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

School of Computer Science and Statistics

Using Deep Reinforcement Learning to increase Traffic Safety in Urban areas whilst maintaining Traffic Flow and Efficiency

David John Neill

April 28, 2021

A Dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Electronic and Computer Engineering)

Declaration

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

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

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

Signed: David John Neill Date:28/04/2021

Abstract

In the western world, health and safety is always a primary concern for car companies as they ensure that they meet all the standards and requirements to keep their passengers safe both while driving and in the event of a collision. A study in 2012 concluded that fatal road traffic incidents occur on average every 50 seconds, and road traffic injuries are sustained every 2 seconds (1). According to WHO, approximately 1.35 million people die each year as a result of road traffic collisions. As a result, this study aimed to find a way to increase traffic safety whilst maintaining or improving traffic flow and efficiency. To complete this task, numerous different techniques in the areas of Machine Learning and Artificial intelligence were considered. The most suitable option involved using Deep Reinforcement Learning to solve the given issue. Existing work showed promising results for using Deep Reinforcement Learning in Traffic control and congestion problems. Initially, this only applied to single-agent environments; however, with the introduction of Independent Deep Q Learning and more powerful computing, this allowed for multi-agent algorithms.

Therefore this thesis focused on using Deep Q Learning for the single-agent scenarios and Independent Deep Q learning for multi-agent scenarios. A single agent consisted of 1 intersection with four roads meeting at the intersection. While the multi-agent scenario consisted of 4 intersections, each with four roads meeting at the intersection. The traffic simulator used for these simulations was called SUMO. During the simulation, key metrics such as vehicle average speed, number of collisions/number of vehicles performing emergency braking and the cumulative waiting time for each episode. Deep Q Learning and Independent Deep Q learning ran through numerous different experiments, each with different parameters such as observation space and memory size. Throughout the simulations, two different actions sets were tested and analysed.

In the single-agent low traffic scenario, we saw that the more extensive action set, large observation scope and smaller memory produced the best results by decreasing collisions by 72% and increasing speed by 150%. The same trend was produced when the single-agent high traffic scenario was run.

The observation scope size and memory size were then taken and used in the multi-agent scenarios. In the multi-agent low traffic scenario with the more extensive action set out preformed both the baseline and the smaller action set by reducing collisions by 48% and increasing the average speed by 127%. However, both action sets reduced the average waiting time to 0. In high traffic demand, the same trend is repeated where the more extensive action set reduces collisions by 60% and increases average speed by 158%. From these results, it concluded that the thesis completes its objective of increasing traffic safety whilst either maintaining or improving traffic flow and efficiency.

Acknowledgements

First and foremost, I would like to thank my project supervisor Prof Ivana Dusparic for all her time, expertise and willingness to answer all of my questions. Thank you.

Secondly, I would also like to thank my family, my mum, dad, and uncle, for their continued support. Lastly, to my lovely fiancée, Alex, for supporting me unconditionally throughout the past year. Thank you.

Contents

List of Figures

List of Tables

1 Introduction

This thesis addresses the issue of traffic safety and is one of the first studies to use Deep Reinforcement Learning to help improve traffic safety whilst maintaining traffic flow and efficiency. Standard Reinforcement Learning approaches suffer from an issue called "the curse of dimensionality". Therefore neural networks are introduced to work around this problem and to improve performance for large complex problems. Therefore, this thesis will use Independent Deep Q learning (IDQL); this allows each agent to learn independently and treats other agents as part of the environment. When using IDQL, the issue of the environment becoming non-stationary can become a problem. This thesis ensures this does not occur by ensuring that the observation space of agents does not overlap within the simulation. We firstly evaluate Deep Q Learning (DQL) in a single agent scenario and analyse how traffic safety has been affected. We then evaluate IDQL in a multi-agent scenario and analyse how traffic safety has been affected. This chapter highlights the problem we are trying to solve, the thesis's aims, objectives, assumptions, main contributions and finally presents a structure to the remainder of the thesis.

1.1 Problem Statement

In the western world, health and safety is always a primary concern for car companies as they ensure that they meet all the standards and requirements to keep their passengers safe both while driving and in the event of a collision. The safety of passengers in a moving vehicle has come under the spotlight due to the increased number of cars on the road and capabilities of cars reaching greater speeds than ever before. A study in 2012 concluded that fatal road traffic incidents occur on average every 50 seconds, and road traffic injuries are sustained every 2 seconds (1). According to WHO, approximately 1.35 million people die each year as a result of road traffic crashes, and 20-50 million more people suffer non-fatal injuries, with many incurring a disability as a result of their injury. Not only does road traffic injuries affect the injured and their family but also the economy. WHO proposes that countries lose 3% of their gross domestic product due to road traffic collisions (10). The majority of road traffic collisions are caused by driver error; whether this be driving too fast for the road conditions, unfamiliar with the road they are on, lack of driving skill, cognitive and mobility impairment, the list goes on. In normal conditions, the most efficient way to reduce road traffic collisions is to reduce

one's speed. However, this has a detrimental effect to the driver, traffic flow/efficiency and the economy at large. Therefore numerous studies have been conducted on how integrating hardware and software systems such as adaptive cruise control, cooperative adaptive cruise control, and autonomous behaviour affect traffic safety and flow. Using these technologies can certainly improve traffic safety and flow. So why stop there? With the introduction of faster and more powerful computers, the areas of machine learning and artificial intelligence can be used to help improve traffic safety. So the aim of this thesis is to try and teach semi-autonomous vehicles to navigate past through challenging sections of the road network with no collisions occurring whilst ensuring that traffic flow and efficiency are either maintained or improved.

1.2 Thesis Aims and Objectives

The general objective of this thesis to address the gap in using Deep Reinforcement Learning to help improve traffic safety. Existing studies tend to focus on optimising traffic control and preventing traffic congestion. There has been much success in implementing Deep Reinforcement learning for optimising traffic control and preventing traffic congestion. Therefore, we believe that if a DRL based techniques are used for traffic safety in a multi-agent scenario, it would have a positive impact on safety. Such an approach can be applied to realistic intersection scenarios of any city worldwide. This thesis presents DRL as a suitable option due to its ability to learn the agent's behaviours without a model of the environment and its ability to perform well in large complex problems. Therefore this thesis analyses the requirements needed for a multi-agent DRL based method to improve traffic safety. It presents the design and implementation of the given technique followed by an evaluation of how the techniques preforms.

1.3 Thesis Assumptions

In designing, implementing and evaluating this multi-agent Deep Reinforcement Learning problem, there were several different assumptions that had to be made about the environment in which it was deployed in. Making these design assumptions allowed the scope of the thesis to be limited and therefore limits the number of issues it addresses. The evaluation environment assumptions are imposed by the capabilities and limitations of the traffic simulator used.
The first assumptions in this thesis assumes that agents are stationary, i.e. their given location is fixed, and they do not move during the simulation. In the design section, it can be seen that the agents are associated with a junction, which are stationary as they do not move. Furthermore, Agents are assumed to be failure-free, and issues arising from these agents are not addressed in this work. In the design of this project, the state space is assumed to be a post-processing representation of images of the given junction. Furthermore, it is assumed that each agent will have the same observation capabilities, i.e. same use of imaging technology,

same image type/size/quality and same viewpoint.
The timing of all agents within the system are synchronised, i.e. all agents make decisions simultaneously at each time step. This thesis does not investigate the effect an asynchronous design would have on the performance of the proposed multi-agent DRL approach. In the simulations performed in this thesis, the traffic lights are permanently set to green. This is done to force the simulation to produce collisions as, by default, the traffic simulator tries to prevent collisions and emergency braking from occurring within the simulation. Furthermore, the starting position, destination and route of vehicles are predefined. If a vehicle is outside the given observation scope of the agent, then the control of the vehicle is handed back to the traffic simulator. As a consequence of this, if there is no available road space for vehicles to join the simulation, then they are not inserted into the simulation.

1.4 Thesis Contribution

This thesis identifies and motivates the need for a Deep Reinforcement Learning based multi-agent technique for traffic safety problems which has not been considered before. This thesis aims to answer the question of, Is Deep Reinforcement Learning a suitable technique to use to help improve traffic safety? It presents the different difficulties and challenges faced to implement a DRL technique into a traffic safety problem. The thesis outlines the different requirements needed to propose such a technique. The main contribution of the thesis is the design, implementation and evaluation of a DRL method for multi-agent traffic safety. Pre-existing DRL techniques only allow for one policy within a given environment. The introduction of Independent Deep Q learning allows for each agent to learn its own policy independent of other agents and assumes these agents are part of the environment. The unique nature of IDQN allows it to learn and consider other agents' policies to improve both local and global performance. The use of IDQL can cause the environment to become non-stationary; to fix this issue, a fingerprint can be used, or the observation scopes of the agents must not overlap. Finally, this thesis evaluates both DQL and IDQL for single and multi-agent scenarios to see how it affects traffic safety. From these evaluations, it can be clearly seen that both DQL for single-agent and IDQL for multi-agent have a positive impact on traffic safety and outperforms the road network controlled by the traffic simulator.

1.5 Thesis Structure

The structure of this thesis is as follows. Chapter 2 discusses the background in related work to this project. It begins by looking at key concepts in the areas of Deep Learning and Reinforcement Learning. It then focuses on how these areas have been combined to form a new area of artificial intelligence and machine learning called Deep Reinforcement Learning. It discusses the basic concepts followed by techniques that have been used to help improve its performance. Presented in this chapter is a section on how different cooperative

Driving Strategies affect Traffic Conditions. Furthermore, a small section is included that discusses the potential when combining RL and personalised driving. Lastly, this chapter presents the state of the art studies for single and multi-agent environments using both Reinforcement Learning and Deep Reinforcement Learning to help solve traffic control and congestion problems. Chapter 3 describes the given design of the approach and defines key terms such as state space, reward function and action space. Chapter 4 describes the implementation of the design stated in chapter 3. Chapter 5 presents the evaluation of the suggested study in both a single and multi-agent environment using the DQL and IDQL methodology respectfully. Chapter 6 concludes the thesis by briefly summarising the contents of the work and finishes by outlining the issues that remain open for future work.

2 Background and Related Work

2.1 Deep Learning

2.1.1 Definition

Deep learning is a form of machine learning that uses algorithms based on the structure and function of the human brain. More specifically, it utilises concepts from the brain's biological neural networks and concepts from the areas of Mathematics and Statistics. How did Deep Learning get its name? Originally neural networks only had 2-3 layers, while now these types of networks are a lot deeper as then contain 10+ layers, hence why the word "deep" is now used. "Learning" can be split up into three different areas unsupervised, semi-supervised and supervised, which all are utilised in Deep Learning.

Deep learning has become increasingly popular and more widely used in today's world. This has mainly been caused by developing more efficient algorithms, more powerful computers and the increased size and availability of datasets. In deep learning, a computer is trained to perform specific tasks using a neural network. Neural networks can be used for tasks such as object detection, prediction and speech recognition. In a neural network, there are three main layers input, hidden and output layers. The input layer is where data is passed into the neural network. It will then pass through the hidden layers. The hidden layer tends to have to be 2+ layers inside it. These layers are mathematical functions that are designed in a given way to get the desired output. This information is then passed to the output which computes a final answer.

With numerous layers, Deep Learning can identify complex trends or structures within data. Backpropagation is an algorithm for supervised learning that is used in neural networks to help train the network. It utilises a 1st order iterative optimisation algorithm called gradient descent, and the calculation of the gradient propagates backwards through the network. There are numerous different types of neural networks in deep learning, each with its advantages and disadvantages.

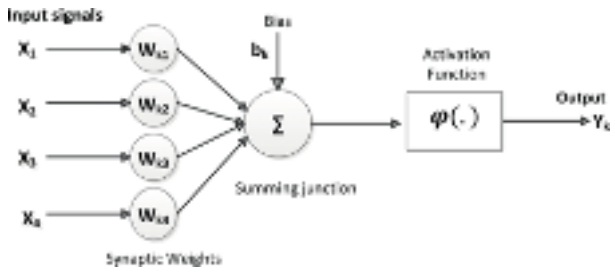


Figure 2.1: Model of an artificial neuron according to McCulloch and Pitts

2.1.2 Neural Nets

A neural network or artificial neural network is several neurons connected together. These networks take in large amounts of data and uses an algorithm to identify and classify unknown patterns in the data. With time the network will train itself further to fully optimise its performance (11). The first idea of an artificial neuron was put forward by Warren McCulloch and Walter Pitts in 1943. Figure 2.1 depicts an visual representation of this.

The first artificial Neural network was not created until 15 years later by Rosenblatt, and he named it the perceptron. It consisted of 2 input cells and one output cell. Another mathematical model that helped the development of this area was Backpropagation created by Werbos in 1975 (12). Backpropagation is a widely used algorithm in machine learning to help train artificial networks. This algorithm will fine-tune the network's weights, based on the error rate obtained in the previous step. Backpropagation was first introduced into neural networks in 1985 by Rumelhart, Hinton and Williams (1986)(13). However, the theory Backpropagation dates back to the 1960s, where Henry Kelly derived the basics of continuous Backpropagation in the context of control theory.(14)
As mentioned in section 2.1, there are three general layers in artificial neural nets. These are called input, hidden and output layers. The input layer has typically one layer, and its function is to take the input data and pass it on to the hidden layers. The number of nodes in an input layer is also the number of components being passed to the network. Therefore, hidden nodes have no direct access to any data outside this network. Each neuron will apply a given weight to the input data and then directs the information through an activation function at the output. There are numerous different activation functions such as Linear, Sigmoid and Tanh (15). These functions help develop improved and more efficient deep learning models. A neural network is considered deep if it contains more than one hidden layer. The hidden layers are then connected to the output layer, which communicates the results. Some of the more widely used neural networks include Convolutional Neural Networks, Recurrent Neural Network, LSTM, Feed-forward Neural Network, etc. Convolutional Neural Networks are easier to train compared to other neural nets, and they generalise much better compared to networks with full connectivity (16).

2.1.3 Convolutional Neural Nets

Convolutional neural nets (CNN) are mainly used to analyse and extract information/features from images, video or even speech recognition (17). CNN's have also been used as a recommendation system (18). In recent years CNNs have become increasingly popular due to the creation of networks such as GoogleLeNet (19), and ResNet (20). A CNN's role is to reduce the input into a smaller form, so that is easier to process, without losing any features from the input data. The input data is usually in the form of a 2d or 3d array. In a standard CNN, there are two distinct parts. The 1st part contains convolution and pooling while the 2nd layer is a fully connected network.

The convolution operation uses a given kernel to extract high-level features. This kernel can either cause the input dimensions to decrease, increase or stay the same depending on the type of padding applied. Pooling is then used to reduce the spatial size of the convolved feature whilst extracting dominant features. The two most common methods of pooling are max pooling (21) and average pooling. (22). Pooling will reduce the amount of noise and distortion in the input. The sequence of convoluting and pooling can keep repeating until a number of features have been extracted.

The input is now in a suitable format to be flattened into a column vector. This new column vector is then fed into a feed-forward neural network which acts as a classifier. Backpropagation can also be utilised throughout the entire network to allow all the weights to be trained.

2.1.4 Optimization Algorithms

Optimisation algorithms are used in neural networks to change attributes such as weights and learning rates. The overall aim is to reduce the cost function J(θ). The most widely used algorithm is Gradient descent. Gradient descent computes the gradient of the cost function with respect to the parameter(θ) over the entire training set. However, as the gradient is being calculated over the entire dataset to preform one update, it can make this algorithm very slow and can get trapped at local minima (23). For this reason, new algorithms have been created and implemented. Some of these include AdDelta, AdaGrad, Momentum, Adam and RMS prop.

RMSProp

RMSProp is similar to Gradient Descent algorithm with momentum. RMSProp prevents the oscillations in the vertical direction. This means that our learning rate can be increased and hence converging faster. RMSProp's equation for finding the gradient can be shown in equation 1. Where γ is recommended to have the value of 0.9

$$G_i^t = \gamma G_i^{t-1} + (1-\gamma)(\frac{\delta J(\theta)}{\delta \sigma j^t})^2 \tag{1}$$

AdaGrad

AdaGrad is a gradient-based optimisation algorithm that adapts the learning rate depending on how frequent the feature is. In other words, it adaptively updates parameters based on a sum of squared gradients per parameter (24). This means it has a different learning rate for every parameter at every time step. To perform this update, we use the below equation in 2.

$$\theta_{i,t+1} = \theta_{i,t} - \frac{\eta}{\sqrt{G_t, ii + \epsilon}} g_t, i \tag{2}$$

AdaGrad will modify the η learning rate after each time step and for every parameter, and it does this based on past gradients that have been computed for theta. Dividing the learning rate by the sum of past square gradients will eliminate the need to tune the learning rate manually. However, the accumulation of the square past gradients can produce a sizeable real number and cause the learning rate to become infinitely small, rendering not usable after this point.

ADAM

ADAM or Adaptive moment estimation algorithm builds on two previous well-known optimisers. The method combines the advantages of AdaGrad and RMSProp. AdaGrad has the ability to deal with sparse gradients while RMSProp can handle non-stationary objectives. ADAM uses the estimation of the first and second-order moments of the gradient to alter the neural network's learning rate.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t \tag{4}$$

In equation 3 shows the first moment which is the mean and is donated as m_t. While equation 4 shows the second moment which is the uncentered variance and is noted as v_t. g_t is the gradient of the current mini batch. Both βs are new hyper parameters of the algorithm and they have default vales of 0.9 and 0.999 respectively. At the beginning of the experiment both m_t and v_t are initialised as zero vectors, this will cause a bias towards 0. In order to prevent this from happening m_t and v_t are bias corrected with:

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{5}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{6}$$

And the final procedure is to the update the weights using the following:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{7}$$

2.2 Reinforcement Learning

2.2.1 Definition

Reinforcement Learning algorithms will try and learn to perform a specific task in a given environment so that it will return the highest reward and perform the task optimally. An agent is placed in an environment where it will undertake certain random actions. These actions will then result in either a positive or negative reward. Depending on this reward, it will either keep doing or stop doing the specific action in the future. The algorithms aim is to return the maximum amount of reward from the system. The Agent will try to find the highest reward by exploring the environment through a trial and error process (25).

Thus far, we have already mentioned two critical terms of Reinforcement Learning, agent and environment. We must give definitions of these two terms and other key terms in Reinforcement Learning. These terms will be used with these definitions unless stated differently.

1. Agent: An agent is an object or component that decides what action to undertake in the environment. In this case of this paper, a junction will be the Agent.
2. Environment: The environment is the "world" in which the Agent will interact with. In reality, the environment is usually a class or a function that takes the Agent's location/current state and action as the input, and it outputs the agents next state and the reward.
3. Policy: A policy is the set of rules that governs the actions that the Agent will take at any given time. Throughout the learning process, the policy can change multiple times. This is because the Agent seeks to learn the optimal policy to receive the highest reward from every action and state. In mathematically terms, we define the policy π in terms of the Markov Decision Process. The state is defined as S, and the action is defined as A, i.e. π*:S $\implies$ A
4. Reward: This is rewarded to the agent when it moves from state s_t to s_{t+1}. A reward may have both positive and negative value depending on the action taken. The agents aim to maximise this value, and in doing so, it can alter the policy.
5. State: A state S is the representation of the environment at given time-step. If the

Agent is active, it represents the Agent and how it interacted with the environment at time step t. At a different angle, an observation is how the current state of the environment looks through the "eyes" of the Agent.

6. Value Function: A Value Function $V_{\pi}(s)$ is an estimation of the total rewards the Agent will receive from following the policy from state S

In Reinforcement Learning, there are two types of algorithms. One of these is called Model-free as these rely on a trial and error approach to solve the problem. The second type is a model-based algorithm. A model-based approach uses a predictive model and therefore requires planning by the system. However, when the system grows in space and size, a model-free approach is then utilised.

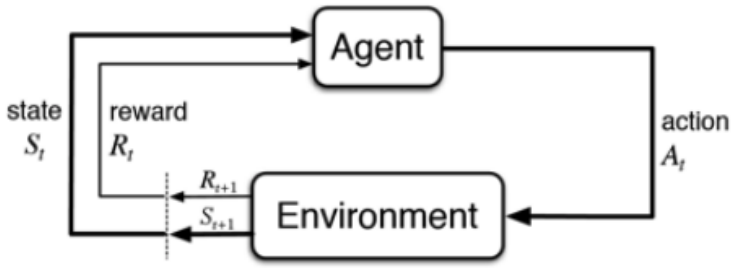


Figure 2.2: A visual representation of Reinforcement Learning

Given this information, we can summarise a given step in the Reinforcement Learning System. In each time step t, the Agent will perceive a given state S_t.The Agent will then select a given action A_t which is based on a policy π. The Agent and the given action they interact with the environment. From this, the Agent receives a reward r_t and transitions to a state of S_{t+1}. A visual representation of this can be seen in figure 2.2. The total reward can be given by equation 8, as shown below.

$$R_t = \sum_{t=0}^{\infty} \gamma^t R(s_t) \tag{8}$$

In equation 8, γ is the discount factor and has a value of between 0 and 1 e.g $0< \gamma \leq 1$. This is used to help ensure that the algorithm can converge as fast as possible.

$V^{\pi}(s)$ is a value function that is used to evaluate a policy given. This can be seen in equation 9 below:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \epsilon S} p(s'|s, \pi(s)) V^{\pi}(s') \tag{9}$$

By rearranging the value function $V^{\pi}(s)$, a policy π can be obtained. As shown in equation

10 below:

$$\pi(s) = argmax[R(s,a) + \gamma \sum_{s' \epsilon S} p(s'|s,a) V^{\pi}(s')] \tag{10}$$

Another key equation that is used throughout the domain of reinforcement learning is the Bellman equation. This equation can be seen in equation 11.

$$V^{*}(s) = max[R(s,a) + \gamma \sum_{s' \epsilon S} p(s'|s,a) V^{*}(s')] \tag{11}$$

Solving the set of equations above allows the computer program to determine a value for the optimal value function and, therefore, compute the optimal policy.

2.2.2 Q-Learning

The Q in Q-learning stands for quality. The quality represents the amount of reward that can be retrieved from a given action. Q-Learning is an off-policy learner and model-free reinforcement learning algorithm. An Off-policy Learner indicates that the learning of the optimal policy is independent of the Agent's actions. In comparison, an on-policy learner algorithm learns the policy that is being carried out by the Agent, including the exploration steps (26). To find the optimal policy for a given agent and environment equations 12 and 13 can be used.

$$\pi(s) = \arg\max_{a \epsilon A} Q(s,a) \tag{12}$$

where the Q value, Q(s,a) can be given as:

$$Q(s,a) = (1-\alpha) Q(s,a) + \alpha [R(s,a) + maxQ(s',a')] \tag{13}$$

where α is the learning rate and has a value between 0 and 1, s and a are the given state and action respectively.

To begin this algorithm, a Q table is initialised with the size [state, action] and all values are set to zero. After this, the Agent then begins to interact with its environment and make updates to the state action pair in Q[state, action]. For the Agent to interact with its environment, two main concepts are used. These two concepts are exploration and exploitation. Exploration is when the agents next step are completely random. Acting randomly is important as it allows the Agent to discover new states that otherwise would not be selected during exploitation. A common technique used to ensure that Agent acts randomly is the ϵ-greedy exploration algorithm. Exploitation is when we use values that we have already recorded in our Q table

to derive our next step. Inevitably there is a trade-off between exploration and exploitation. The Agent then chooses an action either acting randomly or by using the Q table. After the Agent completes the action, a reward is then calculated, and the Q table is then updated with the new Q value. This process will repeat a number of times until the value function Q(s, a) is optimised. Figure 2.3 shows a visual representation of Q Learning algorithm (27).

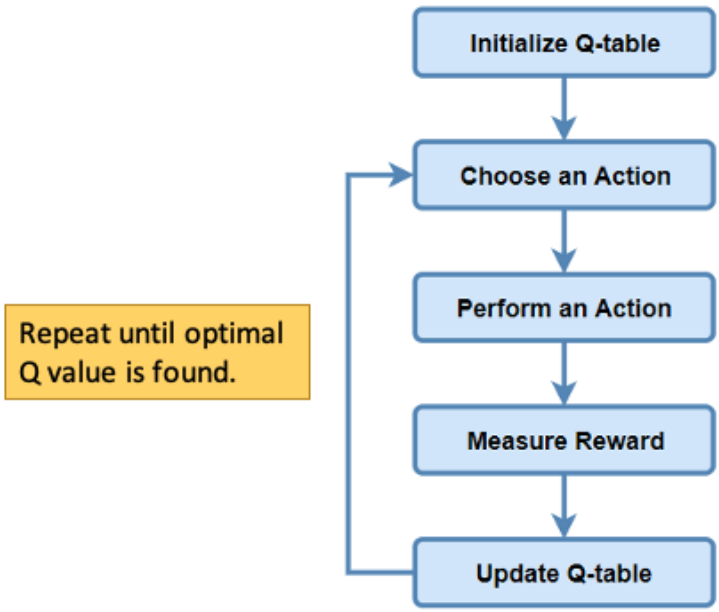


Figure 2.3: A visual representation of Q Learning

2.3 Deep Reinforcement Learning

2.3.1 Definition

Deep Reinforcement Learning is an area of research in machine learning and artificial intelligence that utilises concepts from the specialised areas of deep learning and reinforcement learning. From the area of reinforcement learning, it uses the fundamental concepts of exploration, exploitation, agents being rewarded or penalised. While from the area of deep learning, it utilises the concepts associated with deep neural networks. However, when a large state and multiple action spaces are available, it can make it unfeasible to learn Q values for a given state. This is known as the curse of dimensionality. For this reason, deep learning and reinforcement learning are implemented together.

2.3.2 Deep Q networks

To help solve the issue of dimensionality, as mentioned in section 2.3.1. Deep Q Networks combines concepts from deep neural nets from deep learning and Q learning from reinforcement learning. Deep Q-Networks was a technique proposed by the DeepMind team at Google in 2015 (28). The team successfully implemented the technique and tested it on a wide range of Atari games. In some cases, it achieved super-human performance. Using this technique has become the benchmark for Deep Reinforcement Learning.

Deep Q-Networks also use two new concepts that are not used in either DL or RL. These techniques are experience replay and target network. These are used because RL is known to be unstable or to diverge when a certain approximator is used in a neural net to represent the Q value. Hence these concepts will eradicate these issues.

Experience replay is a technique that stores the given agent's experience at a given timestamp in replay memory. During training, it will then randomly access these experiences in replay memory, and it is then used to help train the network and prevent correlation occurring between consecutive samples. Experience replay has three main advantages over standard online Q-learning. Firstly, data efficiency increases as each step of experience is potentially used in many weight updates. Secondly, if samples are taken directly from the simulation, this will cause inefficient learning due to the strong correlation between consecutive samples. Randomising these samples with experience replay helps to break these correlations and reduce the variance of the updates. Thirdly, when learning on-policy, the current parameters determine the next data sample that the parameters are trained on. By using this method, the behaviour distribution is averaged over the preceding states and hence causing the learning to stabilise and prevent variation or divergence in the parameters (29).

The second technique uses a second neural network called a target network(28). This second network improves the network's stability, and it is used to generate the "target value" during

Q-Learning. The target network will clone the original network every C iterations/updates. A Q-value is then taken from both the target network and original network and the two values are then compared through the use of a loss function. For this process to occur, two networks need to be used, as it allows for a delay in the value of Q for both networks. This delay makes the probability of divergence or oscillations occurring, very low.

A Deep Q Network parameterises an approximate value function Q(s,a;θ_i) using a neural network, where θ_i are the weights of the network at iteration i. The experience replay then stores the agent experience $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ at every timestep into a given dataset D_t. This dataset is of finite size, and when full, the oldest experiences can be replaced by newer ones. Mini-batches are then drawn uniformly at random from the given dataset $U(D)$ and are applied as Q updates to the network during training. The loss function of the Q learning update at iteration i can be given by the following loss equation (equation 14). It can be seen that the loss equation calculates the squared difference between target Q and predicted Q.

$$L_i(\theta_i) = E_{U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)]^2 \tag{14}$$

where θ_i and θ_i^- are the weights of the network at iteration i and weights of the target network at iteration i, respectively. As mentioned above, the target network weights are only updated every C updates and remain the same until the next update. Algorithm 1 below shows the pseudo-code for Deep Q Learning(29).

2.3.3 Double Deep Q Networks

Double Q learning was first proposed by Hasselt(30) in 2010, and it was a solution to the problem of large overestimation's of the Q value in Q learning. In normal Q-Learning, the agent will always choose a given action that results in a maximum Q value. However, at the beginning of the simulation, the agent knows nothing about the environment, so therefore it needs to estimate the first value of Q(s,a) and then update after each iteration. However, these Q values can contain a large amount of noise, and hence we are not certain that the agent chooses the best action. The noise from estimating the Q values can cause large positive biases in the updating process.

To prevent this overestimation, Double Q-learning decouples the selection and the evaluation. To do this, two value functions are learned by assigning experiences randomly to update one of the two value functions, resulting in two sets of weights, θ_i and θ_i'. For each update, one set of weights is used to determine the greedy policy and the other to determine its value(31). After decoupling the selection and evaluation, we can rewrite the target as:

Algorithm 1: Deep Q-learning with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
Initialize target action-value function $\hat{\mathrm{Q}}$ with weights $\theta^{-} = \theta$
for *episode = 1, M* **do**
  Initialize sequence $\mathrm{s}_1 = \{\mathrm{x1}\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  for t = *1, T* **do**
    With probability ϵ select a random action a_t
    otherwise select $\mathrm{a}_t = \arg\max_a \mathrm{Q}(\phi(s_t),\mathrm{a};\theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $st+1 = \mathrm{s}_t,\mathrm{a}_t,\mathrm{x}_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t,\mathrm{a}_t,\mathrm{r}_t,\phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j,\mathrm{a}_j,\mathrm{r}_j,\phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j, & \text{if episode terminates at step } j+1 \\ r + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}), & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameters θ
    Every C steps reset $\hat{\mathrm{Q}} = \mathrm{Q}$

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t) \tag{15}$$

The new Double Q-Learning error can be written as:

$$Y_t^Q = R_{t+1} + \gamma Q(S_{t+1}, \arg\max_a Q(S_{t+1}, a; \theta_t); \theta_t) \tag{16}$$

However, as mentioned in Hasselt's paper, it is not optimal for Deep Q-Network architecture to fully decouple the target network as it already provides a suitable second value function without the need to introduce more networks. To evaluate the greedy policy, we use the online network but use the target network to estimate its value. Therefore, the Double Deep Q Network can be written as:

$$L_i(\theta_i) = E_{U(D)}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^{-}) - Q(s, a; \theta_i)]^2 \tag{17}$$

The main difference between Double Q-Learning and Double Deep Q Network is that the weights of the second network θ_i^{-} are replaced with the weight from the target network θ_t^{-}. These weights are replaced for the evaluation of the current greedy policy. However, the target network is still updating using the same method outlined in section 2.3.2

2.3.4 Improvements to Deep Reinforcement Learning

In the area of Deep Reinforcement Learning, continuous improvements are being published. The research team at the leading edge is the DeepMind team at Google. Recently they proposed a paper that contained many improvements to the pre-existing techniques. In this section, two of these techniques will be discussed. Firstly, Prioritised Experience replay and secondly a dueling network (32).

Prioritized Experience replay

In 2016 a new type of experience replay was proposed, and it was named Prioritised Experience replay (33). This new methods aim was to make experience replay more efficient. In normal experience replay, the Deep Q network will sample uniformly from the replay buffer in the order they were experienced. While prioritised experience replay aims to replay important transitions more frequently, and therefore learn more efficiently. (33) recorded that it sped up learning by a factor of 2, and it achieved a new state of art benchmark as it out preformed normal experience replay in 41 out of 49 Atari games. Prioritised experience replay samples more frequently transitions from which there is most to learn as measured by the temporal difference error (TD). It samples transitions with probability p_t relative to the last encountered absolute TD error as shown in the equation below.

$$p_t \alpha |(R_{t+1} + \gamma_{t+1} \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))|^{\omega} \tag{18}$$

Where ω is a hyperparameter that is used to determine the shape of the given distribution. This means that new transitions are inserted into the replay buffer with maximum priority, providing a bias towards recent transitions. The algorithm for Double DQN with proportional prioritisation can be seen in Algorithm 2 (33).

Algorithm 2: Double DQN with proportional prioritization

Input: minibatch k, step size η, replay period K and size N, exponents α and β, budget T.

Initialize replay memory D = ϕ, $\triangle = 0$, p_i
Observer S_0 and choose $A_0 \sim \pi_\theta(S_0)$
for t=1,T **do**
  Observe S_t, Rs_t, γ_t
  Store transition $(S_{t1}, A_{t1}, R_t, \gamma_t, S_t)$ in D with maximum priority $p_t = max_{i<t} p_i$
  if $t \equiv 0$ *mod* K **then**
    for j=1, k **do**
      Sample Transition j/ sim $P(j) = p_j^\alpha / \sum_i p_i^\alpha$
      Compute importance-sampling weight $w_j = (N \cdot P(j))^\beta / \max_i w_i$
      Compute TD-error $\delta_i = R_j + \gamma_j Q_{target}(S_j, \arg\max_a Q(S_j, a)) - Q(S_{j-1}, A_{j-1})$
      Update transition priority $p_j \longleftarrow |\delta_j$
      Accumlate weight change $\triangle \longleftarrow \triangle = w_j \cdot \delta_j \cdot \nabla_\theta Q(S_{j-1}, A_{j-1})$
    end
    Update Weights $\theta \longleftarrow \theta + \eta \cdot \triangle$, *reset*$\triangle = 0$
    From time to time copy weights into target network $\theta_{target} \longleftarrow \theta$
  end
  Choose Action $A_t \sim \pi_\theta(S_0)$
end

Dueling Network

The architecture of a Dueling Network was proposed by the DeepMind team at Google in 2016 (2). A dueling network is a type of network used to help calculate a better Q value. A dueling network is used to calculate the value of V(S) and A(s,a) separately. Where V(S) is the value of being at that state while A(s,a) is the advantage of taking that action a, at a given state s. V(S) and A(s,a) are calculating separately using two different streams and are then combined together using a convolutional layer to then produce an estimated value for the state action pair Q(s,a). The Q value can be decomposed, as shown in equation 19. Where $V^\pi(s)$ expresses how optimal it is to be in state s, while A(s,a) expresses how more advantageous it is to take that action over other actions.

$$Q^\pi(s, a) = V^\pi(s) + A(s, a) \tag{19}$$

These two different streams can be seen in Figure 2.4. The dueling network automatically produces separate estimates of the state value function and advantage function without extra supervision. This new network allows the algorithm to learn which states are useful without actually having to explore each action for each state. The final equation used to obtain the Q values can be seen in equation 20 below (2):

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left[A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a; \theta, \alpha) \right] \tag{20}$$

where θ is the common network parameters, α is the advantage stream parameters, and β is the value stream parameters.

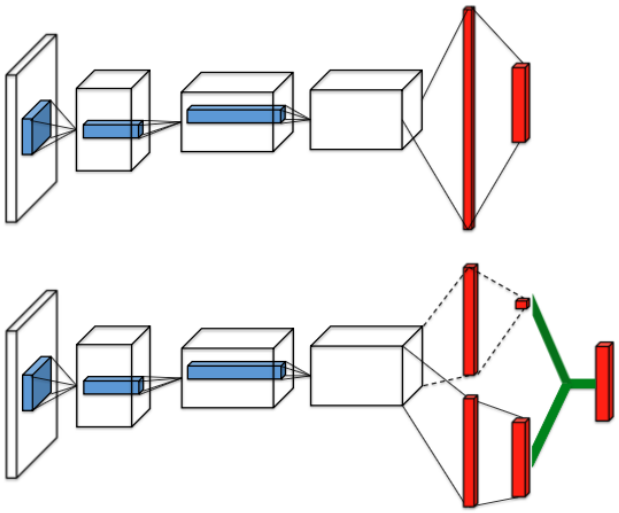


Figure 2.4: On the top, a normal single stream Q-Network can be seen and the dueling Q network on the bottom. The duel network has two streams that both estimate the state-value, the green output module implements equation 20 to combine them. Therefore allowing the dueling Q network to output Q value like the single-stream Q-Network (2).

2.3.5 Multi-agent Deep Reinforcement Learning

If we consider a fully cooperative multi-agent setting in which n agents identified by $a \in A \equiv 1, .., n$ participate in a stochastic game, G, described by a tuple $G = \langle S, U, P, r, Z, O, n, \gamma \rangle$. The environment consists of given states $s \in S$ and at every time step t, each agent takes an action $u_a \in U$, forming a joint action $u \in U = U^n$. State transition probabilities are defined by $P(s'|s, u) : SxUxS \rightarrow [0, 1]$. A global reward function given as $r(s, u) : SxU \rightarrow \mathbb{R}$ is shared between the different agents.

For each agent's observation $z \in Z$, they are governed by a given observation function $O(s, a) : SxA \rightarrow Z$. Each agent a conditions its behaviour on its own action-observation history $\tau_a \in T \equiv (ZxU)*$, according to its policy $\pi_a(u_a|\tau_a) : TxU \rightarrow [0, 1]$. After each transition, the action u_a and new observation $O(s, a)$ are added to τ_a, forming $\tau_a^{'}$. (34)

Independent Deep Q Network

Independent Q Learning (IQL) was a technique proposed by Ming Tan (35) in 1993, and it is considered one of the most simple and popular approaches to multi-agent reinforcement learning. Each agent trains themselves independently, and in doing so, the agent can learn its own optimal policy. Each agent a_1 is independent of each other, and in doing so, it will treat the other agents $a_2...a_n$ as part of the environment. Each will identity its own policy, and like Q learning, it will only be affected by the state and actions that the agent takes. IQL is very

useful as it does not have scalability problems, unlike centralized learning when it comes to learning a joint Q function that countdowns on U. The value of |U| will grow exponentially with the number of agents. However, IQL introduces a new issue, the environment becomes non-stationary from the point of view of each agent, as it involves other agents who are themselves learning at the same time, ruling out any convergence guarantees. This issue can be rectified using Fingerprints as discussed in section 2.3.5 below.(34)

Independent Deep Q Networks is a combination of both IQL and Deep Q networks. Where Deep Q Networks have been extended to incorporate a multi-agent environment. Each agent a will observe a given state s_t at time t and it will select a give action u_t^a. This will result in a team reward that will be shared among all the agents in the environment. Tampuu (36) integrates DQN with independent Q-learning, where an agent a independently and simultaneously learns its own Q-function $Q^a(s, u^a; \theta_i^a)$ (37). Since the environment in which the project will be run is somewhat observable, an IDGQN could be applied, where each individual agent would have its own DQN based on its action and corresponding states.

In section 2.3.1 it was discussed how important the use of experience replay memory was for a Deep Q Network. However, the experience replay combined with IQL causes a problem. The non-stationarity introduced by IQL means that the dynamics that generated the data in the agent's replay memory no longer reflect the current dynamics in which it is learning. It has been well documented that IQL can still learn well without memory replay as long as each agent can gradually keep track of the other agent's policies. However, if experience replay is enabled it can confuse the agent by using obsolete experiences. To remove this issue, researches have either limited experience replay to short, recent buffers (38) or disabled it altogether. However, these workarounds limit the sample efficiency and threaten the stability of multi-agent reinforcement learning. Therefore it is essential for A DQN to use experience replay. One possible solution to this problem is to use a technique called Multi-Agent Fingerprints.

Multi-Agent Fingerprints

(34) proposes two different methods to incorporate experience replay into multi-agent reinforcement learning. The first method that he proposes in this paper is called Multi-Agent Importance Sampling, which is similar to prioritized experience replay. While the second method is named Multi-Agent Fingerprints. In the evaluation section of this paper, it can be clearly seen that the second method, Multi-Agent Fingerprints, out preforms Multi-Agent Importance Sampling in forward feeding networks, which will be used in this research.

Foerster also describes the weakness of independent Q learning in this paper. This weakness of IQL is that by treating other agents as part of the environment, it ignores the fact that such agents' policies are changing over time, rendering its own Q-function non-stationary. However, the Q-function could be made stationary if it is conditioned on the other agents' policies.

To solve this issue, a method called hyper Q-Learning was used (39). In Hyper Q-Learning, each agent's state space is changed by estimating the other agents' policies computed via Bayesian inference. This will in turn, reduce the agents learning problem to a standard single-agent problem. The environment will be stationary but will be much larger. This will therefore increase the dimensionality of the Q function. If we consider including weights of the other agents networks, θ_{-a} in our observation function e.g $O'(s) = (O(s), \theta_{-a})$. We could theoretically learn a mapping from the weights and is own trajectory into expected returns. However, if other agents are using deep models, then we have to ignore their weights θ_{-a} as they are too large to include in the Q function.

A key observation is to help stabilize experience replay because the agent only needs to condition on those values of θ_{-a} that are actually present in replay memory. The sequence of policies that generated the data in this buffer can be thought of as following a single, one-dimensional trajectory through the high-dimensional policy space. Therefore to stabilize experience replay, each agent's observations were along the trajectory where the current training sample originated from (34).

2.4 Co-operative Driving Strategies and their effect on Traffic Conditions

2.4.1 Definition

In the past 20 years, there has been much development of key technologies in vehicles. These new technologies provide basic control assistance (such as traction control, anti-lock braking system). These improvements are primarily focused on increasing safety and increasing driver and passenger comfort. As defined in (40) an intelligent vehicle is a vehicle that is enhanced with perception, reasoning, and actuating devices that enable the automation of driving tasks such as safe lane following, obstacle avoidance, overtaking, following the vehicle ahead, assessing and avoiding dangerous situations. Furthermore, as Bishop (41) described in his article, Intelligent vehicles should operate at the tactical level of driving (throttle, brakes, steering) as contrasted with strategic decisions such as route choice. Even though these new technologies are aimed at increasing traffic safety, they can also have positive effects on traffic flow and efficiency.

In this paper, we will consider three different levels of Intelligent Vehicles. These include adaptive cruise control vehicles (ACC), co-operative adaptive cruise control (CACC) and fully autonomous vehicles. As mentioned above, moving the control of the vehicle from the human to the vehicle itself can improve almost metric in regards to traffic management and safety.

There are two main areas of control that these systems can utilize. Firstly, Lateral Control refers to the use of the vehicles steering and the ability of a vehicle to detect lanes on the road. Secondly, longitudinal control. There are two main elements in longitudinal control, vehicle dynamics and power-train dynamics. Vehicle dynamics are influenced by longitudinal tire forces, aerodynamic drag forces, rolling resistance forces, acceleration, velocity, braking and gravitational forces. The vehicle's longitudinal power-train system consists of the internal combustion engine, the torque converter, the transmission, and the wheels (42).

Cruise control systems have been in place since the mid 20th century. These systems allow the driver to maintain a given speed without any input. However, nowadays, numerous high-end cars will contain ACC. ACC was first developed by Toyota in the late 1990s (43). The aim of ACC is to monitor and maintain a given headway between a car and the car in front. To keep track of this distance, it will use a combination of either lidar or radar sensors and will also keep track of the cars current speed. Hence, the car's speed is automatically controlled to maintain a given distance to the car in front. The cars speed may increase or decrease with a minimal amount of acceleration or deacceleration to keep this gap. In recent years further enhancements have been made to ACC, and these new improvements have created a new system called CACC. CACC allows for vehicle-to-vehicle communication.

In terms of vehicle communication, there are two distinct types. The first one is vehicle-to-vehicle communication (V2V), and this is when two or more vehicles are directly connected to each other and communicate their actions with each other. The second type of vehicle communication is vehicle-to-Infrastructure (V2I). V2I communication the wireless exchange of information between vehicles and the road infrastructure such as road signs, traffic lights etc. (44). CACC allows vehicles to follow more closely, accurately and safely with braking and accelerating done together in sync. CACC solves the issue of string instability in ACC systems (45).

Autonomous Vehicles are vehicles that are capable of performing both lateral and longitudinal control without any human interaction within the system. It can complete autonomous driving through the use of the vehicles sensors, cameras and other equipment. One of the most advanced autonomous vehicles is the google driving car, Waymo. The system has currently driven over 32 million kms in the real world and over 10 billion in simulation. Like ACC, it uses Lidar and radar sensors; along with this, it also has microphones to help detect emergency services. However, having fully autonomous vehicles along do not hugely improve traffic flow or efficiency. To see a more significant improvement in Traffic flow or efficiency, autonomous vehicles still need to use V2V or V2I communication.

2.4.2 Traffic Flow

This section will discuss both the state art of methods for improving traffic flow and how each of the different Intelligent Vehicles affects traffic flow in different situations. As ACC vehicles became more readily available, it quickly became a very active area of research for academics. One of these areas was how the Traffic flow would be affected by introducing ACC vehicles. One of the first papers conducted by Broqua in 1991 (46) concluded that if the penetration rate of ACC vehicles was above 40% and the target headway was set to a value of 1 second. Then the overall throughput of the given traffic flow would be increased by 13%. However, this study was partially contradicted by studies such as (47) and (48). These papers showed that ACC had minimal effect on traffic and was shown that certain aspects of the system would need to be changed to impact traffic flow positively. However, since those early papers, there have been considerable advancements in the simulation software used to model both ACC vehicles and a traffic simulation. As well as this, ACC vehicles have been made more readily available in the real world, and hence there are numerous times more data available to researches now compared to the 1990s.

One of the first papers that gained recognition was by Kesting (49). He investigated how the strategy of ACC could be changed to have a positive impact on traffic. More specifically, his research aimed to increase the capacity and stability of flow in traffic. In his research, he used a scenario where an on-ramp merging into a one-lane road creates a bottleneck during high traffic demand. He concluded that current ACC systems were used for driver comfort and safety, and then adding new measures such as constant time headway between vehicles

or by adding jam-avoiding parameters could help delay or suppress traffic breakdowns. He also showed that even if the penetration rate of ACC vehicles were as low as 30%, this would positively affect the traffic and possibly cause Traffic jams to disappear completely. In future work, Kesting proposed an ACC-based traffic assistance system with an active jam-avoidance strategy. This method will vary the driving strategy based on the current traffic situation. He again produced better results with his new system, only requiring a 5% penetration rate to see a noticeable difference in traffic flow and reduced travel times.

At the turn of the decade (2010), the emergence of CACC systems began to be widely investigated. One of the first papers to be published was Ploeg (50) in 2011. He proved both theoretically and experimentally that CACC, which is based on standard ACC sensors and a wireless inter-vehicle communication link, allows for time gaps significantly smaller than 1 s while maintaining string stability. He also concluded that improves traffic throughput/flow and a decrease in emission could be expected. Other papers such as (51) and (52) agree with this conclusion that, in general, the use of CACC in traffic can increase traffic flow and throughput. These papers also conclude that traffic stability can be improved by introducing CACC vehicles into the system. Regarding these papers, there is much contradiction as to what penetration rate does CACC vehicles become effective. This value is very subjective as the penetration rate will vary depending on the environment and network in which the simulation was run.

Furthermore, in recent years Autonomous vehicles have seen a surge in development, and along with this, there has been additional research conducted on autonomous vehicles and the impact they will have on the traffic system at large. In terms of traffic flow, autonomous vehicles like ACC and CACC vehicles improve traffic flow. Wu proposed that a penetration rate of 6% was enough to efficiently stabilize traffic under all uniform flow conditions on a given highway (53). Lu (54) conducted research to see the impact of autonomous vehicles in an urban setting; again, her results were very similar to that of Wu. That is, autonomous vehicles improve traffic stability, flow and throughput. It increases these characteristics as it allows vehicles to have a higher speed whilst maintaining a lower headway to the car in-front compared to human drivers. Other research not mentioned also agrees with Wu's and Lu's research.

The overview of ACC, CACC and autonomous vehicles shows that incorporating any of these methods will improve traffic flow. The use of these methods will prevent harsh braking/accelerating that is custom for human drivers. This is one of the leading causes of traffic congestion and traffic bottlenecks.

2.4.3 Traffic Safety

Traffic safety is one of the most important factors we need to consider regarding intelligent vehicles. If these vehicles begin to cause more accidents than human drivers, then they are of

no use to society. In ACC vehicles, they have the ability to brake up to half the cars maximum braking potentially. This is to ensure driver and passenger comfort. However, only having this level of braking available to the system may not prevent the occurrence of crashes in emergency conditions. Li (55) investigated the effect that ACC vehicles have on traffic safety in traffic oscillations on freeways.

In conclusion, he found that the effect of ACCs have on traffic safety largely depend on parameter settings. These parameter settings included perception-reaction time, the initial gap between vehicles, and deceleration ability. If the settings were set correctly, then ACC systems would have a positive impact on traffic safety. In general, he found that better safety effects were found when the time gap was larger, the time delay was smaller, and the maximum deceleration rate in emergencies was greater. Furthermore, if the penetration rates of ACC vehicles were increased, there would be a corresponding positive impact on traffic safety.

Marsdens work (56) also agrees with Li that ACCs do have the capability to improve traffic safety. However, Marsden brought up an important part regarding ACCs and there use in heavy traffic situations. In ACCs, the driver has the option to overrule the system if emergency braking is needed to prevent an accident. When this emergency stop has been concluded, there is ambiguity about whether the driver will re-engage the ACC during congested driving conditions. Hence, it could be said that ACCs do not improve traffic safety when traffic is congested as it is not being utilized.

Traffic safety for CACC vehicles on freeways was investigated by Li in (57). In this research, Li investigated how the overall safety would be affected by adding CACC vehicles to a freeway. To help analyze the results, two surrogated safety measures were introduced. These included time exposed time-to-collision (TET) and time-integrated time-to-collision (TIT). Both measures are derived from the time-to-collision (TTC) metric, where the TET is defined as the summation of all moments that a driver approaches a front vehicle with a TTC-value below the threshold value TTC*. While TIT is defined as Integral of the TTC-profile during the time it is below the threshold. The definition of TTC is the time until a collision between the vehicles would occur if they continued on their present course at their present rates (58).

In his study, Li successfully showed that both his theoretical and simulation results conformably indicate that the CACC dramatically improved traffic safety by showing the reduction of risk for rear-end collisions. Furthermore, if both key factors, desired time headway and time delay, were set to the appropriate values, both TET and TIT could be reduced by more than 90%. However, it was also noted that if the penetration rates of CACC vales were reduced, this negatively affected traffic safety.

CACC vehicles communicate with each other via wireless communication, whether it be V2V or V2I communication. However, if this communication link breaks down, the vehicle automatically returns to using just an ACC system. This is known as degradation and can happen for a number of reasons, such as communication failures, driver manipulations, and cyber-

attacks. As mentioned in (59) degradation has a significant negative impact on traffic safety. An interesting observation from this paper regarding CACC degradation is that if the vehicles in the middle of the platoon had degradation, these would have the least effect on traffic safety. Another observation from this paper is that when non-adjacent degradation occurs, this is much riskier than adjacent degradation at the front positions of a fleet.

There have been numerous studies on the effect of autonomous vehicles on traffic safety. The majority of these papers firstly analyze how safety is impacted depending on the penetration rates of autonomous vehicles. (60) (61) (62) all suggest that autonomous vehicles have positive impact on traffic safety. Both (60) and (62) concluded that there was an optimal value for the penetration rate that was sufficient to reach near-maximum benefits. In both papers, it can be seen that the value was around 40%. These papers also suggested that safety could be further positively impacted if the autonomous cars followed a more cautious car following approach. However, this suggestion would reduce traffic flow and reduce the capacity of the road. The research conducted by (60) evaluated the impact of safety in 3 different road scenarios. The impact was evaluated on a motorway, national road and in an urban setting. This was different to (62), and (61) who evaluated it on 1 type of road. Across all three road types, it was shown that autonomous vehicles do have a positive impact on traffic safety on average. The paper also concluded that accidents were more likely to occur in an urban setting as it contained 435 intersections with very complex junction layout and actuated traffic light controllers.

As discussed in section 2.4.3, it is clear that under the right conditions, ACC, CACC and autonomous vehicles all have a positive impact on traffic safety. Utilizing any of these methods will reduce the risk of an accident occurring. In summary, it was found that autonomous had the most positive impact on traffic safety compared to the other methods.

2.4.4 Lane Changing

In this section, the different technologies used to help vehicles change lane are discussed. This section may not have direct relevance to this project, but lane changing has vast impacts on traffic flow and safety as the majority of traffic collisions and jams stem from sudden lane changing, which in-turn causes emergency breaking.

As stated in the definition of ACC vehicles, these type of vehicles do not have the correct onboard technology to allow them to change lane by themselves. However, ACC vehicles do have the ability to react to other vehicles changing lanes in front of them. They can de-accelerate the vehicle while limiting the number of disturbances to the traffic behind. There have been systems added on to ACC to help with lane changing procedures. One of these systems was proposed by Kim in (63). While this might seem like the ideal solution, it is not. It only concentrates on the host vehicle's longitudinal and lateral movements while ignoring inter-vehicle relationships between the host vehicle and surrounding vehicles. Furthermore, the

controller designed in (63) assumed that the lateral trajectory was similar to sine or trapezium wave and that the longitudinal speed was constant. These assumptions would not relate to the real-world environment.

Both CACC and autonomous vehicles have the capabilities to perform lane-changing manoeuvres. Whether this be through onboard sensors that detect the lines on the road and the distance between other vehicles. The use of vehicle-to-vehicle or vehicle-to-infrastructure communication can also be used to help ensure safe lane changing.

2.5 Personalized Driving

This section includes a brief discussion on personalised driving and its potential when combined with RL. The papers described in this section aim to learn the driver's policy using a range of techniques in the areas of Deep Learning and Reinforcement Learning. However, before these policies can be learnt and applied, the driver's behaviours need to be monitored and computed. One of the first papers to propose a method to measure driving style through a smartphone was Johnson (64). His paper aimed to increase awareness and promote driver safety. He proposed a system that uses Dynamic Time Warping and smartphone-based sensor-fusion (accelerometer, gyroscope, magnetometer, GPS, video) to detect, recognise, and record the driver's given action. The inbuilt algorithm was able to distinguish the difference between aggressive and non-aggressive driving. It was able to determine this by the number of events the smartphone was recording. Some of these include aggressive braking, aggressive acceleration, swerve right/left, excessive speed and right/left turn. In the results, he concluded that the Dynamic Time Warping algorithm and the smartphone sensors were more than adequate to detect these events. This paper also highlights how easily this algorithm could be distributed to a broad audience due to the high availability of smartphones in the 21st century.

The area of using RL or DRL in personalised driving is very much state of the art. The two paper discussed in this section have been published in the last number of months. Firstly, (65) presents a self-learning method for personalised driving decisions. To collect information about the driver, a number of different metrics are required. During the experiment, metrics such as acceleration, speed and braking are monitored and stored for analyses once the data collection has concluded. During the data collection phase, a personalised expression is also developed. With this information and the use of a Deep Deterministic Policy Gradient algorithm, a suitable driving decision system is then designed for each of the individual drivers. To test their algorithm, it was not feasible to test in the real world for various reasons. So, the TORCS driving simulation platform was used. During the testing, the algorithm performed positively. However, during training, the algorithm was slow to converge and did have numerous issues such as causing collisions and leaving the road.

Secondly, (66) presents a methodological tool to produce personalised and quantified policies to improve the driver's behaviour. The data collected in this study was through the use of a

smartphone. The information collected allowed metrics such as number of harsh accelerations per 100 kilometres, number of harsh brakings per 100 kilometres, mobile phone usage while driving expressed as a percentage of driving time, and speeding expressed as a percentage of driving time over the speed limit. The driving policies developed in this study are based on those metrics. For a drivers trip to be considered optimal in this study, all the previous metric values should return a value of 0. To produce a personalised and quantified policy, a R realisation of the Q learning algorithm is used. The algorithm will then return a given action that the driver should take to help optimise their driving. It was also concluded that drivers might need to take one large action to improve their driving or take multiple small actions. During this study, there was no actual reward for the driver if they tried to drive optimally. An interesting further study could be planned to see how the experiment would be affected if the drivers were offered a reward if they drove optimally.

2.6 Deep Reinforcement Learning in Traffic Control, Safety and Flow

As mentioned in the previous sections, RL's main objective is to get the optimal policy through exploration and exploitation in the training phase according to the different inputs and outputs of the agent during the simulation. The issue of reinforcement learning with traffic simulations is the exponentially increasing complexity of the dimensionality in regards to the number of agents and their characteristics. Therefore it does not become feasible to use traditionally RL techniques such as table based Q learning. Using these traditional techniques would become very expensive both in terms of computing power, memory, and time. Furthermore, traditional techniques such as using a function approximator based Q learning method will also have issues capturing the dynamics and randomness nature of traffic flow. Therefore Deep Learning is then introduced to solve some of these problems. DL has a special characteristic that allows it to concurrently solve both modeling and optimization of the given systems using ideas such as Deep Neural Network and Deep Q Networks. Combining both RL and DL allows for the use of neural networks to train and to give an appropriate action after receiving a certain input so that the award is maximized. Using these methods will eliminate the issue of dimensionality.

The above areas of reinforcement learning have been extensively used in traffic control systems. So in the proceeding sections the discussions will be more focused on traffic control rather than traffic safety and flow. However an important point to realize is that when traffic control is fully optimized it will have positive impacts on traffic flow and congestion.

2.6.1 Multi-Agent Reinforcement Learning

Four different papers are discussed in this section that utilises techniques in RL. The papers discussed try to reduce or eliminate the problems that are caused by Multi-Agent Reinforcement Learning. These include the curse of dimensionality and the need for coordination between agents to achieve optimal global policy.

(67) presents an approach that aims to improve traffic flow whilst using reinforcement learning. This paper shows that a traffic flow optimisation problem can be formulated as a Markov Decision Process (MDP). An MDP is a mathematical framework that is used to Model sequential decision-making problems. A Q learning method is also utilised to learn the given policy so that the highway's maximum driving speed can be obtained and reduce traffic congestion. For this to happen, a Speed limit policy algorithm was introduced. This is when the Q value determines the speed of the highway in the next time step. This speed limit policy could be considered as a hidden state task as the state description is not Markovian, and other information is not included in the MDP. To store an appropriate Q function in memory, a linear approximator method called tile coding is used. This allows Q-Learning to be used when the state space is continuous and improves learning efficiency because it generalises learning experiences.

(3) presents an algorithm for collaborative multi-agent, multi-policy optimisation in a decentralised autonomic system based on W-learning. It allows for an optimisation of traffic management goals and the learning and decision-making phases. The name of this algorithm is Distributed W-Learning (DWL). DWL is an RL algorithm for multi-policy optimisation in agent-based systems. In DWL, intelligent agents are allowed to communicate and work with each other in order to meet multiple polices. This characteristic of heterogeneity has shown to improve the overall performance of the agents. From Figure 2.5, it can be seen that both local and remote policies are calculated. The local policy is calculated for the local agent. While the remote policy is calculated, taking into account the local agent's neighbours. The given local and remote Q and W values are updated using both the Q-Learning and W-Learning algorithms. The W-learning algorithm allows the agent to learn the importance of each policy in each state. The values, along with a given action, is then presented to the agent. The action with the highest associated W-value is then executed.

(4) presents a method called particle swarm optimisation, which is a population-based method. Using these techniques will allow a global optimal solution for a multi-modal function in an arbitrary short space of time. The particle swarm optimisation method took its inspiration from the social behaviour of birds within a flock, and it was initially designed for use in continuous optimisation problems. In this paper, the agents had two different manners in which they could learn. Firstly, they could learn through their own experiences. Secondly, they can learn by exchanging information between other agents. (4) proposed an architecture where the agents could communicate with two different levels. Firstly, Intra-level, this is where agents interact

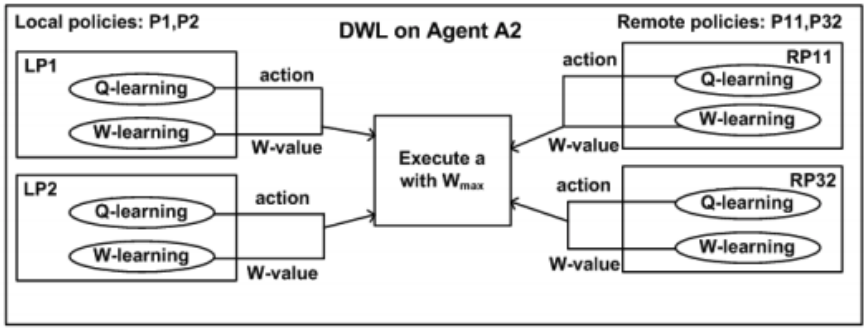


Figure 2.5: Distributed W-Learning (3).

with each other on the same level, also known as horizontal interaction. Secondly, Inter-Level is were agents in different levels of hierarchy communicate with each other. In Figure 2.6, a visual representation of Tahifa's architecture can be seen. Furthermore, the paper explained that the traffic agents communication to other traffic agents was Intra-level. At the same time, communication between the monitor agent and the traffic agent was Inter-level.

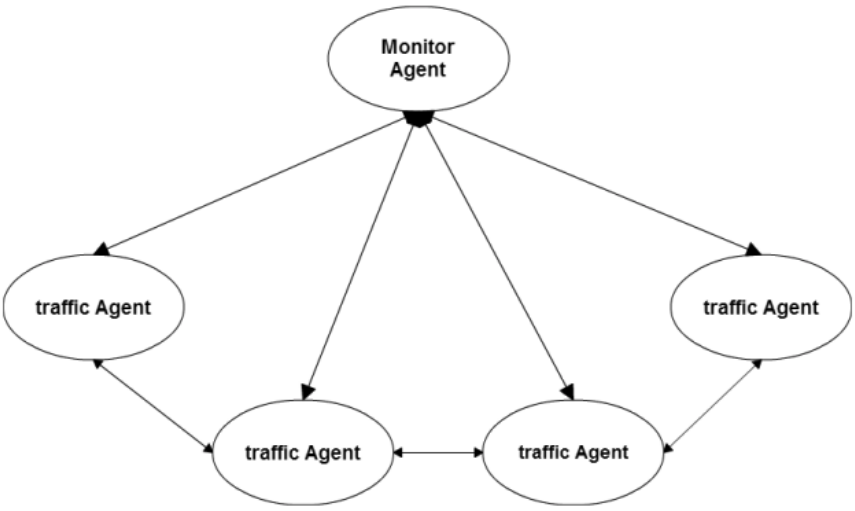


Figure 2.6: (4) proposed Architecture.

(68) presents Multi-Agent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controller (MARLIN-ATSC). The MARLIN learning approach presents a new control system that maintains an explicit coordination mechanism while addressing the curse of dimensionality problem. To address this problem, the approach utilises two techniques. Firstly, by exploiting the principle of locality of interaction among agents. This approach aims to estimate a local neighbourhood utility that maps the effect of an agent to the global Q-value function while only considering the interaction with its neighbours. The second technique utilises the Modular Q-Learning Technique. Modular Q-learning partitions the state space into partial state spaces that consist of two agents. This in turn, causes the partial space

always to be of size $|S|^2$ regardless of the number of agents present in the environment. In MARLIN, agents can be implemented in 2 different ways. (1) Independent Mode is were each agent learns and reacts independently to other agents in its environment. (2) Integrated Mode, each agent coordinates the signal control actions with the neighbouring agents by implementing a MARLIN learning algorithm. In MARLIN, each agent plays a game with all its adjacent intersections in its neighbourhood. The game that is played is known as a Markov game (69) or a Stochastic game. Each agent will learn numerous modules were each module will correspond to a given game. Using the principle of modular Q-learning, the agent learns the joint policy with one of its neighbours at a time. The state and action spaces are distributed in such a way that allows this learning to take place. Table 2.1 below summaries the existing research in Multi-agent reinforcement learning.

Research	Algorithm	State	Rewards
(67)	Markov Decision Process and Q-learning	Speed limit of the given highway	reduction of traffic count
(3)	Distributed W-Learning	Number of vehicles waiting and presence of public transportation	reduction of traffic count and no public transport waiting.
(4)	Swarm Reinforcement Learning	Average queue length	number of vehicles waiting and crossing.
(68)	Q-Learning and MARLIN	Queue Length	Reduction in the Total Cumulative Delay

Table 2.1: Summary of the existing research in Multi-agent reinforcement learning.

When comparing the papers above there is a number of interesting analysis that can be made. Firstly, the way that agents learn in each of the four papers. In all but one paper, the agents learn locally first and then uses a communicating technique to share their experiences and learning's to other agents in the environment. Only MARLIN-ATSC do not use this approach. Instead, they presented an idea where the agents learn alongside their neighbours without any prior knowledge of the given environment. MARLIN-ATSC proposed that the coordination between agents can be achieved by considering the joint state and joint action for the other agents. However, (3) allows the local goal to be independently modified. This also allows the agents to have different goals and a different number of goals.

Another interesting analysis to discuss is the goals that each paper is trying to optimise. In the case of the first paper, (67), the goal of this research was to reduce traffic congestion on a given highway. The reward function of this work was directly related to the delay incurred by the vehicles. So the reward function was computed by the amount of time the vehicles spent on the highway. In (3), there are two main goals that are trying to be optimised. They

try to optimise the global waiting time (GWO) and prioritise public transport (PTO). In this paper, GWO is rewarded if the overall traffic count is less in the current time step compared to the previous one (e.g. more traffic was cleared from the intersection than arrived). At the same time, the PTO is rewarded if there are no public transport vehicles present. This implies that the aim is to keep public transport moving.

On the other hand, in Swarm RL, the reward function is based on the number of vehicles waiting in an edge, the number of vehicles crossed the junction coming from an edge, the number total of vehicles present in a junction. There is also a penalty introduced if the number of vehicles crossing the roads is superior to the number of vehicles waiting for it. Lastly, in MARLIN-ATSC, the reward is defined as the reduction in the Total Cumulative delay. This cumulative delay is the summation of all the vehicles delays that are currently in the intersections' upstreams.

The final discussion is about the evaluation and results of each of the four different papers. The first paper by Walraven was conducted in a traffic simulator called METANET. Walraven conducted this simulation using eight sections of highway, each with two lanes and 3km in length. In this study, the use of tile coding was compared against the use of the neural network to update the Q-value function. It was proven that the neural network outperformed tile coding by roughly 2 percent. Overall, his algorithms positively impacted simulations, whether the algorithm had to learn policies for single or multiple scenarios. Secondly, (3) was conducted in an "in-house tool" developed by Trinity College Dublin. The simulation was conducted in an environment with 270 junctions, 62 of which are signalled controlled. The experiment simulated 12 hours of realistic traffic. From the results, it showed a considerable reduction in waiting times ranging from 73% to 88%.These values and a range of tests and evaluations showed that the DWL algorithm is suitable for multi-agent multi-policy optimisation.

In the investigation of the use of Swarm RL, the researchers used an open-source simulator called SUMO. The network used consisted of 4 junctions in a grid presentation, with each section of the road having two lanes and a length of 600m. The results compares and contrasts Q-Learning against Swarm Q-Learning(SQL). It concluded that SQL reduced vehicles' waiting time compared to an architecture with a standard multi-agent Q-learning algorithm. This was down to the fact that there was cooperation between agents in SQL. Lastly, MARLIN-ATSC, this research used a traffic simulator called Paramics. The network described in this paper contained 59 junctions in the lower downtown core of the City of Toronto, Canada. The simulation used real traffic rush hour data. Overall from this study, the results were extremely positive. Firstly, the average intersection delay in the independent mode was 27%, and in the integrated mode were 39% at the network level. Secondly, the travel savings were also significantly reduced, 15% in independent mode and 26% in integrated mode. Table 2.2 summaries the environment and results of each paper.

Research	Number of Junctions	Junction Structure	Environment	Overall impact on environment	Goals Achieved?
(67)	2	Regular. 2 sections containing on-ramps	METANET	Reduced traffic congestion	Yes
(3)	270	Irregular. All structures are different	"In-House tool"	73% to 88% reduction in waiting times	Yes
(4)	4	Regular. All junctions the same.	SUMO	SQL preformed better than Q-Learning	Yes
(68)	59	Irregular.	Paramics	Both modes had a large reduction on waiting times.	Yes

Table 2.2: Summary of the environment and results of each paper.

2.6.2 Single-Agent Deep Reinforcement Learning

In the following section, four pieces of work have used Deep Learning and Deep Reinforcement Learning to help solve traffic control and congestion problems. The use of deep learning and neural networks help deal with the issue of dimensionality that regular reinforcement learning brings.

(70) built two different RL algorithms to predict the best possible traffic signal for a junction. The methods that the paper proposes are deep policy-gradient (PG) and value-function based agents. The PGs objective is the optimise a given policy by using gradient descent. The value function is based on the Deep Q-Learning Algorithm. The study also utilises a target Q network to address the instability problem of a stand-alone Q-network. The architecture used in this paper is similar to that of Mnih in (29). Firstly the algorithm takes a snapshot of the SUMO-GUI and converts the image to grey-scale and resized to 128x128. This allows the system to memorise past experiences. The current image is then stacked with the past three images from the system and past as a 128x128x4 image to the network. The network consists of 3 hidden layers. Two of which are convolutional layers, and one is a fully connected layer with 256 nodes. A non-linear rectifier follows each hidden layer. The last layer is, therefore, different depending on the RL algorithm that is being used. For the Q value function, the output layer is a fully connected linear layer with a number of output neurons. While the PG algorithm, the last layer has two output nodes. The node is a softmax output resulting in a probability distribution over the actions A. The second node is a single linear output node resulting in the estimate of the state-value function. To evaluate the proposed method, a shallow neural network was built and used.

(5) presents a DNN to learn the Q-function. The DNN used in this paper is a deep-stacked auto encoders neural network (SAE). This network will take the given state as the input and outputs the Q-value for each given action. The building blocks of this DNN are the autoencoders. The definition of an autoencoder is that the target output is set to be equal to the input. The autoencoders are then "stacked" in a greedy layerwise manner, so the network's weights can be initialised. During the learning process, the SAE network is trained by minimising a loss function over samples of experience until a given time. The network is a four-layer SAE neural network that has two hidden layers. In the hidden layer, a sigmoid activation function is used. Figure 2.7 shows a visual representation of the SAE neural network used in (5).

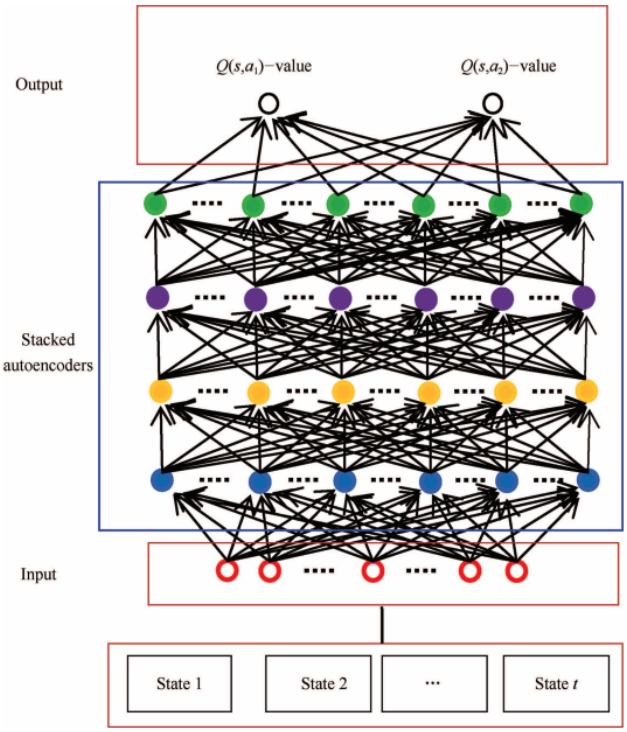


Figure 2.7: SAE neural network used in (5)

(6) presents the use of a Deep reinforcement algorithm to extract useful information and features from raw, real-time traffic data. To do this, he utilises a Deep Q-Network alongside techniques such as target network and experience replay to help stabilise the learning. The input to the neural network consists of the position matrix (P), speed matrix (S) and the latest traffic signal state (L). Both P ans S are fed into similar neural networks but will have different parameters. Each matrix is then passed through 2 convolution layers with a ReLU activation function. After this, the matrices are then flattened and are then concatenated with L. Layers 3 and 4 are fully connected layers with 128 and 64 nodes respectively, it also uses the ReLU activation function. The last layer is the output, which presents the Q-value for every given action. Figure 2.8 represents a visual representation of the network described above.

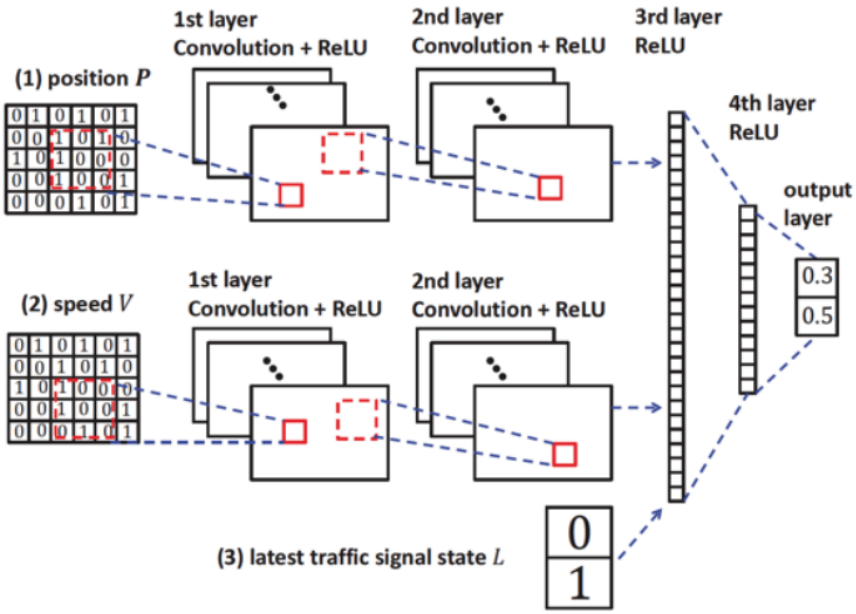


Figure 2.8: The DQN used in (6)

As mentioned before, (6) uses experience replay and a target Network to help stabilise learning. As described in this paper, the estimated Q-value is learnt from the DNN, and it is then passed to the target network, which will help it learn the optimal Q value. This target network has the same architecture as seen in Figure 2.8. When training the DNN, it is customary to try and minimise the means squared error. However, if the input size is too large, then it will become computationally expensive. To get around this issue, Gao used the stochastic gradient descent algorithm RMSProp with a mini-batch of size 32. Overall, when the DNN is being trained, it will randomly draw 32 samples from replay memory which in turn will form 32 input data and target pairs. These values will then be used to update DNN parameters which in turn will update the target network. Figure 2.9 shows a visual representation of the training process in (6).

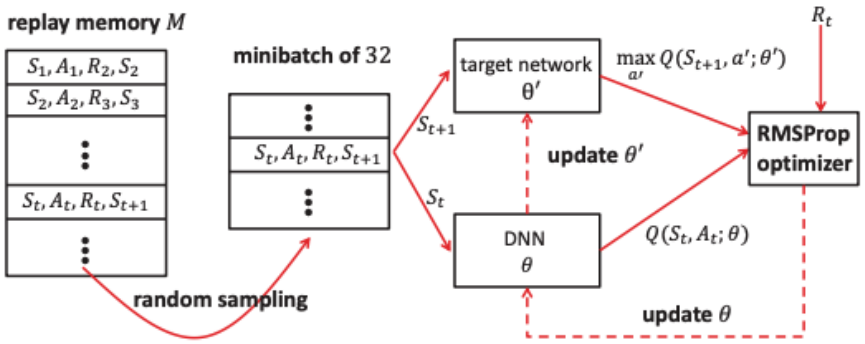


Figure 2.9: The training process used in (6)

(7) presents the use of deep reinforcement learning to control the traffic light cycle using the collected data from the different sensors. This paper uses a convolutional neural network

(CNN) to approximate the given Q-values. Along with this, they use state of the art techniques such as s dueling network, target network, double Q-learning network, and prioritised experience replay. In summary, the network is called a Double Dueling Deep Q Network (3DQN). The CNN consists of 3 different layers. With the input to the CNN having a size of 60x60x2. Where the 60x60 is the size of the grid at the intersection and the "x2" is in reference to both speed and position of the vehicles. This information is captured in a snapshot image. In this CNN, each layer contains pooling, convolution and activation. In this paper, the activation function that is used is leaky ReLU. After the three convolution's, the output is then flattened to a fully connected layer of size 128x1. This layer is then split into two equal parts. The first part is used to calculate the value, while the second part calculates the best action. In the simulation, there are nine possible actions, so the action matrix is of size 9x1. Both parts are then combined again to calculate the Q value. The tentative action will penalise the values if the actions are illegal or cause accidents. Therefore allowing an optimal Q value at the output of the network. Figure 2.10 shows a visual representation of the architecture discussed here.

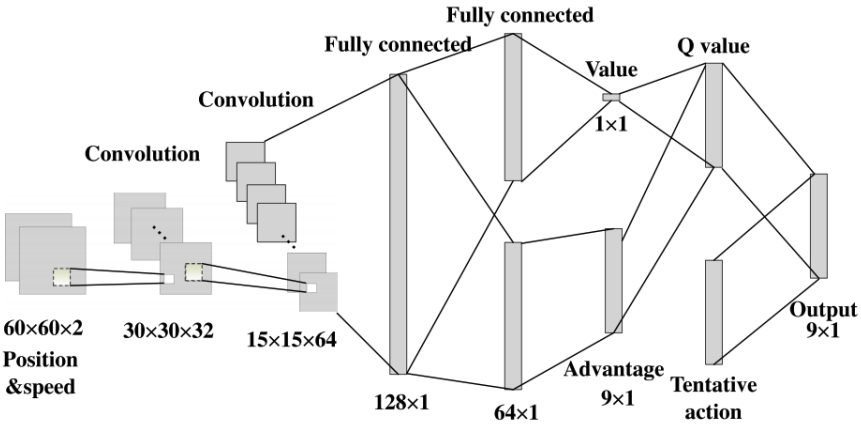


Figure 2.10: The Double Dueling Deep Q Network used in (7)

From an overarching perspective, Figure 2.11 shows the overall working architecture of this papers research. It can be seen that both the state and tentative actions are fed into the architecture as the input. This architecture will save the current state and action as well as the next and action into memory. The architecture utilises prioritised experience replay to generate mini-batches which are then used to update the primary neural network's parameters. To increase the stability of the network, a target network is also used. To remove the problem of overestimation, both a double DQN and a dueling network are used. This also improves performance and helps calculate the optimal Q values, leading to the optimal policy.

Table 2.3 below summaries each piece of work in this section.

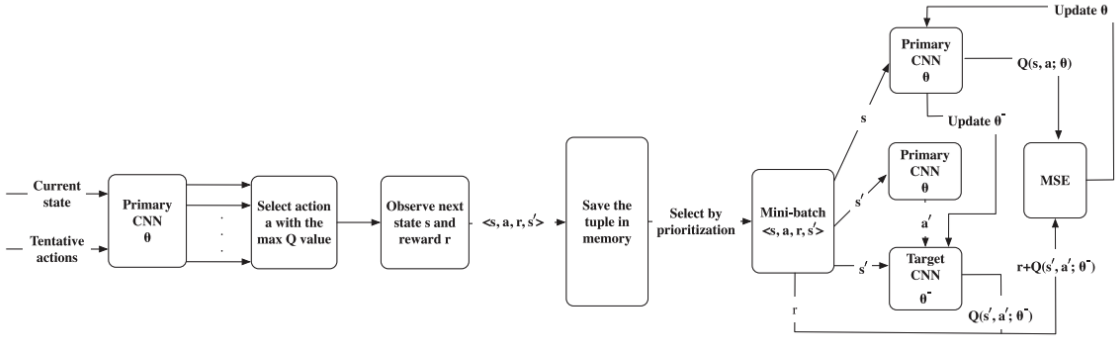


Figure 2.11: The overall Architecture used in (7)

Research	Algorithm	Architecture	State	Reward
(70)	Policy Gradient and Q- Value function	CNN	4 images of the intersection in the SUMO simulator	Difference between the total cumulative delays of two consecutive actions
(5)	Deep Q learning	Deep stacked auto encoders neural network	Speed, Queue length	Difference between opposite directions e.g North/south
(6)	Deep Q Network with Experience Replay	CNN with a Target Network	Image of intersection to extract position and speed values of vehicles	Time of vehicles spent at intersection decreases
(7)	Double Q Learning with Prioritized Experience Replay	CNN with Dueling Network and Target Network	An image of the intersection to extract position and speed of vehicles	Increase the efficiency of an intersection and reduce the waiting time of vehicles

Table 2.3: Summary of the techniques used in each paper in Section 2.6.2

One of the most interesting comparisons between the four papers is what one uses as its state representation. Three out of the four papers use an image as the state representation, with only (5) not using one. (5) may not use an image because its architecture is an SAE network instead of a CNN network. Due to the fact that the majority of papers use images as the state, it would therefore imply that these deliver good results. The variables retrieved from each image where relative speed and position of the vehicles, but they are processed differently in each paper. It is also important to mention how each paper rewards the system. In (70), the reward is defined as the difference between the total cumulative delays of two consecutive action. While in (5) the reward is defined as the absolute value of the difference between opposite directions, e.g. North/South, West/East. Thirdly, in (6) the reward was given to the system if the amount of time spent at the intersection decreased for the given vehicle.

Lastly, in (7), the reward was based on the algorithm's ability to increase the efficiency of the junction and reduce the amount of time the vehicle was waiting. Overall it can be seen that the papers had different rewards for their system. With the only similarity being between (6) and (7) were they both rewarded the agent if they reduced the waiting time.

The final discussion in this section is about the evaluation and results of each of the four different papers. Firstly, (70) uses a traffic simulator tool called SUMO to run the simulations. To evaluate there proposed solution, the use of a baseline traffic controller(BTC) was used. This BTC gives an equal, fixed time to each phase of the intersection. Their methods performed significantly better than the baseline values. It was clear from the graphs in this paper that the more epochs performed, the higher the average reward per simulation. This increase in reward reflects the agent's ability to learn an optimal policy in a stable manner. This paper also compares their proposed method to a shallow neural network (SNN) which contained one hidden layer. From the results, it was shown that their proposed solution out preforms the SNN. There was a 62% and 72% reductions in the average cumulative delay and queue length for the PG method and 68% and 73% reductions for the value-function-based method compared to the SNN. Therefore it was concluded that the PG and value function could learn the policy better than the SNN.

Secondly, (5) uses a traffic simulator tool called PARAMICS. In this paper, they compare their method against a conventional RL traffic controller. From their results, it was shown that their DRL traffic controller outperformed the conventional RL traffic controller. They saw that the average delay could be reduced by 14%, which equates to roughly the vehicles spending 13 fewer seconds at the intersection. Furthermore, it was also shown that the number of fully stopped vehicles reduced by 1020 vehicles when the DRL controller was used. This paper's last result showed that the queue length could be decreased by using the DRL controller.

Thirdly, (6) uses a traffic simulator tool called SUMO to run the simulations. One of the first results shown from the experiment was that the staying time of vehicles at an intersection decreased with each episode. After 800 episodes, it was seen that the staying time was stable at a small value which suggests that the method converges to an acceptable policy and that experience replay and target network work effectively. The proposed algorithm was then compared against two other traffic signal control algorithms. These were the longest queue first algorithm and fixed time control algorithm. The proposed algorithm showed a reduction of up to 86% compared to the fixed time control algorithm. While it showed a reduction of up to 47% compared to the longest queue first algorithm. These results were for two out of the four roads. The remaining roads showed that all algorithms performed the same with no improvement.

Lastly, (7) uses a traffic simulator tool called SUMO to run the simulations. (7) compares their proposed methods against four other different strategies. The first two force the traffic lights duration to be fixed. In this case, for 30 and 40 seconds. The third method that it is

compared to is the Adaptive Traffic Signal Control method. Lastly, it was also compared to a Deep Q- Network. The paper then compares them in 3 different scenarios. These scenarios include: (1) Cumulative reward (2) Average Waiting time (3) Average Waiting time during rush hour. From the results given, each scenario follows the same trend. The proposed 3DQN starts poorly, but as the number of episodes increases, the 3DQN begins to perform better than the other four strategies. Following these results and trends, it was shown that it could reduce the average waiting time by over 20%.

2.6.3 Multi-Agent Deep Reinforcement Learning

This section will review some of the most updated papers using Multi-Agent Deep Reinforcement Learning to solve traffic control and congestion problems. The proposed work in this report will use some techniques used in this and previous sections to enhance the algorithm's performance.

(8) presents a study that combines RL and Graphical convolution to prevent congestion. Graph neural networks (GNN) are deep learning models that preserve the dependence of graphs using nodes and edges. In this study, the nodes of the graph will be the vehicles and the communication link between the vehicles are represented by the links. GNN are very useful as they have high interpretability and performance. The input into this system is a CAV network that collects operational information. This information is then passed as node features into an FCN encoder. The output of this is known as node embeddings. These embeddings are the feature matrix passed to the Graphical Convolutional Network (GCN). An adjacency matrix representing the network topology for data flow and decision dependency is also passed as input to the GCN. The GCN then outputs a feature embedding map which is then passed to an actor and critic model to learn the value action and its given action. The actor and critic model also use the system to perform joint driving manoeuvres between all agents. This architecture can be seen in Figure 2.12 below. To ensure that the actor and critic network both converge, the use of Deep Deterministic Policy Gradient (DDPG) is needed. This method is very similar to that of a DQL, but a DDPG is more useful when using it for continuous actions. This study also uses experience replay and target networks to ensure the optimal policy and Q values are found.

(9) presents a method called multi-agent recurrent deep deterministic policy gradient (MARDDPG) algorithm. This is based on a deep deterministic policy gradient (DDPG) algorithm for traffic light control (TLC) in vehicular networks. The network requires five different matrices as its input. These are the vehicle's position (P), speed (V), number of pedestrians(M), traffic light phase (L) and the queue length (D). V, P and D all go through a range of convolutional layers with a ReLU activation function. While M and L are passed through a fully connected network. These five inputs are then concatenated back together and are then passed through an LSTM network followed by a Fully connected layer with a softmax activation function. The output of this network is to advise the agent on a given action to take. Figure 2.13 shows a

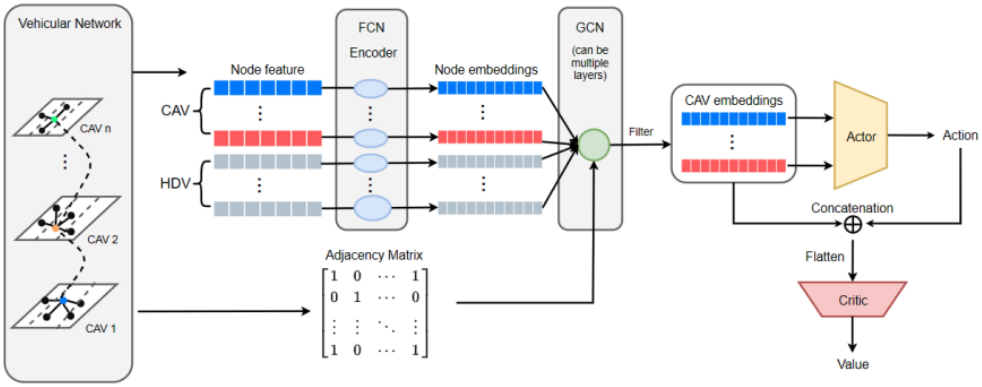


Figure 2.12: The model architecture used in (8)

visual representation of the Actor-network used in (9).

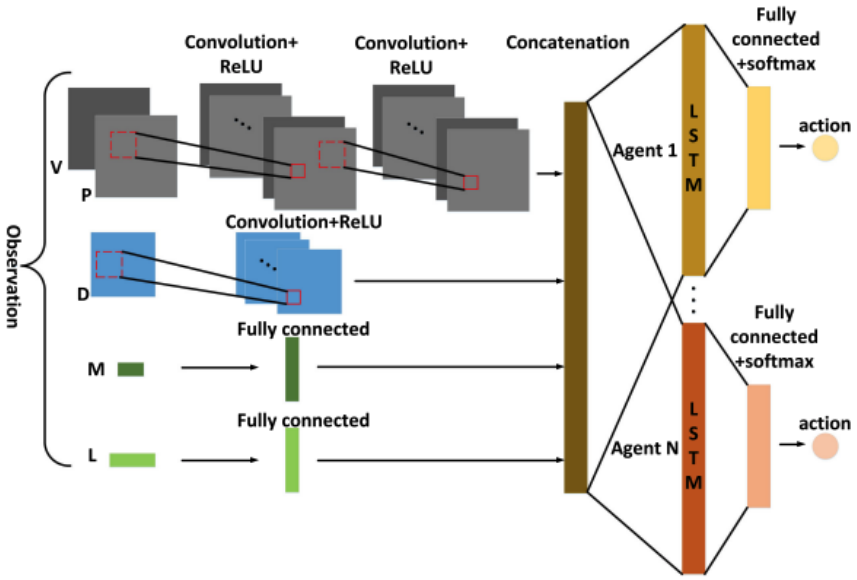


Figure 2.13: The Actor network used in (9)

Once the given action has been observed in the environment, the observation is then passed to the critic network. The critic is similar in network structure to the actor-network. However, the input to the critic network includes the global states of all intersections and the global actions of all agents. The output layer of the critic network contains one output that returns the reward value. This overall architecture can be seen in Figure 2.14.

To prevent the environment from being non-stationary, each agent utilises global states and actions during the training process so that the other agents' policy can be estimated. Therefore, it allows the agent to update its policy according to this estimate and find the global optimal policy. This will also mean that each agent in the environment will have its own actor and critic network. To ensure the algorithm converges, each critic network receives the observation and action of other agents at a given time step to calculate the Q-value during

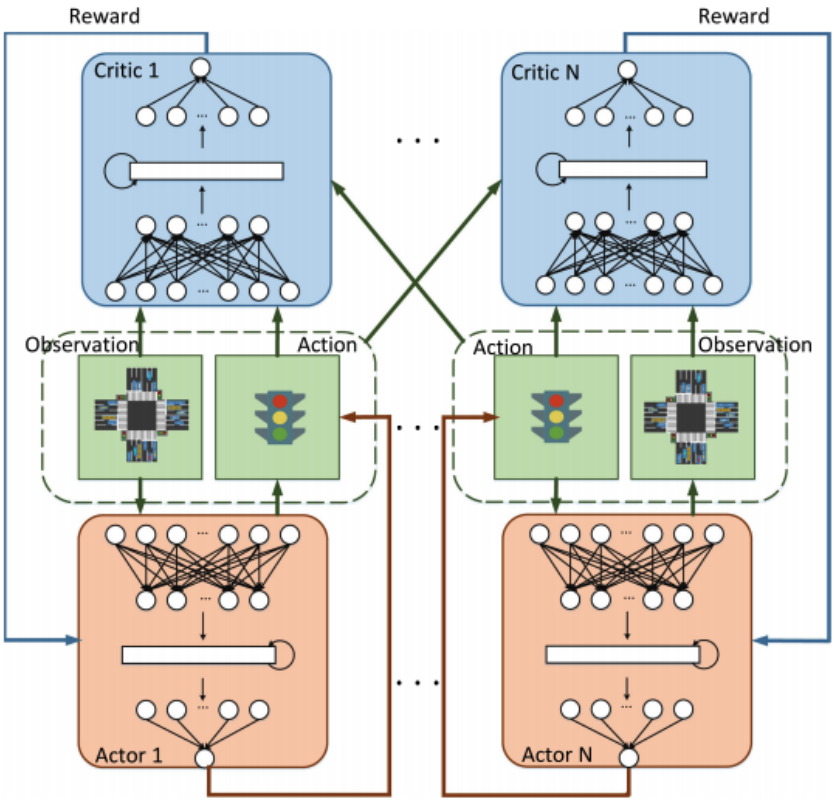


Figure 2.14: The overall architecture used in (9)

offline training. This algorithm also uses experience replay and target networks to ensure the optimal policy and Q values are found.

(71) presents a method called Cooperative Deep Reinforcement Learning. In this method, the architecture that is used resembles that of a CNN network. The input to this network is a 10x10x4 matrix consisting of the vehicles position and speed and the traffic signal phases of the current and other intersections. Also used in this paper is a network called a Deep Residual Network (ResNet). This network is used to help speed up the training process while providing good results. The network itself uses Independent Q-Learning (IQL) and experience replay to help find the optimal policy. In IQL, each agent will learn its own policy independently but will still interact with other agents as if they were part of the environment. As well as this, each agent will be able to track other agents policies. This paper discussed the issue of non-stationary and how it could be a problem. To resolve this issue, it was decided that only one agent would be trained during one episode. During this phase, the agent does not need to communicate with other agents. The other agents in the simulation would behave according to their current policies. During each training episode, only one agent will receive the award and therefore, only that agents ResNet and experience replay will get updated. In this paper, every agent will have their own ResNet and experience replay memory. After a given number of iterations, the current policy will be updated throughout the simulation, and hence a new agent is then chosen.

Table 2.4 below gives a brief summary of the papers discussed in Section 2.6.3.

Research	Algorithm	Architecture	State	Reward
(8)	Deep Deterministic Policy Gradient, target network and experience replay.	GCN, Actor and critic network, FCN encoder	node features and the adjacency matrix	throughput of vehicles
(9)	multi-agent recurrent deep deterministic policy gradient and target network and experience replay	CNN and LTSM	Vehicle position, speed, number of pedestrians, traffic light phase, queue length	Total waiting time for each vehicle on all roads, difference of travel time, The total number of vehicles that passed the intersection, the given condition of the traffic light
(71)	Independent Q-Learning with Experience Replay	CNN and ResNet	vehicles position,speed, traffic signal phases of the current and other intersections.	Cumulative delay and also considers the drivers patience

Table 2.4: Summary of the techniques used in each paper in Section 2.6.3

A variety of different state representations are used in each of the three papers. Firstly, (71) has the state space defined as the node features and adjacency matrix. Node features contain information such as vehicle velocity and position. Secondly, (9) has five state variables: vehicle position, speed, number of pedestrians, traffic light phase, and queue length. Lastly, (8) defines its state representation as the vehicles position, speed and the traffic signal phases of the current and other intersections. Another interesting comparison is that each paper in this section uses an image like structure that is passed to the network. This method was also used in Section 2.6.2, which shows that this method is the preferred method among researchers as it produces good results. Each paper also utilises techniques such as experience replay and target network to help stabilise the learning. However, it was more complicated to implement compared to Section 2.6.2. Another interesting feature of all three papers is that each agent in a given environment will have its own individual network, and during one-time step, only one agent can be trained at a time. This was done to resolve the issue of the system being non-stationary.

The final discussion in this section will be the comparison of the results and evaluations of each paper. Firstly, (8) uses a traffic simulator tool called SUMO. The system is rewarded if the throughput of vehicles is high, and it is penalised if there is a high-speed variance. In this paper, they tested their algorithm in 2 different scenarios. Firstly, it was tested in a

Moderately Congested Scenario. It showed positive results as it showed the congestion at the bottleneck was both less severe and shorter in duration. This occurred because CAVs would sacrifice there own travel time by moving to a "dropped lane" to create gaps for human-driven vehicles (HDV). Secondly, the algorithm was tested in a severely congested scenario. The results were not as promising as CAVs could no longer find great benefit in only allowing HDVs to merge out of dropped lanes as there was now too many vehicles on the road.

Secondly, (9) also uses SUMO as its traffic simulator. In this paper, five different rewards are used. These include the sum of the queue lengths, the total waiting time for each vehicle on all roads at the intersection, the difference of travel time between actual time and travelling at max speed time, the total number of vehicles that passed the intersection in a time step and the given condition of the traffic light. This paper has compared its algorithm to 4 others. These are (1) Fixed-time Traffic Light Control, (2) Self-Organizing Traffic Light Control, (3) Independent Deep Q-learning and (4) Deep Deterministic Policy Gradient. The algorithms were then tested in 2 different scenarios. In the first scenario, MARDDPG performs the best as it learns the fastest and has the highest stable reward value. In the second scenario, the proposed algorithm also performs better than the other four algorithms. However, in this scenario, the reward is smaller and takes more time to stabilise compared to scenario 1. Further tests were run in each scenario, were they compared their algorithm with a slightly different version of their algorithm. It was also tested under peak pedestrian rate and under varying penetration rates (VANET enabled cars).

Lastly, (71) like the other two papers in this section, also use the traffic simulator SUMO. In this paper, the reward is defined as the cumulative delay, and the reward also considers the driver's patience. When analysing the results, it can be seen that the algorithm proposed is compared to 3 other algorithms. These algorithms are Self-Organising Traffic Lights, Deep Q-Learning and Deep Q-Networks. In terms of rewards, the proposed algorithm converges slower than the other algorithms, but it stabilises at a higher reward value. This trend tends to mirror in the additional tests run. For example, in the average delay experiment, the proposed method converged slower than the other algorithms but converged to a more optimal value. Therefore it was concluded that the CDRL algorithm can minimise the average delay of the vehicle and reduce the probability that a number of vehicles will be sitting stationary for an extended period of time.

2.7 Summary

This chapter covered some of the basic concepts in Reinforcement Learning and Deep Learning. As well as this, it also looked at how these two techniques can be used in unison to achieve optimal performance in given problems, and this technique is known as Deep Reinforcement Learning. It also explained the current state of the art methods such as Double Deep Q Networks and techniques that are used to improve stability, performance and learning. It was then discussed how ACC, CACC and autonomous vehicles affect traffic flow and safety. At the end of this chapter, the state of the art work was presented for single and multi-agent environments. The majority of the work discussed looked at how DRL could be used to optimise traffic control. Recent articles also showed at DRL could be used in either a single or multi-agent environment to improve traffic flow and congestion. We also discussed how personalized driving decisions can be derived from observing driver behavior. In the remaining chapters we build on both these streams of work by learning driving decisions (rather than traffic light control decisions) in order to reduce the number of collisions in a network whilst maintaining or improving traffic flow and efficiency.

3 Design

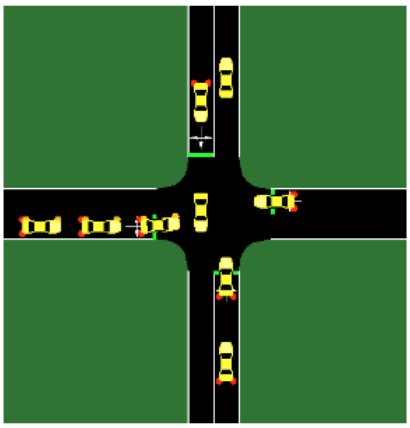
This chapter explains the approach used to model the traffic safety problem as a Deep Reinforcement Learning problem and the different elements implemented to do so. Firstly this section includes the explanation of some key terms such as state representation, action space and the reward function and how I have defined each one in the simulations. This section also explains the different algorithms used to tackle both single and multi-agent scenarios. Along with this, it also describes the Neural Network Architecture used in this approach. As mentioned in Section 2.3.5, a multi-agent environment can become non-stationary, and the design of my IDQN to prevent this from occurring is presented in this section.

3.1 Formulation of the Traffic Safety Problem

To formulate the traffic safety problem as a DRL experiment, multiple different configurations could have been used that would have lead to a positive outcome. In a DRL experiment, the agent interacts with the given environment, which consists of roads and vehicles. So in this approach, the agent was constructed to be a junction and interact with connecting roads and vehicles. The agent would have the ability to change any given characteristic of the vehicles travelling along the road at a given discrete time step. It would also have the ability to change the max speed allowed on a connecting road into or out of the junction. In a given timestep, the agent would observe a given state $s_t \in S$ and then it chooses a given action $a_t \in A$, which corresponds to either changing the characteristics of the vehicles or the road. Once this action a_t has been performed, the agent moves to the next state s_{t+1} and receives a given reward r_t depending on the previous step's outcome. The following sections describe how each of these values (a_t, s_t, r_t) are calculated.

3.1.1 State Representation

In my approach, the state representation is an image like 3d representation of the simulation at a given timestep. This 3D structure contains two 2D matrices. The first matrix is a binary representation of the location of the vehicle (position matrix (Figure 3.1b)). While the second matrix is also a 2d representation of the vehicle's velocity (velocity matrix (Figure 3.1c)) at the same given timestep. These two matrices are then stacked together to form the given 3D matrix. This approach has been popular and is used in several different academic papers such as (7), (6) and (9)



a) Simulation State Space

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

b) Position Matrix

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7.8 & 14.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10.6 & 0 & 0 \\ 1.5 & 0 & 0 & 11.6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6.8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 9.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

c) Velocity Matrix

Figure 3.1: State Representations

To create these matrices, we have to map the simulation's continuous space to a discrete environment where values can only be whole integer values corresponding to indexes in the matrix. We have to split up the junction into a square grid format where each grid will be of size S.

Each individual grid will correspond to one element in the 2D matrix. In the first matrix (position matrix), if there is a car present in the given grid position, then the corresponding position in the matrix will be assigned a value of 1. If no vehicle is present, then a value of zero will be assigned. The second matrix (velocity matrix) follows the same principle as

the position matrix. Except it will pass the speed of the vehicle if one is present and zero otherwise.

3.1.2 Action Space

As mentioned in Section 3.1, once the agent has observed a given state, s_t, it has to choose a given action from a list of possible actions, $a_t \epsilon A$. It is difficult to select an appropriate action space due to a huge number of possible actions. It was decided that two given action spaces would be used and then directly compared against one another to see the performance difference. The two chosen action spaces are described below.

Note: Each action will only alter the values of vehicles that are present in the state space.

The first set contains four different actions that the network can select. Therefore, the action space is defined as A=0, 1, 2, 3. Where 0 forces all vehicles to increase their speed by 15%. 1, forces all vehicles to decrease their speed by 5%. 2, forces all vehicles to increase their speed and acceleration by 15%. While action number 3 will force all vehicles to decrease their speed and acceleration by 5%.

The second set contains eight possible actions that the network can select. The action space is defined as A = [0, 1, 2, 3, 4, 5, 6, 7]. Where actions 0-3 are the same as defined in set 1. Action 4 forces all the vehicles to increase their speed and acceleration by 15%, but it also forces the vehicle to decrease their minGap by 10%. MinGap is defined as the minimum distance between 2 vehicles. Action 5 forces the vehicles to decrease their speed and acceleration by 5% and increase their minGap by 10%. Action 6 increases the vehicles' speed and acceleration by 20% while also increasing the MaxSpeed of the road by 10%. Lastly, Action 7 decreases the speed and acceleration by 5% while decreasing the max road speed by 10%.

Early testing showed that the second set of actions performed better in terms of the number of collisions and the average speed in the road network. See Figure 3.2 and 3.3 below. However, when using the second set of actions, the simulation on the rare occasion would fail to converge. This involved vehicles reducing their speed to 0m/s and not moving through their route. This error would only occur in the first 200 timesteps of the simulation. After this, the simulation would run smoothly. This error would occur when a high number of collisions occur quickly at the beginning of the simulation. This, in turn, causes the reward value to be negative, and the given action will be to slow the cars down. Therefore invoking a negative cycle of accident, negative reward, slow down the vehicle and so on. Hence causing the car to reduce its speed to 0m/s.

During the simulation, the agent can only select one given action at each time step t. That given action will last a one-time step until the next action is selected in the next time step.

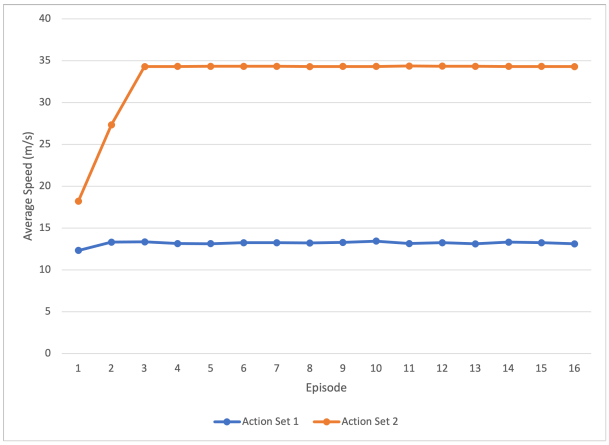


Figure 3.2: Speed difference in each Episode for the 2 different action sets

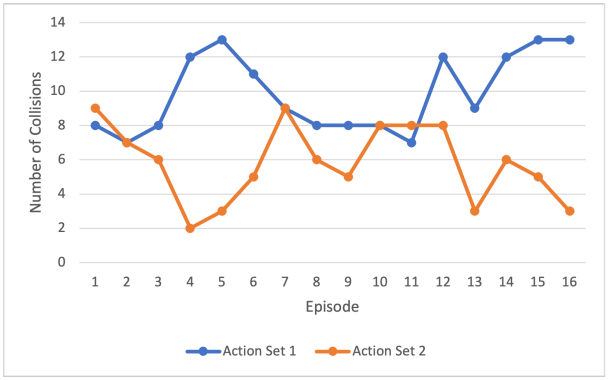


Figure 3.3: The number collision in each Episode for the 2 different action sets

3.1.3 Reward Function

In reinforcement learning problems, the reward function is one of the most challenging parts to define as it is very influential in order to find the optimal policy for the given problem. The reward is defined as a scalar value r, that is returned after an agent performs a given action in the environment.

Let r_t be the reward from the simulation at a given timestep. Let W_t and V_t represent the total cumulative waiting time and velocity respectively for all vehicles in the observation scope of the network at time t, as can be seen in Equations 1 and 2.

$$W_t = \sum_{i}^{t=i} w_i \tag{1}$$

$$V_t = \sum_{i}^{t=i} v_i \tag{2}$$

Let C_t denote the number of collisions occurring in the simulation and let E_t denote the number of vehicles performing emergency breaking in a given timestep. The reward function

intends to reward the agent when there are no accidents, low waiting times, and higher average speeds. A negative reward will be rewarded if a vehicle performs emergency braking or a collision occurs. Therefore, it is in the agents best interest to keep the accident count low whilst letting vehicles maintain a high average speed. The reward function in this approach is defined in Equation 3

$$r_t = \begin{cases} -10 * C_t + -1 * E_t & \text{if Emergency breaking or Collisions occur} \\ \frac{V_t}{t}^{1.5}/2 - W_t & \text{if waiting time} > 100 \text{ seconds otherwise} \\ \frac{V_t}{t}^{1.5}/2 & \text{otherwise} \end{cases} \tag{3}$$

3.2 Deep Neural Network Architecture

The neural network developed in this thesis will be used to train the agent in both single and multi-agent environments. The network used utilizes different techniques, concepts and structures from some of the academic papers mentioned in Chapter 2. The network will have slight modifications to suit the specific problem outlined in section 3.1. The structure of the network remains primarily unchanged, with there only being two differences. Firstly, the input size is different due to variance of the observation scope of the various papers. Secondly, the output size varies depending on the set of actions chosen.

Figure 3.4 illustrates the Deep Neural Network used in this research. The first six layers contain three layers of convolution and three layers of max pooling. The first convolution layer contains 32 4x4 filters with a stride value of 1. Max pooling is then applied with a pool size of (2, 2) and with 'valid' padding. The second convolution layer contains 64 2x2 filters with a stride value of 1. Max pooling is then applied with a pool size of (2, 2) and with 'valid' padding. The third and last convolution layer contains 128 2x2 filters with a stride value of 1. Max pooling is then applied with a pool size of (2, 2) and with 'valid' padding.

The output from this convolution layer is then flattened into a dense layer. Another fully connected layer of size 128x1 is then added to the network. A duelling network now splits into two streams of fully connected layers. Both the value and advantage stream have a fully connected layer of 24 nodes. The final hidden layer of the value and advantage stream are both fully connected where the value stream has a size of 1x1. While the advantage stream has a size of 1x(number of actions). These two streams are then concatenated together to output a given number of Q-values (number of Q-values = number of actions).

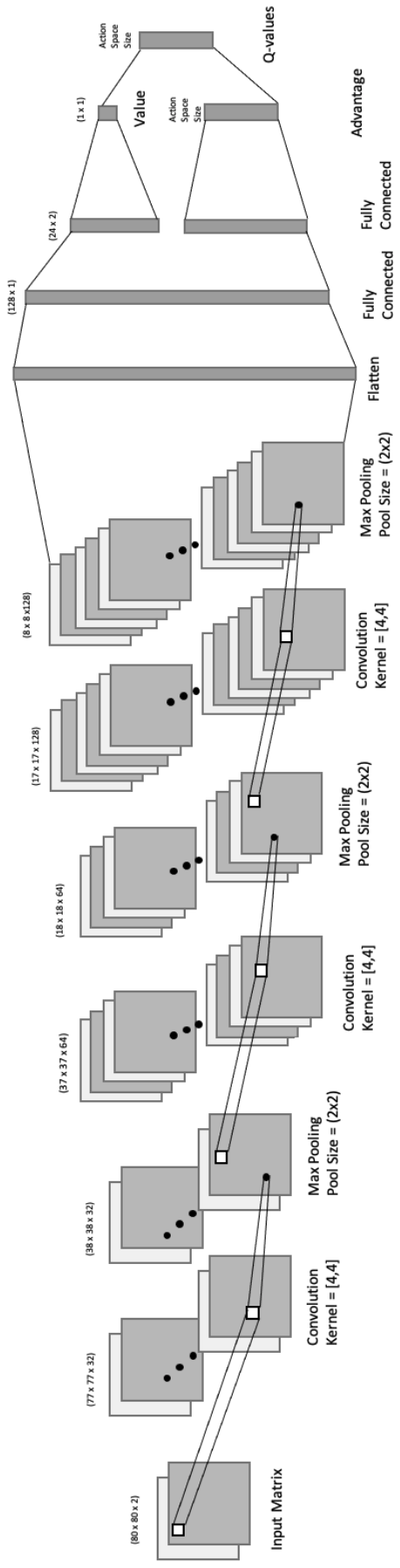


Figure 3.4: Deep Neural Network Architecture

3.3 Single-Agent Training

The agent is trained using a DDQN and utilizes techniques such as Prioritized Experience replay, duelling network and a target network. Algorithm 3 below describes the process for single-agent training. The observation state is taken from the simulation and is then passed to the Deep Neural Network, which will then calculate a given Q-value for each possible action.

Once the vehicles begin to enter the road network, a four tuple of the current state, action, reward and the next state, $< s_t, a_t, r_t, s_{t+1} >$, are saved into memory of size M. Once this memory is full, it will begin to replace all the oldest entries with new ones. Using the Q-Network and a target Network, a given loss value is calculated, and if the loss value is smaller than a loss value in the Prioritized Experience replay, then the four tuple is saved into the Prioritized Experience replay, and the replaced value is deleted. Data is sampled from the main memory in batches of size B. This is then combined with the Prioritized Experience replay. As mentioned above, a second neural network called a target network is used to help increase stability during the learning phase. The target networks parameters/weights will get updated every C steps to the same values as the main network.

When there are no vehicles in the observation scope, no learning or training are performed. In the initial stages of training the DDQN, no update or computing of the loss values are undertaken. This is to allow the experience replay to gather data into its memory buffer. The DDQN also contains a Dueling network which helps to reduce overestimation and possibly improve performance. The training algorithm used is very similar to that in Algorithm 1 and 2 in Chapter 2. Table 3.1 below shows the parameters used and their retrospective values.

Parameter	**Value**
Episodes	30
TimeSteps	48,000
Learning rate	0.001
Eploration ϵ	$1 \rightarrow 0.01$
Time steps from start ϵ to ending ϵ	4,800
Target Network update	100
Discount Factor γ	0.95
Memory size M	10,000
Batch Size B	10
Priority Memory Size P	5

Table 3.1: DRL Hyper-Parameters

Algorithm 3: DDQN for Single Agent Training

Input: memory buffer size M, Priority Memory P, minibatch size B,
greedy ϵ, learning rate α, learning frequency γ,
times steps T, target network update C.
Initialize simulator with time steps T
Initialize optimizer with learning rate α
Initialize action-value function Q with random weights
Initialize replay memory
Initialize target action-value function
while *there are cars present in the simulation* **do**
    With probability ϵ select
    a random action a_t otherwise select $\max_a Q(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in Memory
    if *Memory entries > minibatch size* **then**
        Sample minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from memory
        based on priorities
        Minimize the error in Bellman's equation and compute Loss
        Update the priorities in the replay buffer from the Loss
        Update networks weights
    else
        Do nothing
    end
    Every C timesteps Update Target Network
    Update $s_t = s_{t+1}$
end

Figure 3.5: Algorithm 3: DDQN for Single-Agent Training

3.4 Multi-Agent Design

Section 3.3 describes the process of how DQN and other techniques such as DDQN, Dueling network, Prioritized Experience replay and target network can be applied in a single agent environment. This section describes how these techniques can be extended to a multi-agent environment. For example, the Deep Q-Network is extended to an Independent Deep Q-Network as described in Chapter 2. The extended design is as follows:

State Space:

The State-space remains unchanged from Section 3.1.1. Chapter 2 concluded that training agents in the multi-agent scenario could cause the simulation to become non-stationary. To ensure this problem does not occur, the state spaces have been designed not to overlap and therefore removing the possibility of non-stationary. Figure 3.6 below shows the observations scope for each agent in this simulation. Where each agent can only see and interact with road vehicles inside its own red box.

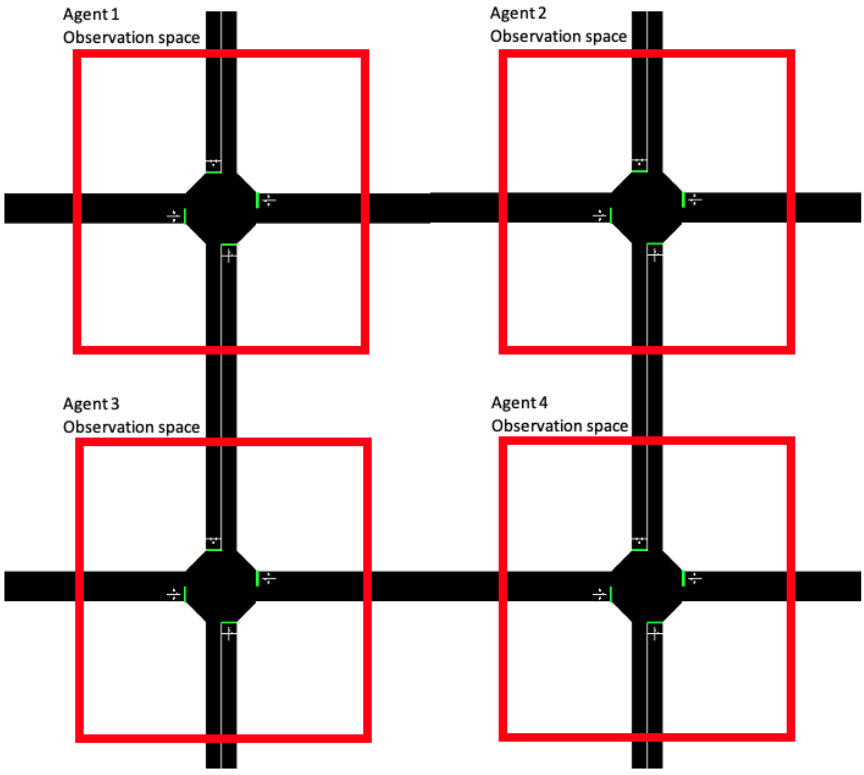


Figure 3.6: State Space for the Multi-agent simulation

Action Space:

The action space is still defined the same as in Section 3.1.2. However, different agents can perform different actions as they are acting independently of each other.

Reward Function:

The individual agent's reward function does not change. However, the global reward function is added. The global reward function is the average of the individual agent's rewards. The aim will be to keep this global reward as high as possible.

Neural Network Architecture:

Each agent will use the same architecture outlined in Section 3.2. Each agent will have its own Deep Neural Network. This so that the agent can learn its own optimal policy due to the definition of an Independent Deep Q-Network, where each agent is trained independently of each other.

3.4.1 Modularity and Scability

One thing that needs to be mentioned is the modularity and scalability of the approach outlined above. This thesis only considers a network consisting of 4 junctions, each one with its own independent network. This approach's independent nature allows it to be easily scalable to large city road networks, with the only drawback being the amount of computing power needed to train the neural network.

In real-world scenarios, junctions or road intersections will not always appear as they do in Figure 3.6. In Figure 3.6 the road network is carefully constructed so that overlapping of the observation space does not occur and cause the simulation to become non-stationary. However, it can be seen from Figure 3.7 that in the real world, junctions can be constructed very close together, and it could then be challenging to define an observation space for both junctions where the observation space does not overlap. To get around this issue, there are two possible solutions. Firstly, each junction could be awarded its own observation space and hence overlapping would occur. To prevent non-stationary becoming an issue, the junction or intersections that are overlapping would train their networks in alternative timesteps. Secondly, a comparison of the two networks could be completed based on the junction's importance, the number of collisions, average speed of speed vehicles, etc. Through this comparison, the appropriate junction would be selected and trained while the second junction would be left untrained. However, dealing with this issue is out of scope for this thesis and would be addressed in Future Work.

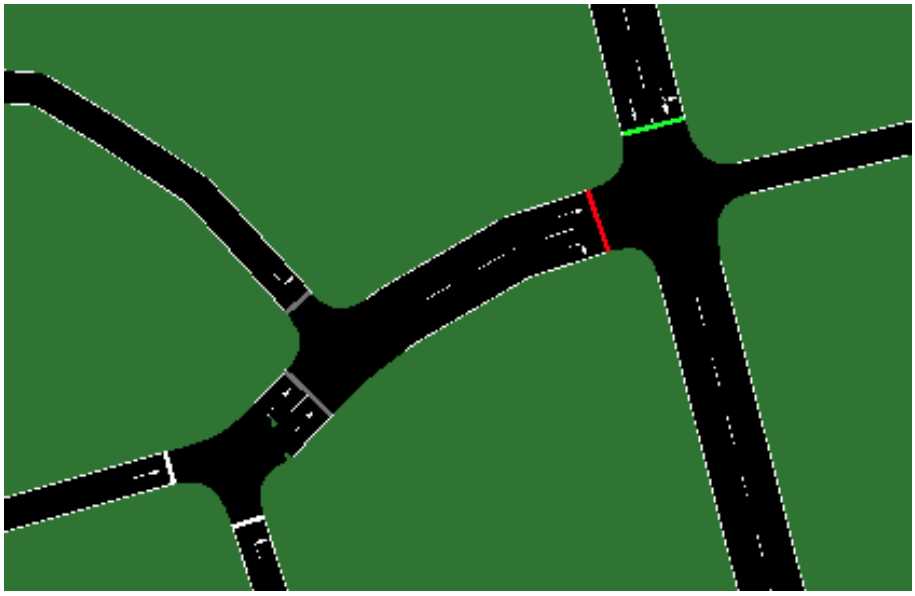


Figure 3.7: A set of junctions where overlapping of observation space would occur

3.4.2 Training

Training for the multi-agent environment can be seen in Algorithm 4, it is clear to see that the multi-agent Algorithm is very similar to that of the single agent. The only difference being extra loops to allow for training of all agents as each agent will have its own neural network. Each network gets trained in every timestep in synchronous time because the agents share the same simulation environment.

3.5 Summary

This section has outlined some of the critical components that are being used to help improve traffic safety in a multi-junction scenarios. It defined key terms such as state representation, action space and the reward function. Additionally, it also described the neural network architecture used to solve this problem. The Algorithm used to train the given network in both the single and multi-agent environments were illustrated. Lastly the modularity and scability of the design was also discussed in this chapter. The implementation of the given design is presented in Chapter 4.

Algorithm 4: IDQN for Multi-Agent Training

Input: Memory buffer size M, Priority Memory P, minibatch size B, greedy ϵ, learning rate α, learning frequency γ,
times steps T, target network update C.
Initialize simulator with time steps T
Initialize all Agents A
Initialize optimizer with learning rate α
Initialize action-value function Q with random weights
Initialize replay memory
Initialize target action-value function
while *there are cars present in the simulation* **do**
    for *every agent A* **do**
        With probability ϵ select
        a random action a_t otherwise select $\max_a Q(s_t, a; \theta)$
        Execute action a_t in emulator
    end
    Retrieve Global Reward
    for *every agent A* **do**
        Observe next state s_{t+1}
        Store transition (s_t, a_t, r_t, s_{t+1}) in Memory
        if *Memory entries > minibatch size* **then**
            Sample minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from memory
            based on priorities
            Minimize the error in Bellman's equation and compute Loss
            Update the priorities in the replay buffer with Loss
            Update networks weights
        else
            Do nothing
        end
    end
    Every C timesteps Update Target Network
    Update $s_t = s_{t+1}$
end

Figure 3.8: Algorithm 4: DDQN for Multi-Agent Training

4 Implementation

This Chapter describes the implementation of the single and multi-agent Deep Reinforcement Learning systems. The simulation environment used to model the traffic network and how the agent communicates with the road network in real-time are also detailed in this Chapter.

4.1 Simulation Environment

The simulation environment used to model a real-world traffic network is called SUMO. SUMO (Simulation of Urban MObility) is a free, open-source, microscopic and continuous road traffic simulation software that is designed to handle large road and traffic networks (72). In this work, SUMO version 1.8.0 is used. One of SUMO's most valuable features is called TraCI. TraCI is a python API and allows the user to interact with the given network. Using this API allows access to components such as Vehicles, traffic lights and roads. The given agent can modify and monitor different properties of the given vehicles and roads contained in its observation space depending on the action taking. The API also allows us to recover critical metrics such as the location and speed of each vehicle to build the position and velocity matrices. SUMO has the ability to implement pedestrians and numerous different vehicle types such as bicycles, trucks, SUVs, etc. For the purpose of this project, a single-vehicle type will be used. The vehicles are modelled to act like that of an average car and be driven in a way similar to that of a human driver. By default, SUMO tries to prevent collision as much as possible as it is designed to monitor traffic flow/congestion. This, therefore, made it very difficult to cause collisions in open traffic. Therefore to ensure that collisions could be created, a junction with traffic lights had been created. However, the lights for every direction will show the colour green. This ensured that collisions and emergency braking would take place inside the simulation.

4.2 Deep Reinforcement Learning

As mentioned before, the Deep Reinforcement Learning algorithms used in this thesis, use techniques such as Double Deep Q-Network, Priority Experience Replay, target network and

a duelling network. To implement the given design from Chapter 3, Python 3.7 was used. To implement the neural network, powerful AI libraries such as Tensorflow[1] version 2.4.1 and Keras[2] version 2.3.1 were also used. The methodology and implementation of DQN and other techniques are roughly mirrored to those mentioned in published papers in Chapter 2. Figure 4.1 below shows the classes that have been created for this DRL problem. An open-source platform called Open AI baseline could have been used to implement a DQN agent. However, building the network and system from scratch allowed for more control over the design of the whole system. [1] [2]

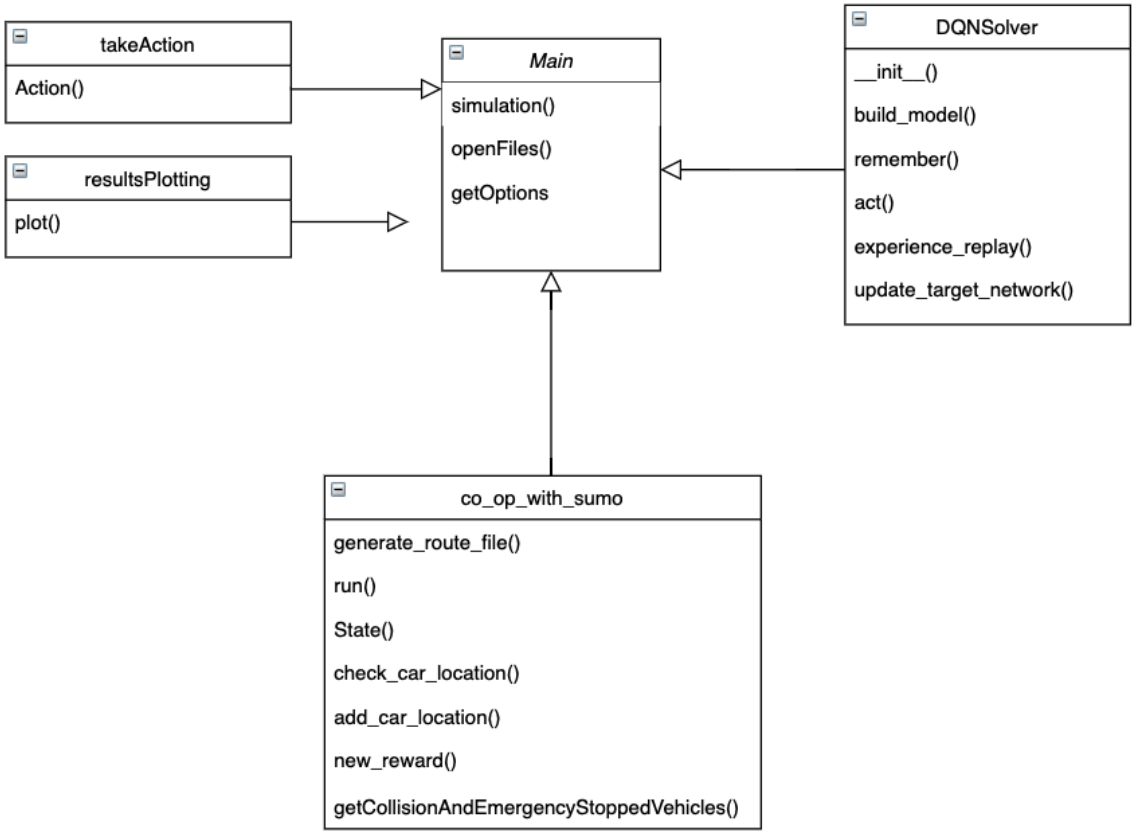


Figure 4.1: Class Diagram for Single Agent DRL Traffic Problem

4.2.1 Co_op_with_sumo

Co_op_with_sumo is the given class that directly interacts with SUMO and SUMO's Python API TraCI to recover needed parameters for other classes in the system. The following methods implemented in Co_op_with_sumo are as follows:

- ***generate_route_file(self):*** generates the traffic for a given road network.

[1] https://www.tensorflow.org/
[2] https://keras.io/

- ***run(self, crash_location_list):*** This is the main method of the file where all the methods mentioned below are called from.
- ***State():*** Subscribes to all the vehicles in the road network at the beginning of every timestep. Allows access to given parameters, as such speed and location.
- ***check_car_location(crash_location, location):***Checks to see if the given vehicle is within the observation scope of the given agent
- ***add_car_location(self, crash_location, location, state matrix, speed):***If the vehicle is within the observation scope of the agent, then this method will add the vehicle to the input matrix
- ***new_reward():*** If no collisions or emergency braking are occurring, then this method calculates the reward based on the waiting time and velocity of the vehicles in the observation scope.
- ***getCollisionAndEmergencyStoppedVehicles():***Returns the vehicles that have been involved either in an accident or have preformed emergency breaking.

4.2.2 Main

The main purpose of this class is to load the SUMO-gui and create the simulation environment. This class is considered the main class of the system, and other classes are called from here. The main class contains the following methods:

- ***simulation():*** Main method in this class contains a while loop that allows the simulation to run through every timestep until there is no more traffic left in the network. This method will also call classes that allow for DQN set up, DQN training and the output of results.
- ***openFiles():*** Opens given files at the beginning of the simulation to allow for result calculations
- ***getOptions():*** Allows for a different set up of the SUMO-gui

4.2.3 DQNSolver

The DQNSlover class contains the given methods that allow for the implementation of a DQN Network, Priority Experience Replay, Dueling Network and target Network. The following methods that are implemented in DQNSolver are given below:

- ***__init__():*** Initialises given parameters for the network
- ***build_model():*** Builds the given network architecture as described in section 3.2 using Tensorflow and keras

- ***remember(self, state, action, reward, next_state):*** Stores a given transition tuple $< s_t, a_t, r_t, s_{t+1} >$ in memory
- ***act(self, state):*** Feeds the current state into the network and returns the given action that the agent should take.
- ***experience_replay(self):*** Trains the given network using memory and priority experience replay memory.
- ***update_target_network(self):***Updates the targets networks weight and parameters to that of the original network

4.2.4 takeAction

takeAction is the only class in the system that has the ability to change parameters inside the simulation directly. The takeAction class only contains one method called action (action, vehicle_list). It takes in the given action and then changes the corresponding parameters of the road and the vehicles in the vehicle_list.

4.2.5 resultsPlotting

resultsPlotting is a simple class that is used to help collect data and then output the results in a graphical format.

4.3 Neural Network Architecture

The Neural network architecture was build using both Tensorflow 2.4.1 and Keras 2.4.3. Figure 4.2 below shows the neural network graph in Tensorflow. It is clear from the image that there are three layers of convolution and max pooling, followed by flattening. Once it has been flattened, it is then passed to 3 dense before being passed to the Lambda layer. This lambda layer is configured to be the Dueling network. The target network is designed to have the same architecture as this.

Finally, Figure A1.1(the figure was placed in the appendix section to show the image as large as possible) describes the whole Neural Network infrastructure. The fact that the model was built using a mixture of Tensorflow and Keras makes this diagram very hard to read and understand. However, the model is still visible in the middle of the image.

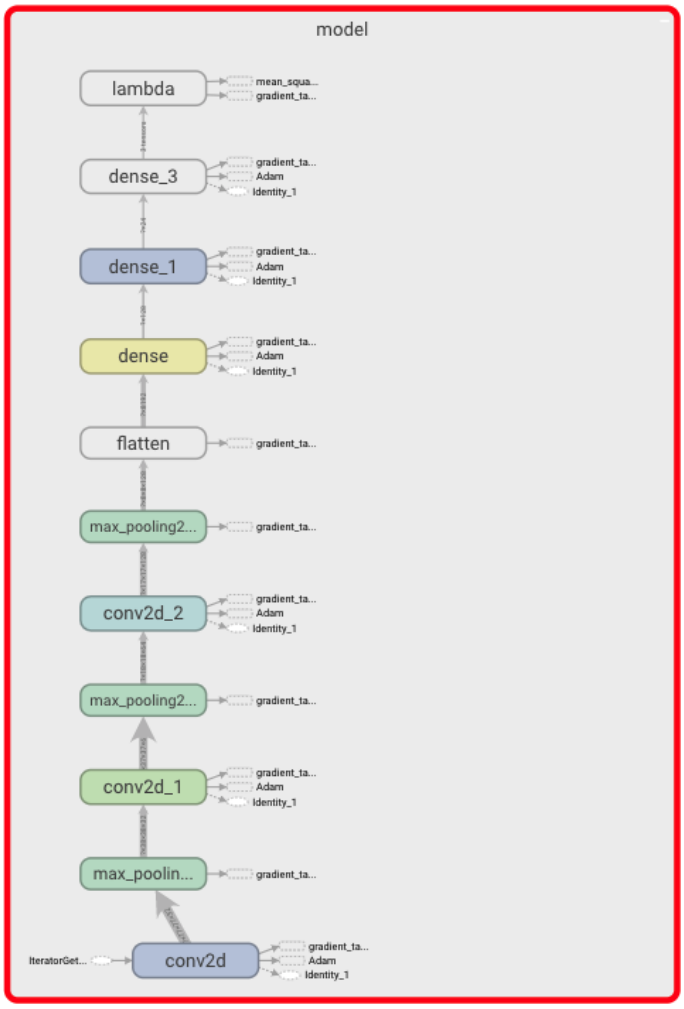


Figure 4.2: The DQN network built using Tensorflow

4.4 Multi-agent Deep Reinforcement Learning

Figure 4.3 below illustrates the classes that have been implemented for multi-agent deep reinforcement learning. From this figure, it can be seen that it is the same as single-agent deep reinforcement learning. The difference between single and multi-agent implementation was on a method level rather than a class level. The changes to the classes can be shown below.

4.4.1 Main

To incorporate multi-agent deep reinforcement learning, the main class was the only class that needed to be altered. In this class, only the method titled simulation was altered. In this method, further instantiations of the DQN were introduced for each of the new junctions in the road network.

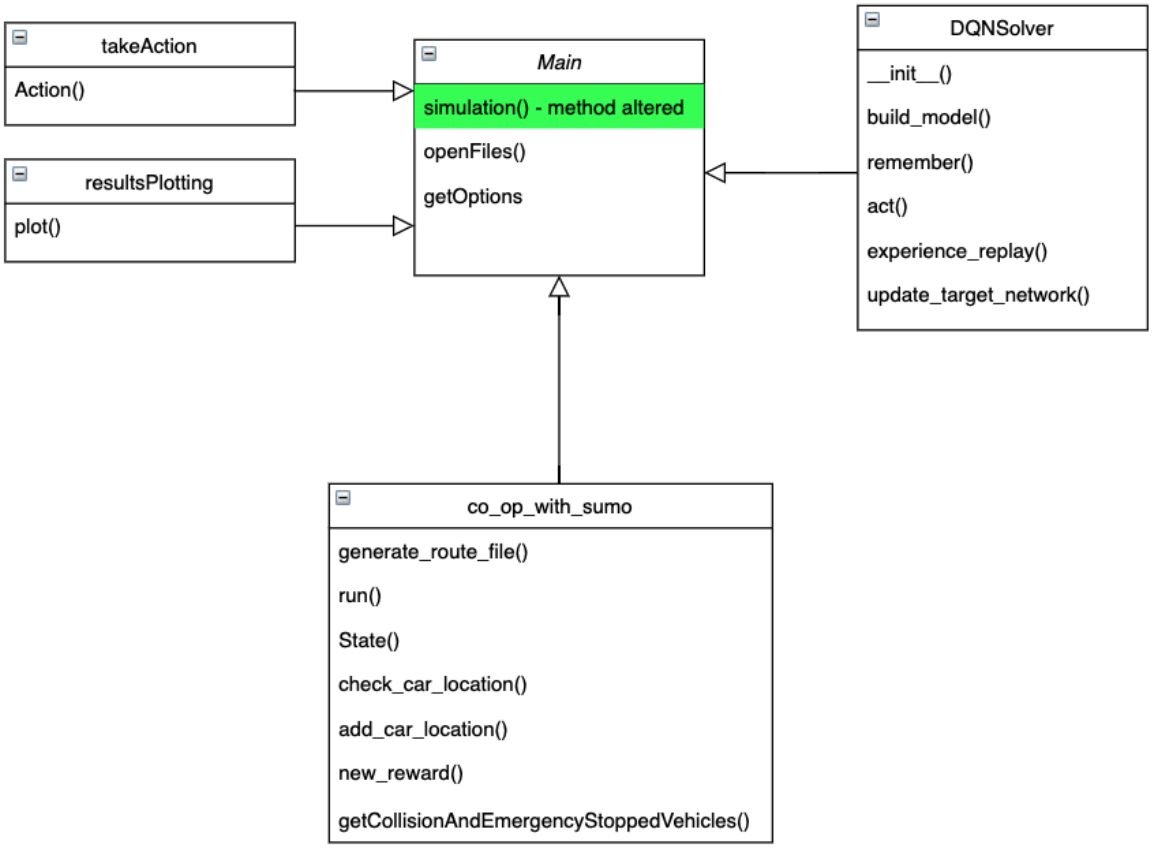


Figure 4.3: Class Diagram for Multi-Agent DRL Traffic Problem

4.5 IDQN Structure

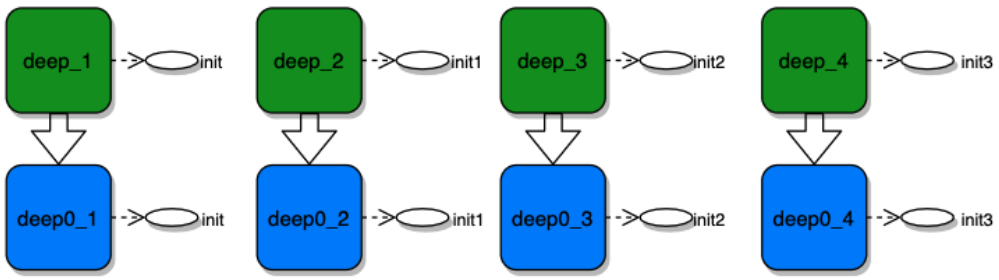


Figure 4.4: Independent Deep Q Networks for 4 agents built using Tensorflow and Keras

Figure 4.4 above shows the overall structure for IDQN agents. Where deep0_1 and deep_1 are the neural networks for the 1st intersection(original and target network). The same naming convention applies to the remaining three junctions. Each agent has its own neural network as described in section 3.4. As described in Chapter 2, the use of a fingerprint was not implemented because the agent's observation state did not overlap. However, a fingerprint may need to be considered in future work if the observation space does indeed overlap.

4.6 Difficulties

When implementing the given design, two issues arose. The first one was in reference to the amount of RAM that was available for the system to utilise. When running the simulations, the experience replay and priority experience replay would store their "experiences" in RAM. Throughout the experiment, the RAM gradually became full. When this RAM had been filled, by default, it should have removed the oldest entries and replace them with the newer ones. However, this default feature did not happen, and it continuously threw an error. To get around this issue, both the experience replay and priority experience replay would be reset to empty after 10,000 steps in the simulation. This fixed the reoccurring error.

The second issue occurring was in regards to vehicles randomly stopping during simulations. This error would occur randomly throughout the experiment. To ensure that this bug would not occur, a few safety measures were put in place. Firstly, at the beginning of the simulation, the action set has a bias towards increasing vehicle speed and acceleration. This was done to prevent vehicles from taking a negative action and slowing their speed. Secondly, in the SUMO set up file, there is a particular value called time-to-teleport that can be set. This value was set to 100. If a vehicle's speed is below 0.1m/s for 100 timesteps in the simulation, it is then teleported to a different position in the simulation with the control on the vehicle handed back to SUMO. With these safety measures, the probability of vehicles randomly stopping was significantly reduced.

4.7 Summary

In this Chapter, we firstly presented implementation for the single-agent DRL. The overall class and method implementation were also described. The neural network architecture was presented in a visual format using Tensorflow/Tensorboard. The implementation of IDQN and the changes that had to be made to the individual files to ensure IQDN was set up correctly was also discussed. Lastly, the difficulties faced while implementing the given solution were also presented in this Chapter.

5 Evaluation

In this Chapter, we firstly present an evaluation of the DQN as a solution for single-agent DRL agents in the Traffic safety problem. Secondly, an evaluation of the IDQN as a solution for multi-agent DRL agents in the Traffic safety problem is also presented. This Chapter also discusses the objectives of the evaluation, along with the given parameters needed to conduct this experiment. It also describes the different metrics that will be used to measure the performance of each system. We also describe the simulations that were used and present and analyse their outcomes.

5.1 Objectives

The goal of the DQN presented in this Chapter is to establish if it is possible to improve traffic safety whilst maintaining traffic flow and efficiency in a single agent scenario. Upon completion, an evaluation of IDQN in a multi-agent scenario would then be undertaken. The overall objectives of these experiments was to find if it is possible to use DRL to increase Traffic safety whilst maintaining or improving traffic flow and efficiency. Chapters 3 and 4 in this report highlight the design and implementation of the given DQN and IDQN that will be used to address this problem. DQN and IDQN can be said to have in succeed in addressing the Traffic Safety problem if it satisfies the following performance requirements:

- The number of accidents has been reduced compared to that of a baseline simulation
- The average velocity of the vehicles in the simulation has either increased or remained the same compared to the value of that in the baseline simulation
- The average waiting time of the vehicles in the simulation has either decreased or remained the same compared to the value of that in the baseline simulation

5.2 Metrics

Due to the unique nature of this project, there were no previous studies to show which metrics should be used. Hence there was a total of 4 metrics used. Only 2 metrics were used for the single agent simulations while all four were used for multi-agent simulations. Every metric is

recorded at every timestep. An episode contains 1600 timesteps, and each time step is one simulated second in SUMO. The metrics used in these experiments are as follows:

- **Cumulative Reward.** This is the most common metric for DRL experiments as there is a direct correlation between the performance of the algorithm and how much reward the agent receives. Throughout the simulation, the agent should aim to collect as much reward as possible. A bigger number means better performance. The reward r_t is taken on every timestep t and is accumulated over the entire episode. This cumulation of the reward is shown in the results. (For Multi-agent scenario only)
- **Number of Collisions and Vehicles Preforming Emergency Braking.** This metric helps measures the effectiveness of the reward function into the primary goal of this project, which is to increase traffic safety. At every timestep t, the simulation will check if either a collision has occurred or if a vehicle has performed emergency braking. At the end of each episode, the sum of this will be calculated and then reset for the next episode. This metric is part of the reward function. Hence this metric is correlated with the rewards of every episode.
- **Average Speed.**This metric also helps to measure the effectiveness of the reward function into the main goal of this project, which is to either maintain or improve traffic flow and efficiency. During each timestep, the average speed of all the vehicles in the agent's observation scope will be calculated. At the end of each episode, the sum of all timesteps will be divided by 1600 to get the average speed for the episode. This metric is part of the reward function. Hence this metric is correlated with the rewards of every episode
- **Average Waiting Time.**This metric also helps to measure the effectiveness of the reward function into the main goal of this project, which is to either maintain or improve traffic flow and efficiency. During each timestep, the average waiting of all the vehicles in the agent's observation scope will be calculated. At the end of each episode, the sum of all timesteps will be divided by 1600 to get the average waiting time for the episode. This metric is part of the reward function. Hence this metric is correlated with the rewards of every episode. (For Multi-agent scenario only)

5.3 Evaluation Scenarios

This section describes the scenarios that we used to evaluate the suitability of DQN and IDQN for the traffic Safety problem. We firstly present the different techniques that will be used to test the DQN in the single-agent environment. Then, we present the techniques that will be used to test the IDQN for the multi-agent environment. Lastly, we describe the different scenarios under which these techniques are going to run.

5.3.1 Evaluation Techniques for Single-Agent

- **Observation Space.** A range of different observation sizes will be used to ensure that the most suitable size is selected. In total, there will be three different observation sizes selected. These include 400mx400m (Large Observation Space (LOS)), 300mx300m (Medium Observation Space (MOS))and 250mx250m (Small Observation Space (SOS)). The dimensions are in reference to the size of the red box (Figure 3.1) in the simulation. The centre of the observation space will match the centre of the given junction.
- **Batch Size and Priority Memory Size.** Both the batch size and Priority Memory size will have two different sets of values for the single-agent experiment. Firstly, the batch size will have a value of 10, and the Priority Memory size will be 5 (this will be defined as reduced memory or RM for shorthand). In the second set, the batch size will have a value of 20, and the Priority Memory size will be 10 (this will be defined as increased memory or IM for shorthand).

5.3.2 Evaluation Techniques for Multi-Agent

- **Observation Space.** From the results of single-agent DQN training, the observation space with the best results are taken and applied to the Multi-agent environment.
- **Batch Size and Priority Memory Size.** From the results of single-agent DQN training, the Batch Size and Priority Memory Size with the best results are taken and applied to the Multi-agent environment.

5.3.3 Scenarios

All the evaluation techniques will be tested using the scenarios described below. The scenarios will be the same for both the single-agent DQN training and the multi-agent IDQN training. The different scenarios are:

- **Action Space.** As mentioned in Section 3.1.2, there are 2 different action spaces. Hence simulations will be conducted on both of these action spaces.
- **Low Traffic Demand.** The Low traffic demand scenario is where the number of vehicles in the simulation is reduced. It represents normal traffic conditions and flow when it is not "rush hour" traffic. However, there is still a suitable amount of traffic that collisions and emergency braking can occur if traffic safety is not upheld.
- **High Traffic demand.** The High traffic demand scenario is where the number of vehicles in the simulation is overwhelming. It represents the traffic conditions during peak times throughout the day. This scenario will certainly produce numerous collisions and emergency braking, even with good traffic safety. In this scenario, if too many

collisions or emergency braking occur, will it have a detrimental effect on traffic flow and efficiency. The idea is to verify how the algorithm performs in overcrowded scenarios

Every technique will use the given parameters and Network layout as described in section 5.4

5.4 Setup

This section will describe the environment in which the single-agent and multi-agent scenarios are run, and the different parameters and configurations were chosen. Firstly, we introduce the network layout built using a SUMO package called NetEdit. We then present the traffic demand generation and the parameters used for this DRL network.

5.4.1 Network Layout

Single-Agent

Figure 5.1 shows the network layout for the single-agent scenario. The road can be seen to contain only one intersection. Since this scenario only has one intersection, it allowed for different parameters of the DRL network to be altered whilst retaining results relatively quickly compared to multi-agent scenarios.

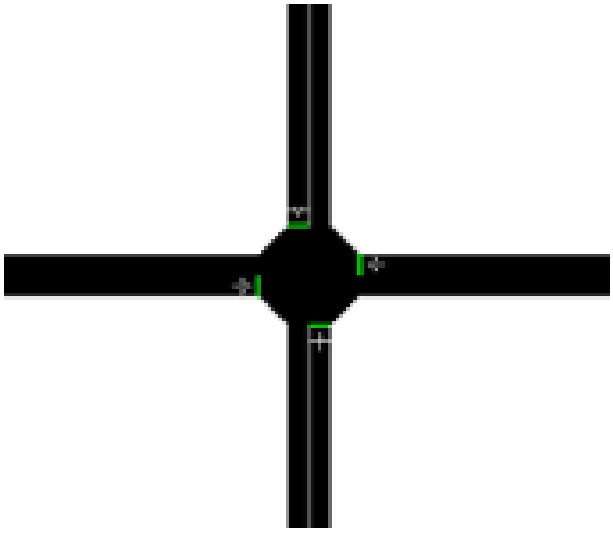


Figure 5.1: Network Layout for Single-Agent Simulations

Multi-Agent

Figure 5.2 below shows the given road network for the multi-agent scenario. The road network contains four intersections. We extended to road intersection to 4 due to the computational resources available. More junctions would help us retrieve more in-depth results. However, this is outside the scope of this project and is discussed in Chapter 6 under future work. Due

to the use of experience and priority experience replay, it was customary for agents to take up several Gigabytes of RAM. This was mainly caused by the storing of the state representation. To maintain the selected value from single-agent training results, the RAM had to be cleared every 10,000 timesteps. However, this does not have a detrimental effect on the results, as shown in Section 5.5.

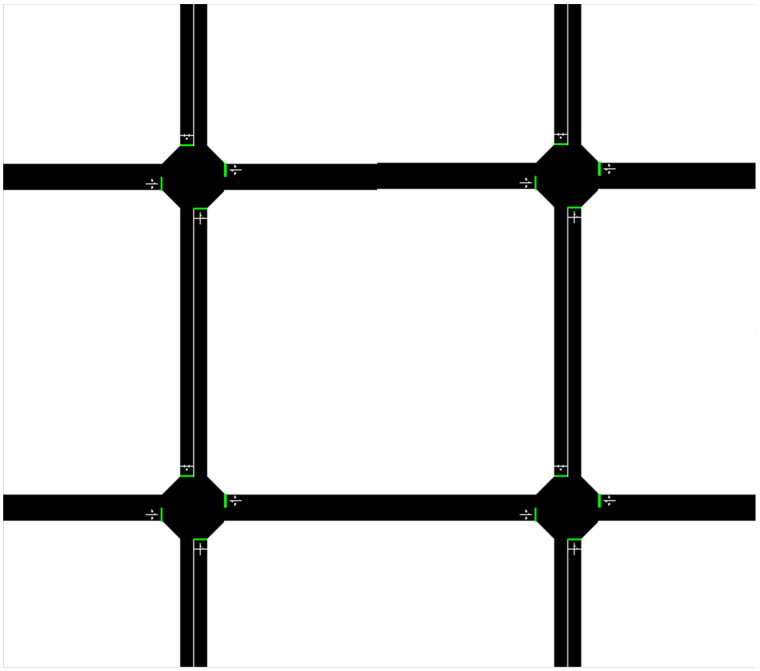


Figure 5.2: Network Layout for Multi-Agent Simulations

5.4.2 Traffic Demand

The traffic demand for single and multi-agent scenarios are slightly different in terms of the number of vehicles entering and leaving the simulation. In single-agent training under low levels of traffic, a total of 640 will pass through the junction in one episode. In high traffic demand, a total of 800 will pass through the junction in every episode. For the multi-agent scenario, a total of 880 vehicles pass through the simulation per episode in low traffic conditions. While in high traffic demand, a total of 1100 pass through the simulation every episode. Throughout the simulation, each episode will contain the same amount of vehicles. This is to ensure a good and fair comparison can be made between the different simulations. Table 5.1 below summaries traffic volumes.

	Low Traffic (Episode)	**High Traffic (Episode)**	**Low Traffic (Overall)**	**High Traffic (Overall)**
Single Agent	640	800	19,200	24,000
Multi-Agent	880	1100	26,400	33,000

Table 5.1: Traffic Volumes in Low/High traffic demand in Single and Multi-agent Scenarios

In both single-agent and multi-agent scenarios, the same vehicle type with the same properties

is used. These properties are listed in Table 5.2. Sigma is defined as the driver's imperfection, and the minGap is defined as the empty space after the leader. All other properties hold their usual meanings.

Vehicle Property	**Value**
Vehicle Type	Car
Acceleration	$1.8ms^{-2}$
Deceleration	$1.5ms^{-2}$
Sigma	0.5
Length of the Car	5m
minGap	2.5m
Max Speed	$35ms^{-1}$

Table 5.2: Vehicle Properties in Single and Multi-agent Scenarios

Single-Agent

Figure 5.3 shows the entry and departure points for the single-agent simulations. In SUMO, every route is defined by specifying a source, destination and the intersections of which the vehicle travels through. Therefore a given route in the single junction scenario could be defined as 1-2-4. The routes created for our evaluation are the following:

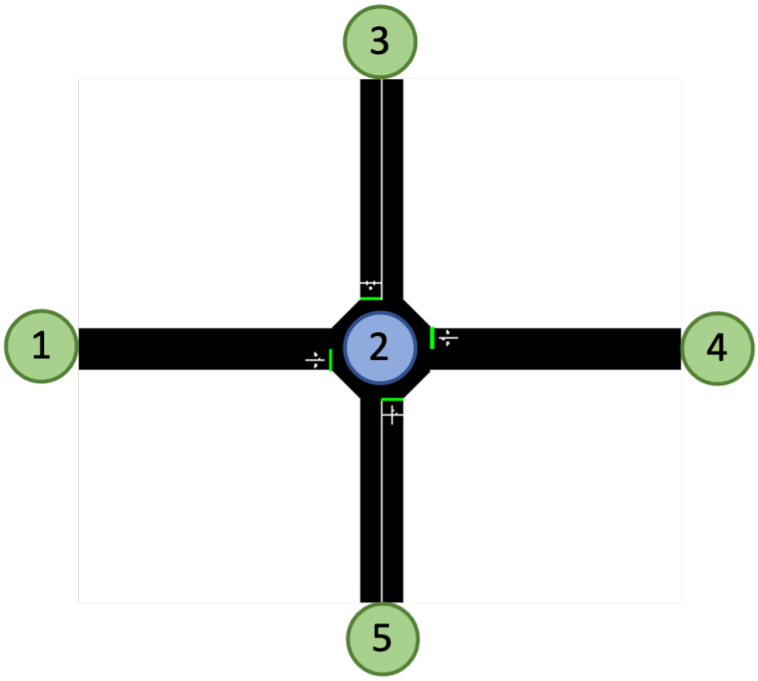


Figure 5.3: Entry and Departure points in the road Network for Single-Agent Experiments. Green points correspond to entry and departure points, while blue points are intersections

- $1 \rightarrow 2 \rightarrow 4$
- $3 \rightarrow 2 \rightarrow 5$

- $4 \rightarrow 2 \rightarrow 1$
- $5 \rightarrow 2 \rightarrow 3$

Multi-Agent

Figure 5.4 shows the entry and departure points for the multi-agent simulations. Therefore a simple route in the multi-junction scenario could be defined as 1-2-3, or a more complicated route could be defined as 4-6-12-11-10. The routes created for our evaluation are the following:

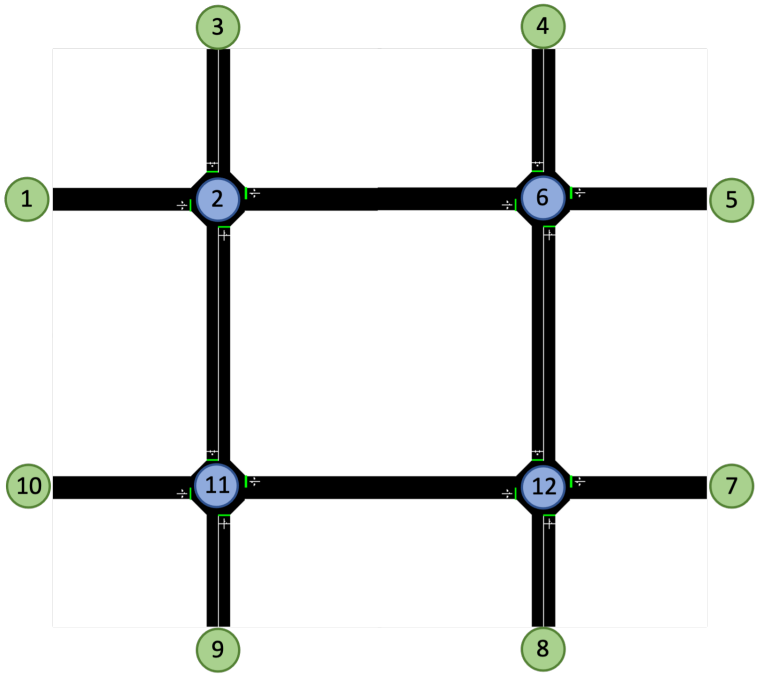


Figure 5.4: Entry and Departure points in the road Network for Multi-Agent Experiments. Green points correspond to entry and departure points, while blue points are intersections

- $1 \rightarrow 2 \rightarrow 6 \rightarrow 5$
- $1 \rightarrow 2 \rightarrow 6 \rightarrow 4$
- $7 \rightarrow 12 \rightarrow 11 \rightarrow 10$
- $8 \rightarrow 12 \rightarrow 6 \rightarrow 4$
- $8 \rightarrow 12 \rightarrow 6 \rightarrow 5$
- $9 \rightarrow 11 \rightarrow 2 \rightarrow 3$
- $10 \rightarrow 11 \rightarrow 9$
- $9 \rightarrow 11 \rightarrow 2 \rightarrow 3$
- $3 \rightarrow 2 \rightarrow 11 \rightarrow 9$

- $4 \rightarrow 6 \rightarrow 12 \rightarrow 8$
- $5 \rightarrow 6 \rightarrow 2 \rightarrow 1$

5.4.3 Hyper-parameters

Table 5.3 and 5.4 shows the different hyperparameters used in the single-agent and multi-agent experiments. By default, Priority experience replay should have its own hyperparameters as outlined in Chapter 2. However, due to the fact that I am implementing my own type of Priority experience replay, it does not need these parameters. Every episode roughly corresponds to 27 minutes of simulation time or 1600 time steps, where every timestep is one simulated second.

Parameter	**Value**
Episodes	30
TimeSteps	48,000
Learning rate	0.001
Eploration ϵ	$1 \rightarrow 0.01$
Time steps from start ϵ to ending ϵ	4,800
Target Network update	100
Discount Factor γ	0.95
Memory size M	10,000
Batch Size B	10/20
Priority Memory Size P	10/5

Table 5.3: DRL Hyper-Parameters for Single-Agent experiments

Parameter	**Value**
Episodes	30
TimeSteps	48,000
Learning rate	0.001
Eploration ϵ	$1 \rightarrow 0.01$
Time steps from start ϵ to ending ϵ	4,800
Target Network update	100
Discount Factor γ	0.95
Memory size M	10,000
Batch Size B	10
Priority Memory Size P	5

Table 5.4: DRL Hyper-Parameters for Multi-Agent experiments

5.5 Results and Analysis

In this section, we analyse the performance of both the DQN in single-agent scenarios and the IDQN in multi-agent scenarios with respect to the objectives outlined in section 5.1. We evaluate the efficiency of DQN and IDQN by comparing all the different techniques mentioned in section 5.3.1 and 5.3.2 in three scenarios. We also describe, analyse and discuss the given results and how they relate to the evaluation objectives.

5.5.1 Low Traffic Demand

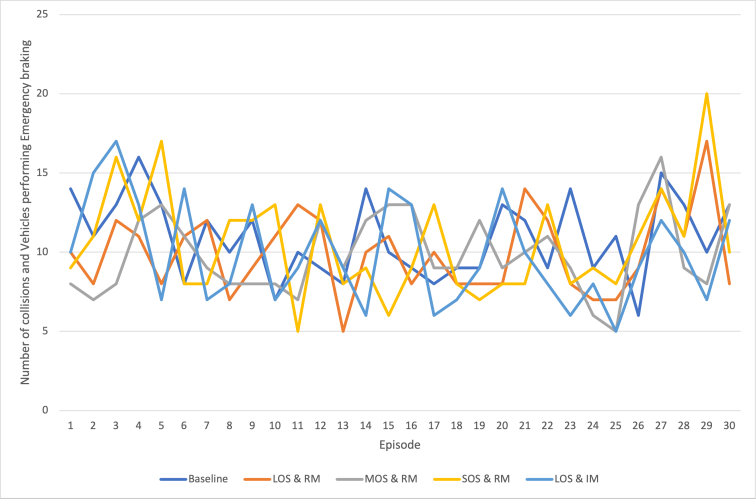
In this section, we first compare the performance of the DQN with two different techniques that are mentioned in section 5.3.1. We also add a baseline value to compare our results to. This baseline value is calculated by allowing SUMO to control the vehicles. From the single-agent experiments, we take the best values from the observation space and Batch Size and Priority Memory Size and pass it to the multi-agent scenario. We then compare the IDQN to the baseline values. We also discuss both Action sets. Table 5.5 below shows the abbreviations and their meanings in the following sections.

Abbreviation	**Meaning**
LOS	Large Observation Scope
MOS	Medium Observation Scope
SOS	Small Observation Scope
RM	Reduced Memory
IM	Increased Memory

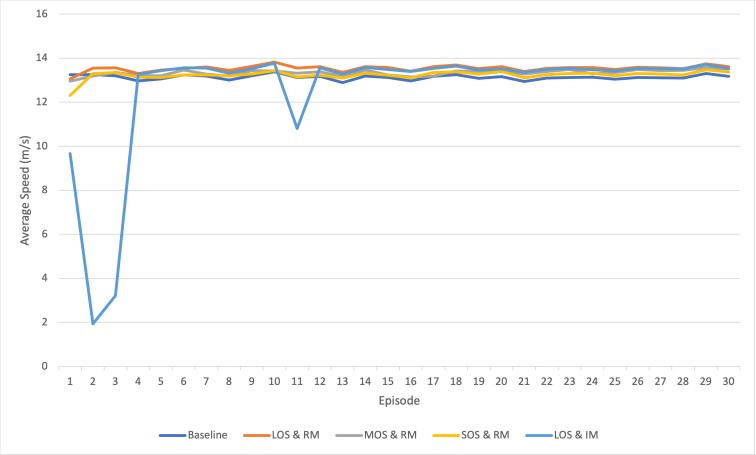
Table 5.5: Abbreviations descriptions

Basic Action Set - Single-Agent

Figure 5.5 below shows the results for single-agent training for Action set 1 (Basic Action Set). More specifically, we look at Figure 5.5a which shows the number of collisions and vehicles performing emergency breaking for each of the different techniques. From this graph, it is very difficult to tell if any of the techniques out-preform the baseline results. To get an overall understanding of this simulation, the overall summary can be found in Table 5.6. From this table, we can see that all the proposed methods outperform the baseline results, with both MOS & RM and LOS & IM producing the best results by decreasing the number of collisions by 10 %. In Figure 5.5b we can see the graph for the average speed of the vehicles for every timestep. Again it is very difficult to read this graph as the results are so closely matched. From Table 5.6 we can see that all proposed methods apart from LOS & IM had a higher average speed than the baseline results. With LOS & RM having the highest average speed of all methods. From these results, we can see that both LOS & RM and MOS & RM perform the best as each method is placed either 1st or 2nd in the two categories.



(a) Number of collisions/ Emergency braking occurring in each Episode



(b) Average Speed of Vehicles in each Episode

Figure 5.5: Results of Low Traffic and Single Agent simulations using Action set 1

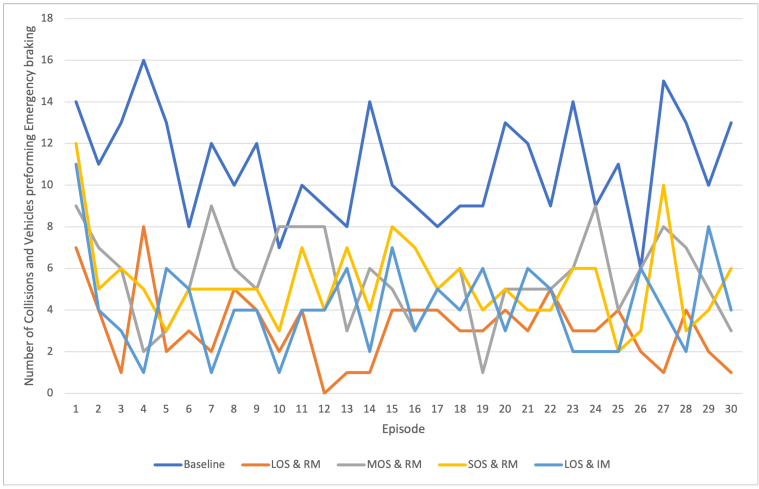
Simulation	Average Speed	Total Number of Collisions/Emergency Braking
Baseline	13.13	327
LOS & RM	13.53	299
MOS & RM	13.33	297
SOS & RM	13.23	316
LOS & IM	12.55	297

Table 5.6: Single-agent, low traffic demand, Action set 1 summary

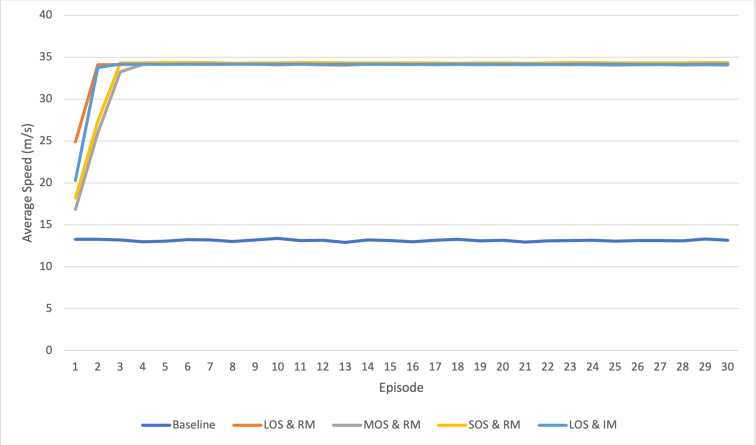
Extended Action Set - Single-Agent

Figure 5.6 below shows the results for single-agent training for Action set 2 (Extended Action Set). To get an overall understanding of this simulation, the overall summary can be found in Table 5.7. More specifically, we look at Figure 5.6a which shows the number of collisions and vehicles performing emergency breaking for each of the different techniques. From this graph, it can be seen that all the proposed methods outperform the baseline results. From

Figure 5.6a and Table 5.7 it can be seen that LOS & RM performs the best as it reduces from collisions by 72% to the baseline result and over 30 fewer collisions to its counterpart. In Figure 5.6b we can see the graph for the average speed of the vehicles for every timestep. Again it is very difficult to read this graph as the results are very closely matched. However, it is clear that once again, all the proposed methods outperforms the baseline results. From Table 5.6, LOS & RM had the highest average speed of all methods. From these results, we clearly say that LOS & RM performed the best in this specific simulation.



(a) Number of collisions/ Emergency braking occurring in each Episode



(b) Average Speed of Vehicles in each Episode

Figure 5.6: Results of Low Traffic and Single Agent simulations using Action set 2

Simulation	Average Speed	Total Number of Collisions/Emergency Braking
Baseline	13.13	327
LOS & RM	33.82	94
MOS & RM	33.24	168
SOS & RM	33.59	159
LOS & IM	33.68	125

Table 5.7: Single-agent, low traffic demand, Action set 2 summary

Comparison of Action Sets - Single-Agent

When comparing the 2 action sets in Low traffic demand and analysing the results from Figures 5.5 and 5.6 and Tables 5.6 and 5.7. It can be seen that Action set 2 (Extended Action Set) outperforms Action set 1, both in terms of the number of collisions/emergency braking performed and average speed. Both action sets will indeed be tested in the Multi-agent scenario to ensure that Action set 2 is the superior action set.

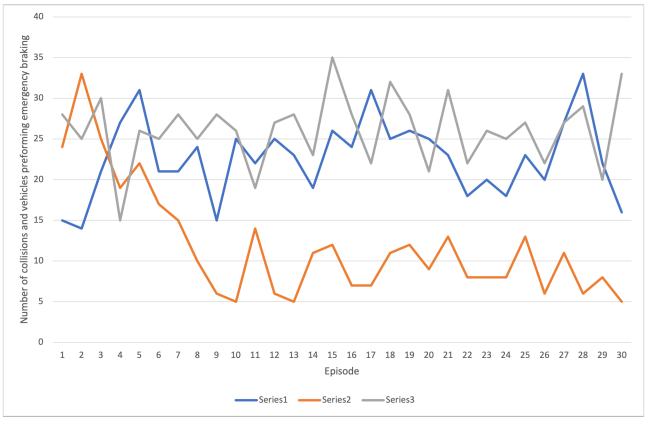
Multi-Agent

From the low traffic, single-agent experiments, we can see that LOS & RM performed the best. So in these experiments, we will be comparing LOS & RM for both Action set 1, 2 and baseline results. Figure 5.7 below shows the results from this experiment. To get an overall understanding of this simulation, the overall summary can be found in Table 5.8. From Figure 5.7a and Table 5.8 we can see that Action set 2 (Extended Action Set) has considerably fewer collisions and emergency braking performed than Action set 1 and the baseline. Like in the single-agent scenario, Action set 1 performs very similar to that of the baseline results. As mentioned in that section, this could be caused by a lack of choice for the agent. However, overall, Action set 1 performs worse than baseline as the number of collisions/emergency braking occurring increases by 15%. Figure 5.7b shows the average speed achieved by the vehicles during the simulation. The general trend stays the same in respect that Action set 2 outperforms Action set 1 and baseline results, and Action set 1 performs similar the baseline values. In Figure 5.7d we see the average waiting for vehicles during the simulation. We see here that both Action set 1 and 2 outperform the baseline as no waiting time occurs. From this, we can conclude that both scenarios reduce the chance of causing a traffic jam and hence improve traffic flow and efficiency. In the last Figure, Figure 5.7c we can see the average reward per episode. Once again, the similar trend as mentioned above for the number of collisions comparison.

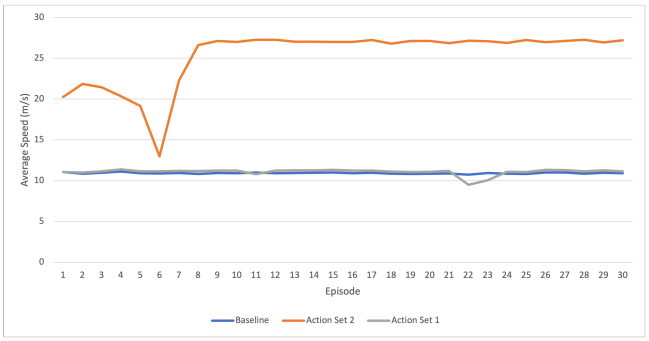
Result Set	Average Speed (m/s)	Total Number of Collisions/Emergency Braking	Total Waiting Time (s)	Average Episode Reward
Baseline	10.91	680	412	120,830
Action Set 1	11.08	781	0	126,202
Action Set 2	25.34	356	0	469,964

Table 5.8: Multi-agent, low traffic demand summary

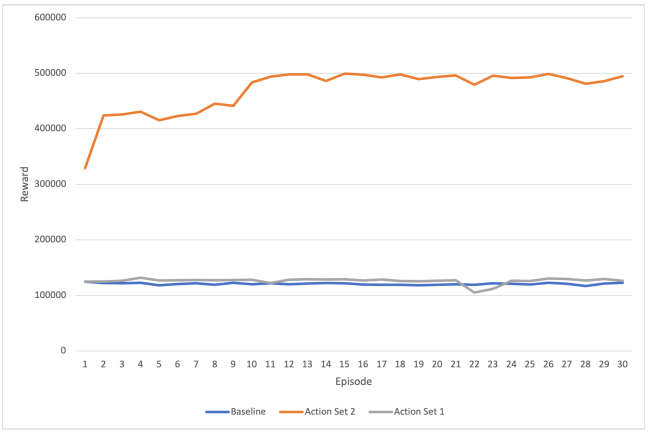
From these results, we can see that Action set 2 out preforms both Action set 1 and baseline results in almost every metric. Using Action set 2, we clearly meet our objectives as mentioned in section 5.1 when there is low traffic.



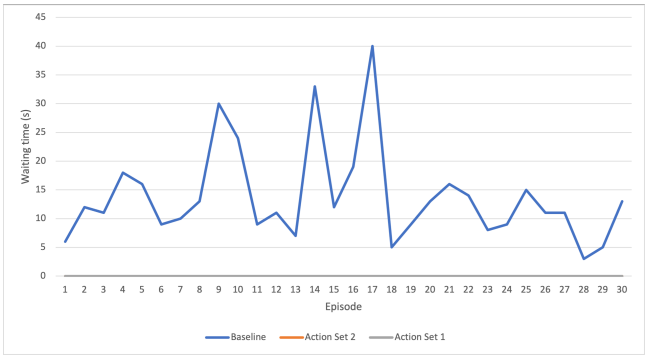
(a) Number of collisions/ Emergency braking occurring in each Episode



(b) Average Speed of Vehicles in each Episode



(c) The Average Reward in each Episode



(d) The Average time in each Episode

Figure 5.7: Results of Multi Agent low traffic simulations

5.5.2 High Traffic demand

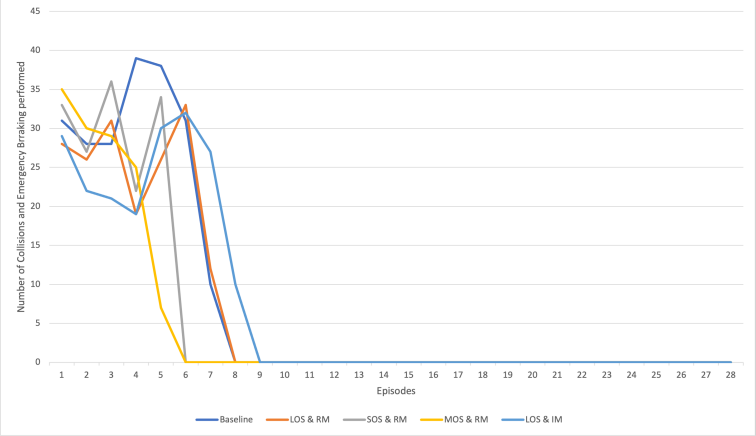
In this section, we first compare the performance of the DQN with two different techniques that are mentioned in section 5.3.1. We also add a baseline value to compare our results to. This baseline value is calculated by allowing SUMO to control the vehicles. From the single-agent experiments, we take the best values from the observation space and Batch Size and Priority Memory Size and pass it to the multi-agent scenario. We then compare the IDQN to the baseline values. We also discuss both Action sets.

Basic Action Set - Single-Agent

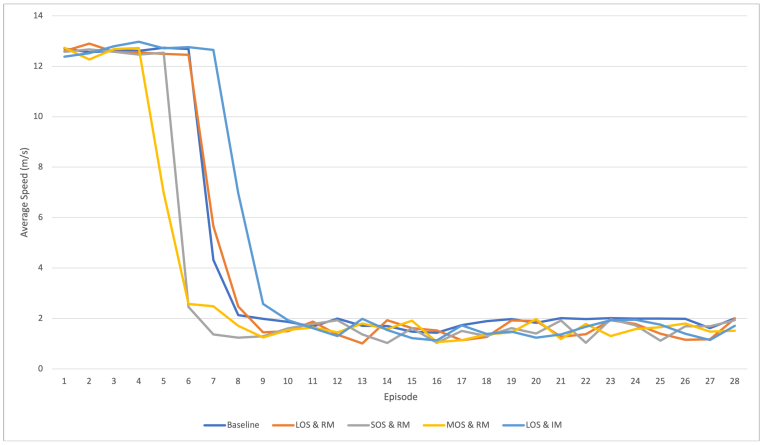
Figure 5.8 below shows the results for Action set 1 (Basic Action Set) in high traffic demands in a single agent scenario. From these graphs, it can be seen that a traffic jam occurs due to the high level of traffic. A traffic jam forms because more vehicles are entering the simulation than there are leaving it. From the graphs, we can see that after about the tenth episode, all proposed methods and baseline have a very low average speed, and in doing so, no collision or emergency braking occur. However, this might meet our primary goal of increasing traffic safety but does meet our second which was to maintain traffic flow and efficiency. From this simulation, there is no "best" method as they all perform very poorly. However, an interesting observation can still be made here. In both low and high traffic demand in the single-agent and Action set 1 scenario, all sets of methods perform relatively similar to that of the baseline results. This could be caused by the lack of actions that the agent can take.

Extended Action Set - Single-Agent

Figure 5.9 below shows the results for Action set 2 (Extended Action Set) in high traffic demands in a single agent scenario. To get an overall understanding of this simulation, the overall summary can be found in Table 5.9. More specifically, we look at Figure 5.9a which shows the number of collisions and vehicles performing emergency breaking for each of the different techniques. From this graph, it can be seen that all the proposed methods outperform the baseline results. This was due to the fact that in the baseline simulations, a traffic jam occurs whilst using Action set 2, no traffic jam occurs. Figure 5.9a and Table 5.7 shows that LOS & IM performs the best as it reduces collisions even though the simulation ran for an extra 23 episodes while the baseline was stuck in a traffic jam. In Figure 5.9b we can see the graph for the average speed of the vehicles for every timestep. Again it is very difficult to read this graph as the results are very closely matched. However, it clear that once again, all the proposed methods outperforms the baseline results. From Table 5.9, MOS & RM had the highest average speed of all methods. From these results, we clearly say that LOS & RM and LOS & IM performed the best in this specific simulation. Even though MOS & RM had the highest average speed, it had over 100 more collision than LOS & RM and LOS & IM. Therefore it was not considered as a top performer.



(a) Number of collisions/ Emergency braking occurring in each Episode



(b) Average Speed of Vehicles in each Episode

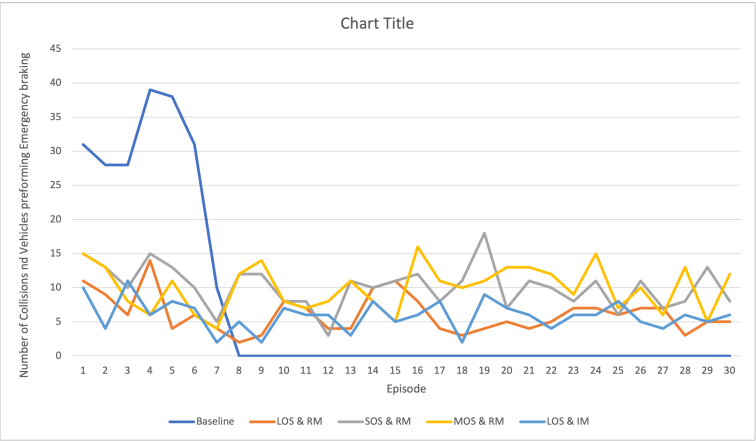
Figure 5.8: Results of High Traffic and Single Agent simulations using Action set 1

Simulation	Average Speed	Total Number of Collisions/Emergency Braking
Baseline	4.08	205
LOS & RM	33.72	183
MOS & RM	33.99	299
SOS & RM	33.45	305
LOS & IM	33.73	178

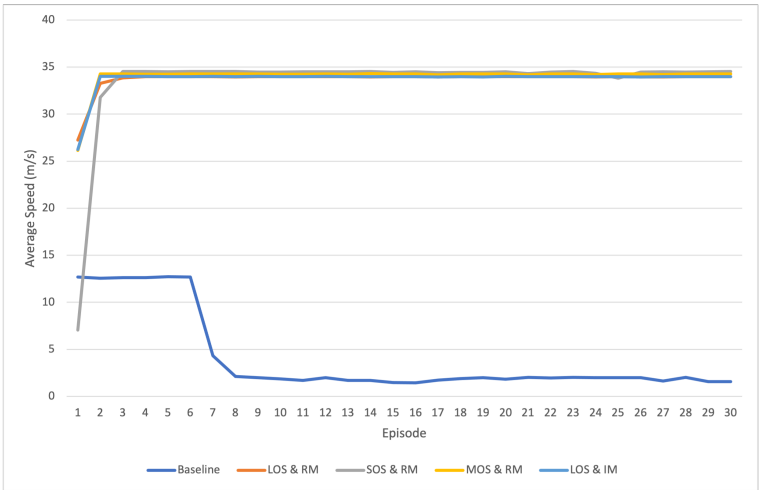
Table 5.9: Single-agent, high traffic demand, Action set 2 summary

Comparison of Action Sets - Single-Agent

When comparing the 2 action sets in High traffic demand and analysing the results from Figures 5.8 and 5.9 and Table 5.9. It can be seen that Action set 2 (Extended Action Set) outperforms Action set 1, both in terms of the number of collisions/emergency braking performed and average speed. Action set 2 is more superior because it removes the traffic jam from the simulation while getting positive results. While using action set 1 does not prevent the traffic jam from occurring. Both action sets will indeed be tested in the Multi-agent scenario to ensure that Action set 2 is the superior action set.



(a) Number of collisions/ Emergency braking occurring in each Episode



(b) Average Speed of Vehicles in each Episode

Figure 5.9: Results of High Traffic and Single Agent simulations using Action set 2

Multi-Agent

From the high traffic, single-agent experiments, we can see that LOS & RM performed the best. So in these experiments, we will be comparing LOS & RM for both Action set 1, 2 and baseline results. Figure 5.10 below shows the results from this experiment. To get an overall understanding of this simulation, the overall summary can be found in Table 5.10. From Figure 5.10a and Table 5.10 we can see that Action set 2 has considerably fewer collisions, and emergency braking performed compared to Action set 1 and the baseline. In total, it reduces the number of collisions by 60%. Like in the single-agent scenario, Action set 1 performs very similar to that of the baseline results. As mentioned in that section, this could be caused by a lack of choice for the agent to take. However, overall Action set 1 performs worse than baseline as the number of collisions/emergency braking occurring increases by 15%. This result is identical to the results in the low traffic scenario. Figure 5.10b shows the average speed achieved by the vehicles during the simulation. The general trend stays the same in respect that Action set 2 outperforms Action set 1 and baseline results, and Action set 1 performs similar the baseline values. In Figure 5.10d we see the average waiting for vehicles

during the simulation. We see here that both Action set 1 and 2 outperform the baseline as no waiting time occurs. From this, we can conclude that both scenarios reduce the chance of causing a traffic jam and hence improve traffic flow and efficiency. In the last Figure, Figure 5.10c we can see that the average reward per episode. Once again, a similar trend is found, where Action set 2 outperforms Action set 1, and Action set 1 performs similar to that of the baseline.

Result Set	Average Speed (m/s)	Total Number of Collisions/Emergency Braking	Total Waiting Time (s)	Average Episode Reward
Baseline	12.52	1233	1226	133,317
Action Set 1	12.72	1411	0	140,621
Action Set 2	31.42	504	0	566,343

Table 5.10: Multi-agent, high traffic demand summary

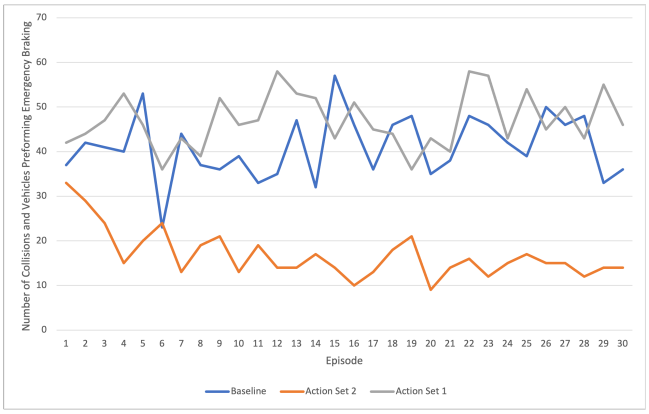
From these results, we can see that Action set 2 (Extended Action Set) out preforms both Action set 1 (Basic Action Set) and baseline results in almost every metric. Using Action set 2, we clearly meet our objectives as mentioned in section 5.1 when there is high traffic volume.

Another interesting observation can be seen in Figure 5.11. In the key of this graph where the given colours are assigned to a given action we also explain what variables are changed in the simulation by adding the acronyms IS,IA,IM,IR,DS,DA,DM and DR after in the brackets. Where I means increase, D means decrease, S stands for speed, A stands for acceleration, M refers to MinGap and R refers to Road Speed.

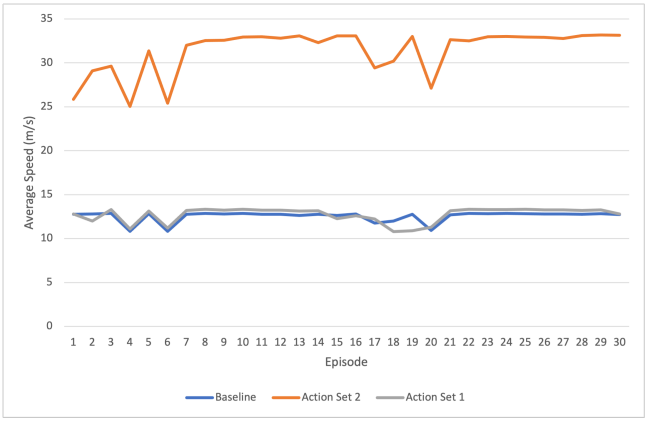
This figure shows which actions are prominent in high traffic simulations. We can see that Action set 1 choice number 3 is the most selected Action. However, in Action set 2, choice number 3 selections go down by over 20 percent. This is due to the fact that the most popular choice for the network in Action set 2 is choice number 5. From this, we could assume that selecting choice five more often would help reduce collisions and improve traffic flow/ efficiency. However, to fully test this proposal would be outside the scope of this work.

5.6 Evaluation Summary

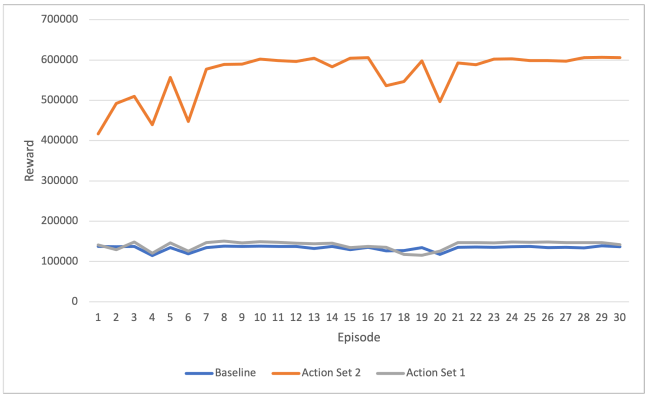
In this Chapter, we have presented details of the evaluation of DQN and IDQN to solve the traffic safety problem. We have also presented evaluation objectives, described the evaluation scenarios, the setup of the environment and presented and analysed the results.
From the analysis of the results, we can conclude that the DQN with priority experience replay is a suitable algorithm to meet the objectives for a single agent environment, as outlined in Section 5.1. From the single-agent experiment, we concluded that the best technique to use was the larger observation space of 400x400(LOS) and the small memory requirements(batch size = 10, priority memory = 5). These values were then used in the Multi-agent scenario.
Using the values from the single-agent experiments and the IDQN with priority experience replay, we can conclude that this is a suitable algorithm to meet the objectives for a multi-agent environment. We find that overall the number could of collisions could be reduced by up to 60%. Both these algorithms are suitable to use when used with Action set 2(Extended Action Set). Action set 2 clearly outperformed the baseline, and Action set 1 (Basic Action Set) in every metric. Finally, we can conclude that if Action set 2 is used alongside IDQN with priority experience replay, it will complete every objective outline in section 5.1. Firstly it reduces the number of collisions, it also increases traffic flow as traffic flow is directly related to the speed of the vehicles, and it also reduces the amount of waiting time in the road network.



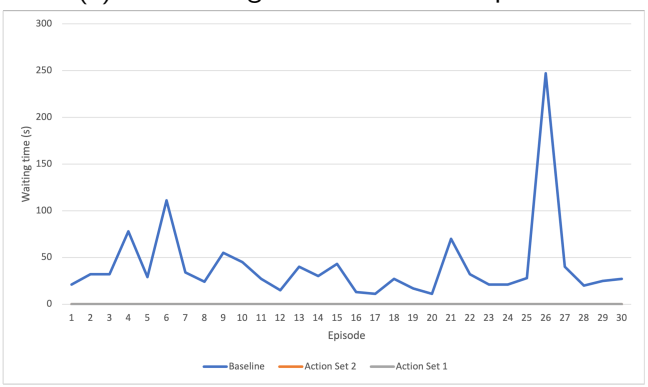
(a) Number of collisions/ Emergency braking occurring in each Episode



(b) Average Speed of Vehicles in each Episode



(c) The Average Reward in each Episode



(d) The Average time in each Episode

Figure 5.10: Results of Multi Agent high traffic simulations

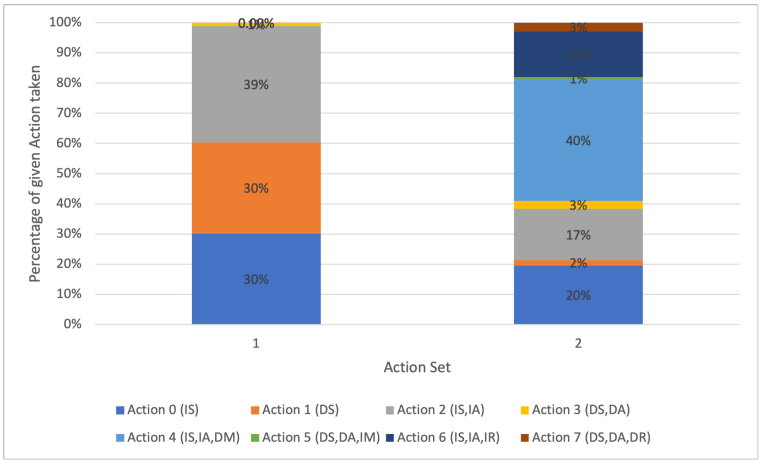


Figure 5.11: The percentage of how often each action is chosen in the multi-agent high traffic demand scenario

6 Conclusion and Future Work

In this chapter, we summarize the entirety of the Thesis and highlight some of its most significant achievements. We then conclude with a discussion on future work based on work completed in this Thesis.

6.1 Thesis Contribution

This Thesis is the only recent work that addresses multi-agent DRL in Traffic Safety Problem.

Chapter 1 outlines the motivation behind the work and discuss key topics such as thesis aims and objectives, problem statement, assumptions, main contributions and presents a structure for the rest of thesis.

Chapter 2 gave the background material used to design and implement the DQN and IDQN. It firstly covered the basics of Deep Learning and Reinforcement Learning. It then went on to mention some of the key term and algorithms used in these areas. Furthermore, it introduced DRL and how it solved the issue of dimensionality. In addition, we also discussed the cooperative driving strategies used and their effect on driving conditions. Lastly, this chapter discusses the use of multi-agent DRL to traffic-related problems such as traffic control and congestion. This related investigation showed that no study has tried to use multi-agent DRL to help improve traffic safety.

Chapter 3 describes the design decision taken to build both the DQN and the IDQN. We defined some key reinforcement learning terms such as state-space, action space, reward function. We also discussed the design of the neural network, single and multi-agent training. Lastly, we discussed the Modularity and Scalability of the multi-agent approach.

Chapter 4 then presents the implementation of a DQN for single agents and the IDQN for multi-agent scenarios. This includes a brief overview of the code, the implementation of the Neural network using Tensorflow and Keras, a description of the simulation environment, SUMO, and the given setup required.

Chapter 5 evaluated the use of IDQN to help improve traffic safety whilst increasing or maintaining traffic flow and efficiency by using SUMO. We first evaluated the performance of

a DQN in a single agent environment. We tested the performance using various parameters such as observation space, batch size, priority memory size and action set. We then compared these values against a baseline figure. These baseline results were calculated by allowing SUMO to control all the vehicles. From the single-agent simulations, we found that the largest observation size along with the smallest memory size produced the best results between low and high traffic demands. These values were then carried forward and tested in the multi-agent scenario. In the multi-agent scenario, we found that Action set 2 (Extended Action Set) out preformed both Action set 1 (Basic Action Set) and the baseline results in almost every metric. We also found that Action Set 1 performs very similar to that of the baseline figures. In conclusion, we saw that using Action set 2 with a large observation space and smaller batch size and priority memory size; we can reduce the number of collisions by up to 60%, decrease waiting time to 0 and increase the average speed of vehicles by over 100%

6.2 Future Work

When designing and evaluating our DQN and IDQN, we identified some areas where future research could be conducted as it outside the scope of this project. Some of these areas include are discussed below:

- **More complicated Junction Types:** In this Thesis, we have used a straightforward junction layout of 4 roads meeting at an intersection equally spread apart. To expand this, we change the junction size to one with more than four roads linked to it and spread unequally around the junction. The junction could also be changed to a roundabout to see how the algorithm would perform.
- **Use Real-World Urban Areas:**This Thesis looks at a simple four junction scenario. To evolve this technique, a real-world urban area could be used. Using OSM wizard (a package supplied by SUMO) allows the user to import real-world road networks. Using this package and the current IDQN algorithm, a comparison of how the algorithm performs in a real word network could be completed.
- **Longer Training times:** In our experiments, we only used 30 episodes per simulation. However, studies usually use 1500-2000 episodes of training. We only used 30 due to the time constraints of the research, but if we had more time, we could have extended the simulation and possibly get improved and converged results. (converged results = flat line graphs were results only differ slightly from episode to episode.)
- **Different Vehicle types:** In this research, we only use one vehicle type. This vehicle type is a standard mid-sized car. To fully develop this we could use a range of vehicles such as trucks, buses, bicycles, motorbikes and SUVs. The given characteristics of different vehicles could affect the end results.
- **Different Traffic Demand Generation:** The traffic demand generation used, follows

a given schedule. However, in the real world, the traffic demand does not follow this strict schedule as traffic will generally deviate from day to day. A suitable approach would be to take real-world street data and en-corporate this into the road network to see what the outcome is.

- **Further Understanding of Actions Taken:** At the end of Chapter 5, we briefly mentioned which actions the agent took in 2 simulations. Future work could develop this idea further and investigate which action the agents take and for what reason is the given Action taken.
- **Scale multi-agent training with distributed computation:** Each agent's Neural Network could be placed on a different computer, and the simulator would also have its own computer which interacts with the agents in a distributed manner. The reason for using a distributed system is to allow us to train several agents using several machines (Horizontal Scaling) rather than be limited by computational resources of only one machine (Vertical Scaling).
- **Pattern detection and adaptation:** The IDQN could learn to detect different fluctuations in the traffic flow and change its policy given those changes in its behaviour.

Bibliography

[1] Mariusz Goniewicz, Adam Nogalski, Meleckidzedeck Khayesi, Tomasz Lübek, Beata Zuchora, Krzysztof Goniewicz, and Paulina Miśkiewicz. Pattern of road traffic injuries in lublin county, poland. *Central European journal of public health*, 20:116–20, 06 2012. doi: 10.21101/cejph.a3686.

[2] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR, 2016.

[3] Ivana Dusparic and Vinny Cahill. Autonomic multi-policy optimization in pervasive systems: Overview and evaluation. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(1):1–25, 2012.

[4] Mohammed Tahifa, Jaouad Boumhidi, and Ali Yahyaouy. Swarm reinforcement learning for traffic signal control based on cooperative multi-agent framework. In *2015 Intelligent Systems and Computer Vision (ISCV)*, pages 1–6. IEEE, 2015.

[5] L. Li, Y. Lv, and F. Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 3(3):247–254, 2016. doi: 10.1109/JAS.2016.7508798.

[6] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito, and Norio Shiratori. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *arXiv preprint arXiv:1705.02755*, 2017.

[7] X. Liang, X. Du, G. Wang, and Z. Han. A deep reinforcement learning network for traffic light cycle control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, 2019. doi: 10.1109/TVT.2018.2890726.

[8] Paul Young Joun Ha, Sikai Chen, Jiqian Dong, Runjia Du, Yujie Li, and Samuel Labi. Leveraging the capabilities of connected and autonomous vehicles and multi-agent reinforcement learning to mitigate highway bottleneck congestion. *arXiv preprint arXiv:2010.05436*, 2020.

[9] T. Wu, P. Zhou, K. Liu, Y. Yuan, X. Wang, H. Huang, and D. O. Wu. Multi-agent deep reinforcement learning for urban traffic light control in vehicular networks. *IEEE Transactions on Vehicular Technology*, 69(8):8243–8256, 2020. doi: 10.1109/TVT.2020.2997896.

[10] WHO. Road traffic injuries. URL `https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries`.

[11] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[12] Paul Werbos. Beyond regression: New tools for prediction and analysis in the behavioral science. thesis (ph. d.). appl. math. harvard university. 01 1974.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[14] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[15] Sagar Sharma. Activation functions in neural networks. *Towards Data Science*, 6, 2017.

[16] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[17] Y. Bengio and Yann Lecun. Convolutional networks for images, speech, and time-series. 11 1997.

[18] Jian Wang, Jiqing Sun, Hongfei Lin, Hualei Dong, and Shaowu Zhang. Convolutional neural networks for expert recommendation in community question answering. *Science China Information Sciences*, 60(11):110102, 2017.

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[21] Jawad Nagi, Frederick Ducatelle, Gianni Di Caro, Dan Ciresan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. pages 342–347, 11 2011. doi: 10.1109/ICSIPA.2011.6144164.

[22] Syed Zubair, Fei Yan, and Wenwu Wang. Dictionary learning based sparse coefficients for audio classification with max and average pooling. *Digital Signal Processing*, 23(3): 960–970, 2013.

[23] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[24] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

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

[26] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[27] Chathurangi Shyalika. A beginners guide to q-learning. URL `https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c`.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[29] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[30] Hado Hasselt. Double q-learning. *Advances in neural information processing systems*, 23:2613–2621, 2010.

[31] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *arXiv preprint arXiv:1509.06461*, 2015.

[32] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[33] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[34] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. *arXiv preprint arXiv:1702.08887*, 2017.

[35] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[36] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4):e0172395, 2017.

[37] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in neural information processing systems*, pages 2137–2145, 2016.

[38] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

[39] Gerald Tesauro. Extending q-learning to general adaptive multi-agent systems. *Advances in neural information processing systems*, 16:871–878, 2003.

[40] Alberto Broggi, Alexander Zelinsky, Michel Parent, and Charles Thorpe. *Intelligent Vehicles*, pages 1175–1198. 01 2008. doi: $10.1007/978\text{-}3\text{-}540\text{-}30301\text{-}5\_52$.

[41] Richard Bishop. Intelligent vehicle rd: A review and contrast of programs worldwide and emerging trends. *Annales des Télécommunications*, 60:228–263, 03 2005. doi: $10.1007/\mathrm{BF}03219820$.

[42] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[43] W. D. Jones. Keeping cars from crashing. *IEEE Spectr.*, 38(9):40–45, September 2001. ISSN 0018-9235. doi: $10.1109/6.946636$. URL `https://doi.org/10.1109/6.946636`.

[44] Kakan Chandra Dey, Anjan Rayamajhi, Mashrur Chowdhury, Parth Bhavsar, and James Martin. Vehicle-to-vehicle (v2v) and vehicle-to-infrastructure (v2i) communication in a heterogeneous wireless network–performance evaluation. *Transportation Research Part C: Emerging Technologies*, 68:168–184, 2016.

[45] Ziran Wang, Guoyuan Wu, and Matthew J Barth. A review on cooperative adaptive cruise control (cacc) systems: Architectures, controls, and applications. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2884–2891. IEEE, 2018.

[46] F. Broqua. Cooperative driving : Basic concepts and a first assessment of "intelligent cruise control" strategies. 1991.

[47] Bart van Arem, JH Hogema, MJWA Vanderschuren, and CH Verheul. An assessment of the impact of autonomous intelligent cruise control. 1996.

[48] Michiel M Minderhoud and Piet HL Bovy. Impact of intelligent cruise control on motorway capacity. *Transportation Research Record*, 1679(1):1–9, 1999.

[49] Arne Kesting, Martin Treiber, Martin Schönhof, Florian Kranke, and Dirk Helbing. Jam-avoiding adaptive cruise control (acc) and its impact on traffic dynamics. *Traffic and Granular Flow '05*, 02 2006. doi: 10.1007/978-3-540-47641-2_62.

[50] Jeroen Ploeg, Bart TM Scheepers, Ellen Van Nunen, Nathan Van de Wouw, and Henk Nijmeijer. Design and experimental evaluation of cooperative adaptive cruise control. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 260–265. IEEE, 2011.

[51] Steven E Shladover, Dongyan Su, and Xiao-Yun Lu. Impacts of cooperative adaptive cruise control on freeway traffic flow. *Transportation Research Record*, 2324(1):63–70, 2012.

[52] Georges M Arnaout and Jean-Paul Arnaout. Exploring the effects of cooperative adaptive cruise control on highway traffic flow using microscopic traffic simulation. *Transportation Planning and Technology*, 37(2):186–199, 2014.

[53] Cathy Wu, Alexandre M Bayen, and Ankur Mehta. Stabilizing traffic with autonomous vehicles. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2018.

[54] Qiong Lu, Tamás Tettamanti, Dániel Hörcher, and István Varga. The impact of autonomous vehicles on urban traffic network capacity: an experimental analysis by microscopic traffic simulation. *Transportation Letters*, pages 1–10, 2019.

[55] Ye Li, Zhibin Li, Hao Wang, Wei Wang, and Lu Xing. Evaluating the safety impact of adaptive cruise control in traffic oscillations on freeways. *Accident Analysis Prevention*, 104:137 – 145, 2017. ISSN 0001-4575. doi: https://doi.org/10.1016/j.aap.2017.04.025. URL http://www.sciencedirect.com/science/article/pii/S0001457517301628.

[56] Greg Marsden, Mike McDonald, and Mark Brackstone. Towards an understanding of adaptive cruise control. *Transportation Research Part C: Emerging Technologies*, 9(1):33 – 51, 2001. ISSN 0968-090X. doi: https://doi.org/10.1016/S0968-090X(00)00022-X. URL http://www.sciencedirect.com/science/article/pii/S0968090X0000022X.

[57] Ye Li, Hao Wang, Wei Wang, Lu Xing, Shanwen Liu, and Xueyan Wei. Evaluation of the impacts of cooperative adaptive cruise control on reducing rear-end collision risks

on freeways. *Accident Analysis Prevention*, 98:87 – 95, 2017. ISSN 0001-4575. doi: https://doi.org/10.1016/j.aap.2016.09.015. URL http://www.sciencedirect.com/science/article/pii/S0001457516303451.

[58] S.M. Sohel Mahmud, Luis Ferreira, Md. Shamsul Hoque, and Ahmad Tavassoli. Application of proximal surrogate indicators for safety evaluation: A review of recent developments and research needs. *IATSS Research*, 41(4):153 – 163, 2017. ISSN 0386-1112. doi: https://doi.org/10.1016/j.iatssr.2017.02.001. URL http://www.sciencedirect.com/science/article/pii/S0386111217300286.

[59] Yu Tu, Wei Wang, Ye Li, Chengcheng Xu, Te Xu, and Xueqi Li. Longitudinal safety impacts of cooperative adaptive cruise control vehicle's degradation. *Journal of Safety Research*, 69:177 – 192, 2019. ISSN 0022-4375. doi: https://doi.org/10.1016/j.jsr.2019.03.002. URL http://www.sciencedirect.com/science/article/pii/S0022437518306509.

[60] Maxime Guériau and Ivana Dusparic. Quantifying the impact of connected and autonomous vehicles on traffic efficiency and safety in mixed traffic.

[61] Lanhang Ye and Toshiyuki Yamamoto. Evaluating the impact of connected and autonomous vehicles on traffic safety. *Physica A: Statistical Mechanics and its Applications*, 526:121009, 2019. ISSN 0378-4371. doi: https://doi.org/10.1016/j.physa.2019.04.245. URL http://www.sciencedirect.com/science/article/pii/S0378437119306181.

[62] Alkis Papadoulis, Mohammed Quddus, and Marianna Imprialou. Evaluating the safety impact of connected and autonomous vehicles on motorways. *Accident Analysis Prevention*, 124:12 – 22, 2019. ISSN 0001-4575. doi: https://doi.org/10.1016/j.aap.2018.12.019. URL http://www.sciencedirect.com/science/article/pii/S0001457518306018.

[63] D. Kim, S. Moon, J. Park, H. J. Kim, and K. Yi. Design of an adaptive cruise control / collision avoidance with lane change support for vehicle autonomous driving. In *2009 ICCAS-SICE*, pages 2938–2943, 2009.

[64] Derick A Johnson and Mohan M Trivedi. Driving style recognition using a smartphone as a sensor platform. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 1609–1615. IEEE, 2011.

[65] Xinpeng Wang, Chaozhong Wu, Jie Xue, and Zhijun Chen. A method of personalized driving decision for smart car based on deep reinforcement learning. *Information*, 11(6): 295, 2020.

[66] Dimitris M. Vlachogiannis, Eleni I. Vlahogianni, and John Golias. A reinforcement learning model for personalized driving policies identification. *International Journal of Transportation Science and Technology*, 9(4):299 – 308, 2020. ISSN 2046-0430. doi: https://doi.org/10.1016/j.ijtst.2020.03.002. URL http://www.sciencedirect.com/science/article/pii/S2046043020300198.

[67] Erwin Walraven, Matthijs T.J. Spaan, and Bram Bakker. Traffic flow optimization: A reinforcement learning approach. *Engineering Applications of Artificial Intelligence*, 52:203 – 212, 2016. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai.2016.01.001. URL http://www.sciencedirect.com/science/article/pii/S0952197616000038.

[68] Samah El-Tantawy, Baher Abdulhai, and Hossam Abdelgawad. Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown toronto. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1140–1150, 2013.

[69] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pages 157–163. Elsevier, 1994.

[70] S. S. Mousavi, M. Schukat, and E. Howley. Traffic light control using deep policy-gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems*, 11 (7):417–423, 2017. doi: 10.1049/iet-its.2017.0153.

[71] Mengqi Liu, Jiachuan Deng, Ming Xu, Xianbo Zhang, and Wei Wang. Cooperative deep reinforcement learning for traffic signal control. In *The 7th International Workshop on Urban Computing (UrbComp 2018)*, 2017.

[72] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of sumo-simulation of urban mobility. *International journal on advances in systems and measurements*, 5(3&4), 2012.

A1 Appendix

Figure A1.1 shows the full DQN network displayed on Tensorflow. This network was built using Tensorflow and Keras which leads to this graph being difficult to read.

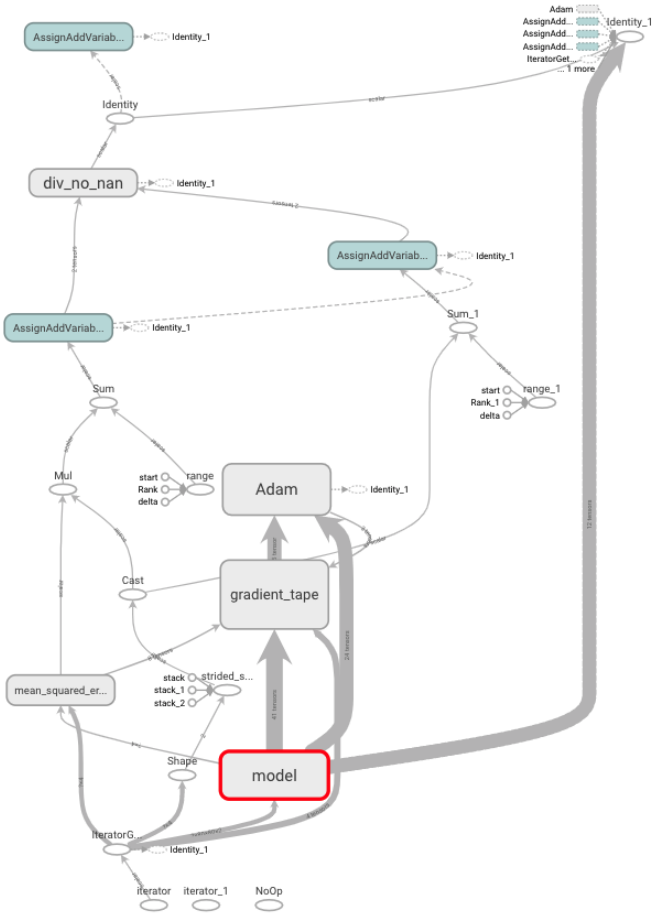


Figure A1.1: The full DQN network built using Tensorflow