DECISION-MAKING OF AN AUTONOMOUS VEHICLE IN THE PRESENCE OF EMERGENCY VEHICLE USING DEEP REINFORCEMENT LEARNING

by

Hamid Shoaraee

M.Sc. Industrial Engineering, University of Tehran, 2019

THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER SCIENCE

UNIVERSITY OF NORTHERN BRITISH COLUMBIA

November 2021

Abstract

Autonomous Vehicles are the future of road transportation where they can increase safety, efficiency, and productivity. In this thesis, we address a new edge case in autonomous driving when one autonomous vehicle is approached by an emergency vehicle and needs to make the best decision. To achieve the desired behavior and learn the sequence decision process, we trained our autonomous vehicle with the help of Deep Reinforcement Learning algorithms and compared the results with rule-based algorithms. The driving environment for this study was developed by using Simulation Urban Mobility as an open-source traffic simulator. The proposed solution based on Deep Reinforcement Learning has a better performance compared to the rule-based solution as a baseline both in normal driving situations and when an emergency vehicle is approaching.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AV, ego	Autonomous Vehicle
emg	Emergency Vehicle
IDM	Intelligent Driver Model
MOBIL	Minimize Overall Braking Induced by Lane Changes
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
SUMO	Simulation Urban Mobility
DQN	Deep Q-Network
DDQN	Dueling Deep Q-Network

Publication From This Research

Hamid Shoaraee, Liang Chen, Fan Terry Jiang. "Decision-Making of an Autonomous Vehicle when Approached by an Emergency Vehicle using Deep Reinforcement Learning". The 19th IEEE International Conference on Pervasive Intelligence and Computing (PICom 2021).

Acknowledgment

First and foremost, I would like to express my gratitude to my supervisor, Dr. Liang Chen, who has an invaluable role in this research. Your informative knowledge, passion, and support encourage me to improve my thoughts and raise the quality of this work.

I want to thank my committee members Dr. Jueyi Sui and Dr. Fan Jiang for their constructive comments and insights that brought the quality of this work to a higher level.

My biggest gratitude to my family for all their love, support, and kindness. My parents, who always providing me opportunities to be a better person and inspired me to take risks in life and pursue all my dreams, LOVE you Baba and Maman.

I have to thank my brother, Saeed, my hero, and his wife and son, Setareh and Ryan, as best friends.

Finally, thanks to all my friends in Prince George especially Reza and Matthias for their help and support throughout my education.

Chapter 1: Introduction

Autonomous driving (AD) or driverless vehicles have gained a lot of attention both in industry and academia. Researchers, engineers, scientists, and automobile manufacturers are working to reach an effective, efficient, and safe solution for fully autonomous driving.

There have been great achievements regarding sensing, perception, planning, and control tasks in autonomous driving. However, some challenges related to decision-making tasks still exist.

In this chapter, an introduction of levels of automation and how the automobile industry has thrived from level 0 of automation to level 4 as a fully autonomous vehicle will be present. After that, we will provide some information about Artificial Intelligence, Machine Learning, and Deep Learning and how they can play a great role as a solution for complicated problems. Next, we will provide information about modular and end-to-end pipelines and the different components of autonomous driving.

1.1. Automobile Automation

Without any doubt, building the first automobile is one of the greatest human achievements. Automobiles serve as a showcase of human accomplishment in fields such as electronic, mechanical engineering, design, manufacturing, art, and many other fields over the past century. Automobiles give people lots of accessibility, freedom, leisure, and they changed the way people thought about transportation. From Karl Benz who made the first automobile in 1885 many inventors and engineers follow his lead to make automobiles a better product.

Automation in the automobile industry is categorized by US National Highway Traffic Safety Administration [1] in 5 levels as shown in Figure 1.1.

The first level of automation is level 0 where the driver has an exclusive control on all controllers (steering, acceleration, and brake). The automation starts from level 1 where the car is equipped with Advanced Driver-Assistance Systems (ADAS). The driver has overall control of the vehicle, but he or she can choose to relinquish limited control over a primary task. Adaptive cruise control, emergency brake, and lane assistance are named as some features of ADAS in this level. When two primary controls are handled by the vehicle instead of a human driver, a vehicle reaches level 2 of automation. For example, a combination of adaptive cruise control and lane centering are considered as the second level of automation. Level 3 is the beginning of the era of self-driving cars: this level call semi-autonomous, where the driver is not required to monitor the road continuously. However, in some situations, the driver is expected to control the vehicle when it is difficult for the self-driving car to control itself.

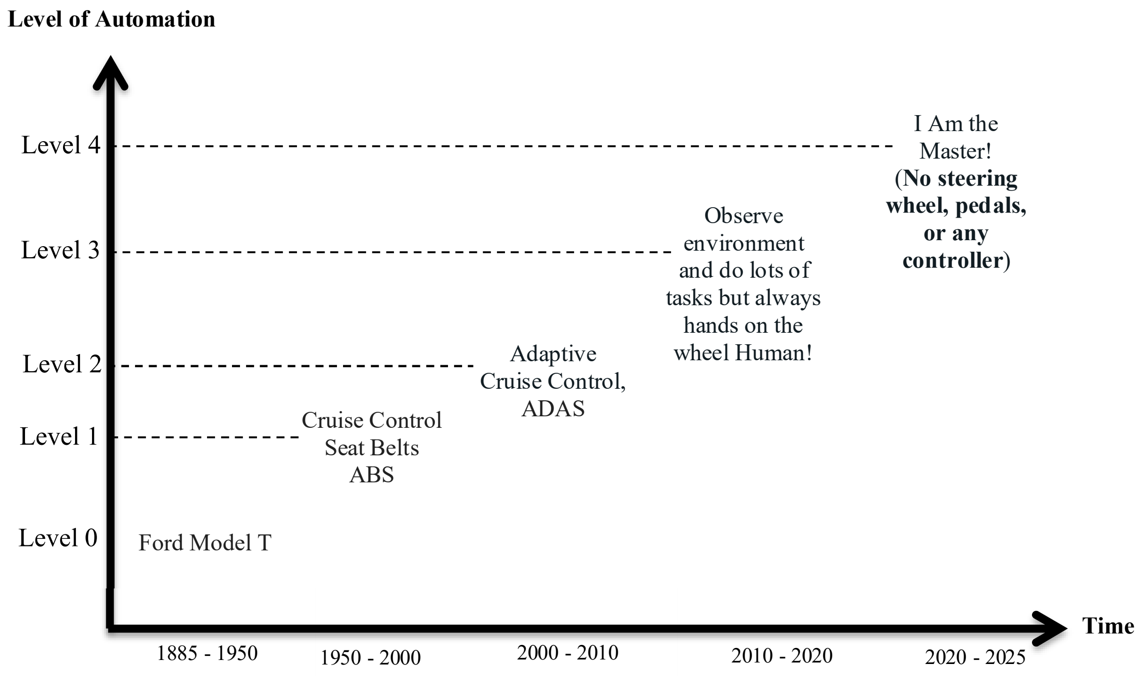
Level of Automation
Level 4
Level 3
Level 2
Level 1
Level 0
Ford Model T
Cruise Control Seat Belts ABS
Adaptive Cruise Control, ADAS
Observe environment and do lots of tasks but always hands on the wheel Human!
I Am the Master! (No steering wheel, pedals, or any controller)
Time
1885 - 1950
1950 - 2000
2000 - 2010
2010 - 2020
2020 - 2025

Figure 1. 1 Level of Automation

For example, in complex situations like unprotected intersections, unmarked roads, and construction zones with a narrow pass, the autonomous driving will be deactivated, and the vehicle gives enough time to transition the control to the human driver.

In level 4, all driving tasks will be observed and controlled by the vehicle without any help from the human driver. At this level, there is no controller such as the steering wheel or gas and brake pedals. Based on our knowledge, none of the car manufacturers and research teams have reached level 4 as fully self-driving at this time. However, based on the trend and development in this field we can expect the ability to fully self-drive in the future. We need to consider based on different categorizations on automation levels some people consider level 5 as fully self-driving automation. There are some studies (e.g. [2][3][4]) that describe the effects of AVs in cities, and how they can lower the cost of transportation, accessibility, and safety. These studies describe the challenges of reaching level 4 as fully autonomous driving.

1.2 Artificial Intelligence, Machine Learning, Deep Learning

Before introducing all procedures that need to consider for Autonomous Driving (AD), we want to provide some information about Artificial Intelligence (AI) and Machine Learning (ML) and explain why classic and heuristic programming does not work for having an AD.

Marvin Minsky in [5] describes the inefficiency of programming in all possible situations as “A computer can do, in a sense, only what it is told to do. But even when we do not know exactly how to solve a certain problem, we may program a machine to Search through some large space of solution attempts. Unfortunately, when we write a straightforward program for such a search, we usually find the resulting process to be enormously inefficient.” Letting machine learning based on searching through a large space of solution is the same thing humans do for learning. Therefore, in AI we want to somehow simulate human intelligence in machines to learn like a human brain. The science and art of gaining knowledge from raw data is called Machine Learning and defined by Tim Mitchell as follows: “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

In general, there are mainly three types of Machine Learning, known as Supervised learning, Unsupervised learning, and reinforcement learning. In this research, we will describe all the details about reinforcement learning as we want to have a solution for the decision-making task of AD based on reinforcement learning.

In supervised learning, we already know both data (features) and labels and we want to find the relationship between input and output. Supervised learning most of the time is categorized into two problems of regression and classification.

In unsupervised learning, the training data is unlabeled, and the machine tries to find patterns from input data without a supervisor. Also, there is Semi-supervised learning when we have a combination of unlabeled and labeled data and we want to learn from partially labeled data.

For a comprehensive study about machine learning and deep learning, we refer readers to two great books by Ian Goodfellow [6] and Aurelien Geron [7].

The relationship between Artificial Intelligence, Machine Learning, and Deep Learning is presented in Figure 1.2. Yann LeCun, Yoshua Bengio, and Geoffrey Hinton defined deep learning in the paper they published in [8] as: “Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer.

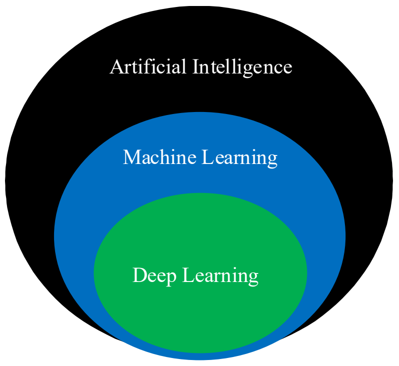

Figure 1. 5 AI, ML, DL

Deep convolutional nets have brought about breakthroughs in processing images, video, speech, and audio whereas recurrent nets have shone a light on sequential data such as text and speech." Learning features that are not human-designed and engineered by the deep neural networks is the power of deep learning which can pave the way for solving complicated problems in Artificial Intelligence like autonomous driving.

We will describe how proposed deep neural networks (Deep Learning) can help to train the autonomous vehicle to handle decision-making tasks with the help of Deep Reinforcement Learning (DRL) without explicit programming (Machine learning), and it enables the AD to be treated like a human driver (Artificial Intelligence).

1.3 Autonomous Driving

Big names like Mercedes-Benz, Toyota, Tesla, Waymo, etc. have tried to reach fully autonomous driving with various technologies from radio control approach to autopilot technologies for many years.

Using Graphics Processing Unit and Tensor Processing Unit with more computational power, developing more complicated deep neural networks that can be trained for different tasks, and collecting and labeling data in an efficient way opens more doors for reaching fully autonomous driving.

There are two common pipelines for solving the autonomous driving problems: 1) modular pipeline 2) end-to-end pipeline.

The input data of both pipelines are the same which can come from different sensors, cameras, Light Detection and Ranging (LiDAR), Radio Detection and Ranging (RADAR), and Ultrasonics data. In the modular pipeline, each of the components works independently, and after the integration between these components, the vehicle can experience autonomous driving.

The second pipeline is End-to-end, where entire driving tasks are considered as one Machine Learning problem, and a neural network handles all the work. Both pipelines have their own pros and cons, for example, the modular pipeline is easier to track which component caused a failure. However, engineering of input and output data between different modules is a very complicated job and is prone to errors, and the result of the modules may not be an optimum decision based on different driving scenarios.

In the end-to-end pipeline, there is no feature engineering and the DNN should learn optimum representation. However, comprehension and tracking the failure reasons are not as easy as compared with the modular pipeline.

The same level of complexity of driving task with other challenges like betting professional game players in Atari 2600 games [9], astonishing performance on complicated games like Go [10] and StarCraft [11], finding protein structure as a significant discovery in genomics [12], and improving computer vision [8] show the possibility of solving end-to-end decision-making of autonomous vehicle with the help of Machine Learning.

Figure 1.3. Shows the main components of the AD system. The sensing component as a first One provides the input data for the AD system. Vision-based AD systems use a simple dash-cam or many cameras around the vehicle. For example, Tesla's autopilot https://www.tesla.com/autopilot uses cameras for 360-degree visibility around the vehicle up to 250 meters in range. Studies like [13][14][15] consider a vision-based system as input data in their models for sensing surrounding objects. The second well-known source of input data is LiDAR [16] as a laser technology that estimates the time of reflected light to the source. Other technologies like RADAR and Ultrasonics are also used in AD systems for collecting input data.

In this research, we consider vehicle state as a high-level measurement technology for collecting input data. All the essential data is extracted in each step from the simulation. We will provide comprehensive information about this method and input data in Chapter 3 when we will introduce the environment and the Simulation Urban Mobility (SUMO) as a simulator for this research.

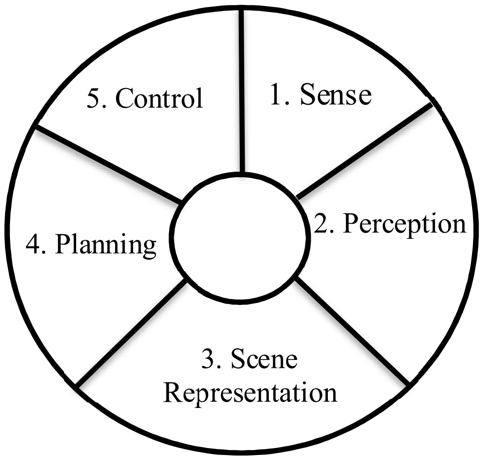

Figure 1. 7 AD Components

After collecting the input data, the next component is perception. Creating an intermediate representation from input data is the goal of the perception task. Object detection, bird-eye view map, lane detection, semantic representation, and high-definition maps are the technologies that give enough perception and localization to the autonomous vehicle.

Scene representation is a higher level of the environment understanding. At this level, information of all the sources needs to combine to create a higher level of knowledge. This combination will happen during the different levels of fusions and in the end, AV knows comprehensive knowledge about the world around the vehicle.

The planning component helps AV to know about the routing and trajectory planning based on the input information such as Global Positioning System or High-Definition Maps. By this level, the AV has all the information for autonomous driving. The last task in the AD pipeline is controlling the vehicle. This task is the output of all the previous tasks and based on the all-stacked information the AV make decision and control steering, acceleration, and deceleration. We refer readers to [17] as a comprehensive study about end-to-end autonomous driving.

1.4 Research Objectives, Contributions, and Structure of This Research

The main objective of this research is to improve the decision-making of autonomous driving systems. We know that there are still some challenges to reach fully autonomous driving. One of these challenges is how the autonomous vehicle can respond to edge cases.

In this research, we want to solve one of these edge cases when an autonomous vehicle is approached by an emergency vehicle. Therefore, the main goal of this thesis is to improve the decision-making of AVs by considering the presence of an emergency vehicle.

Based on our knowledge this research would be the very first research that wants to consider this edge case. As a second contribution of this research, we will develop both rule-based and DRL-based solutions for solving the decision-making of AVs. The rule-based solution will be created by a combination of the Intelligent Driver Model (IDM) and Minimize Overall Braking Induced by Lane Changes (MOBIL). We will design the three DRL-based solutions with two different neural network architectures, and for the DRL-based solution, we will create a highway environment and all the stakeholders of the simulation with help of Simulation Urban Mobility (SUMO) as an open-source simulator. Finally, this research shows that the performance of the proposed solutions is robust by doing different experiments and considering different criteria.

We will do a comprehensive analysis to find which of the proposed solutions has a better performance.

The rest of this thesis will be organized as described: In Chapter 2, we will review the literature and recent work on Reinforcement Learning, Artificial Neural Networks, and Decision-making tasks for autonomous driving. In chapter 3, we will start with a problem description and after that, we will describe both proposed DRL-based and rule-based solutions. We will describe how to develop the environment, action, agent, reward, and all stakeholders that will need for a DRL-based solution. In chapter 4, we will take different experiments and show our proposed solution is robust based on different criteria. Also, computational results and comparisons between different methods will be provided in this chapter. Finally, in Chapter 5, we will conclude this thesis by summarizing our work through this research and we will propose some potential future research at the end.

Chapter 2: Background

This chapter will focus on the previous studies that related to this research. The chapter starts with a review of the literature on Reinforcement Learning and how value-based algorithms can help us to solve the decision-making of autonomous vehicles. Then the background knowledge of Artificial Neural Networks and how they evolved from the simple unit of Perceptron will be described, and decision-making tasks and edge-cases of autonomous driving that were considered by previous researchers will be explored as well.

2.1 Reinforcement Learning

The third type of Machine Learning is Reinforcement Learning (RL) when an agent trains based on a trial-and-error process to learn the optimum policy.

"Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them" (Richard S. Sutton, 2018) [18].

In other words, an agent will learn what to do (take actions) in different situations (states) based on the reward or penalties gained from the environment after taking actions. Interactions of the agent with the environment and learning the best policy $\pi^*(a_t|s_t)$ that works as a map for an agent to take the best action *($a_t \in A$)* in each state *($s_t \in S$)* to maximize the cumulative discounted reward is the goal of reinforcement learning.

There are two types of RL algorithms, value-based and policy-based algorithms. This research will use value-based RL algorithms to solve the decision-making of the AD problem.

Definition of some terminologies can help us for understanding the RL concept. The terms will be introduced through a story that many people experienced in their daily life.

Imagine you have a dog; we need a good name for her. We call her Ava!. You take Ava out for a walk and want to teach her each time you throw a ball she follows the ball and brings it back. Each time Ava brings the ball back you will give her a treat for her good job and if she loses the ball or does not bring the ball back means no more treat! The concept of RL is the same, there is an agent (Ava), and the agent can perform actions. In Ava's case, following the ball or not, and each action creates a new situation (state) for an agent. Based on the action that the agent (Ava) chooses there is a reward or penalty (treat) and by doing a considerable number of experiments we can suppose Ava learns to follow the ball and bring it back. All these experiments happen in an environment, in Ava's example open areas like parks. In RL concept everything outside of the agent is considered as the environment.

Figure 2.1 shows all the stakeholders of RL in a sequential decision process where the action of the agent not only has an immediate reward but also changes the long-term reward of the agent. This sequential decision process follows the finite Markov Decision Process, and it has Markov property when each state has all the relevant information on the history. In other words, in a Markov Decision Process (MDP), the next state is just related to the current state and action, and not the history of the previous states [19]. Therefore, RL based on the MDP can be defined as a tuple *(S, A, T, R, γ)*, where an agent in state S take action A with state transition probability of T, receive the reward of R, which is discounted with γ for balancing between a long time and short-time effect of rewards.

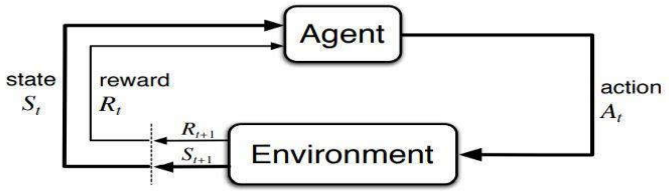

Figure 2. 3 Stakeholders in Markov Decision Process. (Richard S. Sutton, 2018)

The sequence of the agent's experience based on MDP consider as follow:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots \tag{2.1}$$

It means an agent in each timestep with knowledge S_i from the environment takes an action A_i and in the next step, it will receive the consequence of the action as the reward of R_{i+1}.

The policy can define as π when an agent chooses one of the actions and caused the transition to the next state and reward. The sequence of state, action, and reward are built by the following policy as:

$$\pi: S \rightarrow p(A_i \in A|S). \tag{2.2}$$

The transition probability equation (2.3) shows the probability of an agent in $S_t \epsilon S$, take action $a \epsilon A$ and will have reach to S_{t+1} and reward of R.

$$p(s_{t+1}, r|s, a) = P\{s_{t+1} = S', r\ |S_t = s, A_t = a\} \tag{2.3}$$

Based on probability theory the sum of probabilities of taking all actions in each state is equal to one as (2.4):

$$\sum_{a \in A} \sum_{s_{t+1} \epsilon S} P(s_{t+1}, r|s, a) = 1 \tag{2.4}$$

The agent's goal is to maximize the cumulative long-term reward that the agent can receive from the environment. The sum of all the rewards that received after time step t is calculated as (2.5) where T is the final time step.

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T \tag{2.5}$$

For the reward calculation, we need to consider the trade-off between future rewards and short time rewards. This concept is achieved by discount the reward with γ as a discounted parameter in (2.6)

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.6}$$

The discount factor $0 < \gamma < 1$ where $\gamma = 1$ is like a short memory and $\gamma = 0$ is a long-time memory. Mostly the discount factor selects on the range $0.95 < \gamma < 0.98$ for having a combination of both short and long-time effect of reward.

The goodness of each state is based on a policy π calculated by the state-value function as described in (2.7)

$$V^{\pi}(s) = E[\textstyle\sum_{t=0}^{\infty} \gamma^{t} r_{t}] \tag{2.7}$$

The expected value of each state is based on a policy of π in a stochastic case when the agent can reach different states *(s')* by taking action *(A)* calculated by the Bellman equation in (2.8)

$$V_{\pi}(s) = \textstyle\sum_{a} \pi(a,s) \sum_{s',r} p(s',r|s,a)[r + \gamma v_{\pi}(s')] \tag{2.8}$$

If the policy of π can find the optimum value of each state, $V^{*}(s)$, then this policy can have optimum value for the state-action function. The value of the state-action function is named Q-value and we want to find the Q^{*} as an optimum value in (2.9).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\, max_{a} Q(S_{t+1}, \alpha) - Q(S_t, A_t)] \tag{2.9}$$

This equation is known as Q-learning and defined by (Watkins 1992) [20] as a value-based RL algorithm. Monte-Carlo and Temporal Difference are two approaches that can improve the modeling of model-free RL algorithms. In the Monte-Carlo update, we consider the difference between the value of each state and the expected return as (2.10)

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{2.10}$$

The other method that is mostly used to update the model-free RL algorithm is the Temporal Difference (TD) update. In the TD update, we use the bootstrapping method based on the Bellman equation and by using the value of the successor state, we can estimate the value of the current state as (2.11)

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)] \tag{2.11}$$

The difference between the estimated value of the S_t and the real value of the state is known as TD error and denoted by δ_t as in (2.12)

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \tag{2.12}$$

Due to the fact that we are working with a model-free RL algorithm and we do not know about state transition probability, we are interested to work with the value of each (state, action) pair, Q(S_t , a_t) instead of the value of each state $V(S_t)$.

There are two main model-free algorithms: SARSA (state, action, reward, state, action) as an on-policy algorithm and Q-learning as the off-policy algorithm. In SARSA, the algorithm chooses the action based on the policy derived from Q. However, Q-learning is an off-policy algorithm. The following algorithm shows how Q-learning as an off-policy, value-based algorithm works.

We will use Q-learning in the next chapters to solve the decision-making problem in autonomous driving. We need to mention all the previous information and math were obtained from Reinforcement Learning, An introduction by Richard S.Sutton and Andrew G.Barto book [18].

Algorithm1. Q-learning

```
Preconditions: $\gamma$: discount factor, $\alpha$: Learning-rate, $\lambda$: decay factor
        Initialization: $Q(s,a) = 0$
for each episode of all iterations do:
        While $S$ is not termination do:
            Choose action $a \epsilon A$ based on policy $\pi$ (e.g., based on $\varepsilon$-greedy algorithm)
            Execute the action $a$ and get the next state $S'$ and reward $r$
            $Q(S,A) \leftarrow Q(S,A) + \alpha[r + \gamma\ max_a Q(S',\alpha) - Q(S,A)]$
            $S \leftarrow S'$
        end while
end for
```

As we mentioned there is a dilemma between exploration, and exploitation. The trade-off between exploration and exploitation is answered by ε-greedy algorithm. Before we introduce the ε-greedy, we need to define the concepts of exploration and exploitation. Imagine you are at the top of the mountain as Figure 2.2 and want to reach the lowest part of the mountain.

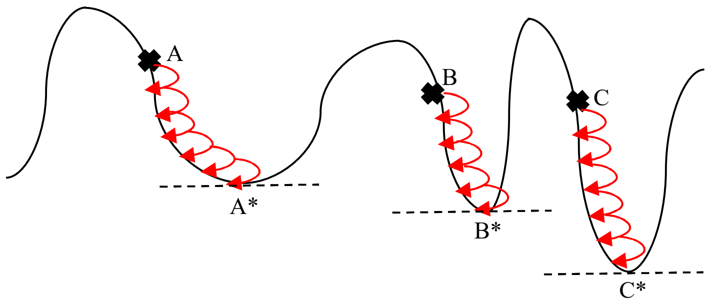

Figure 2. 4 Exploration V.S Exploitation

As we can see in Figure 2.2, we can start from any point of the mountain and the goal is reaching the minimum point. For simplicity, we consider three candidate points as (A, B, C) and by starting from A and B we reach the local minimum (A* and B*) that are not the optimum points. However, if we start moving downhill from point C, we will reach the optimum point (C*) as the global optimum.

The task of choosing the starting point in solution space is called exploration and moving small steps from the initial point to the optimum point is called exploitation. The agent needs to first do some exploration and after that exploit the solution space to reach to the optimum solution.

The trade-off between exploration and exploitation is handled by the epsilon greedy algorithm [21]. Generally, in ε-greedy we generate a random number, and if the random number is greater than the ε agent will take exploration. Otherwise, if the random number is less than ε the agent will do exploitation, and it means the agent chooses the best action on that state for exploitation.

At this point we review the essential knowledge about the reinforcement learning and Q-learning as an off-policy algorithm. Before moving to the next section and describing artificial neural networks and deep neural networks, we want to review on-policy algorithms as another type of RL algorithm.

On-policy algorithms directly optimize the policy performance and these algorithms do not use the old data. However, off-policy algorithms like Q-learning reuse all the previous data based on the Bellman equation to learn optimum policy.

The idea of the on-policy based algorithm like the Vanilla Policy Gradient (VPG) is that it optimizes the policy in the way the agent takes the actions that leads to have more reward with greater probabilities and take the actions with fewer rewards with lower probabilities.

The policy update with standard gradient descent, stochastic gradient descent, or Adam as an optimizer. For more information about on-policy algorithms we refer readers to VPG (Sutton et al. 2000) [21], Proximal Policy Optimization (PPG) in (Schulman et al. 2017) [22], deterministic policy gradients (DPG) [23], and Soft-Actor Critic (SAC) by (Haarnoja et al, 2018) [24].

The problem with Q-learning and using Q-table to store all the values is that by increasing the number of the states and actions, the computational cost will increase dramatically.

Most of the time in the real world we have lots of combinations of state-action. Therefore, storing and quarrying all the information from a tabular format is impossible. The second problem with the tabular version of Q-learning is that the relationship of different states that have similarities cannot be comprehended.

Gaining the knowledge and information between different states can occur when we use the Deep Neural Network as a great function approximator.

The great success of the Deep Neural Network (DNN) in various tasks like computer vision, natural language processing, and speech recognition brings this idea to combine these two concepts to build Deep Reinforcement Learning (DRL) as a combination of the Deep Neural Network (DNN) and Reinforcement Learning (RL).

2.2 Artificial Neural Network and Deep Learning

Aiming to create a solution based on brain activities and transferring data into knowledge with help of brain cells (neurons) caused the creation of an Artificial Neural Network (ANN).

The simple mathematical representation of the neurons can be found in early AI works like [25]. By this time, researchers try to have a different combination of the neurons and different ways they can stack neurons to make ANNs more capable to solve complicated problems like a human brain. What makes ANNs so special compared to the other solving approaches is the capability of considering non-linearity as most of the phenomenon in this world have some degree of non-linearity in their behaviors.

The smallest particle in ANNs structure is called perceptron as shown in Figure 2.3. The perceptron (neuron) has input, activation function, and output as subcomponents. The artificial neural network with the different chain-like structure of neurons has a great capability of approximate the non-

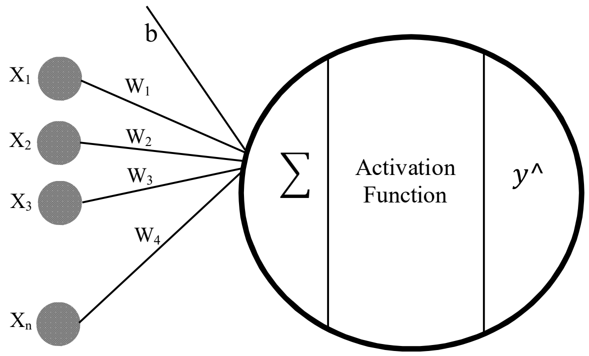

Figure 2. 6 Perceptron the Smallest Component of ANNs

linearity behaviors. Finding the best combination of all the weights and biases of all neurons in the neural network is called learning.

There are different configurations of neural networks, and the simplest neural network is called multilayer perceptron (MLP) [26]. In the MLPs the first layer is called the input layer, where all the preprocessed data from outside of the network feed into the network. After the input layer, neural networks have one or more hidden layers that all the processing is performed in these layers. Finally, the network will be terminated with the output layer with several output neurons that based on the different tasks can predict, describe, discover relationships from input data.

Figure 2.4 shows a MLP network with more than one hidden layer that mostly considers as Deep Neural Network or Deep Learning. As we can see all the neurons are connected to neurons in the next layer and for this reason, these types of networks consider as fully connected networks.

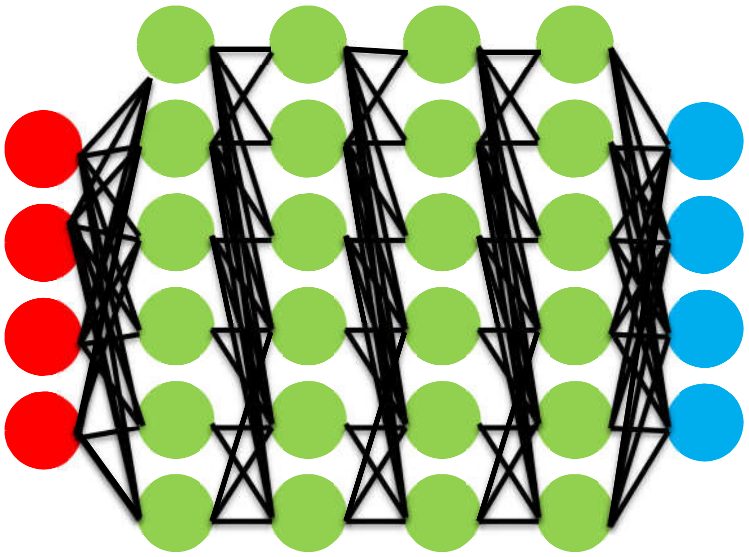

Figure 2. 7 Deep Neural Network

We will propose a deep reinforcement learning solution with help of both a Fully connected Neural Network (FNN) and a Convolutional Neural Network (CNN) as they can learn the complex relationship between input data.

2.3 Decision-Making Task for Autonomous Vehicles

The main two controllers in Autonomous Vehicles (AVs) include longitudinal and lateral control. Longitudinal control mainly means control the speed of the vehicle.

The adequate speed brings a safe distance to the following vehicle, applying the brake in emergency situations, and staying with the constant speed with control of brake and throttle pedals are part of the longitudinal task.

lateral control means how the autonomous vehicle steering in different driving situations. Change the lane safely without making trouble for the neighbor vehicles in the target lane, keep in the center of the lane, and turning are part of the lateral control.

For a full review of car-following and lane-changing models, we refer readers to studies [27][28][29] as comprehensive studies about this topic. We consider two famous models that were mostly used in previous studies of the decision-making of AVs.

These models consider as rule-based solutions for the decision-making of AVs. In other words, they are based on the mathematical formulation, and they have a perfect result in certain situations but not in all situations. However, by creating a solution based on Machine Learning generalization problem of rule-based solutions can solve.

2.3.1 Rule-based Controllers

One of the rule-based models that can control longitudinal trajectories perfectly is the Intelligent Driver Model (IDM) [29]. The IDM as an accident-free model sets the speed of the Autonomous Vehicle by considering the minimum gap and different speeds between two vehicles as the equations of (2.13) and (2.14).

$$V_{IDM} = \alpha\left[(1 - \left(\frac{v}{v_0}\right)^{\delta} - (\frac{d^*(v,\Delta v)}{d})^2\right] \tag{2.13}$$

$$d^*(v,\Delta v) = d_0 + vT + \frac{v\Delta v}{2\sqrt{\alpha\beta}} \tag{2.14}$$

Where v_0 is the desired velocity, δ acceleration exponent, the gap between two vehicles denoted by d, speed difference (approach rate) denoted by Δv, the minimum gap distance d_0, maximum acceleration α, desired deceleration β, and safe time headway T.

Figure 2.5 shows AV and the leader vehicle that we need to consider in IDM. We will use the terms autonomous vehicle (AV) and ego from now to the end of this thesis interchangeably.

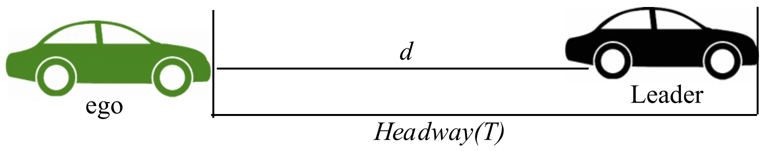

Figure 2. 10 Autonomous Vehicle (ego) and Leader Vehicle Follow IDM

The IDM can play a role as a longitudinal controller when the ego needs to set and control the speed of the vehicle and avoid rear-end collision.

The second model helps the ego to change the lane as a lateral controller. The lateral model called Minimize Overall Braking Induced by Lane Changes (MOBIL) was introduced in [30]. This model minimizing the effect of lane changing on the positive and negative acceleration of the other neighbor vehicles.

Considering neighbor vehicles before changing the lane in the MOBIL model is provided by (2.14) as follows.

$$a'_e - a_e + p[(a'_n - a_n) + (a'_o - a_o)] > a_{th} \qquad (2.14)$$

Safety constraints in this model will check the possibility of changing the lane by considering the traffic effect on the target lane. So, the deceleration of the successor vehicle after changing a lane in a target lane follows $a_n \geq -b_{safe}$ constraint where a_n is the acceleration of the vehicle in the target lane and b_{safe} is a safety acceleration parameter.

a_e, a_n, a_o are the acceleration of the ego, new follower, and old follower, respectively. a'_e, a'_n, a'_o are the accelerations of the vehicles if ego changes lanes, respectively. Also, p determines the politeness factor, and the lane will change if the sum of weighted acceleration is greater than the changing parameter a_{th} on the right-hand side of equation 2.14.

Table 2.1 IDM and MOBIL Parameters

Index	Definition	Value
δ	Acceleration exponent	4
d	Minimum gap between two vehicles	2 (m)
T	Safe time headway	1.6 (S)
β	Desired deceleration	1.8 (m/s^2)
α	Maximum acceleration	2 (m/s^2)
b_{safe}	safety acceleration parameter	4 (m/s^2)
P	Politeness factor	1
a_{th}	Changing Threshold	0.1 (m/s^2)

All the parameters of both IDM and MOBIL will be presented in Table 2.1. we will use a combination of the IDM and MOBIL as a rule-based solution.

2.3.1 RL-based controllers

The reason researchers start considering RL-based solutions instead of the rule-based solution is that rule-based solutions can have a good solution for certain situations.

For solving the generalization problem of rule-based algorithms, researchers in [31][32] tried to optimize the performance of rule-based algorithms with help of the genetic algorithm as a metaheuristic algorithm. However, rule-based approaches are vulnerable to unforeseen circumstances and need to modify their features based on new situations. Therefore, recent studies proposed a new approach, Deep Reinforcement Learning (DRL), for decision-making of AVs as a combination of the Reinforcement Learning (RL) and Deep Neural Network (DNN).

For example, studies like [33] trained ego for control steering, throttle, and brake in simulation world that they create in CARLA [34] as one of the open-source simulators.

Traffic fluidity is considered by [35] when the ego needs to control the speed and avoid collision with other vehicles. The recent studies with help of DRL want to solve complicated edge cases of autonomous driving. For example, researchers in [36] proposed a DRL model to train the ego to perform the on-ramp merge task involving accelerating, decelerating, and steering. In this research, they use a Long Short-Term Memory (LSTM) architecture to learn the relationship between ego and surrounding vehicles and internal state from LSTM cell feed into DRL architecture.

Wang and Chen in [37] proposed a new model based on DRL for continuous state and action space for merging tasks in the highway environment. The reward function in this study is designed for doing a highway merge like a human driver where smoothness, safeness, and promptness are key attributes in reward function.

More studies want to solve merging tasks with DRL solutions like [38] where they introduced Deep merging as a DRL-based solution with pictures input data and [39] proposed a DRL-based model for on-ramp merging with traffic light input data.

Another edge case that many researchers try to have a DRL-based solution for is overtaking task. In overtaking task, the ego wants to pass the next vehicle safely. For overtaking ego needs to control both speed and lane change decisions. For example, researchers in [40] introduced to DRL-based solutions and find that if the agent control both lateral and longitudinal task with the help of DRL solution it can provide better result compared to lateral control by DRL and longitudinal control by IDM. Researchers in [41] proposed two DRL-based models to control speed and lane change decisions for overtaking in a highway driving environment. Researchers in [42] consider overtaking tasks with two different highway traffic scenarios with and without oncoming traffic and they proposed DRL-based solutions for proposed problems.

Researchers at the University of California Berkeley developed FLOW [43][44] https://github.com/flow-project/flow as four benchmarks for reinforcement learning in mixed-autonomy traffic. These benchmarks included different driving scenarios: Circle, Figure-eight, Grid, Merge, and Bottleneck. They want to develop a solution for a mixed-autonomy traffic system, where autonomous vehicles control by RL, interact with a human driver. Flow as a python library interface with other python RL libraries like RLib [45], rllab [46], and Simulation Urban Mobility (SUMO) [47] as an open-source traffic simulator.

There are some studies that consider uncertainty, these uncertainties can come from noisy sensor data or interaction with other users. For example, researchers in [48] used the combination of

Monte-Carlo tree search and deep reinforcement learning for continuous highway driving and highway exit cases. Researchers in [49] proposed two solutions based on game theory for the decision-making of AVs and compare the result with human-like driver decisions. They test the proposed solutions on lane changing, merging, and overtaking scenarios. For more clarification, related previous work on the decision-making task of AVs is summarized in Table 2.2.

Table 2.2 Summary of Related Work

Work	DRL/ Rule-based	Solving method	Task	Simulator
[29]	Rule	IDM	Longitudinal control	N/A
[30]	Rule	MOBIL	Lateral control	N/A
[31]	Rule	(IDM + MOBIL) optimized by Evolutionary algorithm	Lane change and Speed control	N/A
[32]	Rule	(IDM + MOBIL) optimized by GA	Lane assignment	N/A
[36]	DRL	LSTM	On-ramp merge	N/A
[37]	DRL	Q-function approximation	Merging with continuous action/state	N/A
[38]	DRL	Deep merging/ picture input data	On-ramp Merging	N/A
[39]	DRL	DQN + graph CNN	On-ramp Merging	SUMO
[41]	DRL	DQN	Speed and lane control	Open AI/ Python
[43]	Both	IDM, MOBIL, and DQN	Speed and lane control	FLOW/SUMO
[33]	DRL	Actor-Critic	Normal Driving	CARLA
[35]	DRL	Proximal Policy Optimization	Normal Driving, Interaction with human driver	SUMO

For a full review of DRL-based solutions for autonomous driving, we will refer readers to [50] and [51] as comprehensive studies related to intelligent transportation systems and autonomous driving. For different edge cases and tasks, previous researchers proposed DRL-based or rule-based solutions in the previous studies. Based on our knowledge, there is no study that answered the decision-making of the AVs when approached by an emergency vehicle. For reaching fully autonomous driving and level 4 autonomous vehicles need to answer to all the driving edge-cases. Also, making optimum decisions by AVs in emergency situations can minimize the emergency time response. Therefore, in this study, we want to develop and propose a new edge-case for the decision-making of AVs when the ego is approached by an emergency vehicle.

Chapter 3: Methodology

In this chapter, we will start with a problem description. After that, we will describe all the components that we need for the RL concept and introduce our solution based on Deep Q-Network (DQN). We will propose two types of neural networks and at the end, we will propose our solution based on Dueling Deep Q-Network (DDQN) as an improvement of DQN.

3.1 Problem Description

The primary purpose of this research is to create a solution for the autonomous vehicle to drive like a human driver when an emergency vehicle approaches. When a human driver notices the presence of an emergency vehicle, they reduce their speed and pull over to the side of the road. In the presence of emergency vehicles, the ego should use the same techniques as a human driver. To that end, the ego must first learn how to drive normally without any collision with other vehicles and travel within the road boundaries. Following that the ego will learn how to respond to the emergency vehicle appropriately.

The objective function that needs to be minimized is shown in (3.1), where $N_{out\_of\_road}$ is the number of times that the ego changes lanes and travels out of the road, $N_{accidents}$ is the number of times the ego crashes with other vehicles, $T = t_{obs}$ denotes the exact time that ego recognizes the emg in the observation range, $N_{same\_lane\_emg}$ denotes the number of times that ego recognizes the emg and they have the same lane, and N_{speed_vio} is the number of times that ego recognizes the emg and the speed of ego is greater than the speed of the emg.

$$Minimize \left(\sum_{t=0}^{T} N_{out\_of\_road} + N_{accidents}\right) + \left(\sum_{t=obs}^{T} N_{same\_lane\_emg} + N_{speed\_vio}\right) \qquad (3.1)$$

This equation shows all the objectives of this research as ego learns how to handle the normal driving task that is no collision with others and travel out of road boundaries and the second part of the equation wants to assure a decent response after the ego sense the emergency vehicle.

For minimizing the above equation, the ego needs to control both longitudinal and lateral trajectories. Before that, we need to build an environment to help the ego to experience all the different driving scenarios especially when the ego is approached by emg.

For creating this environment there are some open-source libraries that can help researchers create different traffic simulations. Some research trains their model based on real-world data. For example, researchers in [52] proposed virtual image synthesis and transformation for autonomy (VISTA) to train ego based on real-world data.

Training based on capturing real-world data is susceptible to unforeseen driving situations and in some cases can cause trouble. To that end, other researchers use simulation to generate a world around the ego and extract data from that environment for training. We will create a simulation environment with help of SUMO and during different episodes, we will train the ego for both normal driving and responding to the emergency vehicle. The following graph in Figure 3.1 shows the high-level workflow for solving this problem.

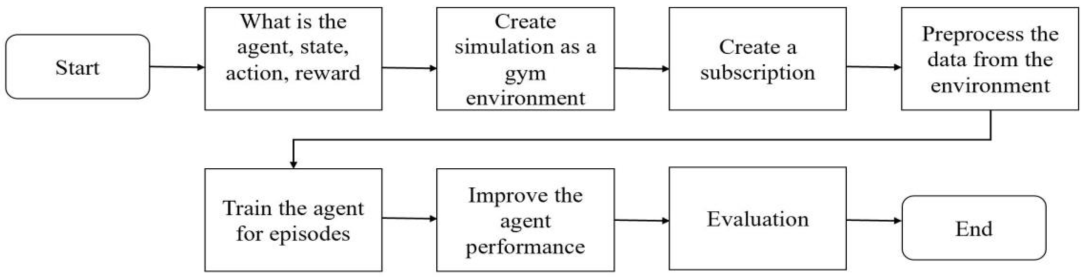
Start
What is the agent, state, action, reward
Create simulation as a gym environment
Create a subscription
Preprocess the data from the environment
Train the agent for episodes
Improve the agent performance
Evaluation
End

Figure 3.1 High-Level Workflow

3.2 RL Framework

The Reinforcement Learning framework has five main components as an agent, environment, state, action, and reward that we will introduce as follow. We defined these components based on the proposed problem.

3.2.1 Agent

In the RL framework, an agent needs to sense the state in the environment and take action to change that state to the new one. In this research, the agent is the autonomous vehicle (AV) let's call it ego wants to learn the best longitudinal and lateral decisions when approached by an emergency vehicle in a highway environment.

When the simulation starts, we consider a box with length (*obs_range)* around the ego with the number of vehicles observation (*nb_obsrv)* limitation.

The state array for the ego consists of positions (x_i, y_i) and speed (V_i) of all the vehicles in the (*obs_range)*. The agent (ego) senses the environment at each step of the simulation and based on this data takes the best action and transit to the new state.

The ego can observe the environment with the limitation of the 10 vehicles. This observation includes the position (x_i, y_i) and the speed (V_i) of the surrounding vehicles. So, we consider the following array with 31 of elements that create the state of ego in each step. We consider 20 elements for positions and 10 elements for the speed of surrounding vehicles and a binary variable that shows the presence of the emergency vehicle as shown in Figure 3.2.

1/0	X_1	Y_1	V_1	….	X_{10}	Y_{10}	V_{10}

Figure 3.2 Input Data Representation

3.2.2 Environment

As we mentioned in Chapter 2, the environment is one of the most important components of the RL framework. The agent can experience different states and take different actions in an environment. One of the famous toolkits for creating different RL environments is OpenAi Gym https://gym.openai.com/.

For autonomous driving, we need to build a traffic simulation environment, and there are some candidates like Simulation Urban Mobility (SUMO), CARLA, and Udacity Self-Driving Car Simulator https://github.com/udacity/self-driving-car-sim.

The benefit of using open-source libraries like SUMO is that we can trust the accuracy of the information extracted from these traffic simulations. The negative point of using traffic simulators is that because of the generalization problem of these simulators, it could be difficult to build an environment and all the stakeholders on top of them.

Based on all our needs, in this research, we prefer to use SUMO as an accurate and compatible traffic simulator. SUMO as a 2D simulator with a combination of Traffic Control Interface (Traci) [53] can handle different traffic scenarios. It is worth mentioning that both CARLA and Udacity Self-Driving Car Simulator can work as 3D simulators. However, they are associated with significantly higher computation costs as they rely on Graphics Processing Units.

For building the environment, we started with creating three .xml files. The first .xml file is simulation.node.xml which is defined all the nodes of simulation. We consider two nodes *A* and *B* as the beginning and ending of the route with a 2000-meter length.

The first node A has *(x = 0, y = 0)* coordinate and the second node B has *(x = 2000, y=0)* coordinate. The second .xml file simulation.edg.xml determines the edge between two nodes. The third .xml file is simulation.rou.xml which determines the type of vehicles that travel in the route. We consider three types of vehicles in this study, the first type of vehicle is the emergency vehicle *(emg)*, the second type of vehicle is the passenger *(veh)*, and the third type is the autonomous vehicle *(ego)*.

The speed factor determines the desired speed which multiplies with the road speed limit. An emergency vehicle with a speed factor of 1.5 means this vehicle can travel 50 percent above the road speed limit. Table 3.1 shows all the parameters related to these three types of vehicles.

Table 3.1 Vehicle Types

Type	Length(m)	Width(m)	maxSpeed(m/s)	minSpeed(m/s)	speedFactor
emg	16.0	2.55	50	20	1.5
ego	4.8	1.8	30	15	1
veh	4.8	1.8	30	20	1

The last essential .xml file is simulation.net.xml which has node file and edge file as an input and for the output, it generates the network file. The simulation.net.xml creates all the lanes and connects all the nodes with edges.

For more information about how to define different nodes, edges, routes, and networks we refer readers to SUMO documentation in https://sumo.dlr.de/docs/Tutorials/Hello_World.html.

The simulation configuration file will be created by combining all the previous .xml flies, and we will have a highway with a length of (*highway_lentgh*) and three lanes (l_0, l_1, l_2) from rightmost to the left where both left and right lane changing is allowed.

At the beginning of each experiment, vehicles are spawn into the environment with different parameters which some deterministic and some random.

The emergency vehicle (emg) always starts travel from $(x_{begin} = 0)$ in the emergency lane (l_1) and has a random departure speed of [20, 50] *(m/s)*. The ego vehicle will be departure with a random position in the first 200 meters of the highway in front of the emg vehicle in one of three lanes. The ego can travel in all the lanes (l_0, l_1, l_2). Other vehicles with the type (*veh)* are responsible to make the longitudinal and lateral control harder for the ego, they spawn in (l_0, l_2) with random speeds of [15, 30] *(m/s)* with IDM as a longitudinal control.

Highway environment that was built with the help of SUMO shown in Figure 3.3. The green vehicle is ego, blue vehicles are passenger-type vehicles, and the vehicle with the emergency sign is an emergency vehicle.

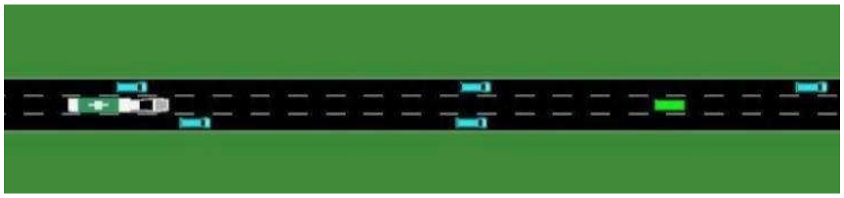
Figure 3.3 Highway Environment

All the configurations were implemented with the help of SUMO which is working as a server.

We need a command-response exchange between the client and SUMO. This connection between our python code which is responsible for training the agent and deployment of the actions with SUMO as a server handle by Traci.

Figure 3.4 shows the connection between the Python module (DRL-code) and SUMO as an environment.

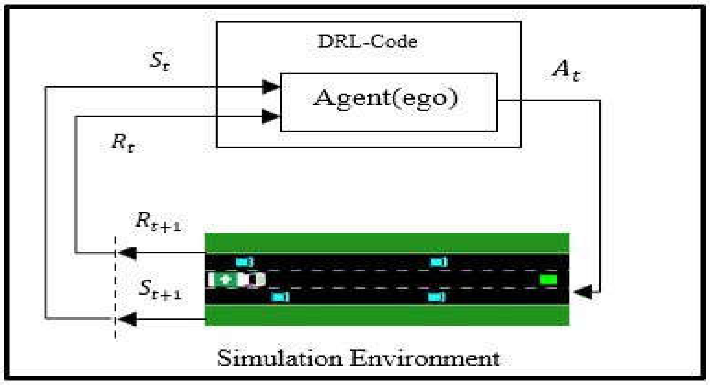

Figure 3.4 DRL and SUMO Connection Through Traci

With help of Traci, it is possible to retrieve the values of all the objects in simulation and change their behavior online. For more information about Traci, we refer readers to the Traci documentation on https://sumo.dlr.de/docs/TraCI.html.

Some termination conditions caused one episode to finish, and the environment reset to the new one. Table 3.2 shows all the termination conditions.

Table 3.2 Termination Conditions

Reason	Definition
Out of road	The ego travel in the right lane l_0 and chooses action change right lane or if the agent is in the left lane l_2 and chooses action a_1.
Accident	An accident of the ego with other vehicles.
Out of sense	Emergency vehicle passes the ego becoming out of observation range after recognition.
Endpoint	ego or emergency vehicle reach the endpoint.

3.2.3 Actions

Based on different states the ego needs to take different actions. We consider 5 actions that the ego can take for controlling the lateral and longitudinal movements. Table 3.3 shows all the actions that we consider in this research.

Table 3.3 Actions

Action	Definition
a_0	Stay in the current lane, keep speed
a_1	Change to the left lane
a_2	Change to the right lane
a_3	Stay in the current lane, increase speed 3 m/s
a_4	Stay in the current lane, decrease speed 5 m/s

These are actions that the ego can take in each step of the simulation. Agent (ego) by taking one of the above actions change the state of to the new state which can bring a reward for the agent.

3.2.4 Reward

An agent will train to take the best actions based on the reward that gain from the environment. This reward in the Ava example was a small biscuit after bringing the ball back. In this research, we consider negative rewards as a penalty. Therefore, if the ego makes a bad decision, it will cause a penalty for the ego and the goal is to minimize the penalty in each step of the simulation.

Table 3.4 shows all the penalties and the definitions for each penalty.

Table 3.4 Penalties

Definition	Penalty
The ego travels out of road boundaries	$P_{out\_of\_road\_penalty}$= -200
The ego has a collision with other vehicles	$P_{collision\_penalty}$ = -150
Each time ego changes a lane	$P_{Change\_lane\_penalty}$ = -1
Each step that ego and emg have the same lane	$P_{Same\_lane\_penalty}$= -5
Each step that the agent recognizes and has a speed greater than emg	$P_{Speed\_penalty}$= -1

If the ego travels out of the road or if it has an accident with other vehicles the episode will be terminated. However, for other penalties, ego can continue the episode. The reason we consider a small penalty for changing the lane is that we don't want to ego change the lanes continuously. Also, we consider a penalty for speed violation that check ego learns to decrease the speed after recognizing the emg. For each step of one episode, the summation of all the previous penalties is consider as a penalty that the agent will get from the environment.

3.3 Deep Reinforcement Learning for Autonomous Driving

Google DeepMind researchers develop the deep Q-network (DQN) [9], and they showed that the DQN as a combination of the Neural Network and Reinforcement Learning could have an appropriate result on the Atari 2600 games and gain a comparable outcome to professional human gamers.

Figure 3.5 shows the workflow in a single step of one episode based on the DRL. We want to approximate Q*(s, a) in each step to take the best action. Approximating the Q(s, a) with Neural Network can be a challenging task, due to the need of lots of data and more importantly high correlation between consecutive states, and actions can cause oscillation on weights of the network and divergence.

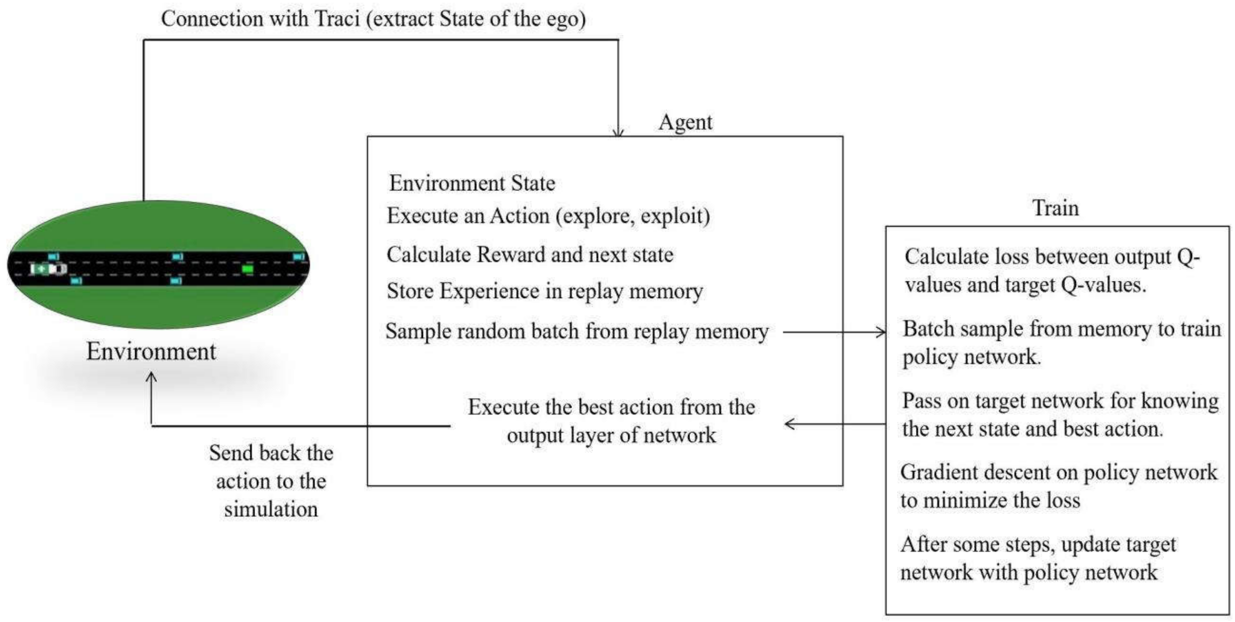
Connection with Traci (extract State of the ego)
Agent
Environment State
Execute an Action (explore, exploit)
Calculate Reward and next state
Store Experience in replay memory
Sample random batch from replay memory
Execute the best action from the output layer of network
Train
Calculate loss between output Q-values and target Q-values.
Batch sample from memory to train policy network.
Pass on target network for knowing the next state and best action.
Gradient descent on policy network to minimize the loss
After some steps, update target network with policy network
Environment
Send back the action to the simulation

Figure 3.5 Workflow in A Single Step of Simulation and Training

To solve the divergence problem, researchers use two concepts experience replay memory and the second network as a target network. Each experience with form of tuple (s, a, s', r) store into the memory with a capacity of *max_cap* when the memory filled with *max_cap* it will remove the oldest experiences. This memory is called experience replay memory. Therefore, instead of feeding the network with one state, the input value will be in the size of *minibatch_size*.

This random minibatch can reduce the correlation between input data.

DQN algorithm includes two networks named policy network and a target network. The loss will be calculated based on the Q-value from the action of the experience tuple and the target value of that action. At the first, we don't know the value of Q-value in the target network, and we want to approximate it. To calculate the target value, we need the second pass with the next state S' if the next state will not be a termination state.

Then, we can obtain the maximum Q-value among the possible actions that can be taken for that next state and use this value in the Bellman equation to calculate the target Q-value of the action. For reaching the optimum network the following loss function in (3.2) needs to be minimized, and after some steps, the target network θ^- will be updated from the policy network.

The target network will be updated after *target_update_counter* times.

$$L(\theta) = E_{s,a,r,s'}[\left(r + \gamma\, max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2] \tag{3.2}$$

Where $max_{a'} Q(s', a'; \theta^-)$ is optimum value of Q-value from the target network and $Q(s, a; \theta)$ is Q-value from the policy network. This loss function will be updated with help of gradient descent in (3.3).

$$\nabla_\theta L(\theta) = E[r + \gamma\, max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))\, \nabla_\theta Q(s, a; \theta)] \tag{3.3}$$

The reason we consider two networks and not using the policy network for the second pass is that when our policy network update to be closer to the target Q-values also the target Q-value is also moving in the same direction because we use the same network architecture and same weights to calculate these values. To solve this problem, we consider the second network as the target network and update the weights of this network after some steps which is denoted by *target_update_counter* as a hyperparameter.

To summarize all the previous procedures the following algorithm is presented as our solution based on DQN for solving the decision-making problem of autonomous driving.

Algorithm 2. proposed solution based on DQN

```
Initialization:
    Initialize replay memory with maximum capacity max_cap.
    Initialize the policy network with random weight.
    Clone the policy network as a target network.

For i in range number_of_simulation:
    start simulation with Traci
    reset the environment
    add the vehicle and set all the controllers
    extract state from one step of the simulation
    preprocess the input data
    Based on $\varepsilon$-greedy algorithm (exploration, exploitation)
    Select an action
    Execute the action in the simulation
    Calculate the new state and reward
    Store the new experience in experience replay memory
    Sample random batch from the experience replay memory
    Preprocess states from the batch
    Pass the processed batch to the policy network
    Calculate loss between target Q-values and output Q-values as
        $L(\theta) = E_{s,a,r,s'}[\left(r + \gamma\ max_{a'}Q(s',a';\theta^{-}) - Q(s,a;\theta)\right)^{2}]$
    Gradient descent to update the weights of the policy network to minimize the loos.
        $\nabla_{\theta}L(\theta) = E[r + \gamma\ max_{a'}Q(s',a';\theta^{-}) - Q(s,a;\theta))\ \nabla_{\theta}Q(s,a;\theta)]$
    After reaching to target_update_counter update the weight of the target network with
    weight of the policy network.
```

With this algorithm after a considerable number of training steps, we will end up with a trained neural network that can choose the best actions based on the optimum Q-values.

Before the description of the neural networks in more detail, we need to normalize the input values that are extracted from the simulation in each step. To that end, we consider following normalization in 3.4, 3.5, and 3.6 for positions and speed normalization in each step of the simulation in (*obs_range).*

$$Position\_xi = \frac{Position\_xi}{lentgh\_of\_road} \tag{3.4}$$

$$Position\_yi = \frac{Position\_yi}{lentgh\_of\_road} \tag{3.5}$$

$$speed_{\ vi} = \frac{Speed_{vi} - Min\_speed\_vi}{Max_{\_speed_{vi}} - Min\_speed_{vi}} \tag{3.6}$$

3.3.1 Neural Networks Architecture

In this research, we consider two types of neural networks structures. The first network is a fully connected feed-forward network and helps us to have DQN$_{FNN}$ solution. This network is a multi-layer perceptron with 31 input neurons as we elaborated in the agent section, in each time step of the simulation agent need to consider a maximum of 31 input variables. The output layer consists of five neurons as we introduced five actions in Table 3.3. This neural network will have 3 dense layers with 128 neurons and all the neurons connected to the next level to create a fully connected network.

We consider Rectified Linear Unit (ReLU) [54] as an activation function. ReLU as an activation function, output the values greater than zero exactly as they are, otherwise, output the other values equal to zero. ReLU is a popular activation function that allows gradient backpropagation without vanishing or exploding. Figure 3.6 shows how ReLU as an activation function works.

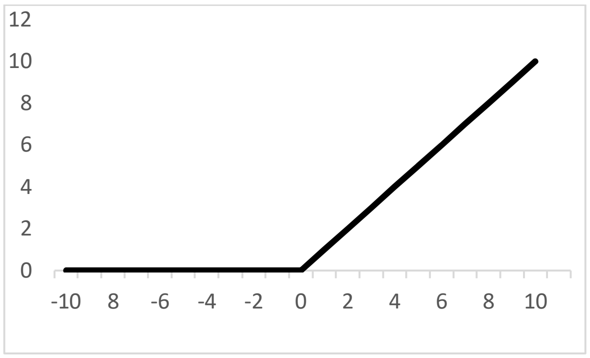

Figure 3.6 ReLU Activation Function

The second network was designed with convolutional layers and create DQN$_{CNN}$ solution. Convolutional layers can learn complex and useful patterns from the input data.

The first layer includes 32 filters, a kernel size of 3, and ReLU as an activation function. The second Conv layer has 64 filters, kernel size of 1, and ReLU as an activation function. The third layer includes 128 filters, kernel size of 1, and ReLU as an activation function. There are some max-pooling layers as aggregation layers after each Conv layer.

The summary of both DQN$_{FNN}$ and DQN$_{CCN}$ are presented in table 3.5 and table 3.6.

Table 3.5 DQN$_{FNN}$ Structure

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 31)]	0
dense_1 (Dense)	(None, 128)	4096
dense_2 (Dense)	(None, 128)	16512
dense_3 (Dense)	(None, 128)	16512
dense_4 (Dense)	(None, 5)	645

Table 3.6 DQN_{CNN} Structure

Layer (type)	Output Shape	Param #
all_input (InputLayer)	[(None, 31, 1)]	0
conv1d (Conv1D)	(None, 31, 32)	128
max_pooling1d	(None, 8, 32)	0
conv1d_1 (Conv1D)	(None, 8, 64)	2112
max_pooling1d_1	(None, 6, 64)	0
conv1d_2 (Conv1D)	(None, 6, 128)	8320
max_pooling1d_2	(None, 1, 128)	0
dense (Dense)	(None, 1, 128)	16512
dense_1 (Dense)	(None, 1, 128)	16512
dense_2 (Dense)	(None, 1, 128)	16512
dense_3 (Dense)	(None, 1, 5)	645
reshape (Reshape)	(None, 5, 1)	0

For implementation and all the computation behind neural network structures, we used TensorFlow https://www.tensorflow.org/ as a machine learning platform and on the top of the TensorFlow we used Keras https://keras.io/ as a deep learning API that is written in python.

3.3.1 Dueling Deep Q-Network

Researchers in [55] improved the performance of the DQN and introduced the dueling version of this algorithm. The dueling version is the optimized DQN that has a modification on the output layer.

They introduced the Dueling Deep Q network (DDQN) with two streams. Figure 3.7 shows the DQN structure and Fig 3.8 shows DDQN structures.

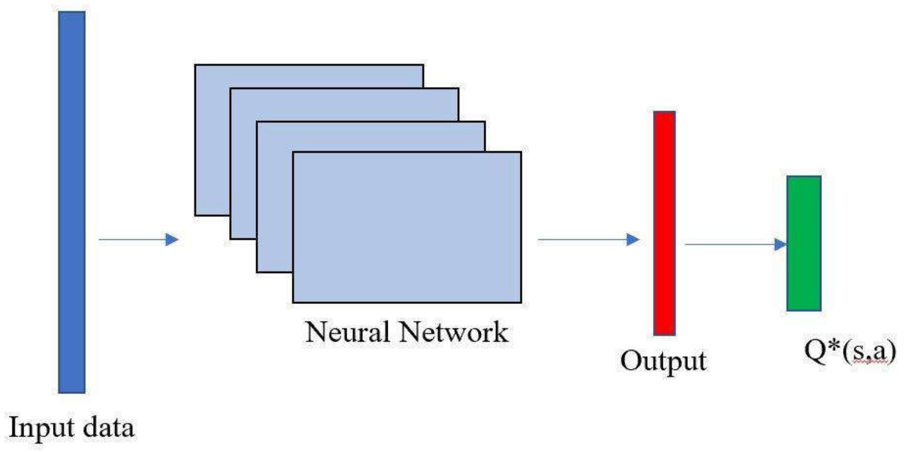

Figure 3.7 DQN Structure

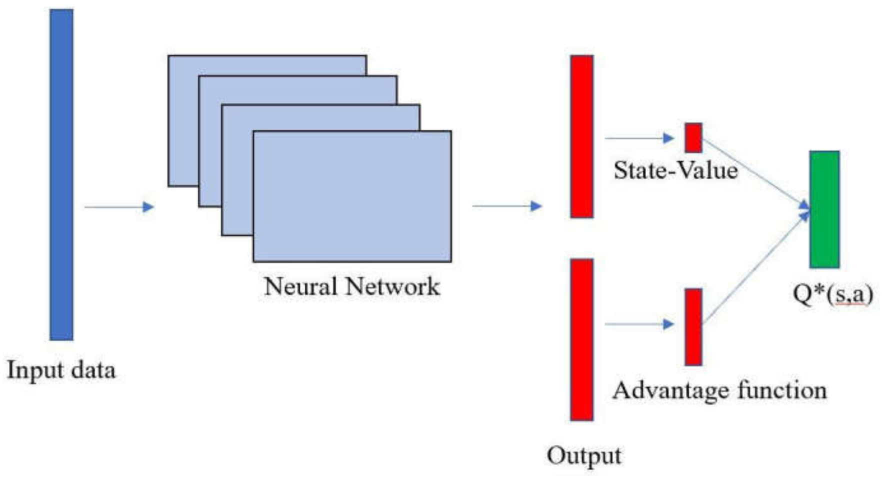

Figure 3.8 DDQN Structure

The dueling network has two steams: one is state-value (the expected reward of state *S*) as a scaler and the second stream determines the advantage of each action. By a combination of these two streams, we will have Q-value as normal DQN. The critical point behind DDQN that makes it better compared to DQN is that it is unnecessary to calculate the value of each action in each state Q (s, a) for every step due to the fact that, not all the states have the same learning capability.

For example, knowing how to change the lane or speed is very important in a situation that ego recognizes both (*veh)* and *(emg)* vehicles in (*obs_range)* compared to the case that there is no vehicle around the ego in the observation range. The Q-value in the DDQN structure is calculated by (3.7).

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

The above equation used the concatenate the value of each state and action advantages. Researchers in [55] proposed the Q-value of the DDQN as (3.8).

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + [A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum A(s,a';\theta,a)] \tag{3.8}$$

The convolutional layers parameter denoted by θ, and parameters of two streams of fully connected layers denoted by α and β. All the other components of both DQN and DDQN structure are the same instead of the last layer when the last dense layer makes two streams and after that concatenate to create the Q-values. This strategy in dueling architecture brings better convergence and results for the DDQN compared to the DQN structure.

In this chapter, we focused on the problem definition and solution based on Deep Reinforcement Learning. We introduced DQN$_{\text{FNN}}$, DQN$_{\text{CNN}}$, and DDQN as solutions for the proposed problem. In the next section, we will describe the implementation process and computational results in more detail.

Chapter 4: Computational Result

In this chapter, we will review the implementation process and the result of both rule-based and RL-based solutions. After that, we will compare the performance of different solutions that are presented in Chapter 3 by considering various criteria.

4.1 Implementation

For the implementation part of this research, we use Python as a programming language. The following Python APIs and libraries also are used as shown in Table 4.1.

Table 4.1 APIs and Python Libraries

API/ library Name	Task Description
Sumolib	Help to generate the simulation components
Traci	Real time control of the simulation
NumPy	Mathematical functions, matrix operation, random number generator
Matplotlib	Visualization
Math	Mathematical functions
TensorFlow	The main platform of ML and DL
tensorflow.keras	Keras on the top of TensorFlow as a DL API
tensorflow.keras.optimizers	Adam as an optimizer
tensorflow.keras.models	Create a sequential model
tensorflow.keras.layers	Create different type of neural network layers

Figure 4.1 shows the main classes, methods, and python files that were determined for coding purposes.

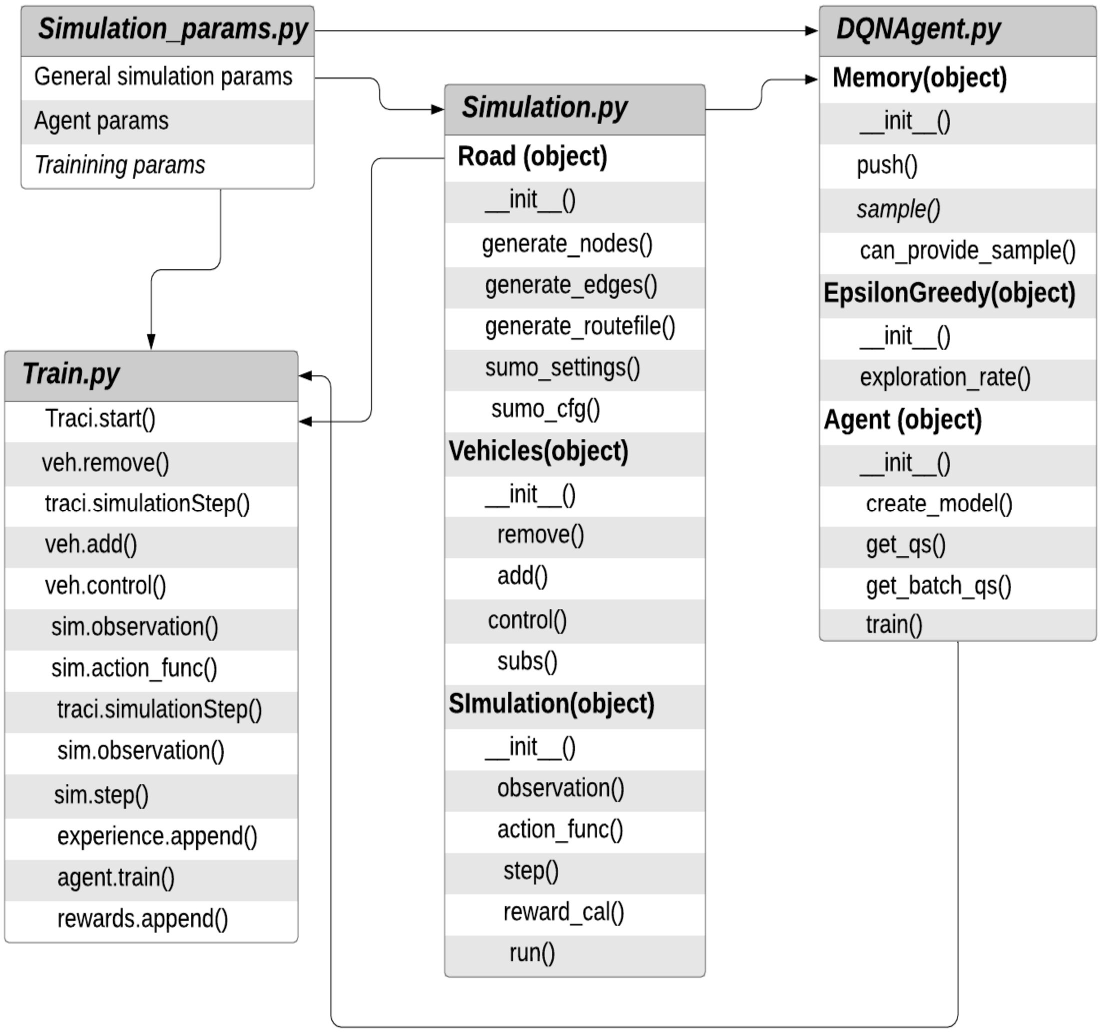

Figure 4. 1 Upper-Level View of Implementation

The first python file is *simulation_params.py* this file includes all the essential parameters. All the parameters related to the simulation, agent, and training process.

The second python file is *simulation.py* and it has five methods for generating nodes, edges, route, setting, and configurations.

The next class is vehicles include removing, adding, control, and subscribing methods. At the beginning of the simulation, all the vehicles remove first from the simulation and make sure the simulation has no vehicle from the previous episode. Then new vehicles add to the simulation with *add()* method. The *control()* method sets all controllers that vehicles need during simulation. This method makes sure which vehicles use the rule-based controller and makes sure ego has no default controller. As we mentioned before we need to connect to the SUMO as a server and subscribe. Therefore we will subscribe for all the useful information with *subs()* method. This subscription gets all the position, speed, and type of all vehicles that are in the observation range of the ego.

The last class of the *simulation.py* do a major part of the simulation.

The *observation()* method returns the state of the ego by concatenation of the vehicle type, position, and speed of the vehicles.

The next method is action*_func()* that executes the action, this action can be a random action or action predicted from the neural network.

After execution of the action, the next method is *step()* which determines what is the consequence of the actions that the agent took. This method includes some queries about the collision of the ego with other vehicles, travel out of the road boundaries, reaching to the end of the route, termination conditions, relative speed and position of the ego to the emg, changing lane, out of access condition, number of times ego and emg have the same lane, and speed violation.

The *step()* method is responsible for all the queries and calculations in each step of one episode.

The *reward_cal()* method helps to calculate the reward of the agent in each step of episodes based on the parameters presented in Table 3.4.

The last method of *simulation.py* is the *run()* method which is responsible to start the simulation and initialize all the previous methods for the number of the episodes.

The *DQNAgent.py* is responsible for creating all the agent needs. The first class is *Memory(object)* that creates the experience replay memory. The *push()* method, fill the experience replay memory with new experiences. These experiences are collected based on the tuple *(s, a, s', r)*.

The *sample()* method extracts a batch with *minibatch_size* from the experience replay memory, and *can_provide_sample()* makes sure there are enough experiences in the memory.

The second class is *EpsilonGreedy(object)* that is responsible for $\varepsilon - greedy$ algorithm. *exploration_rate()* determine the value of the ε in each iteration. This value starts from one and decays in each iteration.

The last class of the *DQNAgent.py* is responsible for creating the neural network, prediction, and training.

The *create_model()* is responsible for creating the fully connected layers in DQN_{FNN} and convolutional layers in DQN_{CNN} and return the model. Predict the Q-value with single transition calculated by *get_qs()* method. The Q-value prediction based on the transition with *batch_size* is calculated by *get_batch_qs()* method.

The last method of *DQNAgent.py* is the *train()* which is the main part of the agent, and it needs *termination_state* and *minibatch* as input of the method.

The main part of the *train()* presents as the following pseudocode which has the primary roles for training the agent.

```
For a sample (current_state, rand_action, new_current_state, reward, done) in minibatch:
    If done is False:
        Calculate the maximum q_value predict by target network
        new_q_value is equal = reward + discounted maximum q-value
    Else:
        new_ q_value = reward
create the list of the future states as input data.
create the list of the new_ q_value for each action as target data.
fit the model with input data and target data.
```

All the previous Python files import to the last Python file as *Train.py* which is connect all the essential modules which is defined in the previous files together.

With this configuration, the agent starts training and because of the nature of the exploration, exploitation the agent makes bad decisions and gain penalties during the first episodes. By having more and more episodes and experience different situations the ego starts learning how to make the best decisions in different situations.

4.1.1 Parameters and Hyperparameter

One of the essential python files is Simulation_params.py which is provide all the parameters and hyperparameters.

Table 4.2 shows all parameters relates to the simulation and Table 4.3 shows parameters related to Agent and RL framework. Also, there are some other parameters related to the rule-based algorithm and vehicle types that were introduced in Table 2.1 and Table 3.1.

Table 4.2 Simulation Parameters

Parameters / Simulation	Value
Number of episodes, *episode_num*	5000
Number of vehicles, *N*	20
Minimum road speed, V_{min}	15 (m/s)
Maximum road speed, V_{max}	60 (m/s)
Highway length, $d_{highway}$	2000 (m)
Width of lane, *w*	3.2 (m)
Number of lanes, *n*	3

Table 4.3 RL Framework Parameters

Parameters / RL-framework	Value
Observation range, *obs_range*	100 (m)
Maximum vehicle sense, *Scap*	10
Initial epsilon, $\varepsilon_{\text{start}}$	1
Decay epsilon, $\varepsilon_{\text{decay}}$	0.0001
Learning rate, η	0.001
Discount factor, γ	0.99
Maximum memory size, $\text{M}_{\text{maxsize}}$	500
Update target network, σ	5
Mini batch size, $M_{minibatch}$	32
Minimum memory size, $M_{minsize}$	100

4.2 Computational Result

The goal of this research is having an autonomous vehicle can address both normal highway driving and when the autonomous vehicle is approached by an emergency vehicle.

To solve this problem, two solutions based on a rule-based solution as a baseline with a combination of the IDM and MOBIL and the second approach as a DRL-based approach were proposed. we proposed DQN_{FNN}, DQN_{CNN}, and $DDQN_{CNN}$ as DRL-based solutions.

For evaluating the performance of the ego after some episodes we consider five criteria each of which is defined as follow:

1. *Number of accidents*

   Calculate the number of times that ego caused an accident. If the ego caused the accident the episode will be terminated.

2. *Number of out of road*

   Calculate the number of times that ego travels out of the road boundaries. If the agent is in the right lane l_0 and chooses action a_2 or if the ego is in the left lane l_2 and chooses action a_1 can cause out of the road and the episode will be terminated.

3. *Number of times ego has the same lane with emergency vehicle*

   Calculate the number of times that both the vehicles have the same lane. The number of the times that ego has the same lane with emg counted for each step of one episode. having the same lane with emg in one step is not a reason for the termination of the episode.

4. *Speed violation*

   Calculate the number of times that ego observes the emergency vehicle, and it has a speed greater than the speed of the emg.

5. *Number of times emergency reach to the end before ego*

   Calculate the number of times an emergency vehicle reaches the end of the route before ego. As the departure position is $x_{begin} = 0$ and ego departure is a random position in the first 200 meters of the route. If the ego has a good performance on both longitudinal and lateral actions by changing the lane and decreasing the speed appropriately then emg reaches the end of the route before ego which is proof of the good performance.

6. *Reward function*

   In the RL framework, the reward that the agent gets from the environment based on the actions it takes in a different state is a sign for the performance evaluation.

   Mostly in the RL framework, we want to maximize the reward that the agent can get from the environment. However, in this research, as all the consequences of the actions are defined based on penalties, the agent wants to minimize the penalty that gets from the environment.

4.2.1 Rule-Based V.S DRL-Based

In this section, DRL-based and rule-based solutions performance will be compared based on all the six criteria introduced in the previous section.

The ego experience different situations in 5000 simulation episodes, each of which had a different number of steps base on the performance of the ego and termination conditions.

Some parameters need to be generated randomly in these episodes. However, due to the fact, these random numbers are generated as a random seed, the result of different solutions can compare with each other.

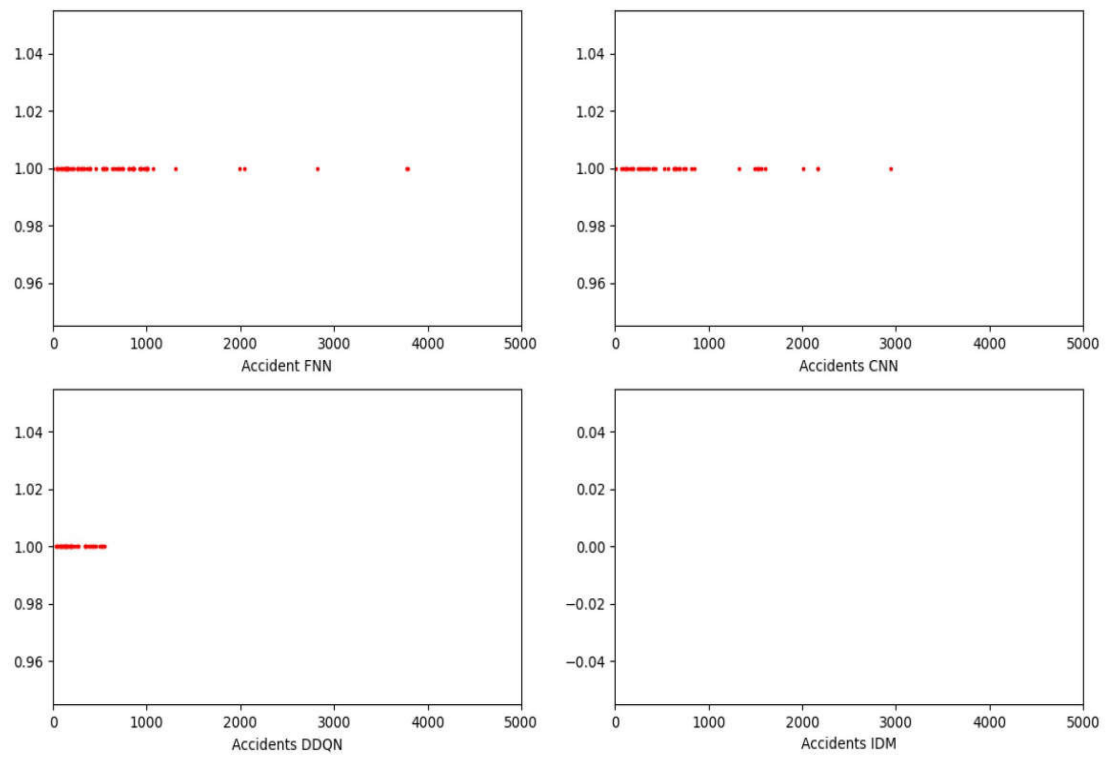

Figure 4.2 Number of Accidents

Figure 4.2 shows the number of accidents in 5000 episodes for four solutions.

Accident IDM representative of rule-based solution as a combination of IDM and MOBIL. As we can see there is no accident in 5000 episodes with IDM and the reason behind this is that MOBIL performs conservatively on changing the lane and IDM sets the speed in a way that caused no accidents.

The other three solutions are representative of DRL-based solutions and as we can see DDQN has the best performance with fewer numbers of accidents.

All three approaches have more accidents in the first 1000 episodes and that's due to the bigger exploration rate when agents start with more exploration and decay the exploration rate to do more exploitation. The performance of DQN_{CNN} is better than DQN_{FNN}.

All the accidents in 5000 episodes, for DQN_{FNN} is 71, DQN_{CNN} is 46, and DDQN has 35 accidents. The second important criteria that we introduced is the number of times that the ego travels out of the road as shown in Figure 4.3 which determines that the ego can learn about the road boundaries and change lanes appropriately when traveling in the right lane or left lane.

As we can see the ego equipped by IDM and MOBIL has not out of road boundaries due to fact that the ego with rule-based solution knows the boundaries of the route and when it is in the right lane or left lane prevent taking the wrong lateral actions.

The performance of DQN_{FNN} and DQN_{CCN} are close to the out of roads times and DDQN has a better performance compared to the other two. As we can see DDQN agent learns after 500 episodes, but DQN_{FNN} and DQN_{CCN} need about 1000 episodes to learn about road boundaries.

In 5000 episodes, DQN_{FNN} has 815 times of out of the road, DQN_{CNN} has 776 times of out of the road, and DDQN has 537 times of out of the road in the episodes which agent try to explore more compared to exploit.

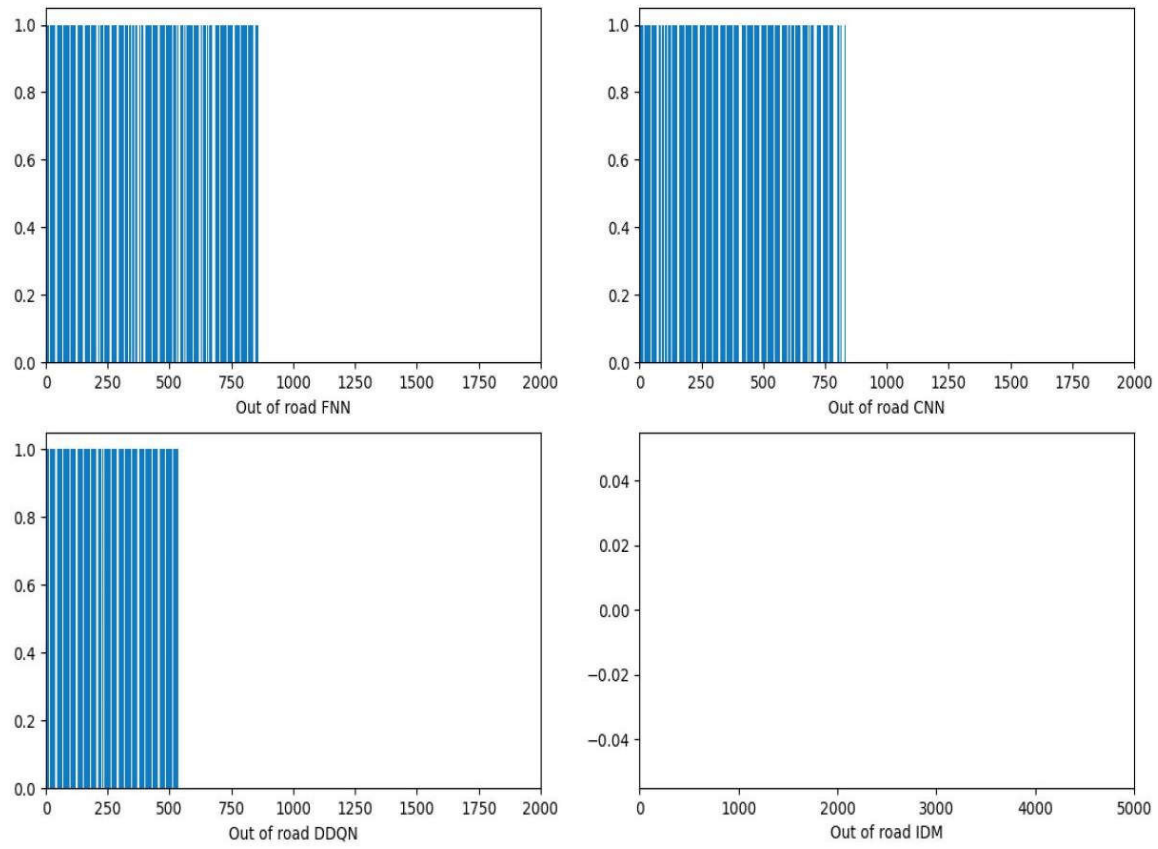

Figure 4.3 Number of Out of Road

The third evaluation is the number of times ego has the same lane as the emergency vehicle in the observation range. We defined the emergency vehicle which starts travel from $(x_{begin} = 0)$ in the emergency lane (l_1). Each time that the ego observes the emergency vehicle in the observation range, and they have the same lane (l_1) counted and shown in Figure 4.4.

In these plots, having fewer number times that ego has the same lane with the emg means better performance.

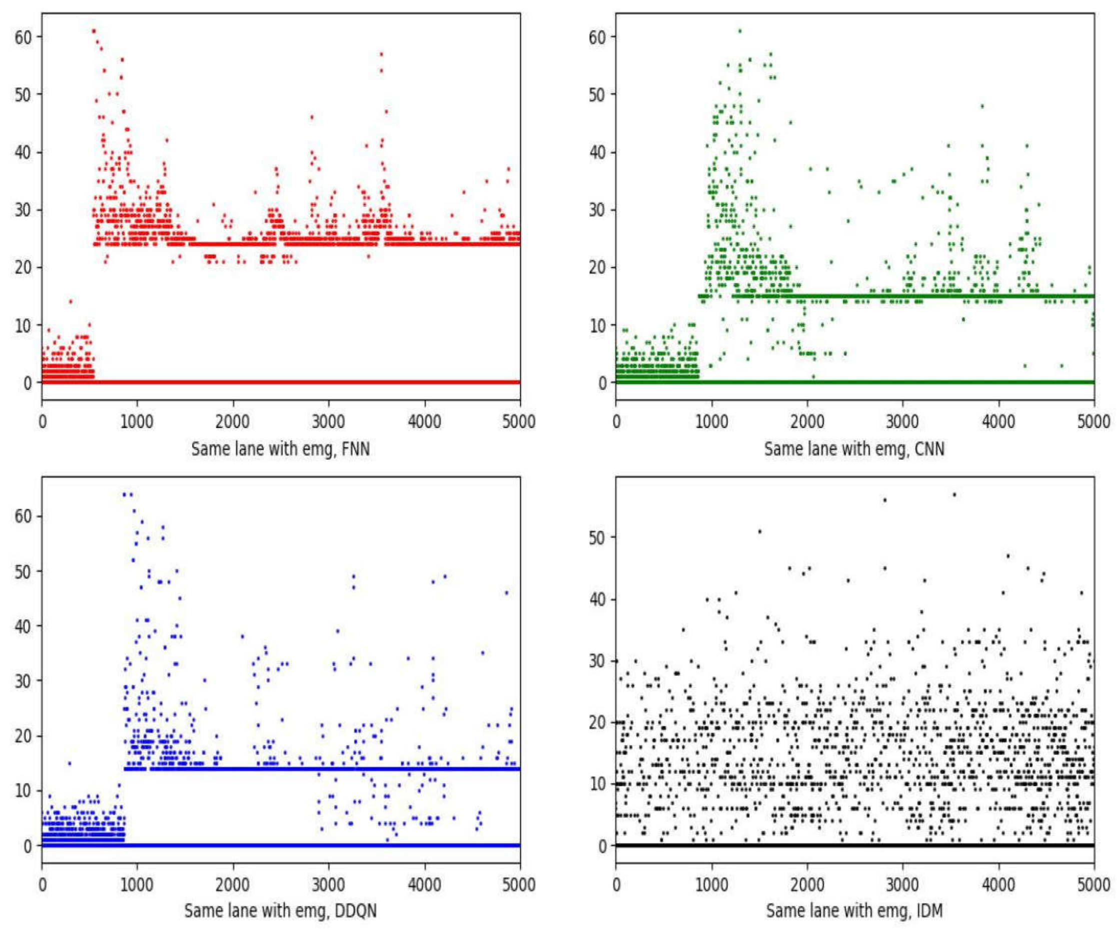

Figure 4.4 Number of Times Autonomous Vehicle (ego) has a Same Lane with Emergency Vehicle

Three DRL-based proposed solutions (red, green, and blue) have some similarities in their performance. For example, in the first 1000 episodes, the number of the same lanes with emg is mostly bound between 0 to 10 because in the first 1000 episodes the ego explores, and after few steps, the ego is terminated by an accident or out of the road.

After 1000 episodes the agent start learning about the number of times that has the same lane with emg. To have a fair comparison between all the four solutions we consider the mean and standard of deviation as shown in Table 4.4.

Table 4.4 shows that the DDQN has a better mean and standard deviation on the number of times ego has the same lane with the emergency vehicle in 3000 episodes compared to the DQN_{CNN} and DQN_{FNN}.

Table 4. 4 Mean and Standard Division Comparison

Method	Mean	Std
DDQN	6.859	7.65
DQN_{FNN}	12.63	12.63
DQN_{CNN}	7.97	8.28
DQN_{IDM}	5.18	8.75

Both DDQN and DQN_{IDM} have similar performance and it is hard to decide which one has a better performance on the number of times that ego has the same lane with emg.

The mean number is almost the same and DDQN has a better standard deviation compared to DQN_{IDM}. Therefore, we determine another measure for finding which one has a better performance on the number of same lanes with emg. Again, we consider 3000 episodes between episode 2000 to 5000. The reason not considering the first 2000 is that as we mentioned in those episodes DDQN has an accident or out of the road due to the exploration and as a result fewer number of times that has the same lane with emg.

For completing the comparison between ego DDQN and ego IDM, Figure 4.5 can help us.

This Figure works like a zero game when one of the players A or B is the winner of the game. From episode 2000 to episode 5000 the method which has a fewer number of same lanes with the emg is the winner of the game.

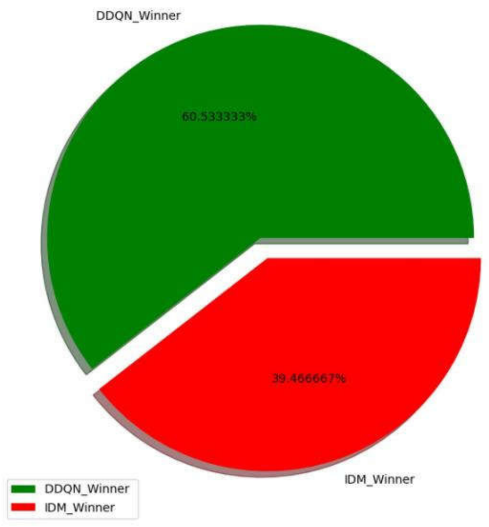

Figure 4. 5 Comparison Between DDQN and IDM on Same Lane with Emergency Vehicle

Figure 4.5 shows the outperformance of DDQN compared to the IDM on the number of times that ego has the same lane from episode 2000 to 5000. The ego DDQN has fewer times that has the same lane with emg in the same episode compared to ego_{IDM}.

The next evaluation is speed violation, as the number of times that ego sense emg in the observation range and it has a greater speed compared to the emg. Figure 4.6 shows the number of speed violations for DQN_{FNN}, DQN_{CNN}, DDQN, and IDM as a rule-based solution.

As we mentioned before to have a more accurate comparison, we don't consider the first 2000 episodes, where the ego mostly has an accident or is out of the road even before recognizing the presence of the emg in the observation range.

Figure 4.6 shows that DDQN with a mean and standard deviation of (10.48, 9.02) has a better performance compare DQN_{FNN} (15.28, 6.96), DQN_{CNN} (14.89, 6.98), and IDM (16.87, 17.44) as a rule-based solution. The reason the combination of IDM and MOBIL has a poor performance on speed violation is that with IDM the ego mostly wants to reach a certain speed and it doesn't change that speed by observing the emergency vehicle.

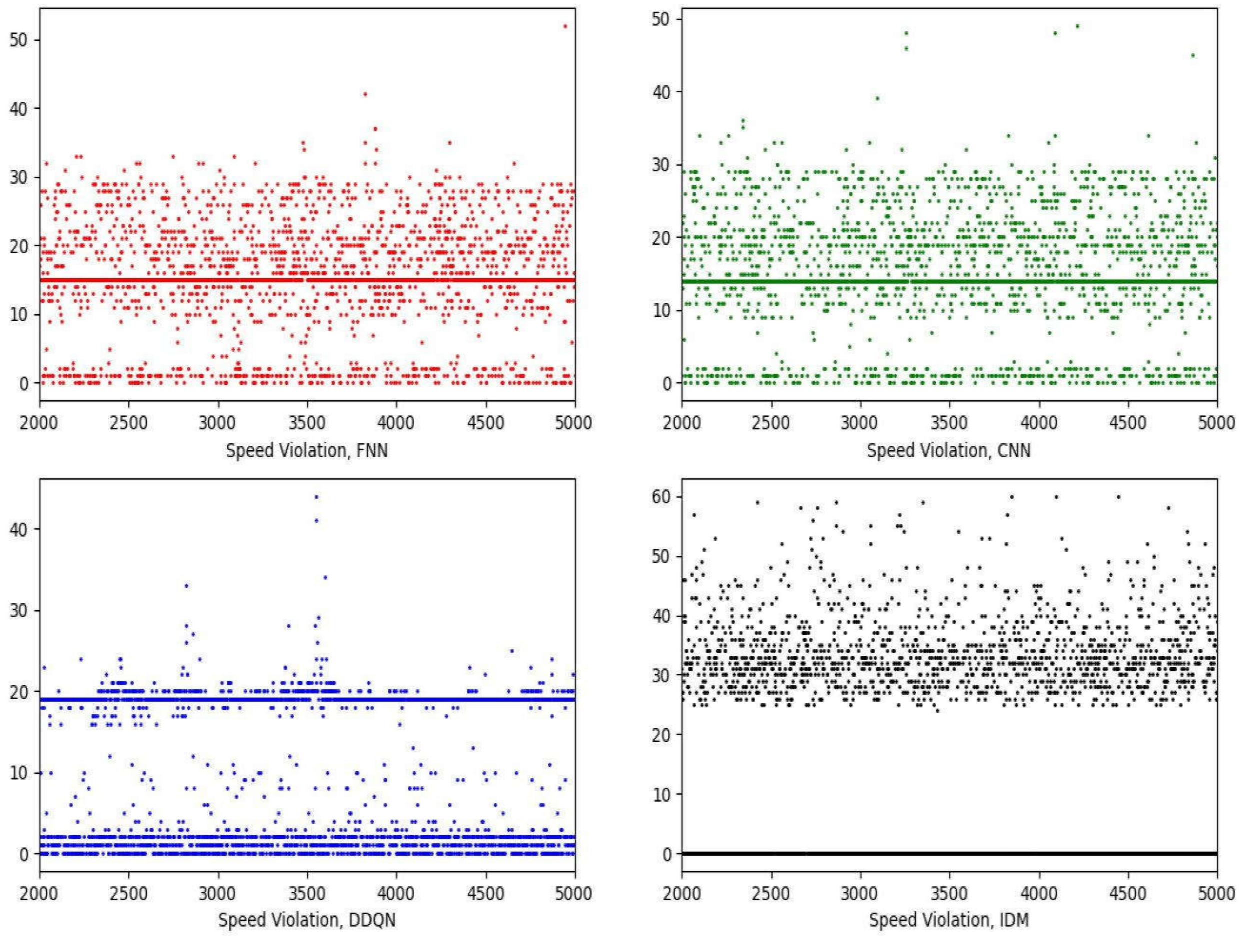

Figure 4. 6 Number of Speed Violation

Figure 4.7 shows that DDQN (red) and IDM (blue) at first has a similar number of speed violations between [0,5] for almost half of episodes between (2000, 5000).

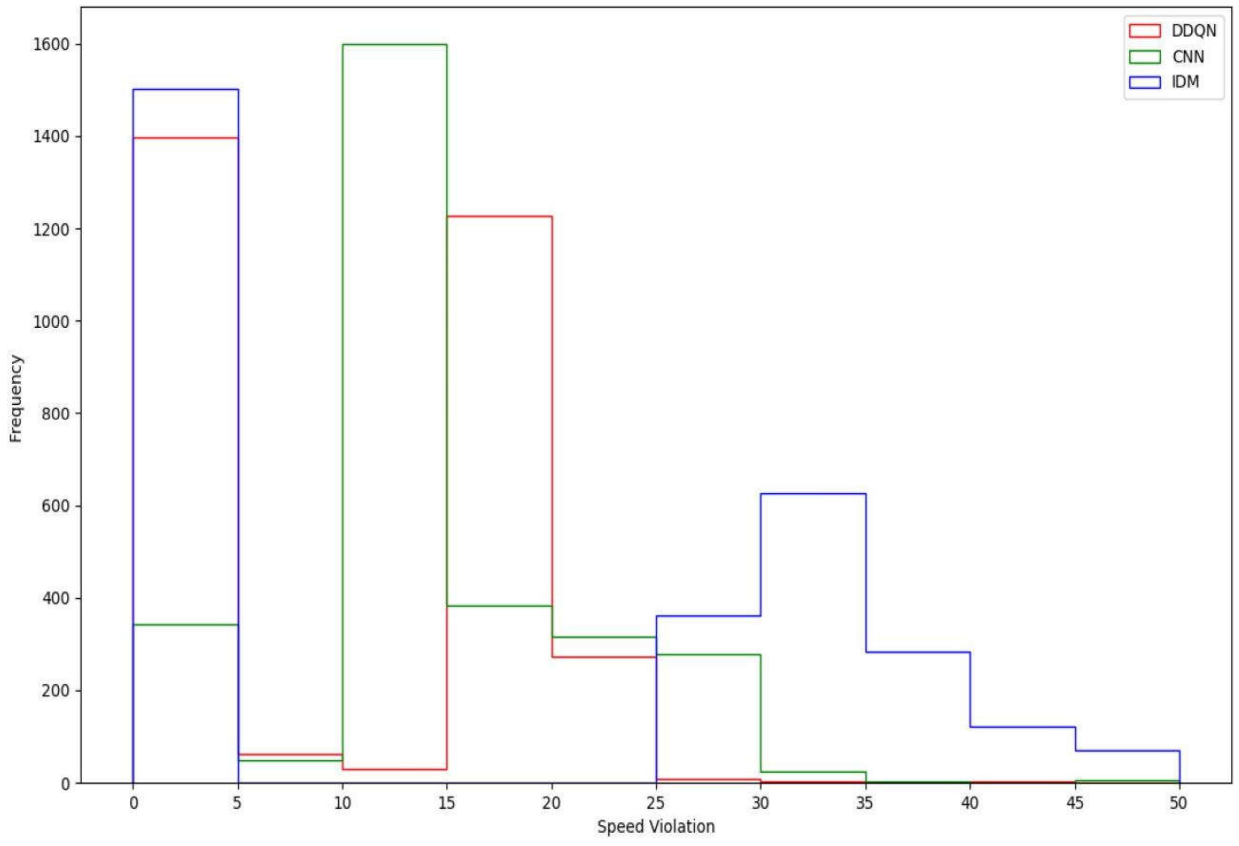

Figure 4. 7 Speed Violation Comparison

The DDQN has a better performance as we can see the other episodes has a speed violation fewer than 25 times. Otherwise, IDM for other episodes has 25 or more speed violations. For $\mathrm{DDQN}_{\mathrm{CNN}}$ for most episodes, the number of speed violations is between (10, 15) times.

The last evaluation feature is the number of the time emergency vehicle reach to the end of the route before the ego as an evaluation for both changing the lanes and decreasing the speed of the ego. The ego starts the travel in front of the emg so by changing the lanes and decreasing the speed appropriately the emg should reach to the end point before ego.

Figure 4.8 shows the number of times that emg reach the end of the road before the ego.

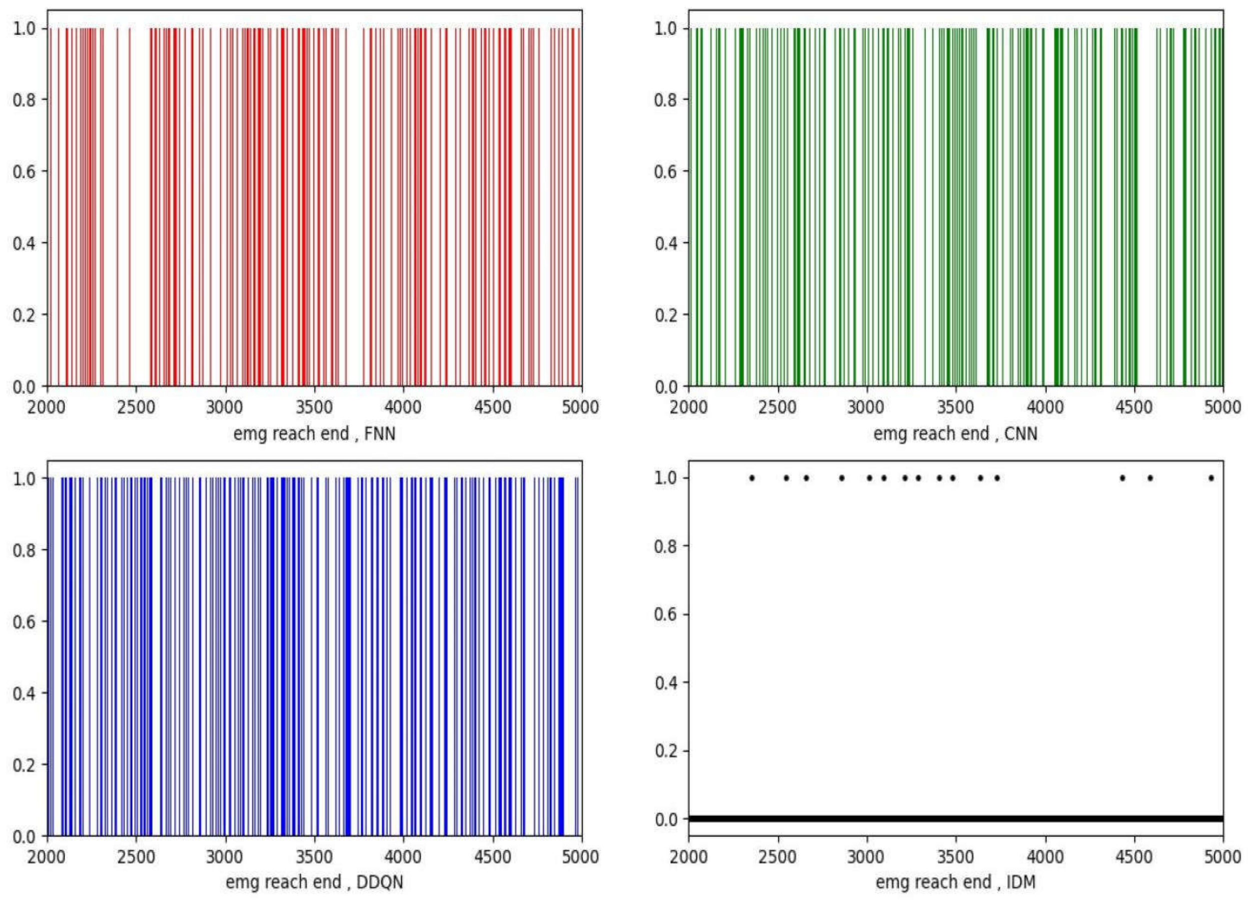

Figure 4.8 Emergency Vehicle Reaches to The End of Route Before ego

Figure 4.8 shows that all the DRL-based solutions have a better performance compared to rule-based solutions.

The reason for the bad performance of the rule-based solution is that IDM never decreases the speed because of the existence of the emergency vehicle and the only reason that IDM changes the speed is to avoid accidents.

For 3000 episodes between [2000, 5000] the number of times that emg end the road before ego with DQN_{FNN}, DQN_{CNN}, and DDQN is 979, 1001, and 1477 respectively. Therefore, we can conclude DDQN has a better performance compared to other solutions about the number of times that emg reach the end before ego.

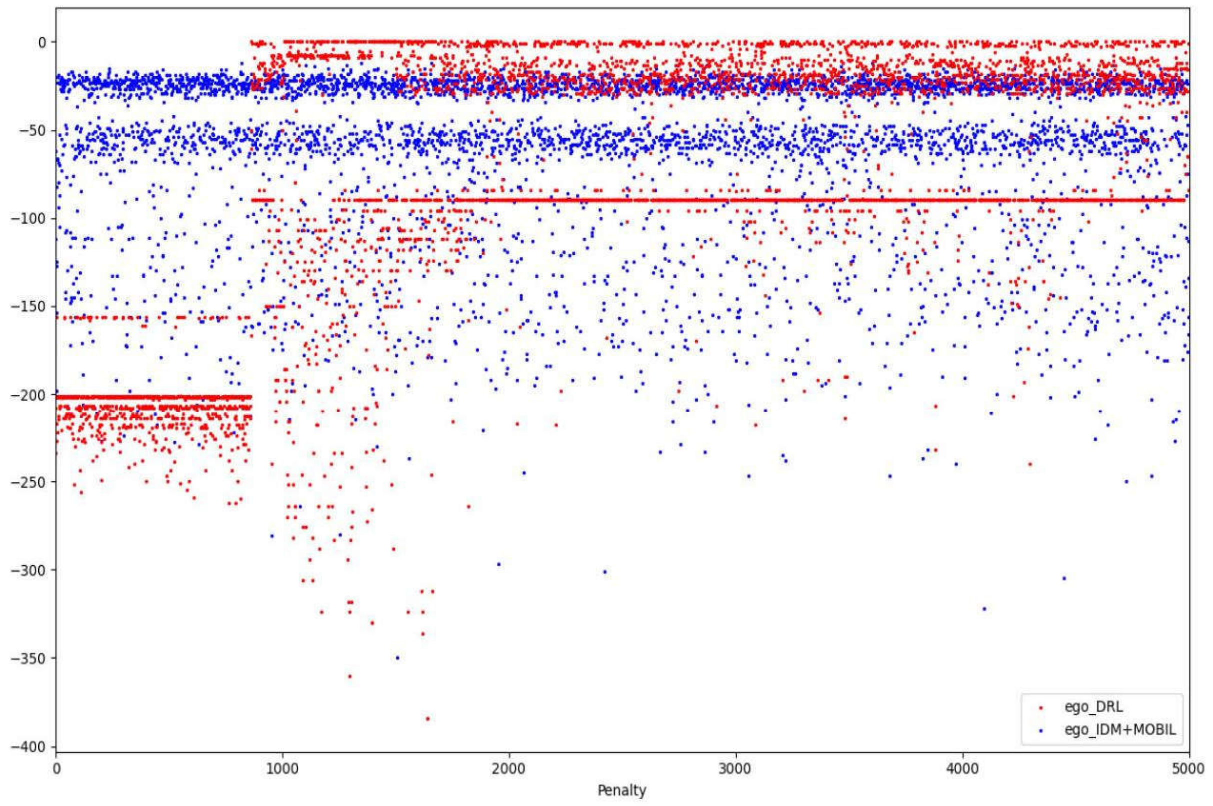

Figure 4.9 Penalty Comparison Between DDQN and IDM+ MOBIL

Figure 4.9 shows the penalties gained by ego equipped by DDQN and compares with penalties that agent gets by using a combination of IDM and MOBIL. Figure 4.9 shows the outperformance of the ego$_{DDQN}$ compared to the ego$_{IDM}$ as we can see the fewer penalties for ego$_{DDQN}$ (red dots) compared to the ego$_{IDM}$ (blue dots).

As we expected the DDQN has a better performance compared to the DQN$_{FNN}$ and DQN$_{CNN}$ on the normalized average reward as shown in Figure 4.10 and the mean normalized penalty of the 10 consecutive episodes.

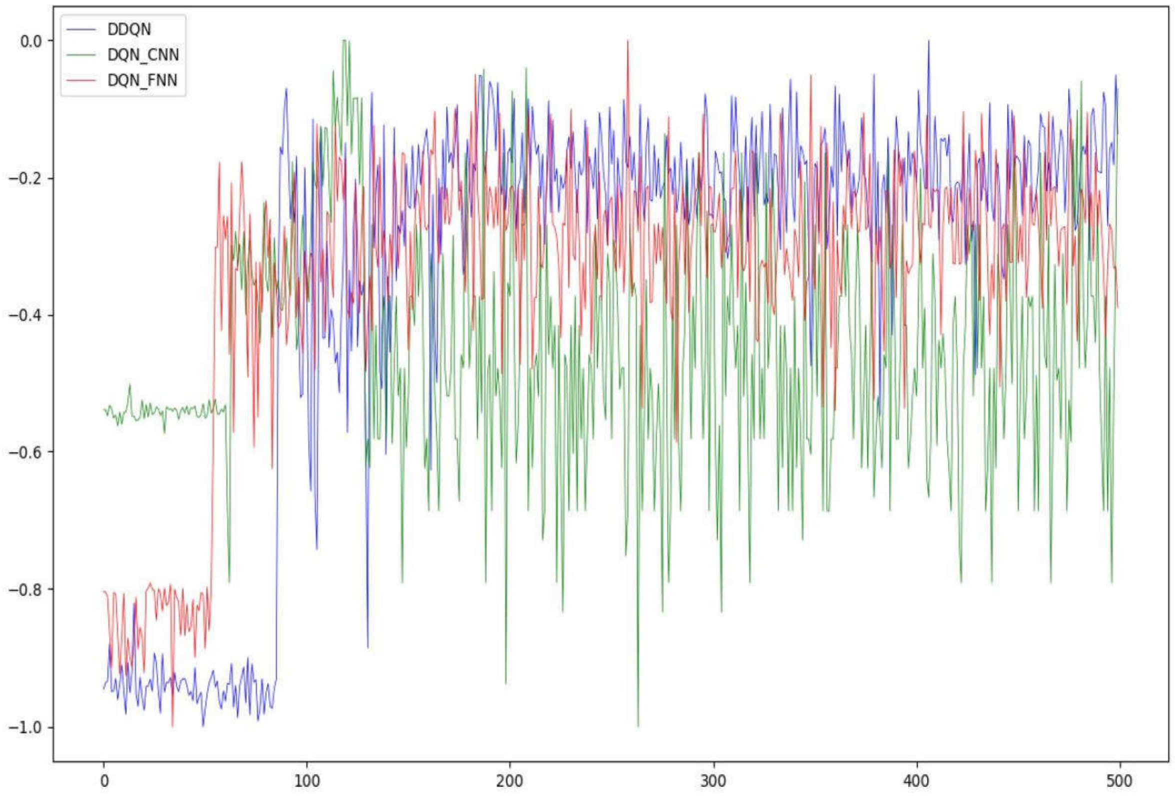

Figure 4.10 Penalty Comparison of Three DRL Based Solutions

Based on the previous experiments we notice for some criteria like the number of accidents or number of times that ego travel out of the road, the rule-based solution with a combination of IDM and MOBIL has a better performance. However, the proposed DRL-based solutions after some training steps reach the same level of performance as a rule-based solution with no accident and out of the road.

Another important feature is the number of same lanes with the emergency vehicle, and we found that our proposed DRL-based solution has a better performance compared to the rule-based solution.

The ego needs also to control the speed and has a speed less than the emergency vehicle after observing it. To that end, we compare the performance of the DRL-based solutions with the rule-based solutions on the number of speed violations and found that the proposed DDQN has a better performance on the speed violation compared to the combination of the IDM and MOBIL.

The last feature that we consider is the number of times the emergency vehicle reaches the end of the road before ego. This feature shows us the performance of the ego on both minimize the speed and change the lane.

We designed the simulation on the way that ego starts the travel in front of the emergency vehicle and if the ego can minimize the speed or in some cases open the emergency lane for the emergency vehicle means the emergency vehicle can end before the ego. The performance of the proposed DRL-based solutions is better compared to the rule-based solution on the number of times that the emergency vehicle reaches the end of the route before ego.

By considering all the previous criteria we can conclude the agent trained with the proposed DRL-based algorithm has a better performance compare to the rule-based solution as a baseline.

The proposed DRL-based solutions are capable to solve the generalization problem of the rule-based solutions both on normal highway driving and the emergency vehicle approaching situation. In this chapter, we started with a description of the implementation process. After that, essential parameters and hyperparameters are presented. At the end of this section, we investigated the performance of the proposed solution, and the comprehensive computational results are presented.

Chapter 5: Conclusion and Future Research

Automobiles as the main component of road transportation have been evolved for years. These days we can think about autonomous vehicles which can drive almost like a human driver. Autonomous vehicles which can operate without a human driver can play a major role in improving transportation systems, reducing traffic congestion, reducing accident rates, increasing safety, and generally increasing travelers' satisfaction. However, for having fully autonomous driving, some challenges need to be solved. For example, when the autonomous vehicle needs to make decisions in difficult situations or scenarios which are not happen quite often, but they are essential for reaching fully autonomous driving.

This research aimed to answer one of those complicated incidents as a new edge case in autonomous driving when an autonomous vehicle is approached by an emergency vehicle. Therefore, the autonomous vehicle needs to control both longitudinal and lateral trajectories.

As the result of this research, we want to have an autonomous vehicle that can make the best decisions when approached by the emergency vehicle as the same as the human drive.

The way that all types of vehicles respond to the presence of an emergency vehicle can bring life or death to one person. Therefore, we consider and defined this problem as a new edge case for autonomous driving.

In this research, we proposed two solutions for the decision-making of autonomous vehicles.

The first solution as a rule-based solution is the combination of two models, Intelligent Driver Model (IDM) and Minimize Overall Braking Induced by Lane Changes (MOBIL).

The reason these approaches are called rule-based is that they are created based on the mathematical formulation as an exact algorithm.

The problem with the rule-based solution as a combination of the IDM and MOBIL is that they don't have an accurate solution for all the driving solutions or in the other words they are not generalized for all the driving scenarios.

This lack of generalization brings this idea to use the neural networks as great function approximators. All the power of the neural networks is that they can play as complex approximator functions which has a better performance compared to the engineered functions like a combination of the IDM and MOBIL as a rule-based solution.

We used this power of the neural network with the concept of reinforcement learning to train an agent, the autonomous vehicle, to take the best actions in different states. By a combination of neural networks and reinforcement learning, DRL-based solutions are created. We started with a simple fully connected neural network at first, DQN_{FNN}, and make the network more complicated by adding some convolutional layers to the network and create DQN_{CNN}. To have a better result, we try the dueling version of the DQN as DDQN where we considered a difference between the state-value and advantage of each action.

We compared the result of the proposed solutions based on designed different criteria as we thought they are essential for autonomous driving and answering the presence of the emergency vehicle. We did a comprehensive quantitative analysis and to find that which of the proposed solutions has a better performance. Based on these experiments we found that DRL-based solutions, DDQN and DQN_{CNN} can have a better performance compared to the rule-based solution.

For all these experiments and creating the environment for training the agent, we built a simulation with help of Simulation Urban Mobility (SUMO) as an open-source traffic simulator and Traffic Control Interface (Traci) for making a connection between different components of simulation and the autonomous vehicle.

5.1 Limitation of This Research

We consider some assumptions in this research, these assumptions mostly bring some limitations to the result and the performance of this research especially when we want to transfer the result of this research to the real world.

The data generated on this research in the simulation environment is not from the real-world traffic data. However, we try to consider all the parameters and assumptions like what happened in real-world highways.

The reason that we use the simulation is that collecting a considerable amount of traffic data in a highway environment especially when an emergency vehicle wants to travel and approach the autonomous vehicle is almost impossible.

These limitations of the simulation environment need to be considered if we want to implement the result of this research in the real world. For example, we consider all vehicles always are travel in the middle of the lanes, which is not always true in the real world. We consider 3.2 meters as the width of lanes, and vehicles are always located at the center of one lane. Vehicles change the lane from the center of one lane to the center of the adjacent lane.

Another limitation of this research that can address in future research and make this research more and more compatible with real life is the number of actions that we considered.

We considered 5 actions as presented in Table 3.3, these are the actions that the agent can take in each step of one episode. In the real world, a human driver most of the time has a combination of two actions presented in Table 3.3 in one step. For example, we consider action 2, ego change to the right lane, and action 3, ego increase the speed of the vehicle. In the real life at one step a human driver sometimes changes both the lane and speed at the same time.

By considering the combination of the actions we will have 9 actions instead of 5 and that can be more realistic. However, considering 9 actions instead of 5 means calculating 9 Q-values and 9 output neurons and it will bring complexity which needs more computational resources.

All the actions in this research have discrete values. For example, the agent with action 3 increases the speed of the ego by 3 m/s for each step. This value can consider as a continuous value and be more compatible with the real world. By considering the continuous actions policy-based solutions might be more helpful compared to the solutions based on Q-values.

In the next section, we will present some future research that can improve the quality of decision-making on autonomous driving and potentially consider it as potential future research.

5.2 Future Research

There are several open challenges and extensions that can be considered as future studies of this research. The first potential work that we can consider as future research from this thesis is that how we can transfer the result and the agent trained in the simulated environment like SUMO or CARLA to the real world. Generally, integrity is the problem with autonomous driving when the knowledge from simulation needs to transfer to the real world.

The main goal of this research is that the ego learns how to make decisions when approached by an emergency vehicle. In this research, we use the numeric data extracted from the simulation. In the real world the shape, siren, and special color of emergency vehicles make them distinctive from the other vehicles. If there is accessibility to the various type of data for example rearview cameras of the autonomous vehicle, it can be useful to solve longitudinal and lateral decisions with help of imagery data for future research.

The other potential future research which can be considered is that the ability of autonomous vehicles shares one lane with other vehicles to open the way for an emergency vehicle in the case that there is traffic congestion.

In this chapter, we started with the summary of this research and the goal of this research elaborated. We also determine the limits of this thesis and potential future research for further studies.

References

[1] National Highway Traffic Safety Administration. "Preliminary statement of policy concerning automated vehicles." *Washington, DC* 1 (2013): 14.

[2] Daily, Mike, et al. "Self-driving cars." *Computer* 50.12 (2017): 18-23.

[3] Rao, Qing, and Jelena Frtunikj. "Deep learning for self-driving cars: Chances and challenges." *Proceedings of the 1st International Workshop on Software Engineering for AI in Autonomous Systems*. (2018).

[4] Zakharenko, Roman. "Self-driving cars will change cities." *Regional science and urban economics* 61 (2016): 26-37.

[5] Minsky, Marvin. "Steps toward artificial intelligence." Proceedings of the IRE 49.1 (1961).

[6] Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, (2016).

[7] Géron, Aurélien. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media, (2019).

[8] LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." nature 521.7553 (2015): 436-444.

[9] Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." nature 518.7540 (2015): 529-533.

[10] Silver, David, et al. "Mastering the game of go without human knowledge." nature 550.7676 (2017): 354-359.

[11] Vinyals, Oriol, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." Nature 575.7782 (2019): 350-354.

[12] Senior, Andrew W., et al. "Improved protein structure prediction using potentials from deep learning." Nature 577.7792 (2020): 706-710.

[13] Chowdhuri, Sauhaarda, Tushar Pankaj, and Karl Zipser. "Multinet: Multi-modal multi-task learning for autonomous driving." 2019 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, (2019).

[14] Mehta, Ashish, Adithya Subramanian, and Anbumani Subramanian. "Learning end-to-end autonomous driving using guided auxiliary supervision." Proceedings of the 11th Indian Conference on Computer Vision, Graphics and Image Processing. (2018).

[15] Kendall, Alex, et al. "Learning to drive in a day." 2019 International Conference on Robotics and Automation (ICRA). IEEE, (2019).

[16] Li, You, and Javier Ibanez-Guzman. "Lidar for autonomous driving: The principles, challenges, and trends for automotive lidar and perception systems." IEEE Signal Processing Magazine 37.4 (2020): 50-61.

[17] Tampuu, Ardi, et al. "A survey of end-to-end driving: Architectures and training methods." IEEE Transactions on Neural Networks and Learning Systems (2020).

[18] Sutton, Richard S., and Andrew G. Barto. Reinforcement learning: An introduction. MIT press, (2018).

[19] Even-Dar, Eyal, Sham M. Kakade, and Yishay Mansour. "Experts in a Markov decision process." Advances in neural information processing systems 17 (2005): 401-408.

[20] Watkins, Christopher JCH, and Peter Dayan. "Q-learning." Machine learning 8.3-4 (1992): 279-292.

[21] Sutton, Richard S., et al. "Policy gradient methods for reinforcement learning with function approximation." Advances in neural information processing systems. (2000).

[22] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[23] Silver, David, et al. "Deterministic policy gradient algorithms." International conference on machine learning. PMLR, (2014).

[24] Haarnoja, Tuomas, et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." International conference on machine learning. PMLR, (2018).

[25] McCulloch, Warren S., and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics 5.4 (1943): 115-133.

[26] Gardner, Matt W., and S. R. Dorling. "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences." Atmospheric environment 32.14-15 (1998): 2627-2636.

[27] Khodayari, Alireza, et al. "A historical review on lateral and longitudinal control of autonomous vehicle motions." 2010 International Conference on Mechanical and Electrical Technology. IEEE, (2010).

[28] Dixit, Shilp, et al. "Trajectory planning and tracking for autonomous overtaking: State-of-the-art and future prospects." Annual Reviews in Control 45 (2018): 76-86.

[29] Treiber, Martin, Ansgar Hennecke, and Dirk Helbing. "Congested traffic states in empirical observations and microscopic simulations." Physical review E 62.2 (2000): 1805.

[30] Kesting, Arne, Martin Treiber, and Dirk Helbing. "General lane-changing model MOBIL for car-following models." Transportation Research Record 1999.1 (2007): 86-94.

[31] Hoel, Carl-Johan, Mattias Wahde, and Krister Wolff. "An evolutionary approach to general-purpose automated speed and lane change behavior." 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA). IEEE, (2017).

[32] Kim, K., J. V. Medanić, and D-I. Cho. "Lane assignment problem using a genetic algorithm in the Automated Highway Systems." International Journal of Automotive Technology 9.3, (2008).

[33] Jaafra, Yesmina, et al. "Robust reinforcement learning for autonomous driving." (2019).

[34] Dosovitskiy, Alexey, et al. "CARLA: An open urban driving simulator." Conference on robot learning. PMLR, (2017).

[35] Wei, Haoran, et al. "Mixed-Autonomy Traffic Control with Proximal Policy Optimization." 2019 IEEE Vehicular Networking Conference (VNC). IEEE, (2019).

[36] Wang, Pin, and Ching-Yao Chan. "Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge." 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC). IEEE, (2017).

[37] Wang, Pin, and Ching-Yao Chan. "Autonomous ramp merge maneuver based on reinforcement learning with continuous action space." arXiv preprint arXiv:1803.09203 (2018).

[38] Nishitani, Ippei, et al. "Deep merging: Vehicle merging controller based on deep reinforcement learning with embedding network." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, (2020).

[39] Wu, Yuankai, et al. "ES-CTC: A Deep Neuroevolution Model for Cooperative Intelligent Freeway Traffic Control." arXiv preprint arXiv:1905.04083 (2019).

[40] Hoel, Carl-Johan, Krister Wolff, and Leo Laine. "Automated speed and lane change decision making using deep reinforcement learning." 2018 21st International Conference on Intelligent Transportation Systems (ITSC). IEEE, (2018).

[41] Liao, Jiangdong, et al. "Decision-making Strategy on Highway for Autonomous Vehicles using Deep Reinforcement Learning." IEEE Access 8, (2020).

[42] Ronecker, Max Peter, and Yuan Zhu. "Deep Q-network based decision making for autonomous driving." 2019 3rd International Conference on Robotics and Automation Sciences (ICRAS). IEEE, (2019).

[43] Vinitsky, Eugene, et al. "Benchmarks for reinforcement learning in mixed-autonomy traffic." Conference on robot learning. PMLR, (2018).

[44] Kheterpal, Nishant, et al. "Flow: Deep reinforcement learning for control in sumo." EPiC Series in Engineering 2 (2018): 134-151.

[45] Liang, Eric, et al. "Ray rllib: A composable and scalable reinforcement learning library." arXiv preprint arXiv:1712.09381 (2017): 85.

[46] Duan, Yan, et al. "Benchmarking deep reinforcement learning for continuous control." International conference on machine learning. PMLR, (2016).

[47] Behrisch, Michael, et al. "SUMO–simulation of urban mobility: an overview." Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation. ThinkMind, (2011).

[48] Hoel, Carl-Johan, et al. "Combining planning and deep reinforcement learning in tactical decision making for autonomous driving." IEEE transactions on intelligent vehicles 5.2, (2019).

[49] Garzón, Mario, and Anne Spalanzani. "Game theoretic decision making for autonomous vehicles' merge manoeuvre in high traffic scenarios." 2019 IEEE Intelligent Transportation Systems Conference (ITSC). IEEE, (2019).

[50] Tampuu, Ardi, et al. "A survey of end-to-end driving: Architectures and training methods." IEEE Transactions on Neural Networks and Learning Systems, (2020).

[51] Kiran, B. Ravi, et al. "Deep reinforcement learning for autonomous driving: A survey." IEEE Transactions on Intelligent Transportation Systems, (2021).

[52] Amini, Alexander, et al. "Learning robust control policies for end-to-end autonomous driving from data-driven simulation." IEEE Robotics and Automation Letters 5.2 (2020): 1143-1150.

[53] Wegener, Axel, et al. "TraCI: an interface for coupling road traffic and network simulators." Proceedings of the 11th communications and networking simulation symposium. (2008).

[54] Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." Icml. (2010).

[55] Wang, Ziyu, et al. "Dueling network architectures for deep reinforcement learning." International conference on machine learning. PMLR, (2016).