Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Lane Changing in Autonomous Vehicles based on Data Sharing

Vyshnavi Goli

August 19, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of

Master of Science in Computer Science - Data Science

Declaration

I hereby declare that this dissertation is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at
`http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: Date:

Acknowledgements

To my sister Tejaswi, for supporting me emotionally and unconditionally throughout my master's, my mum for always motivating me and pushing me to my limits and beyond all the time. To my supervisor, Prof. Ivana Dusparic and Jernej Hribar, without whose guidance this process of research would not be successfully completed. Finally, to my family for always being there and supporting me.

Thank you so much !!!

Vyshnavi Goli

Trinity College Dublin

August 2022

Abstract

Autonomous vehicles also called 'driver-less' vehicles or self-driving vehicles are increasing constantly and the world, with the introduction of the Tesla auto-pilot system, entered into Level 2 automation of cars. According to it, the vehicle is able to take full control of the system, and handle acceleration, braking, and steering at the same time. This needs some communication to be necessary between the vehicles for the purpose of decision-making. The dissertation uses Reinforcement learning (RL), a branch of machine learning dealing with the creation of smart agents, to specifically address the issue of lane changing in connected autonomous vehicles (CAV). Connected autonomous vehicles (CAV) combine connectivity and automation to aid or replace people in the task of driving. This is achieved using sensor technology, Global Positioning System (GPS), remote processing capabilities, and telecommunication systems. Performing safe and efficient lane changing is a crucial part of automating these vehicles and the RL intelligent agent selects and executes the action by perceiving its surroundings using the data from the sensors, GPS, and other components of the vehicle. It can also be trained using several behavior patterns of drivers thus eliminating driver errors, congestion, traffic accidents, etc.,This thesis mainly focuses on developing a Lane changing strategy using a Deep Q learning Agent.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis addresses the Vehicle to Vehicle communication in Connected Autonomous Vehicles (CAV) and uses it for developing a Lane changing strategy in CAVs using Deep Reinforcement Learning (DRL). It suggests using DRL to combat the dimensionality curse that Reinforcement Learning experiences [6] and uses neural network techniques which are more effective at solving complex, huge issues. This is suitable for the present dynamic environment where there are multiple entities to interact with. The code is available at `https://github.com/GoliVyshnavi/Lane_Change_Thesis`

1.1 Motivation

There are many applications that can be developed by using the data that is shared between two vehicles. A lane change application, intersection management, adaptive cruise control etc., The motivation behind choosing development of lane change strategy is , according to statistics, out of all the crashes that occur due to lane changing, ramp area is where most of the collisions occur. Among them, there are two types of collisions, rear-end and side swipe

collisions, which constitute 12.5 and 85.5 per cent respectively [7]. The former occur due to the target vehicle not being to pick up speeds on par with its surroundings and latter occurs when because of the vehicle that made a lane change is not in a desired position on the lane. Inspired to eliminate these, the main goal of this thesis is to remove both of these collision and create a more efficient strategy that focuses on the safety as well as the efficiency.

1.2 Lane Change in CAV

Vehicle to Vehicle (V2V) is Inter-Vehicle communication and happens when both the vehicles that wants to communicate are present in a specified range from each other. Especially in CAVs, obtaining and processing the information from surrounding vehicles and sensors acts as an advantage for the ego vehicle as it can use it in variety of ways.

Various scenarios needs to be automated as part of CAV development, and some of them are Lane changing, intersection management, issuing warnings, navigating and remote driving [8].

Safety, Comfort, Efficiency, Mobility, and Environmental applications are some of the areas which can be improvised using the data acquired from other vehicles surrounding the ego vehicle and the ego vehicle itself. For enhancing safety and comfort, automated vehicles should be prevented from colliding with each other and monitoring of sudden changes in acceleration using different mechanisms must be done.

This evokes a need for information such as acceleration, position of the vehicles, speed of the vehicles and others as they are needed for taking important decisions while automating a connected autonomous vehicle (CAV).

1.3 Co-operative Lane Change

Co-operative Lane changing is where vehicles collaborate to perform a lane change of one or a group of cooperative vehicles in a safe and efficient manner [9]. It is comparatively new and involves complicated scenarios to be tested but it is necessary for preventing collisions and also for improving overall lane change efficiency. Again, data from other vehicles is of extreme importance for developing it.

Especially while designing a lane change strategy, it must be decided beforehand what kind of co-operative lane change must be performed. During multi-lane-change, other vehicles may need to change lane due to reasons like as part of the ego vehicle's cooperative lane change or speed gain etc.,

1.4 Thesis Assumptions

Although various background works were examined in Chapter 2 that targets CAVs, mix of CAVs and Human driven vehicles (HDV) etc., This thesis implements interaction with only Connected Autonomous Vehicles, and no pedestrian paths. The speeds of the vehicles are assumed to be same for all and constant through out the journey. The target vehicle i.e., the agent is surrounded a specified number of vehicles as in chapter 5.4.2.

1.5 Thesis aims and objectives

The thesis' overall purpose is to address the lane changing in connected autonomous vehicles using data that is shared between them leveraging the use of Vehicle to Vehicle communication in CAVs. An application of it is the Lane changing strategy which is developed aiming to not only improve the safety, comfort of the passengers but also enhance the efficiency of the lane change

using Reinforcement Learning. Deep Q Network based system with optimised state space and reward function is used for this purpose as a DRL agent can make intelligent decisions when configured and learns based on the scenarios it experiences continously. The CAV agent involved has a foresight that makes the decisions efficient even in unforeseen scenarios as this strategy not only uses the data from the ego vehicle itself but also considers information from the surrounding vehicles.

1.6 Thesis Contribution

With safety, efficiency, and comfort as the primary considerations, this thesis identifies and justifies the necessity for an improved DQN-based Lane change approach. It seeks to address the impact of providing a DQN network with more information as well as enhance the reward functions already in place. It outlines the difficulties in implementing lane changes and then suggests solutions based on those difficulties. The design, implementation, and evaluation of a DQN method are the key contributions of the thesis, and unlike other DRL-based techniques, the simulation takes into account various driving styles of other nearby vehicles. The agent learns from its experiences using Experience replay memory, which helps it avoid the instability brought on by the environment's non-stationarity. The agent is taught on its own DRL-based policy. The technique is evaluated in a simulation of traffic and lane change in this thesis's last section. Evaluation results demonstrate superior performance of the modified DQN agent over the agent with less local knowledge.

1.7 Document Structure

The thesis is organized as follows. The cutting-edge research on non-reinforcement learning approaches, advanced reinforcement learning techniques, and deep

learning techniques utilized for the suggested work are explained in Chapter 2. The backdrop for them is mostly focused on RL algorithms and other approaches. It provides lane change techniques for CAVs using single agents as well as multi-agent systems. The planned research's design is described in Chapter 3. The execution of the architecture described in Chapter 3 is covered in Chapter 4. Chapter 5 discusses the evaluation of the proposed study using a single-agent DQN-based methodology. In Chapter 6, this theory is outlined along with the problems that require more research.

Chapter 2

Background and Related Work

This chapter explains the concepts used in developing the lane change model for connected autonomous vehicles(CAVs). The first section (2.1) explains the main concepts needed to work with 'Reinforcement Learning' which is the main agent that acts as the vehicle and the following sections describes the state of the art methodologies used for lane changing aspect of the CAV. At the end, a summary presented explains the research question and also the inspired state-of-the-art methods that is being implemented in the thesis.

2.1 Reinforcement Learning

The branch of machine learning that deals with sequential decision-making is called reinforcement learning (RL) [10]. It involves an agent learning an optimal way to solve a set of problems by mapping situations present in its dynamic environment to actions which are then used to maximize a reward signal. This reward signal is given to the agent for taking a favourable action and encourage the desired behaviour. To reach a certain goal, the agent selects an action from the set of available actions using a trial and error search.

A reinforcement learning system has four parts in addition to the agent and the environment: *a policy (π), a reward signal, a value function*, and *model of the environment*. The main component of psychology-inspired reinforcement learning that controls the agent's behavior is policy. It maps the states in which the agent is present to the actions that can be taken when the agent is in that state. A reward signal acts as a primary objective of the reinforcement problem. It helps in distinguishing the actions that benefit solving the problem by using high and low rewards for positive and negative actions immediately.

With reward comes value function, which determines how much total reward an agent can expect to get in the long run if it took a particular action. Unlike reward, value function gives the total rewards in the long run which is very crucial in making evaluation decisions. We choose a particular action, by looking at its value function but not the amount of reward.

A model is used for planning which actions to choose from when given a state and action by also considering the possible future situations. There are model-based and model free reinforcement learning methods.

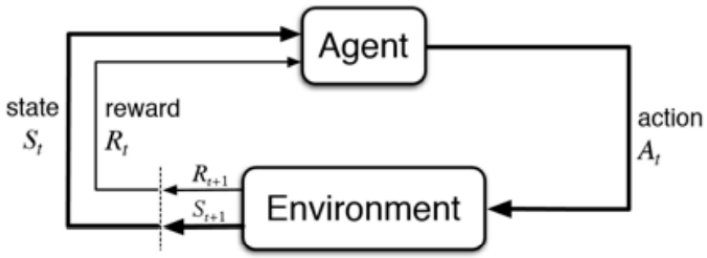

Figure 2.1: Interaction of Agent and Environment in RL

An example of an RL system with an agent interacting with the surroundings is shown in Fig. 2.1. The agent existing in a state *st* selects an action *at* by

adhering to policy from state-space S and action space A. As per the dynamics of the environment, the agent collects its r_t reward upon moving to the state s_{t+1}.

According to Markov Decision Process (MDP), at a time step t, the agent is presented with a state of the environment S_t, and based on it, an action A_t [11] is chosen which is the best one to select. Then, as an outcome of that action, a reward R_t is given to the agent and it is moved to a new state $S_t + 1$ using a state transition probability P, where P is given by [12],

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1....., S_t] \tag{1}$$

The above equation states that the current state of the agent depends only on the immediate previous state. For helping the agent to play the game in a optimal way, it is made to try new random actions once in a while and not only play in the know ways. But if the agent is made to always explore new actions and does not use its wisdom, it may cause the agent to lose. The parameter ϵ is used to balance exploration and exploitation of knowledge.

Discounting(γ) is a value which is in the range [0,1], and tells us how much importance the agent is giving for the future rewards as compared to the immediate rewards when taking an action. If the discount rate γ is 0, then the agents does not take into account the future rewards and only cares about immediate rewards and vice versa for 1.

Then the sum of rewards G_t at a time step t, is modified to include the discounting factor as below

$$G_t = R_{t+1} + _{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2}$$

The equation of Bellman is given by [12],

$$V^*(s) = \max_{a \epsilon A} R_{(s,a)} + \gamma \sum_{s' \epsilon S} \quad (3)$$

The above equation tells that the long term rewards of an action a is equal to the current reward obtained by executing that action and the expected long term reward of the action.

In the next section, a concept called Q-Learning is explained which tells how the agent uses the knowledge gained in order to maximise its rewards and play in a desired way.

2.1.1 Q-Learning

As we have established that our agent should be interested in long term rewards, compared to short term ones, Q learning is an model-free reinforcement learning algorithm [12] that tells our agent which is the optimal action to take given a particular state by calculating its future rewards if that action is taken. As the objective is to find the optimal action given its current state, in order to achieve this, this method may use its own rules or it may not use the policy that it should be following.As there is no necessity for a policy here, this is also an off-policy learning algorithm .

It uses a q-table or matrix of size (state size * action size) and each element is called q-value and is initially set to zero. It is mathematically given by,

$$q_*(s,a) = \max_* Q_\pi(s,a) \quad (4)$$

After we run a episode, i.e., after the agent experiences the task, these ele-

ments (Q [state,action]) change and when this same process is repeated for many episodes, each element in the q-table changes many times and after a while becomes approximately constant and optimal.

Even though there is a barter involving exploration and exploitation, practically all of the table's components converge to optimal values as the agent completes its training by watching more episodes. Then the agent behaves in a greedy way i.e., selects the action having the highest probability value.

The agent can then use it as a guide to decide on the optimal course of action depending on the q value. As a result, the agent uses this q-table as a look-up table to choose an action in a certain condition. Bellman Equation (3) is used for computing the reward and the expected future state at each state.

Discounted Markov Decision Processes with an unlimited horizon can be used with Q-learning. It may also be used for issues that have not been discounted as long as the best course of action is ensured to attain a reward-free absorbing condition and the state is re-established on a regular basis [13].

2.1.2 Deep Reinforcement Learning (DRL)

It is a combination of both Reinforcement learning (RL) and Deep learning and has so many applications in playing games such as Atari, Backgammon, Alpha Go, Alpha zero, chess and shogi. This comes into play due to curse of dimensionality because of big action and state spaces making the q-table huge and unable to traverse quickly as well as hard to compute the q-value estimates for each individual state action pair.In this, neural networks are used instead of components of RL and often the input to these is unstructured like a screenshot of the game environment etc.,

The parameters used by these networks are optimised by algorithms and suitable loss function is minimized. The next sections showcase various other methods used to implement the propose lane changing strategy in a connected automated vehicle (CAV).

Model-based and model-free deep reinforcement learning techniques are also available. In the former, a forward model of the changing environment is estimated via supervised learning on neural networks. Model predictive control is then used to forecast actions for this environment. Actions for this environment are then predicted using model predictive control. Even with these states and actions, the model will often do trail and errors because the true environment changes will be different from the learned environment changes. Optimisation needs to be done to improve the performance of the model and methods such as cross-entropy method, model-free methods are used.

In the latter, the policy is learned without modelling the environment dynamics but depends on temporal difference learning and Q-learning. When in a continuous action space, these algorithms frequently learn both a value estimate and a policy; however, in discrete action spaces, these algorithms typically train a neural network Q-function that forecasts the future returns from executing an action [14].

2.1.3 Deep Q Network (DQN)

The major issue using vanilla Q learning is the use of q- table, for small set of states and actions this is considered a viable strategy. When it comes to using this for problems such as lane changing for autonomous vehicles which involve enormous amount of data the state space becomes huge and computationally infeasible to process the data, to overcome this issue DQN has been introduced.

In DQN, a neural network completely replaces the Q-table and updates the q values of each input state, rather than selecting an optimal q value from the table neural network is used to estimate this value. In Q learning we try to map a q value to a (state, action) pair, while in DQN we map (action, q value) pairs to each input state.

The main network and the frozen target network, which receives updates less often than the main network's weights, are the two networks that are utilized to provide stability [15]. Here DQN uses a concept called experience replay which is a part of reinforcement learning where we capture agents experiences at different times to learn about the environment. These samples are used to train the DQN and are used to update the main and target networks. The main network trains on the batches of samples and updates its weights after k steps, after this the updated weights are passed to target network after N steps. The target network is responsible for performing both the selection and evaluation of the actions. Here the selection of k is crucial as selecting small k values increases the time taken to train the DQN. The final output layer has actions as labels or classes and the related q value as the probability of that action.

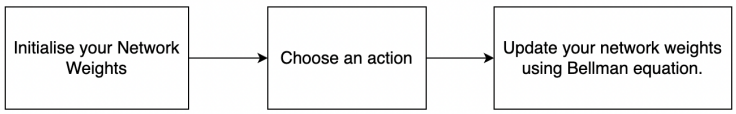

Figure 2.2: Deep Q Network Algorithm

Upon the selecting an action from the output layer, it is performed by the agent and using bellman equation, the weights are again updated in the main and target networks.

2.1.4 Double Deep Q Network (DDQN)

The q values computed are approximates of the transition probability and in DQN, this estimate is used to update the network again. While updating in Q learning, the agent tries to pick the max q values which also include certain amount of approximation error. Taking the maximum estimated value every time as we learn leads to overestimation of the q values leading to maximisation bias. This tricks the agent to pick the action which lead to lower reward at the end.

To overcome this, double Deep Q Network is used which has two independent estimators or models which update each other or modify a target model($Q^{'}$) based on another model (Q). Here to avoid the use of max value this operation is split to use two different networks, online network for selection of action where as target network for evaluation of actions. The weights of the target network are then copied to the online network to evaluate the next greedy action to be considered.

$$Q^*(s_t, a_t) \approx r_t + \gamma Q(s_{t+1}, argmax_{a'} Q^{'}(s_t, a_t)) \quad (5)$$

Hasselt et al. (2015) [1] illustrates this overestimation bias in experiments across different Atari game environments:

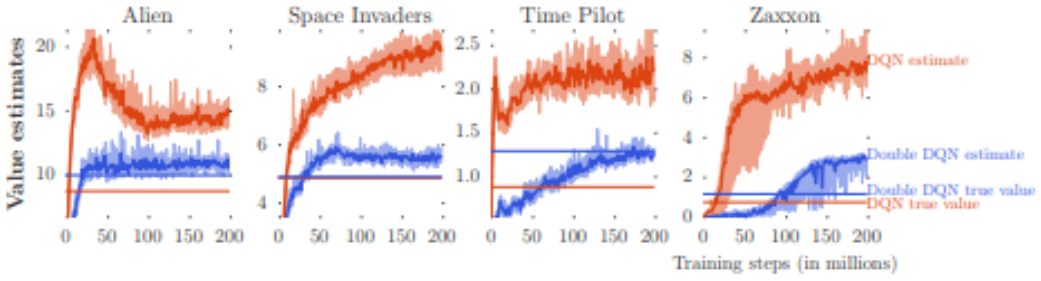

Figure 2.3: Comparison between DQN and Double DQN [1]

2.1.5 Multi-Agent Reinforcement Learning

MARL has become a popular algorithm in strategy games, as well as in the area of robotics and autonomous vehicles where multiple sensors are involved. This algorithm involves multiple agents present in a same environment that are made to interact with each other to achieve a goal. All of these agents tries to learn from others and make the performance of the system better. Experience sharing is one of the important characteristic in MARL where skilled agents serve as teachers where as unskilled agents try to imitate skilled ones.[16]

MARL helps in observing the behaviour of agents and how they effect each other by collaborating, coordinating and competing against each other etc., when assigned a specific goal. Cooperative is where the agents work together towards a common goal; Competitive, where the agents compete against each other; and some mix of them which lead to social dilemmas are the different possible interactions in an environment.

There are two major concepts involved in MARL, one is autocurriculum and cooperative AI. When agents become competitive new tasks are created as each agent tries to hinder the other due which the environment changes leading to multiple phases of learning , this is called autocurriculum framework. Cooperative AI is the class of problems which involve AI that seek to solve problems of cooperation. For example, MARL can be used in cases where two vehicles need to overtake an other one without collision and at a safe distance.

2.2 Related work w/o RL

2.2.1 MOBIL

A generic lane change model called Minimise Overall Braking Induced by Lane Changes (MOBIL) determines the circumstances under which both necessary and optional lane changes occur in an automobile [17]. This considers the risks and utility of the lanes presents and comes up with incentive criterion and safety criterion for making efficient lane changing strategies based on longitudinal acceleration differences.

The criteria just specified prevents collisions, critical lane changes and other even the behaviour of other drivers using a 'politeness factor'. This parameter helps the drivers to make a lane change by co-operating with other vehicles around it and not be completely egoistic. This model considers both the symmetric and asymmetric passing rules present on roads.

One of the two lane changing strategies proposed in [2] is based on MOBIL and is the baseline lane changing strategy developed. Ego-efficient lane changes are done using MOBIL based approach which uses the longitudinal accelerations of the vehicles in the current and target lane.

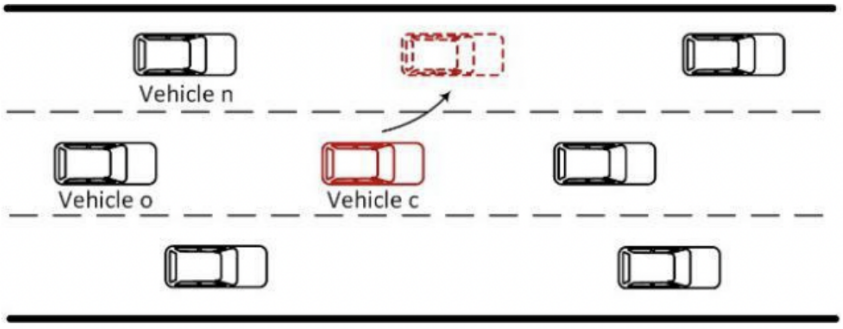

Figure 2.4: Lane Changing scenario in [2]

Figure 2.3 depicts a lane changing scenario in which a vehicle c wants to per-

form a lane change which involves the following vehicle o and the vehicle n in the target lane. Safety and Incentive criteria are calculated and upon satisfying those, a lane change is executed. There are other parameters present in the MOBIL model apart from the two criteria mentioned that can control the behaviour of the lane changing model. Optimisation of those parameters are done using Adaptive fine tuning and Grid search, which are parameter optimisation algorithms.

2.2.2 Extended Intelligent Driver Model

The Intelligent Driver Model(IDM) is a model which is time continuous and that computes the acceleration and desired gap of the vehicle in the next time step using parameters as present in the picture below. The acceleration calcu-

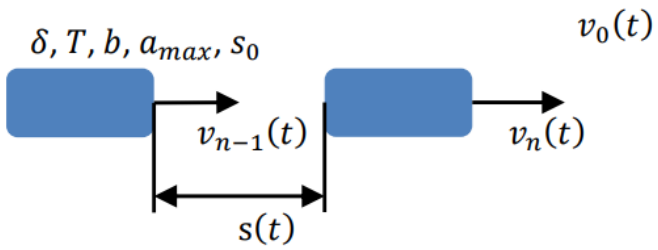

Figure 2.5: Intelligent Driver Model [3]

lated is given by,

$$a(t) = a_{max}\left[1 - \left(\frac{speed(V_{n-1})}{desiredspeed(V_0)}\right)^{\delta} - \left(\frac{desiredgap(S_{n+1})}{gap(S_0)}\right)^{2}\right] \quad (6)$$

which uses the ratios of desired and current velocities, gaps of and between the vehicles.

$$S^*_{n-1}(t) = s_0 + max\left(0, v_{n-1}(t) * T - \left(\frac{v_{n-1}(t) * (v_n(t) - v_{n-1}(t))}{2 * \sqrt{a_{max} * b}}\right)^{\delta}\right) \quad (7)$$

$$a_{IDM}(t+\Delta t) = a_{max}\left[1 - \left(\frac{V_{n-1}(t)}{(V_0)}\right)^{\delta} - \left(\frac{S^*_{n-1}(t)}{S(t)}\right)^2\right] \tag{8}$$

The above equations are for calculating the desired gap needed between the vehicles and acceleration of the vehicle. T is the desired time, b is the desired acceleration, a_{max} is the maximum acceleration, a_0 being the minimum gap and δ, the acceleration exponent.

But due to the usage of gap between the vehicles in the second term while calculating the speed, if any vehicle is far away then it will not go to its desired speed and will never meet the destination. This version of IDM is improvised by changing the characteristics of the model and as for making the acceleration of the desired vehicle much more realistic, two different cases are considered. One is when the vehicle is travelling below the desired gap and otherwise.

$$a(t+\Delta t) = \begin{cases} a_{max}\left[1 - \left(\frac{S^*_{n-1}(t)}{S(t)}\right)^2\right] & \text{for } S^*_{n-1}(t) \geq S(t) \\ a_{free}\left[1 - \left(\frac{S^*_{n-1}(t)}{S(t)}\right)^{\frac{2*a_{max}}{|a_{free}(t)|}\}}\right] & \text{Otherwise} \end{cases} \tag{9}$$

This Enchanced IDM is present in SUMO and other modifications of the IDM exist. For any CAV on the road in real-life scenarios, it has to communicate with the surrounding vehicles for making a lane change else it has to follow typical rules that are needed by any vehicle on the road. The former part is the crucial model for lane changing and the latter is the behaviour of car following models.

To cover both these aspects in the presence of Human driven vehicles (HDV), one needs to know the uncertainty associated with the HDVs. [18] models this behaviour using driver behaviour patterns and uses a longitudinal con-

troller framework for changing between an Reinforcement Learning based lane-change model if any lane change occurs involving a HDV, and an extended intelligent driver model (EIDM), an enhancement of the just introduced IDM based on car-following control model if it does not.

2.2.3 XGBoost model for Lane change

The authors of [19] proposes a deep auto encoder network with XGBoost algorithm to address the problem of lane changing. The issue is divided into identification and decision making. For the identification of the scenarios for the lane change, they have used multiple sensors onboard the vehicles to capture non linear correlations between them. Autoencoder is used to build a model for the time series vehicle data to reconstruct the signals and analyzing the trends in their errors which enable in identifying the behavior of vehicles. To improve the reliability of the results and get a robust model, an adaptive threshold is placed on the errors.

Since there are multiple parameters as well as non linear problems involved in the lane changing decision scenarios, XGBoost with Bayesian optimization is applied to find the optimal parameters. The also propose an online training method to update the parameters with the data batches that are collected by their lane change identification algorithm. Based on the data collected from DAE based lane change identification algorithm, XGBoost based algorithm takes the lane changing decisions. The drawbacks of the proposed model is that it is only applicable in straight lanes or curved lanes in express ways.

2.3 With RL

During the evaluation, reinforcement learning strategies often outperforms the non-reinforcement learning methods discussed in the above section. Therefore, the below content discusses the usage of Reinforcement learning strategies and other methods used in the development of lane changing strategies in autonomous vehicles.

[2] used MOBIL as the baseline and compared it against the performance of Q-Learning algorithm based lane changing strategy. For simulation purposes,

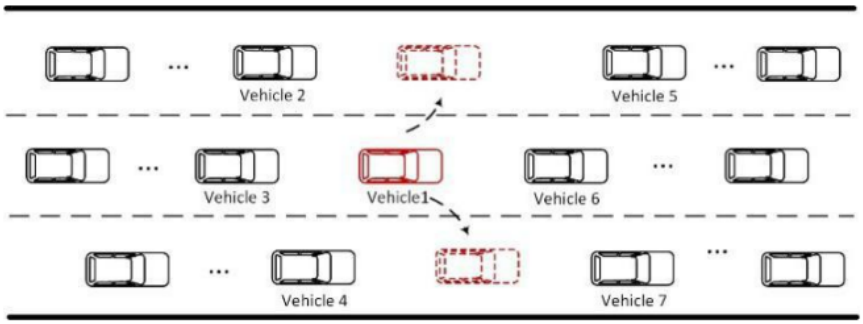

Figure 2.6: Vehicle lane change in Q learning strategy[2]

a three lane highway is used with 2 off-ramps and the reinforcement learning strategy is used to select the state variables which are both mean and standard deviation of the acceleration of the vehicle (Vehicle 1 in fig 2.6) and as well as adjacent vehicles (Vehicles 2-7 in fig 2.6) and remaining downstream ones. No other vehicles other than CAV are considered during the evaluation and the acceleration benefit of the vehicle 1 is considered as the part of the state space when it takes the left (equation 10) or right (equation 11) or no lane change. The reward function is the acceleration gain of CAV itself.

$$\tilde{a}_{1L} = a_{1L} - a_1 \tag{10}$$

$$\tilde{a}_{1R} = a_{1R} - a_1 \tag{11}$$

The results demonstrated via simulation that the lane-changing strategy developed using MOBIL optimization was unable to achieve the goal for CAVs to go faster than human-driven vehicles, while the goal was reached via Q learning.

For making the lane change smoother and much more realistic, [20] created a continuous action space but Q Learning cannot handle it. Another reason behind using this Q Learning based method is to utilise higher number of features present in the environment of the CAV.

Hence, new Quadratic Q function is invented which uses neural networks for working with continuous action space and it as follows,

$$Q(s,a) = A(s) * (B(s) - a)^2 + C(s) \tag{12}$$

where A, B and C are three neural networks made up of different number of layers taking the information about states as input. Optimal action to take is obtained using B which is the complex neural network of all three and upon combining the outputs of these, Q values of the state are obtained.

For evaluating this, a 1000m three lane highway with width of each lane as 3.75m is used for simulation driving environment.In order to teach a smooth and effective lane change behavior, the reward function is defined using yaw rate, yaw acceleration, and lane changing time.

[21] article proposes a Digital Twin (DT) empowered mobile edge computing (MEC) architecture. It states that by using MEC, the sensing and computing capabilities of the CAVs can be strengthened to guarantee realtime safety. The virtualization and offline learning capabilities of the DT can be leveraged

to enable the CAVs to learn from the experience of the physical MEC network and make lane-changing decisions via a 'foresight intelligent' approach. Deep reinforcement learning is adopted to train the lane-changing strategy and results validate the effectiveness of our proposed architecture.

AI is combined with Digital twin and intelligent simulations were carried out for predicting unexpected traffic scenarios and enabling long term decisions by simulating the mobile edge computing network. The aggregated Line of Sight and Non Line of Sight lane environment data is constructed as the training dataset. The NLOS data is only used for estimating the impact on the traffic flow for a lane change decision taken. Deep Q learning is used for obtaining the optimal policy.

The authors of [4] proposed a DRL strategy for making lane changing decisions for the CAV. It also uses Deep Set and Q-learning for handling variable input size for the model and uses information from the vehicles surrounding the CAV and all the vehicles in the connectivity range. This is more advantageous in the sense that taking only a limited number of features for modelling purposes may cause the CAV to use the information from only a certain number of vehicles around it where as with variable input, it can sense the situation of more vehicles thus making decisions of the CAV more efficient. Also, the local information as well as the downstream vehicle's information is fused together for the very same purpose and it also aids the CAV for long term planning.

DRL method is used for calculating an approximation of the Q function which helps to choose an optimal policy by having fully connected neural network (Figure 2.7) for encoding input from each downstream vehicle in the connectivity range, vehicles in the lanes, and the CAV's information itself into

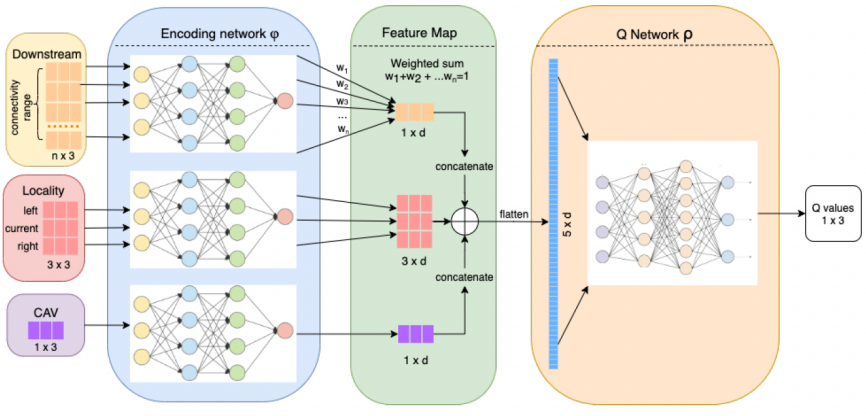

Figure 2.7: Proposed architecture in [4]

a higher dimension space [4].

This is of variable length depending on the number of vehicles present and all this information is fused together to obtain a fixed size input. Different weights are used as per the importance given to each piece of information. This research lets the CAV experience both success and loss and used a experience replay buffer along with Deep set Q learning for storing these experiences.

Experience Replay is a method which uses a separate large table containing a tuple of the data discovered for each state and action like the state, action, reward, next state instead of Q learning. This is because, for SGD optimisation, the training should be independent and when Q learning is used the successive information of the state, etc., is highly correlative and therefore is not suitable for converging the Q function. Instead a large table is used where the information obtained from each experience is stored in a buffer or a table and the training data is sampled from it by selecting random data for the prior two steps or in any such similar way. As there is low variance in between two successive rows, this has little effect on the credibility of the data.

Then a subset of these experiences are selected for learning and used to build a q-network.

The work proposed in [22] does fusion of information using a novel DRL based approach namely Graphic Convolution Q Network which combines Graphic Convolution Neural Network (GCN) and DQN. It considers the CAV network where each node represents a vehicle in a graph and the edges denote the connection i.e.,information dissemination path. CAV is developed in such a way that it enables decision making in short term as well as long term by gaining information from its sensors as well as the vehicles present in long range.

Each Q network is created for the information of a single agent and to train all the vehicles present in the graph, so many networks needs to be trained and the number of parameters for each networks increases and hence, it is not scalable.Hence, a shared centralised Q network that accommodates parameter sharing is used. Through that, multiple agents are trained and each one makes a decision at each time step and the target is to make all the agents achieve the same goal. Decision processor is Deep Q learning agent and the information dissemination, communication are modelled with GNN.

The state space involves 3 blocks of information - nodes features i.e., for any vehicles/node in the graph, speed, location, lane position, intention (Leaving the first and second ramps, continue straight ahead on the highway) are used. Next is adjacency matrix describing the relationship between vehicles and a mask which is used for keeping in check the index of Connected Autonomous Vehicles.

Two types of rewards like Speed reward and Intention reward is given for making sure the chosen actions increases system efficiency and all the CAVs

merge out from the prescribed ramp. Penalties for collision and incorrect lane changing are also provided. Rule based and LSTM models are used as baseline models.Simulation parameters are 500m long, 3-lane freeway with 2 ramps at 200m and 400m locations. The speed limit for the road segment is set as 50km/h for all the CAVs and 36 km/h for HDVs. The model was trained using experience replay and first 150 epochs are 'warm-up' phase and also the model is tested under mixed traffic densities.

[18] treats vehicles in the lane as a platoon and a deep reinforcement learning-based proactive longitudinal control strategy for the CAV to assist the HDV lane change is proposed to reduce the impact of the lane change decisions on the smooth operation of the mixed platoon.

CAV longitudinal control strategy consists of two components: a behavior predictor and a longitudinal controller. A Long Short Term Memory (LSTM) network is utilized to construct behavior predictor. A model is used to predict the future longitudinal state (e.g., longitudinal spacing, speed difference, and acceleration) of the target vehicle, as well as its intention to change lanes. Using another proposed method for predicting lane-change trajectory, it predicts the lane-change trajectory of the target vehicle if the lane-change intention is indicated.

Two switching models make up the longitudinal controller: a car-following control model and a lane-change assistance model. A longitudinal controller will apply the car-following control model based on the extended intelligent driver model (EIDM), if no lane change is indicated by the behavior predictor, to achieve a smooth car-following behavior in a mixed platoon. As soon as the behavior predictor determines a potential lane change, the longitudinal controller switches to the lane-change assistance model, which uses the

Table 2.1: Existing Research in Lane changing strategy of CAV and the necessary details

Application areas for listed references					
Paper	**Application area and Objective**	**Approach used**	**Evaluation Scenario**	**Simulator used**	**State Space**
[4]	Lane Change,Information fusion,collision free	DRL and DeepSets	500m,4-lane loop,mixed traffic(CAV,HDV)	SUMO	Relative distance, Relative speed, Relative lane
[23]	Lane Change,Safety, Efficiency, Passenger comfort	Multi-agent RL	520m gym-based highway,mixed traffic(CAV,HDV)	TORCS	Relative distance(longitudinal, lateral), Relative speed(longitudinal, lateral),Number of detected vehicles
[2]	Lane Change	MOBIL,RL, Q learning	9.3km stretch,lane drops,off-way ramps	AIMSUN	mean,std dev of acceleration in the all the lanes,current acceleration,benefit due to left/right lane change
[21]	Lane Change,safety, efficiency	DT based MEC network, DQN	3 lane, 5km length	MEC network simulator coupled with road traffic simulator	CAV speed, avg lane speed,current lane

[18]	HDV Lane Change, car following, mental safety,behaviour predictor for HDV	DDQN, LSTM	2 lane freeway, mixed traffic	NA	position and speed of the surrounding vehicles
[22]	safe and co-operative lane change	GCN combined with DQN to GCQ network	3 lane freeway, 2 off-ramps,mixed traffic	SUMO	relative speed, longitudinal location,lane position,intention, adjacency matrix
[20]	Lane change,safety, smoothness and efficiency	DQL,Q function approximator	1000m, 3 lane	NA	CAV speed, longitudinal acceleration,position,yaw angle,target lane,lane width and road curvature
[5]	Lane change	DDQN	3 lane circular loop	NA	Lateral position,longitudinal velocity,relative and longitudinal position of surrounding vehicles and also its relative longitudinal velocity.
[24]	Lane change,safety, efficiency,comfort	DQN	3 lane highway	SUMO	longitudinal speed, lateral position,acceleration, relative distance with the agent
[19]	Lane change	XGBoost	-	-	-

Markov Decision Process to model CAV-HDV interaction, and a double deep Q-network model to generate longitudinal control commands for the CAV to assist HDV lane changes. A car-following control model will be used again after the HDV lane change is performed, as the behavior predictor indicates "no lane change" once again. A behavior predictor is also incorporated into the double deep Q-network model training process to enhance convergence efficiency.

It takes into account the case of mixed traffic. A k-nearest neighbors (KNN) model creates the target vehicle's lane-change trajectory. Using the MOBIL paradigm, lane-change decision-making is carried out. The drawback is that the KNN utilized might not accurately reflect how legible and predictable CAV motions affect HDV lane-change behaviors in actual driving situations.

[5] offers a model-based exploration technique for lane changes that are at the driver's option and based on unexpected intrinsic reward. The emphasis was on creating a model of the environment for the nearby vehicles and determining how to relate the intrinsic reward based on surprise to the behavior predicted by the model.

According to figure 2.8, the host car is traveling with knowledge of its six closest neighbors—three vehicles in front and three vehicles behind. The host car is represented by a blue box, and the six nearby vehicles are represented by red boxes with their statuses being observed. Environment vehicles in the remaining boxes are not monitored in terms of their states.

Utilizing proximal policy optimization(PPO)-based DRL, [24] suggest an lane change technique that exhibits significant benefits in retaining performance stability while increasing learning effectiveness.The main concerns of them is to develop safety, efficiency and comfort of the CAV. The PPO network con-

sists of Input layer whose output is fed to hidden layer and then an output layer. The state space is calculated using surrounding 5 vehicles, the target vehicle, the leading and surrounding vehicles in the current and target lane. 4 from all the vehicles except ego vehicle are taken for state vector formation and for the ego vehicle 5 data features are taken.

The four data characteristics are acceleration, longitudinal speed, lateral position, and relative distance to the target vehicle. The longitudinal position, speed, acceleration, lateral position, and speed of the ego vehicle are all taken into account. Penalty is given for collisions and also for not providing comfort to the passenger. As for the incentives, it is given for avoiding collisions, increasing comfort of the passenger and for maintaining efficiency.

[24] proposes a complex reward function for maintaining efficiency which is made of 3 factors. These make sure the vehicle is always trying to meet its maximum allowable speed while being in the accurate position and does the required lane change in minimum time.

This proposed method is compared with baseline models such as rule based agent and PPO based method was able to outperform it.

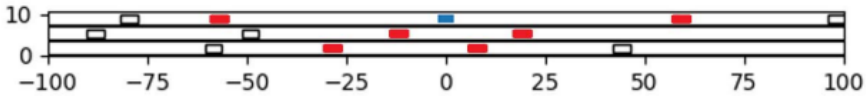

Figure 2.8: Three lane road simulator in [5]

The environment model is built by accounting for both driver behavior and vehicle kinematics, and it is simultaneously taught through reinforcement learning. Lane change, accelerating, strong braking, maintaining speed, and heavy braking are only a few of the 12 distinct acts that make up state space. The longitudinal velocities of nearby autos as well as their relative locations

and velocities are all included in the action space.

The host vehicle's state space includes the host vehicle's lateral position, its longitudinal velocity, its relative longitudinal position with respect to the ith surrounding vehicle, and its relative lateral position with respect to the ith surrounding vehicle. As a result, our continuous state space has a total of 2+3 6(cars) = 20 dimensions. Reward function is expressed as a function of the separation between the host vehicle and its lead vehicle, the host vehicle's lateral position, and the host vehicle's longitudinal velocity.

DDQN, which is used to learn and utilise the environment model, directs the agent's investigation of "unfamiliar" states. The agent would form a perspective on the environment, and using surprise-based intrinsic reward, it would be intrinsically motivated to seek out outcomes that contradict or deviate from that perspective.

[23] suggests the use of the MA2C, a multi-agent advantage actor-critic network, to take into account both CAV and HDV behavior when making multi-AV lane-changing decisions. By utilizing single agent reinforcement learning in the presence of numerous automated vehicles in a mixed traffic scenario, it solves the issues that can arise. As part of the HDV modeling process, it takes the driver's comfort into account when changing lanes.

In order to improve the comfort, effectiveness, and safety of autonomous driving, it also suggests a multi-objective reward function. However, it does not design long-term route and only incorporates information for short-range vehicles. The evaluation in the presence of various traffic scenarios is not put to the test, and it provides no information on how the traffic scenarios are improved.

2.4 Summary

The primary methods employed for the current research were covered in this chapter. The ideas of RL, DRL, and Multi-DRL were introduced.Then, it explained the different state-of-the-art methods for Deep Q-Learning by using DQN and its aggregated techniques which enhance the stability and performance of the training. Then, utilizing DQN and its aggregated procedures, which improve the stability and performance of the training, it described the many cutting-edge methods for Deep Q-Learning. Some of the RL approaches utilized in state-of-the-art methodologies were discussed in the first subsection. The evolution of our strategy for using RL approaches to combat the curse of dimensionality was then discussed. Before delving deeply into the numerous reinforcement learning techniques and other features that increase the model's effectiveness, we first presented a few works for non-reinforcement learning. Upon observing these works, it is observed that most of the papers used DQN to address the lane change problem along with other techniques. This thesis relates to [24] in some of its implementation decisions and modifies it in a way that the state vector is much more advanced as it incorporates data from more number of vehicles. With the agent having more information than the inspired work, it is expected that these changes will in turn reflect in the increased efficiency of the model. The focus of our research is to look at these changes for comparing with the DQN model from [24]. The design and execution of these crucial components are respectively explained in further depth in chapters 3 and 4.

Chapter 3

Experimental Design

The methodology used to create the various parts of the Lane Changing issue is described in this chapter. The design and presumptions used to construct the selected algorithms to address the issue of lane change in CAVs are explained in this section. It explains how the reward function and state space are formed and how they contribute to the accomplishment of many objectives, including safety, efficiency, etc. It also explains how the suggested DQN algorithm relates to the creation of a lane-changing strategy in a CAV and discusses and defends the design choices made when deciding which DQN methods and RL components to use.

3.1 Lane Change Problem

An agent interacts with the environment, which is the road network comprising the ramp region where the traffic splits, in the lane change problem, which is characterized as an RL problem in which a lane change happens at discrete time steps t. The agent particularly notices a state s_t called *epsilon* S at the start of a time step t. It then chooses, executes A, a selected action that corre-

sponds to one of the actions, i.e., the direction of the lane change the agent can make. The agent moves to the next state s_{t+1} *epsilon* S after performing the action at and is rewarded with rt. These elements are defined in the following subsections.

3.2 Grid observation space

For making a informed decision about lane change, the ego vehicle should have data about the surrounding vehicles such as their speed of approaching, distance from the ego vehicle itself etc., for safety and efficiency purposes.

Upon inspiration from [24] using surrounding 5 vehicle's data, this thesis incorporates information from surrounding 8 vehicles which is almost double. This is designed as explained in the following lines.

In any scenario, observation of the ego vehicle which is the DQN agent is in the format of a 3*3 grid in order to calculate its spate space. A observation matrix with dimensions 3*3 consisting of the vehicle id's around the ego vehicle within a threshold distance limit of 10m is designed with the help of lane indexes present in the observation grid on the highway as reference. This information is obtained using TraCi.

The DQN or RL agent is colored as 'red' and all the surrounding autonomous vehicles which will be communicating with agent is yellow in color. The indexes of columns of the matrix aligns with the indexing of the lanes of the road network. It is visually shown in the figure 3.1

The id of RL agent is updated as -1 which lets us know of its presence. Using the vehicle Id's in newly computed observation matrix, information about the vehicles is gathered into a state matrix with the help of TraCi.

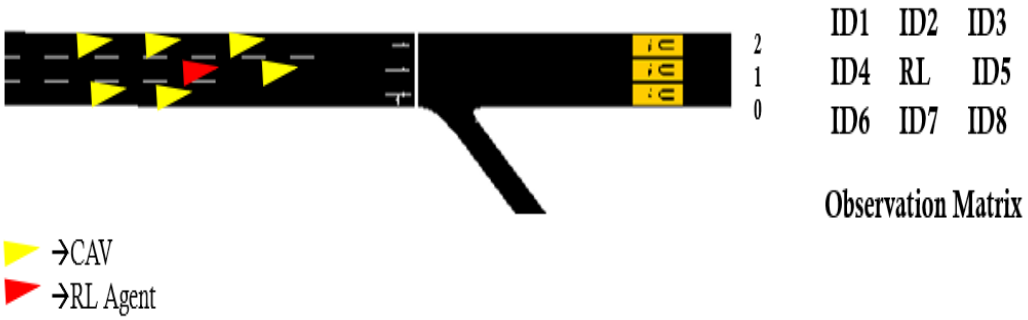

Figure 3.1: Grid Observation space of the ego vehicle

3.3 State space representation

The current state of the target vehicle is depicted as a vector in the state. The state space is borrowed from [24] and in that the surrounding vehicle's longitudinal speed, lateral position,acceleration, relative distance with the agent are obtained as part of the state space. For the DQN agent, 5 data features such as lateral and longitudinal position, speed and longitudinal speed, acceleration.

The state consists of features from the ego vehicle as well as features from vehicles surrounding the CAV within a predefined radius. Each of these arrays containing 4 or 5 elements from the vehicles will be squeezed into a main state vector array.

As the total no.of features for the state space per vehicle is 4, as per the count of vehicles surrounding the agent, state space of it will be the embedding of the features from those vehicles as well. So, at anytime the state is represented using 5 + 4 * (no.of vehicles surrounding the ego vehicle). The figure below shows the formation of the state vector.

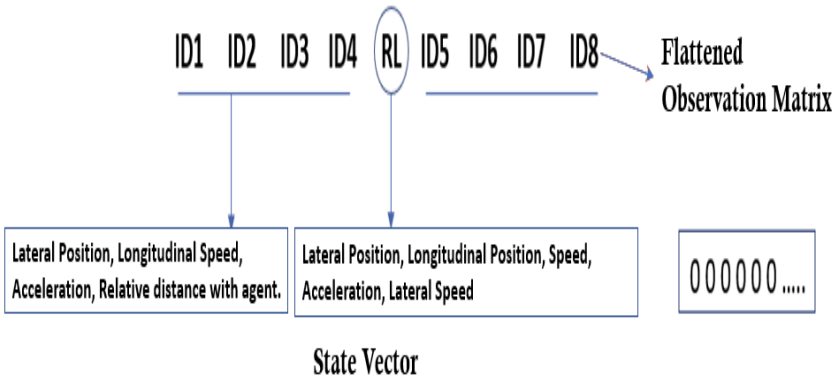

Figure 3.2: Formation of state matrix from figure 3.1

If there are no vehicles around the agent, then those places in the state vector will be zeros.

3.4 Action space

The target vehicle, or agent, chooses an action a_t *epsilon* A after being in a state *St* at the beginning of a time step t. Here is the A is the action space. At any time step t in the environment, the agent has only three actions to make and that is choosing which lane to go in i.e., whether it should be in the current lane, go to left lane or go to right lane. Each of these is represented using 0,1,2 respectively. Thus, action space will be represented as A=$\{0, 1, 2\}$.

3.5 Reward function

Deep reinforcement learning techniques may maximize the intended reward, but they do not always ensure safety throughout the training or implementation stages [25].The design of the reward function targets three features of vehicle driving i.e., comfort, safety and efficiency. This is inspired from [24] and the con of drastic braking is addressed and lead to creation of change

rewards $R_{comfort}$.

Changes in acceleration is monitored and for any high levels of discomfort caused to the passenger, a high penalty is given and otherwise, micro reward is given. This improves comfort of the passengers because the penalty is directly proportional to the impact of the jerk caused to the person travelling in that vehicle. The comfort reward is mathematically represented as follows.

$$R_{comfort}(t) = -\alpha.a_x(t)^2 - \beta.a_y(t)^2 \tag{1}$$

Collisions should be as much minimised as possible and therefore, a high penalty is given for the agent to make it avoid colliding with other vehicles ensuring the safety of the surrounding vehicles and a small reward is given to the agent when it travels without collisions. The bigger penalty discourages the agent to collide at all costs.

$$R_{collision}(t) = \begin{cases} +1 & \text{for } collision = False \\ -100 & \text{Otherwise} \end{cases} \tag{2}$$

The efficiency is formulated in the sense that the vehicle can go as fast as possible without causing discomfort to the passenger and avoiding collisions. This takes into consideration the 3 factors. It validates the desired position of the vehicle in the lane so as to avoid any sideswipe collisions and it also checks if the speed of the ego vehicle is in par with its max allowable speed, thus encouraging it to go faster as possible. In order to reap more rewards, the vehicle should not change lanes too many times. The third factor takes care

of that. Overall, the formula for calculating this reward is as follows.

$$R_{speed}(t) = -|v_y - v_{max}| \tag{3}$$

$$R_{lane}(t) = -|P_x - P^*{}_x| \tag{4}$$

$$R_{change}(t) = \begin{cases} +1 & \text{for staying in lane} \\ -1 & \text{Otherwise} \end{cases} \tag{5}$$

All the above calculated rewards are added together with weights of importance associated for each individual factor. The final incentive for efficiency is given by the following equation.

$$R_{eff} = w_l.R_{lane}(t) + w_s.R_{speed}(t) + w_c.R_{change}(t) \tag{6}$$

The reward function consists of both rewards as well as penalties. To know the final and combined rewards earned by the agent, total rewards is again computed is used while training the model.

$$R_{total} = R_{comf} + R_{eff} + R_{collision} \tag{7}$$

3.6 Design of the model

3.6.1 DQN network

All of the tests carried out throughout this research utilized the neural network, with the exception of the application of the methods described in the earlier sections.

The DQN network uses this architecture and 3.3 describes the Deep Neural

Network. There is an Input layer followed by 4 hidden layers. Artificial input neurons make up the input layer of a neural network, which enters data into the system for processing by artificial neurons in later layers. The procedure for an artificial neural network starts with the input layer.

The first dense layer has input size as 64 and relu activation function. The output of the Input layer is passed to this. Any layer that is closely coupled to the one above it in a neural network indicates that each neuron in that layer is linked to all neuron in the layer above it. In artificial neural network networks, this layer is the one that is most frequently utilized. This dense layer is in turn connected to another series 3 of dense layers with relu activation functions. So, overall, there are 4 dense layers till now and the third dense layer takes an input of size 128. The input to all these layers is obtained from their preceding dense layers. The output of fourth dense layer is given to a lambda which also takes as input the number of actions in the action space. Now, a fifth dense layer is present which takes input from the third dense layer and the output of this is fed to a lambda layer again.

When building Sequential and Functional API models, the Lambda layer allows for the usage of any expression as a Layer. Lambda layers work best for straightforward tasks or brief experiments.

The following diagram 3.3 explains the functioning of the network more visually. These layers form a learner car model.

A learner model and a simulation environment are the system's two main parts. To be more precise, the learner model instructs the target vehicle (agent) to pick up knowledge as it engages with the surrounding traffic. The high-fidelity microscopic traffic simulation package SUMO is used to create the simulation environment, which comprises the road network, traffic, and var-

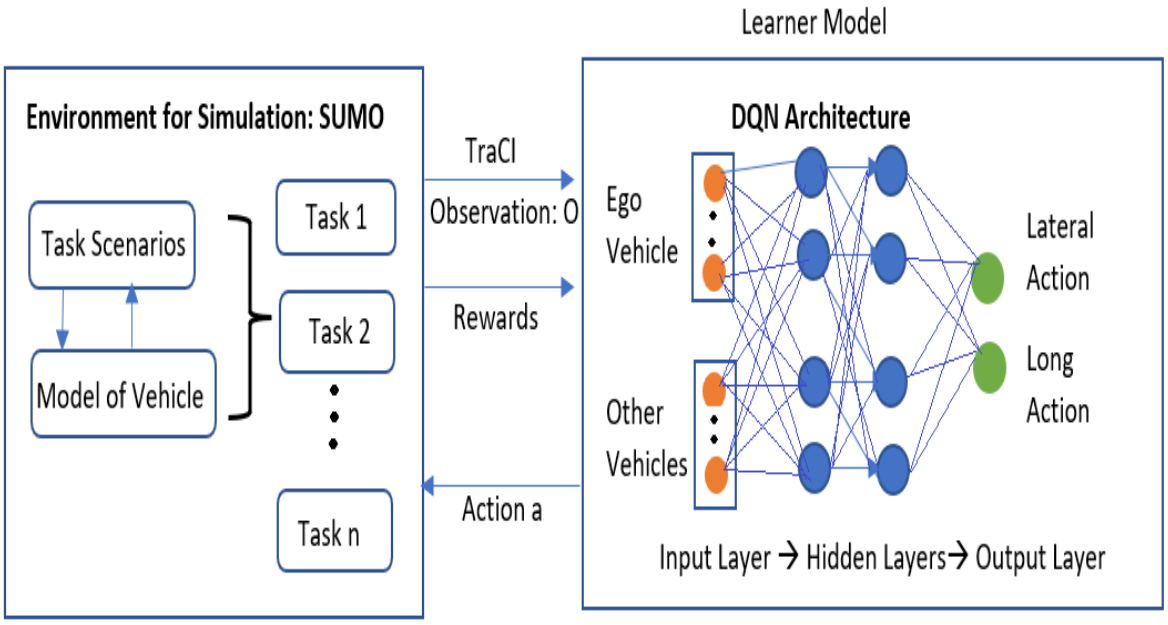

Figure 3.3: Architecture of DQN Network and its interaction with simulation environment

ious task scenarios.

TraCI is used to get the target car's current condition as well as the states of the neighboring vehicles from the SUMO environment. These states are then transferred through the policy to enable safe, slick, and efficient driving techniques on highways. The target vehicle then decides on the course of action in accordance with the specified policy, which is then communicated back to SUMO and hence it can model the movement of the vehicle in the next time step and determine the appropriate reward.

As explained in chapter 2, experience replay is used to store the experiences of the agent while interacting with the environment. These experiences can be best actions and their related choices that made it best. Here, the replay buffer stores the experiences using a tuple consisting of rewards, action, frame of the environment in gray scale and whether the state was terminal or not. These

experiences are used in the learn phase of the network after the buffer is full after making the agent interact with the environment for a while, usually till the buffer is full as per our thesis.

The algorithm combining the concepts of DQN network as well as the just described replay memory is as follows.

Algorithm 1 DQN algorithm with replay memory

Initiate Main Network
Initiate Target Network
Initiate Replay Memory
Initiate the *DQN Agent* to communicate with the environment

while convergence did not happen **do**
/*Sampling Phase
$\epsilon \leftarrow$ setting new epsilon with ϵ-decay
Select action a from state space s using policy ϵ-greedy(Q)
Agent performs action a, takes reward r and goes to upcoming state $s^{'}$
Send move $(s, a, r, s^{'})$ to replay memory
if *Replay memory having experiences to sample from* **then**
/* Knowledge generation Phase
Generate a mini batch of some transitions from replay memory
for *All transitions* $(s_t, a_t, r_t, s_t^{'}, terminal_t)$ in *minibatch* **do**
if $terminal_t$ **then**
$y_t = r_t$
else
$y_t = r_t + \gamma.\max_{a^{'} \epsilon A}$
end
end
Compute loss function L = 1/N $(Q(s_i, a_i) - y_i)^2$
Modify Q utilising Stochaistic gradient descent algorithm by minimising the loss L
For all C steps, copy weights from Q to $Q^{'}$
end
end

3.7 Summary

The choices made for the key elements of the Lane Change problem were described in this section. It discussed the environment's primary elements, including status and action spaces and the reward function. Additionally, it gave details on how the utilization of rewards functioned. It also demonstrates the neural network architecture and training algorithm design. How we used these components will be discussed in the following chapter.

Chapter 4

Implementation

The DQN system's implementation as well as the simulation environment for lane changes are both covered in this chapter. We present the fundamental structure that was used to put the DQN agents into practice. We go over how DQN agents are put into practice and how they interact with the simulated environment as well as how all these combined achieve the goal of solving the lane change problem.

4.1 Simulator and Design of Road Network

To model the traffic flow very close to real life scenarios simulation tools are used. These help build a simulation environment consisting of the road network, traffic and modelling of different scenarios. CARLA and SUMO are the popular simulators available in the market.

CARLA [26] is an open source simulator with a high level of realism for research on autonomous vehicles. It gives the creator access to a variety of sensor measurements that an automobile might obtain, including LiDar, cameras, sensors, and more. Open digital assets (urban layouts, structures, and vehi-

cles) were produced by CARLA for this reason and are available for usage at no cost. Without a doubt, CARLA is among the top products on the market, and major corporations like Toyota use it. It offers more widely utilized market sensors, and its authenticity would make a taught algorithm's output more obvious. Additionally, there is a sizable community of CARLA users as well as additional training. The primary problem with CARLA is that you can't build a custom road, thus you can't test out different settings. This is due to the necessity of owning a costly commercial edition of a program named RoadRunner.This diverted us to SUMO.

An open source traffic modelling and simulation tool called "Simulation of Urban Mobility" (SUMO) [27] is made to manage massive networks. With its extensive toolkit for scenario building, it enables intermodal simulation that takes into account pedestrians. In contrast to CARLA, SUMO contains simple graphics that ensure quick algorithm execution for the user. Sumo was created primarily to simulate traffic situations. As a result, the user has access to a variety of urban components to alter the scene. SUMO collaborates with another program called NetEdit. With the use of a highway, crosswalks, stops, sidewalks, buildings, and other features, the user is able to customize the road as they see fit. It creates an XML file that describes the scenario, which is required by SUMO in order to execute the simulation, together with a configuration file that specifies the basic settings for the vehicles.

In addition, SUMO functions as a TCP-based server that accepts connections from numerous clients, and its data may be accessed using the TraCi API.

As NetEdit is a drag and drop interface, a three lane highway is constructed using junctions and edges to form roadways. The point where two or more edges meet would be ramp areas or round-abouts.

4.2 DQN Agent

The development of a lane change strategy needed several components to put together. The figure 4.1 refers to all the classes that helped solve the problem of lane change. TraCi and Gym is an already existing libraries that help build the needed environment that contains road network. These libraries make it easy to fetch data from the simulation as well as to help the training of the DQN agent model. Deque is a part of python collections and represents a type of queue structure in which data is sto10d. The environment, DQN agent and replay memory are created and they have helper functions that implement different parts of the problem.

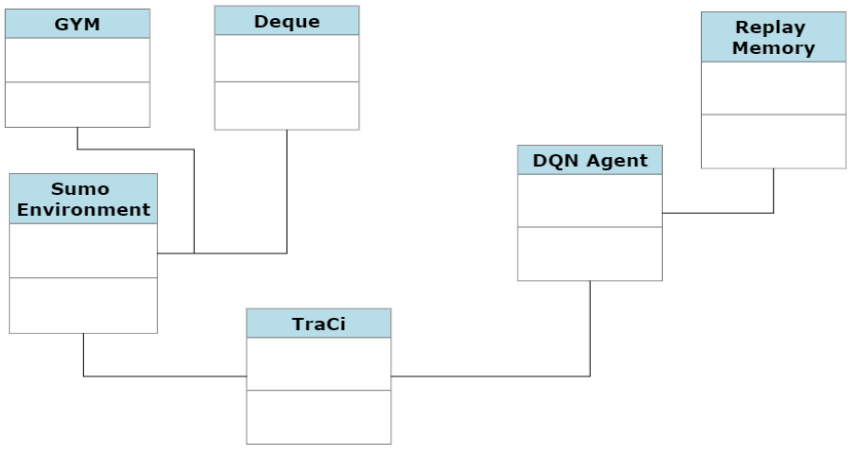

Figure 4.1: Class Diagram of the DQN based Lane Change Problem

4.2.1 Sumo Environment

The environment is of major importance and starting point of the implementation as it is the python program which helps the DQN agent to interact with the simulation in SUMO. Also, it has functions that calculate the rewards and implements the state, action spaces without which the agent cannot function properly.

This environment is created using gym and it acts as a wrapper for the class. In Gym, an environment consists of 4 functions, i.e., initialisation, render, close and step. The gym also comes with other class called 'env', which makes the creation of the environment and the simulation easy. These along with other functions that are provided in this class are as follows.

- **start**(*self, gui = True, no_Vehicle, veh_type*): connects the python program with SUMO road configuration files and TraCi. The simulation and the vehicles are initialised and put into the highway using TraCi commands present in this. It also changes the settings of these vehicles such as lane change mode, speed mode etc.,
- **update_params**(*self*): Another crucial method where the parameters of the DRL based agent is set. Some of these parameters are speed, position, acceleration, lane number etc., Using this, the current state of the agent is updated at any time step *t*.
- **compute_observation_matrix**(*self, threshold_distance*): In the state-of-the-art methods, there is a concept called Information Fusion, which combines information from surrounding vehicles within the connectivity range and not only limit this possibility to the vehicles in contact with the ego vehicle. This method implements the Grid Observation space as specified in the Design chapter and gets the Id's of the vehicles in the given distance.
- **get_comfort**(*self*): deals with the comfort provision to the passenger. There is an acceleration history deque where the acceleration at time steps *t* and *t-1* are stored and this method calculates the difference between these two and return the same. The impact of the jerk caused is directly proportional to the result of this method.

- **get_collision _value**(*self*): Using TraCi's function to get the colliding vehicles list, this method checks if the agent is involved in a collision or not.
- **get_info**(*self,vehicle name*): gets the data features as explained in the Design chapter necessary for the computing the state vector based on their Vehicle Id.
- **get_compute _state**(*self*): creates a state vector as in Design chapter by getting the vehicle information and the observation matrix from get_info (*self,vehicle name*) and get_grid_state(*self, threshold_distance*)
- **compute_reward**(*self, collision,action*): The reward function explained in Design chapter is implemented. these incentives along with the total reward which is the sum of all the individual rewards as well as penalties are returned.
- **step**(*self, action*): specifies the next actions to be taken by the agent after an action is selected to be executed by the DQN network.
- **reset**(*self, gui = False, no_vehicle, veh_type*): loads the network configuration again or used to restart the vehicles in the road network from their initial states depending on the presence of the GUI.
- **close**(*self*): to close the sumo connection that exists between the python program and sumo simulator and shut down both.

4.2.2 Replay Memory

Experienced replay or reply memory maintains a buffer of experiences for making the training process efficient and lets the agent learn from the best experiences it had so far. The agent interacts with its environment till the

replay memory buffer gets full and after that learning phase starts. In this phase, a sample mini batch of the stored experiences is taken and fed to the agent again for learning purposes. It is visually expressed as in following figure 4.2.

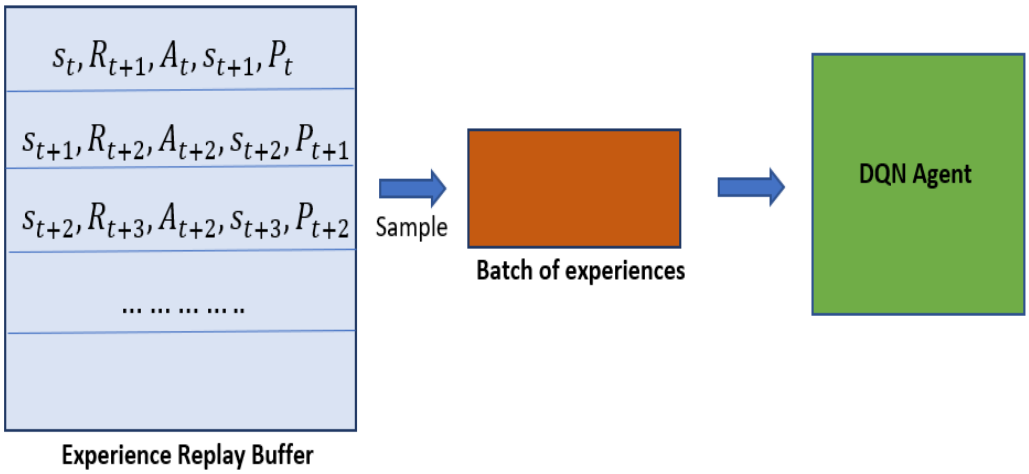

Figure 4.2: Working of Experience replay

The replay memory class implements this functionality and has various supporting methods. The buffer can hold a vector of length 37 which is the same as of the state vector and a maximum of 10,000 experiences can be stored. The user-defined functions are as follows.

- **add_experience**(*self, action, frame, reward, terminal*): The input parameters action, frame, reward, terminal are added to the allocated arrays. Terminal states whether the state is the last episode or not.
- **_get_state**(*self, index*): retrieves a particular experience based on the index from the stored buffer of experiences.
- **_get_valid_indices**(*self*): returns the starting indices of each experience in the all the experiences stored i.e., it gives the valid start indices needed to be used in order to retrieve them.

- **get_minibatch***(self)*: returns 32 transitions of state vector of the agent, new state after executing action, the action itself, rewards obtained and a boolean variable saying the episode is terminal or not individually.

4.2.3 DQN Agent

The design of DQN agent, mainly the neural network model is explained in the design chapter. It is a car model and supporting functions written in the DQN Agent class helps train the network and carry out the evaluation. This class connects with the replay memory class and uses the buffer created in it to store the experiences of the agent interaction while in the epoch at any time step t.

When the class is created, the parameters such as

- Discount factor γ
- Learning rate α
- Replay buffer size
- Learning frequency λ
- Target network
- ϵ and ϵ-decay
- Main network
- Update frequency

Along with these, paths for the model file and training logs to be saved,the loss function, optimiser and the neural network model are created. The methods are as follows:

- **create_logs_directory***(self)*: takes the path of the directory from the class

global variables and creates a log directory where the event files are kept.

- **action***(self, state, primary_nw)*: balances exploration and exploitation and provides the necessary actions to take place in both cases using a predict of the DQN network and random action respectively.
- **training_steps**_*(self, replay_memory)*: takes the replay memory for every 4 actions executed and updates the Q-values for them. Then, it calculates and returns the loss in accordance with the rewards received vs q-values of actions executed.
- **modify_network***(self)*: the network parameters from the main network are updated to the target network.
- **train***(self, env, steps_epoch, epochs)*: The crucial part of the class where the training happens and the epochs are run. Training corresponds to running the overall flow of the reinforcement learning as explained in the background work chapter, much more specifically the deep reinforcement learning methodology. During this, interaction with both sumo environment and replay memory classes is seen. The evaluation metrics are updated at the end of each epoch and the parameters of the main network are updated within loops in the epoch.

There are two Deep Q-Networks used for Deep Q-Learning in TensorFlow implemented in this thesis. Both of these have the same network architecture as explained in chapter 3. A target network and a main network and the former is updated with weights of the latter for every 4 actions that are executed. The main network is updated with the weights obtained after performing all actions that the agent takes in the environment. At the given update frequency

rate, they are retrieved as experiences from the replay memory and Q-values are calculated for each of them. Using the best experiences decided based on the computed Q-values, the target network weights are updated and thus efficiency of the network is increased.

4.2.4 Huber Loss

The Huber loss, a loss function used in statistics and in robust regression, is less susceptible to outliers in the data than the squared error loss [28].This function explains the cost a method of estimating incurs. It is given by the formula below[29].

$$L_\delta(y, f(x)) = \begin{cases} 1/2(y - f(x))^2 & \text{for } |y - f(x)| <= \delta, \\ \delta.(|y - f(x)| - 1/2\delta), & \text{Otherwise} \end{cases} \quad (1)$$

This combines mean square value and absolute value functions mathematically. Graphically represented as in 4.3, according to the value of δ, the graph shifts between mean square loss and huber loss.

Outliers occur when the Q-value approximation is wrong during the learning phase. As, huber loss is less sensitive to them, it helps converge the model to valid mean solution.

It is also particularly useful when the rewards obtained are corrupted occasionally due to its robustness towards outliers.

4.2.5 Optimizer

This subsection's goal is to analyze the tests conducted in order to choose the most effective optimizer among the two most often utilized optimizers—Stochaistic Gradient Descent and ADAM—used in the earlier studies. The assessment

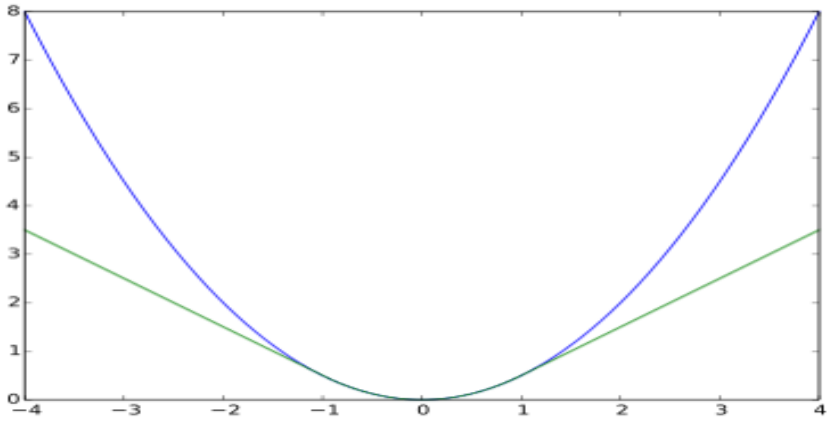

Figure 4.3: Huber Loss function

Table 4.1: Parameters used for choosing between the optimizers

Parameters	**Values**
Episodes	2000
Training epochs	1000
learning rate α	0.0001
Exploration	$0.9 \rightarrow 0.5$
Replay memory start size	33
Discount factor	0.99
Update frequency	4
Replay Memory size	10000
Size of Minibatch	32

was carried out using the hyper-parameters presented in Table 4.1. Both the optimizers are run using their default parameters.

In order to choose which optimizer to use, we only compare the loss of the model so that it is easy to know which one converges the model fast and helps the reach the goal. Upon running the model with its default parameters as explained in table 4.1, Adam reduces the overall loss of the model within the given 1000 runs and adapted to the learning rate, but SGD demands the inputs to be normalised and use regularization in order to reduce the exploding gradient and did not give any valid results. The loss chart when Adam optimiser used is as 4.4

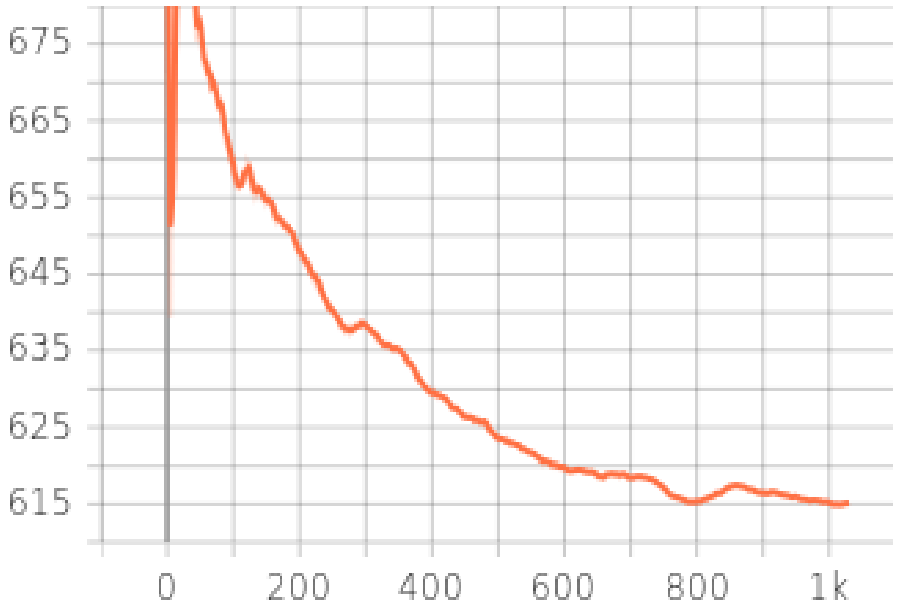

Figure 4.4: Loss rate of Adam Optimizer

As adam performs better than SGD, hence, we choose adam optimizer to carry out the further experimentation.

4.3 Summary

In this chapter a justification is presented for the simulator choice and explains the road network that is designed for the simulation evaluation. Also presented the classes created in python to implement the design of the problem at hand. It includes methods to compute the rewards, states and actions as well as the whole interaction between different parts of the lane change problem. It is created as per the algorithm specified in Chapter 3. Finally, the implementation decisions of loss function and optimiser are illustrated.

Chapter 5

Evaluation

This section presents an assessment of DQN as a solution for the lane-changing issue. The evaluation's goals and the system's hyper parameters that are used to conduct the experiments are described. It also discusses the measures used to gauge the system's efficacy. Experiments used for evaluation, as well as their presentation and analysis of results, are also detailed.

5.1 Objectives

In this chapter, DQN is evaluated to see how well it addresses the problem of lane change in autonomous cars in context of the local environment. The primary goal of DQN's design was to create an agent that could manage the wide dimensional space of real-world driving situations.Chapters 3 and 4 provide descriptions of the development and implementation of DQN, which fulfills the requirements for such a system. However, how well DQN performs in various evaluation settings will determine whether it is effective as a lane-changing approach in CAVs. If DQN complies with the criteria listed below, it can be claimed that it has successfully performed lane change in autonomous

vehicles.They are as follows:

- DQN agent makes use of the available state information and can take the passenger through out the road network with minimum scope for a collision.
- DQN agent can manage meeting the maximum speed allowable for it while also following other necessary criterion.
- DQN agent's driving behaviour actually provides comfort to the passengers without causing any huge jerks to them i.e., with no sudden high changes in acceleration. In other words, the smoothness of the driving is almost or better than the real-life driving.

5.2 Metrics

Metrics are values that are attained during the learning and training phases and are used to track past development. This tracking also helps to tune the performance of the DQN agent by enabling us to train the agent effectively and check for any overfitting or underfitting. We must employ metrics during this tuning process in order to choose the hyper parameters necessary to enhance the model's functionality.

The main metrics that needs to be monitored and that helps to achieve the objectives of the thesis as well as the evaluation are as follows:

- **Average speed vs Maximum**:This encourages the agent to go as fast as possible but within the maximum allowable speed. It is needed because it helps avoid rear-end collisions occurring due to over-cautious slow speeds.
- **Collision rate**: The crucial metric that helps to see the number of colli-

sions occurred overall. This promotes efficiency of the model.

- **Efficiency rewards**: This helps us see the way on which agent leaning on i.e., if it learning then the agent's rewards will be increasing with respect to time. This metric is for monitoring efficiency rewards alone.
- **Comfort rewards**: This metric monitors comfort factor of the reward function and helps us notice increasing or constant comfort provided to the passenger.
- **Safety rewards**: Monitoring of safety i.e., avoidance of collision is checked within this chart. This must have increasing trend overall ideally.
- **Total loss of the model**: This tells us how well the model is trained and is able to converge.
- **Total rewards**: This metric tells the behaviour of the agent during the training and while learning, increasing rewards informs the agent is learning something instead of heavy fluctuations.

5.3 Evaluation Scenarios

In this section, we'll go over the situations that we utilized to gauge DQN's suitability as a lane-changing system. We begin by outlining the methods and scenarios that will be tested and used against DQN.

5.3.1 Techniques and Scenarios

- **DQN with different learning rate:** Varying learning rate tells how fast the agent is able to get trained and perform accurately.
- **DQN with higher speeds for target vehicle than others:** The same criteria for safety is validated here as the maximum allowable speed for

the DQN agent is increased. It should also keep the comfort criteria in check with not much compromise on efficiency.

- **DQN with modified weights for the reward function:** This is to make the reward function incline towards one of the three factors that are part of it.

All these discussed above are tested on the network layout and parameters shown in section 5.4.

5.4 Setup

This section describes the environment in which the tests were carried out as well as the different setups and settings used. The traffic demand creation, the variables for the simulator, and the DQN agent are provided after the network design produced in SUMO.

5.4.1 Network Layout

A elevated nano-scale traffic simulation package called SUMO is utilized to create the simulation environment, which comprises the road layout, congestion, and various task contexts, and has been used to communicate with the training agent [30].The road network used for training and evaluation is 5.1. Vehicles in the network traverse from left to right and the leftmost ramp area is where vehicles from two different road networks join and two way lane acts as an exit road for vehicles in the rightmost ramp area.

off-ramps are an important part of the network where lane changes as well as collisions occur more. Hence, the road network is designed to have 2 off-ramps so that the agent can get trained more efficiently.

The highway is of length 1400m and each lane width is 3.2m. The main route

has 3 lanes where as the ramps have only 2.

Figure 5.1: Designed Road Network

5.4.2 Demand data

During the start of the simulation, the target vehicle which is a DQN agent needs to be placed in the traffic using simulation steps. So, for this purpose, 35 vehicles are added to the road network along with the DQN agent. It is assumed that they all travel at constant speed of 70. These vehicles follows IDM car following model and the lane change model is SL2015. The acceleration is the same for both i.e., 0.8. However, the maximum speeds vary. These along with other parameters are set and the figure 5.2 illustrates the various points including the entry as well as end points of the network during the experiments.

Each path is described by an origin point, a target point, one or more junctions to go through, and so on (blue, red points). A shorter, more complicated route might be designated as 0-2-3-5, whereas a longer, easier route might be designated as 1-2-3-4. As a result, itineraries of various durations are possible. Two routes that are created as are follows:

- 1-2-3-4

Figure 5.2: Designed Road Network

Table 5.1: Parameters used for Vehicles in simulation

Parameters	**Values - 'Vehicle'**	**Values - 'rl-agent'**
Acceleration	0.8	0.8
Maximum Speed	70	50
Lane change model	SL2015	SL2015
Minimum Gap	0.0	2.0

- 0-2-3-5

Two vehicle types are created using the parameters explained in the table 5.1 namely 'vehicle' and 'rl-agent'. These two specify the behaviour of the surrounding vehicles and the DQN agent itself respectively. These attributes help relate the vehicles created to real-life ones.

5.5 Results and Analysis

This section analyzes DQN's efficiency in relation to the evaluation goals listed in section 5.1. DQN's effectiveness is assessed by contrasting it with inspired work produced in various situations. The outcomes are described, examined, and debated in relation to the assessment goals.

The parameters used for evaluation are as per Table 5.1. The weights used for the reward functions and the default values for accelerations etc., and the

Table 5.2: Parameters used for choosing between the optimizers

Parameters	**Values**
Episodes	2000
Training epochs	1000
learning rate α	0.0001
Exploration	$0.9 \rightarrow 0.5$
Replay memory start size	33
Discount factor	0.99
Update frequency	4
Replay Memory size	10000
Size of Mini batch	32

values in the table are decided based on [24].

Along with the parameters of the DQN network, there are others which the vehicles uses like acceleration, maximum allowable speeds

The lane change strategy is tested within various scenarios as listed in section 5.3.1. The analysis and results is as follows.

- Learning rates are varied between α = 0.00001 and 0.0001 for evaluation purposes and it is run for 1000 epochs. The graphs below show that the loss of the model varies alot when different learning rate is used. It means the model is not being able to learn properly in an optimal way within a global scope.

  Usually, under these circumstances, the learning rate should be reduced in order to allow the model to learn a more globally optimal way. Although this makes the training times slower, this massively increases the performance of the model.

  Figures below show the loss of the model with learning rate 0.00001 and 0.0001.

  The above results show not many differences between the modified learn-

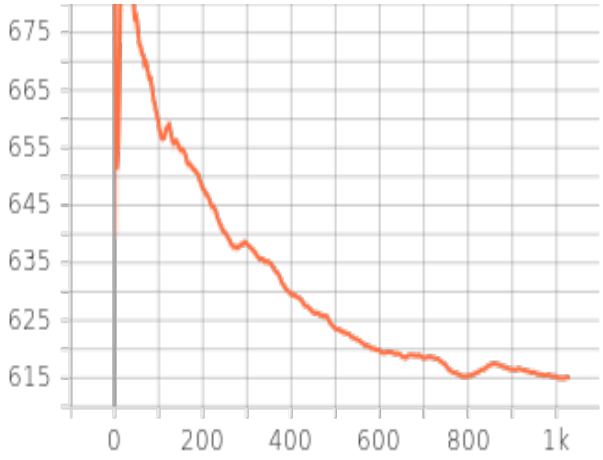

Figure 5.3: Loss rate of model with $\alpha = 0.00001$

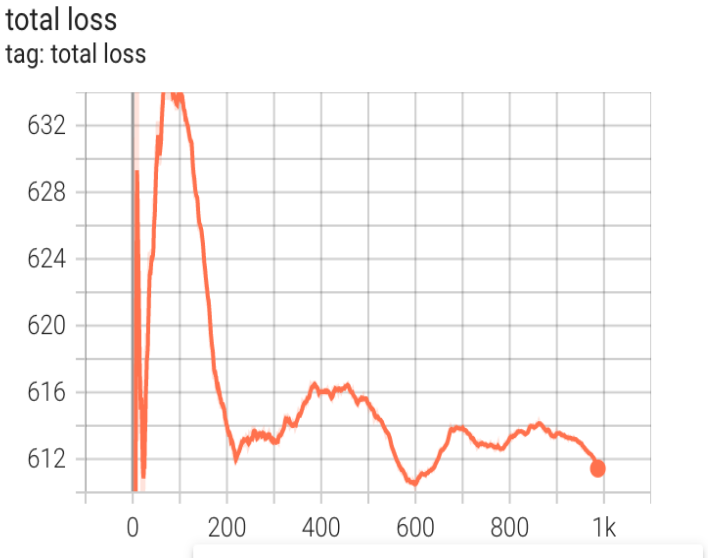

Figure 5.4: Loss rate of model with $\alpha = 0.0001$

ing rates. But in fig 5.4 has more fluctuations starting from 200 epochs to 580. Although the loss value dipped to less than 612, it increased again. So, at the end, the loss value is not decreasing and is fluctuating a little. But fig 5.3 shows no fluctuations and is decreasing overall. Although its not a significant difference, learning rate $\alpha = 0.00001$ is chosen to be used.

- The maximum allowable speed for the agent is 50 and within this experiment the target speeds of both the vehicle and rl-agent are increased to 120 and 100 respectively.

  Agent reaching this speed is tested and all the other parameters are as in Table 5.2. The results of metrics are as below.

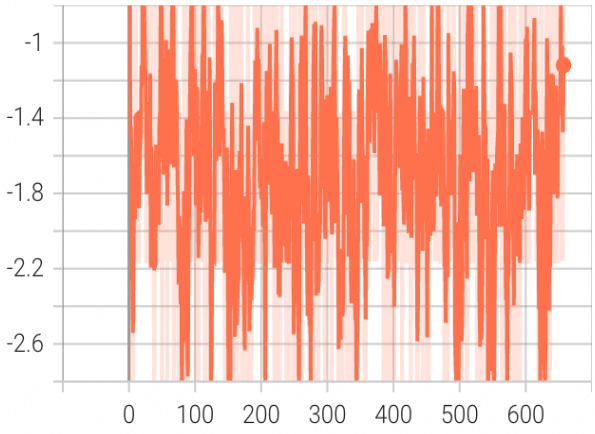

Figure 5.5: Avg vs maximum speed when speed is increased in model

When fig 5.5, 5.6 are compared to collision rate and avg speed vs max speed of the model, it can be seen that several fluctuations are occurring in the first 600 epoch area. Thus this model needs to be trained with different parameters which is out of the scope of this thesis as this deals with a specific aspect of lane changing in motorways and hence, usage of the speeds of 70 and 50 for both rl agent and other vehicles will be continued.

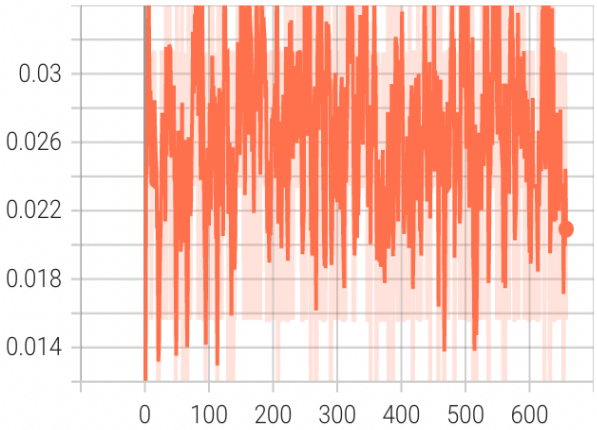

Figure 5.6: Collision rate when speed is increased in model

After careful experimentation and choices taken with the parameters, the figures from both results are as below.

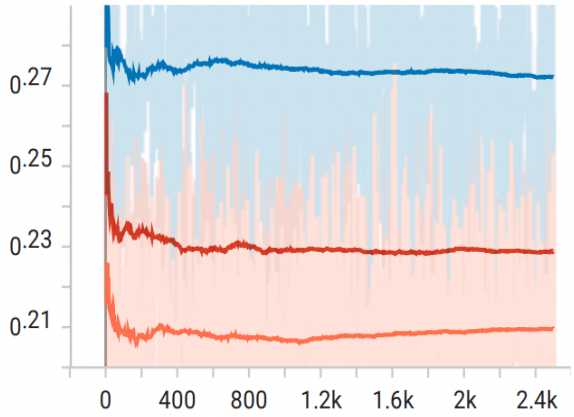

Figure 5.7: Avg speed vs maximum speed of base model

The blue line in the figures of the base model is the required scenario using which the parameters of the model are set and run.

After careful observation, the most crucial metrics which are collision rates and avg speed vs maximum allowable speed are extremely different in both the models. For the model, the collision rate (fig 5.10,5.9) is decreasing overall and this trend is not seen in the base model. But, the value of collision rate in base model=0.007 which is less than the least value of collision rate (0.02) in

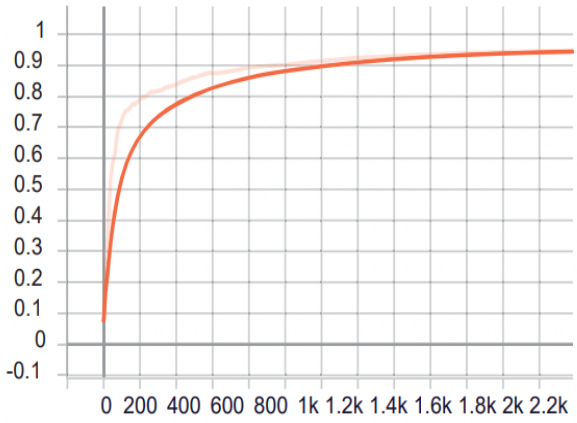

Figure 5.8: Avg speed vs maximum speed of model

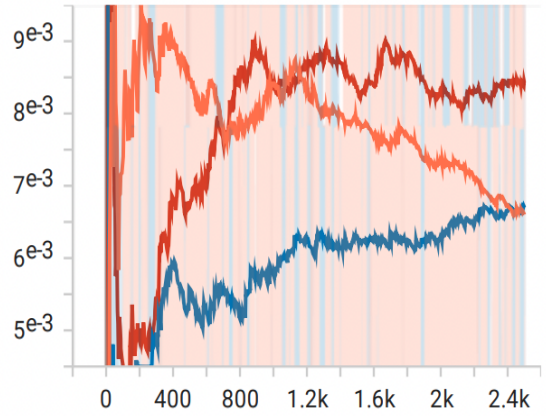

Figure 5.9: Collision rate of base model

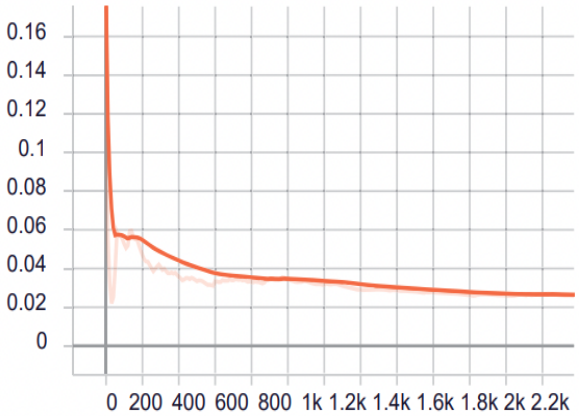

Figure 5.10: Collision rate of model

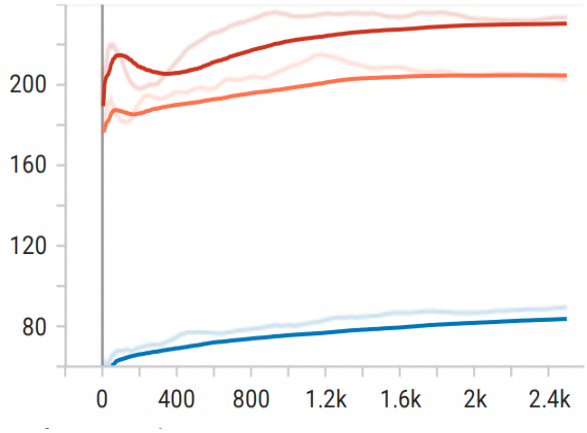

Figure 5.11: Loss of base model

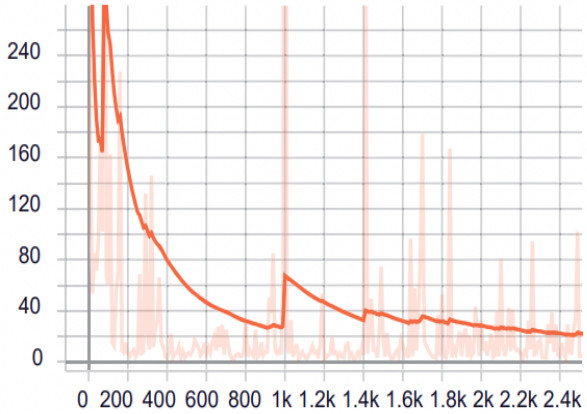

Figure 5.12: Loss of model

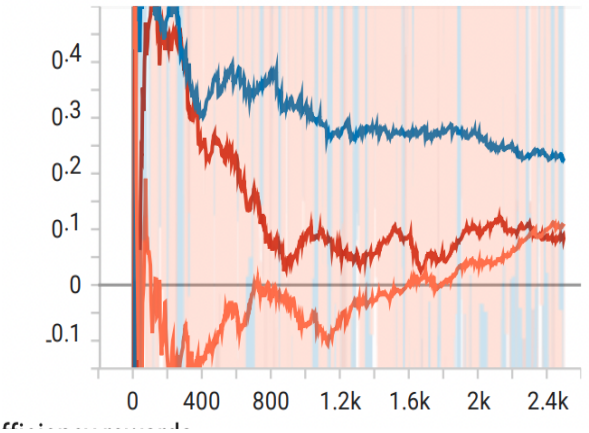

Figure 5.13: Total rewards of base model

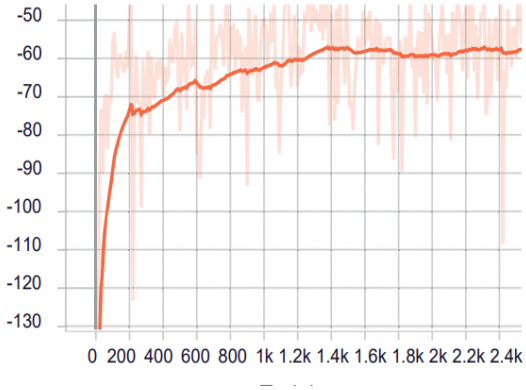

Figure 5.14: Total rewards of model

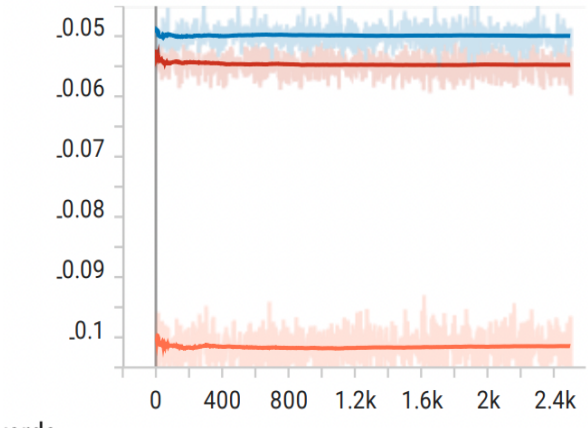

Figure 5.15: Efficiency rewards of base model

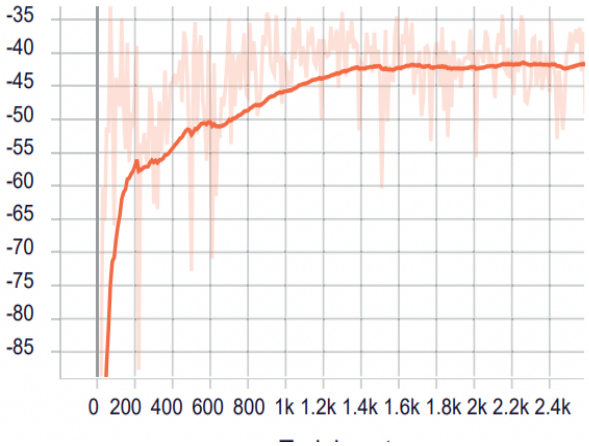

Figure 5.16: Efficiency rewards of model

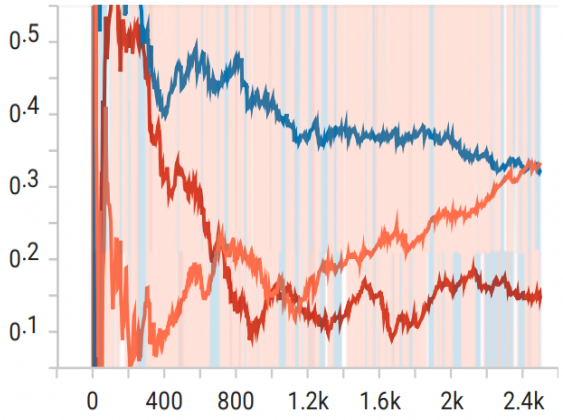

Figure 5.17: safety rewards of base model

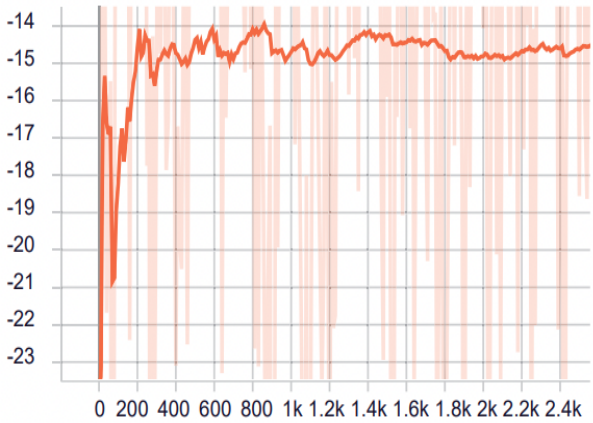

Figure 5.18: safety rewards of model

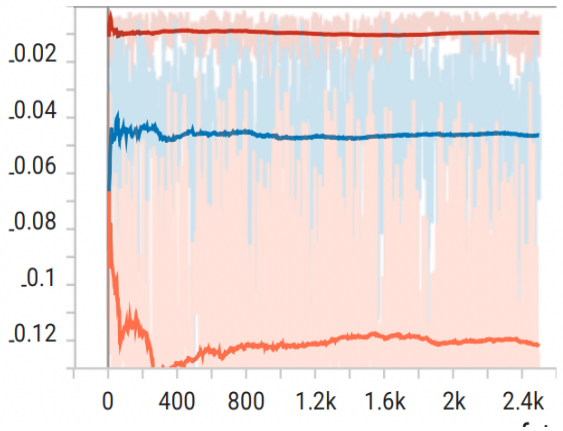

Figure 5.19: Comfort rewards of base model

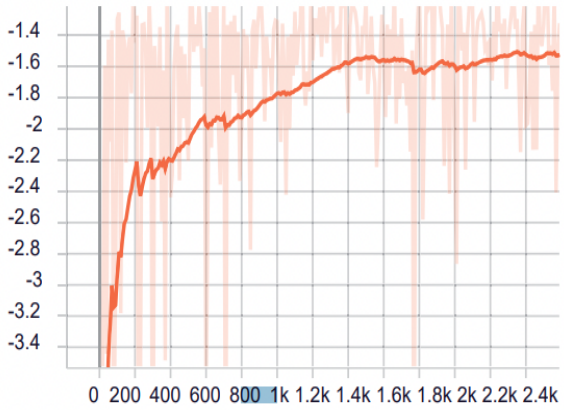

Figure 5.20: Comfort rewards of model

the newly created model.

Though it is not much of a difference, When seen together with the avg speed vs max speed graphs 5.7 and 5.8, the new model is able to meet the goals by trying to reach the objectives of the thesis as is seen in its graph 5.6. It is because there is an increase in speed of new model but in 5.5, the agent is travelling at a constant speed and shows no change in speed overall.

Next, we compare the metric total loss of the model, which tells how much accurately the model is trained. These can be seen in the graphs,5.11 and 5.12. The new model is trained better as the loss of the base model is increasing which is not a good sign.

Now, as it is seen that overall new model is performing better than the base model, this can be confirmed by looking at the various parts of the reward function. Ideally, these should be increasing because, if it learns to do things right, rewards will be added in each epoch and hence an increasing trend should be seen in each of its graphs.

Graphs of new model regarding rewards are in figures 5.14, 5.20,5.18,5.16. All these have an increasing trend which supports the idea that agent is good at

learning the objectives stated by the thesis.

When these are compared with base model, and graphs 5.19,5.17,5.13,5.15, they do not satisfy them. The total rewards, safety rewards are decreasing and efficiency, comfort rewards are constant. This means the newly created model outperforms the base model and satisfies all the specified objectives.

5.6 Summary

The evaluation of DQN as a DRL-based lane change problem was described in length in this chapter. The evaluation goals,scenarios, and environment setup have all been reported, along with an analysis of the findings. The study of the data leads us to the conclusion that the altered DQN outperforms the original and may represent a feasible solution to the lane change issues. Although the strategies produce strong learning and performance, extra adjustments, such as utilizing a different neural network for information fusion, may enhance performance further because it can incorporate a significant amount of information about the nearby vehicles. The created DQN agent satisfies each of the goals specified in section 5.1 at the beginning of this chapter. However, modifications to the hyper parameters can still be made in order to obtain the even more best outcomes.

Chapter 6

Conclusion

The thesis is summarized and its most noteworthy accomplishments are discussed in this chapter. The remaining unresolved research questions associated with this work are then discussed.

6.1 Thesis Contributions

Chapter 1 introduced the problem of lane changing and the concepts involved in it. It then talks about the assumptions made during this thesis regarding the vehicles, creation of a suitable environment for simulation purposes, aims and objectives of it. Thesis contribution explains how the aims and objectives are addressed using a much more technical language i.e., the goals are translated to reward function, usage of vehicles information and penalties. In reinforcement learning, these contribute so much towards the work that is being put in to achieve the goals.

The state-of-the-art techniques that were considered to conduct research on the issue at hand were described in Chapter 2 of the book. It explains different RL and non-RL methodologies that were applied throughout the back-

ground investigation. More precisely, the emphasis is on thoroughly describing the RL approaches since they are essential to the development of a DQN agent model. These studies are all concerned with creating a lane switching method for connected autonomous vehicles. Additionally, it explored the associated state spaces and reward functions used in more recent experiments with DQN. However, they are also described along with other ideas that were utilized with the DQN network model.In conclusion, the nearest competitor is analyzed, and the modified components of this thesis lane change technique are described.

The design choices made in order to construct the DQN approaches are discussed in Chapter 3. It specifically covered the RL elements i.e., creration of each of state, actions, and reward function, neural network architecture, appropriate DRL approach choice, and DQN agent algorithms.

The implementation of DQN for lane change scenarios was described in Chapter 4 in detail. This is a quick overview of the TensorFlow code used to construct the neural network, as well as additional classes created for environment generation and libraries for DRL algorithm and simulator, respectively.

Using SUMO, Chapter 5 assessed DQN as a potential solution for the lane change issue. With various setups, such as variable speeds, learning rates, and accelerations, we assessed DQN's performance. We contrasted the modified DQN agent baseline with the inspired DQN agent [24], from which the basic framework was taken. The results of the studies indicate that the DQN is the optimum strategy for lane-changing tactics. We established the DQN performs better than the baseline through the successful studies.

6.2 Future Work

Various areas were found where DQN's efficiency and usability may be improved during the design and evaluation process, as well as a number of possible research topics. A list of these areas is given below and it also goes over the still-unresolved research problems.

- **CNN based network for incorporating Information fusion:** A neural network can often work with more information as compared to matrix and this gives a possibility of using even more information for computing state vector. This gives the agent more farsightedness in terms of making decisions and increasing efficiency.
- **Using Prioritized replay memory**Instead of just using replay memory, and giving the best experiences for the agent to learn, the same if given priorities can make the agent in the learning phase learn the best of best experiences and helps increase the performance.
- **Introducing behaviour patterns for surrounding vehicles** This thesis uses interaction with connected autonomous vehicles but using human driven vehicles with behaviour close to real-life is also a possibility. This makes the scenarios more closer to real-life ones.
- **Varying demand generation** At present, the number of vehicles around the agent is around 35 and this can be increased or made complex with different routes.
- **More Complex intersection experiments** The road network can be changed to a more complex and one that depicts real-life roads. Training and gauging the performance on this or even evaluating using this could be done.

- **Longer training times** Due to time constraints of the thesis, the model could not be trained for more time and with modifications in model's parameters like ϵ-decay, update frequency etc., This could be done and it might have resulted in a much more efficient strategy.

Bibliography

[1] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL http://arxiv.org/abs/1509.06461.

[2] Yiyue Ma, Long Wang, Yibing Wang, Jingqiu Guo, Lihui Zhang, Simon Hu, Ioannis Papamichail, and Markos Papageorgiou. Developing smart lane-changing strategies for cavs on freeways based on mobil and reinforcement learning. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2027–2033, 2021. doi: 10.1109/ITSC48978.2021.9564678.

[3] Stefan Kaufmann Dominik Salles and Hans-Christian Reuss. Extending the intelligent driver model in sumo and verifying the drive off trajectories with aerial measurements. *SUMO User Conference*.

[4] Jiqian Dong, Sikai Chen, Yujie Li, Runjia Du, Aaron Steinfeld, and Samuel Labi. Space-weighted information fusion using deep reinforcement learning: The context of tactical control of lane-changing autonomous vehicles and connectivity range assessment. *Transportation Research Part C: Emerging Technologies*, 128:103192, 2021. ISSN 0968-090X. doi: https://doi.org/10.1016/j.trc.2021.103192. URL https://www.sciencedirect.com/science/article/pii/S0968090X21002084.

[5] Songan Zhang, Huei Peng, Subramanya Nageshrao, and Eric Tseng. Discretionary lane change decision making using reinforcement learning with model-based exploration. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 844–850, 2019. doi: 10.1109/ICMLA.2019.00147.

[6] Geana A Gershman SJ Leong YC Radulescu A Wilson RC. Niv Y, Daniel R. Reinforcement learning in multidimensional environments relies on attention mechanisms. 2015 May 27. doi: 10.1523/JNEUROSCI.2978-14.2015.

[7] Lane change/merge crashes: Problem size assessment and statistical description. *National Highway Traffic Safety Administration*. URL `https://ntlrepository.blob.core.windows.net/lib/16000/16700/16737/PB2000104631.pdf`.

[8] Guchan Ozbilgin, Umit Ozguner, Onur Altintas, Haris Kremo, and John Maroli. Evaluating the requirements of communicating vehicles in collaborative automated driving. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 1066–1071, 2016. doi: 10.1109/IVS.2016.7535521.

[9] Jerome Tiphene (PSA Group) Mikael Fallgren Wanlu Sun (Ericsson) Nadia Brahmi (Bosch) Erik Ström Tommy Svensson (Chalmers) Diego Bernardez (CTAG) Jesus Alonso-Zarate (CTTC) Apostolos Kousaridas Mate Boban Markus Dillinger (Huawei) Massimo Condoluci Toktam Mahmoodi (KCL) Zexian Li Juergen Otterbach (Nokia) Mathieu Lefebvre (Orange) Guillaume Vivier (Sequans) Taimoor Abbas Peter Wingård (Volvo Cars) Antonio Eduardo Fernandez, Alain Servel. 5gcar scenarios, use cases, requirements and kpis. 2017. URL `https://5gcar.eu/wp-content/uploads/2017/05/5GCAR_D2.1_v1.0.pdf`.

[10] Vincent Franç ois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3-4):219–354, 2018. doi: 10.1561/2200000071. URL `https://doi.org/10.1561%2F2200000071`.

[11] Philippe Preux Olivier Pietquin. Mathieu Seurin, Florian Strub. A machine of few words interactive speaker recognition with reinforcement learning. *Conference of the International Speech Communication Association (INTERSPEECH)*, Oct 2020, Shanghai, China. doi: ff10.21437/Interspeech.2020-2892ff.ffhal-03123999f.

[12] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[13] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4: 237–285, 1996.

[14] Volodymyr; et al Mnih. Playing atari with deep reinforcement learning. Deep Learning Workshop 2013.

[15] Marc Velay. Target networks: Slow and steady wins the race. URL `https://towardsdatascience.com/target-networks-slow-and-steady-wins-the-race-214ed14e97e7`.

[16] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008. doi: 10.1109/TSMCC.2007.913919.

[17] Dr Dirk Helbing Prof. Dr. Arne Kesting, Martin Treiber. Mobil: General lane-changing model for car-following models. 2006.

[18] Yongyang Liu, Anye Zhou, Yu Wang, and Srinivas Peeta. Proactive longitudinal control of connected and autonomous vehicles with lane-change assistance for human-driven vehicles. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 776–781, 2021. doi: 10.1109/ITSC48978.2021.9564458.

[19] Xinping Gu, Yunpeng Han, and Junfu Yu. A novel lane-changing decision model for autonomous vehicles based on deep autoencoder network and xgboost. *IEEE Access*, 8:9846–9863, 2020. doi: 10.1109/ACCESS.2020.2964294.

[20] Pin Wang, Ching-Yao Chan, and Arnaud de La Fortelle. A reinforcement learning based approach for automated lane change maneuvers, 2018.

[21] Bo Fan, Yuan Wu, Zhengbing He, Yanyan Chen, Tony Q.S. Quek, and Cheng-Zhong Xu. Digital twin empowered mobile edge computing for intelligent vehicular lane-changing. *IEEE Network*, 35(6):194–201, 2021. doi: 10.1109/MNET.201.2000768.

[22] Jiqian Dong, Sikai Chen, Paul Young Joun Ha, Yujie Li, and Samuel Labi. A drl-based multiagent cooperative control framework for cav networks: a graphic convolution q network, 2020.

[23] Wei Zhou, Dong Chen, Jun Yan, Zhaojian Li, Huilin Yin, and Wanchen Ge. Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic, 2021.

[24] Diab Bilal Aissaoui Ilhem. Reinforcement learning for lane change on a highway stretch. GitHub, 2021/2022. URL `https://github.com/Ilhem23/change_lane_DQN/tree/master/report`.

[25] R. Ehlers B. Knighofer S. Niekum M. Alshiekh, R. Bloem and U. Topcu. Safe reinforcement learning via shielding. 2017.

[26] Felipe Codevilla Antonio Lopez Alexey Dosovitskiy, German Ros and Vladlen Koltun. "carla: An open urban driving simulator". *In: Proceedings of the 1st Annual Conference on Robot Learning.*, page 1–16., 2017.

[27] Laura Bieker-Walz Jakob Erdmann Yun-Pang Flötteröd Robert Hilbrich Leonhard Lücken Johannes Rummel Peter Wagner Pablo Alvarez Lopez, Michael Behrisch and Evamarie Wießner. "microscopic traffic simulation using sumo". *The 21st IEEE International Conference on Intelligent Transportation Systems.*, IEEE, 2018. URL `https://elib.dlr.de/124092/`.

[28] Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 35(1):73 – 101, 1964. doi: 10.1214/aoms/1177703732. URL `https://doi.org/10.1214/aoms/1177703732`.

[29] Friedman J. Model inference averaging. In: Hastie T Tibshirani R Friedman J editors. Hastie T, Tibshirani R. The elements of statistical learning: Data mining, inference, prediction springer series in statistics. page p. 261–94., Springer (2009). doi: 10.1007/978-0-387-84858-7_8.

[30] L. Bieker J. E. M. Behrisch and D. Krajzewicz. "sumo–simulation of urban mobility: An overview". page pp. 63–68, 2011.