trends in Neurosciences*, vol. 33, pp. 220–229, 5 2010.

[94] L.-j. Lin, *Reinforcement learning for robots using neural networks*. PhD thesis, 1992.

[95] H. Van Hasselt, "Double Q-learning," *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010, NIPS 2010*, pp. 1–9, 2010.

[96] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-Learning," *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pp. 2094–2100, 2016.

[97] B. Leemon C, "Advantage Updating," tech. rep., Wright-Patterson Air Force Base, 1993.

[98] E. Wagstaff, F. B. Fuchs, M. Engelcke, I. Posner, and M. Osborne, "On the Limitations of Representing Functions on Sets," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 11285–11298, 1 2019.