FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Intelligent Traffic Light Management for Automated Guided Vehicle Systems using Deep Reinforcement Learning

Ricardo França D. Cardoso

Mestrado em Engenharia Informática e Computação

Supervisor: Rosaldo J. F. Rossetti, PhD

Second Supervisor: Zafeiris Kokkinogenis, PhD

February 15, 2023

Intelligent Traffic Light Management for Automated Guided Vehicle Systems using Deep Reinforcement Learning

Ricardo França D. Cardoso

Mestrado em Engenharia Informática e Computação

February 15, 2023

Resumo

Um dos motores do desenvolvimento tecnológico é a automação de processos nas empresas. Atualmente, a movimentação de veículos requer interação humana e automatizar esses movimentos é um grande desafio. Uma vantagem competitiva pode surgir se superarmos os principais desafios como gestão de conflitos, cálculo de rotas, previsão de impactos, entre outros. Um cenário com o qual até os humanos podem achar difícil de lidar são os cruzamentos; mesmo com semáforos, rotundas e outros truques de engenharia, evitar engarrafamentos e acidentes pode ser um desafio.

Para conceber um sistema capaz de tomar decisões em tempo real neste cenário, é típico o uso de algoritmos de Inteligência Artificial. Aprendizagem por Reforço (RL) é uma das escolhas mais populares para controlar sistemas complexos. Por meio de um sistema de recompensas positivas, é possível treinar agentes capazes de otimizar as decisões tomadas para aumentar o fluxo de trânsito e diminuir consideravelmente, senão erradicar, cenários de acidentes e impasses. Treinamos e testamos três métodos de RL, nomeadamente, Deep Q-Network (DQN), Double DQN e Dueling DQN.

Os três métodos escolhidos foram treinados e testados no simulador microscópico SUMO desenvolvido pela Eclipse Foundation. Implementamos e testamos seis cenários com diferentes números de vias e configurações de fases de semáforos. A metodologia proposta permitiu obter informações importantes sobre como melhorar a coordenação de interseções simples de AGVs. Os resultados sugerem, por exemplo, que diminuir o número de fases e aumentar o número de vias de rodagem pode ser benéfico. Quanto aos algoritmos utilizados, o DQN e o Double DQN tiveram melhor desempenho em cenários mais simples, enquanto o Dueling DQN parece ser mais adequado para configurações mais complexas de interseção.

Keywords: agv, aprendizagem por máquinas, aprendizagem por reforço, aprendizagem por reforço profunda, automação, controlo de trânsito, interseções, redes neurais, rl, veículos, veículos guiados autonomamente

Abstract

One of the drivers of technological development is the automation of processes in companies. Nowadays, vehicle movement requires human interaction, and automating those movements is a big challenge. It can provide a competitive advantage if we overcome the main challenges like conflict management, routing calculation, impact prediction, and others. One scenario that even humans can find challenging to cope with is intersections; even with traffic lights, roundabouts, and other engineering tricks, avoiding traffic jams and accidents can be challenging.

In order to conceive a system capable of making real-time decisions in this scenario, it is typical to use algorithms of Artificial Intelligence. Reinforcement Learning (RL) is one of the most popular choices for controlling complex systems. Through a positive rewards system, it is possible to train agents capable of optimising the decisions made to increase traffic flow and considerably decrease, if not eradicate, accident and deadlock scenarios. We trained and tested three RL methods, namely Deep Q-Network (DQN), Double DQN, and Dueling DQN.

The three methods chosen were trained and tested under the SUMO microscopic simulator developed by the Eclipse Foundation. We implemented and tested six scenarios with different numbers of lanes and traffic light phase configurations. The proposed methodology allowed us to gain important insight into how to improve the coordination of single intersections operating AGVs. Results suggested, for instance, that decreasing the number of phases and increasing the number of lanes could be beneficial. As for the algorithms used, the DQN and Double DQN performed better in simpler scenarios, whereas Dueling DQN seems to be more appropriate for more complex intersection settings.

Keywords: agv, automated guided vehicles, automation, deep reinforcement learning, intersections, machine learning, neural network, reinforcement learning, rl, traffic control, vehicles

Contents

List of Figures

List of Tables

Abbreviations

AD	Autonomous Driving
AGV	Automated Guided Vehicle
AI	Artificial Intelligence
ATI-TSC	Alan Turing Institute's Traffic Signal Control
CNN	Convolution Neural Network
DL	Deep Learning
DRL	Deep Reinforcement Learning
DP	Dynamic Programming
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
DDDQN	Dueling Double Deep Q-Network
FC	Fully Connected
GPU	Graphical Processing Unit
GUI	Graphical Use Interface
MARL	Multi-Agent Reinforcement Learning
MDP	Markov Decision Process
MP	Markov Process
MP	Memory Palace
MRP	Markov Reward Process
PG	Phase Gate
RL	Reinforcement Learning
TraCI	SUMO's Traffic Control Interface
TSC	Traffic Signal Control

Symbols

$a \in A$	action $\in$ set of actions
$r \in R$	reward $\in$ set of rewards
$s \in S$	state $\in$ set of states
$p \in P$	probability $\in$ state transition matrix
π	policy
α	learning rate
γ	discount factor
τ	trajectory
ε	coefficient between exploration and exploitation
Q^*	optimal Q-value function
$Q(s,a)$	Q-value for taking action a in state s
E	episode return
G_t	future discounted return
G or $R(\tau)$	trajectory return
θ, α, β	weights of the neural networks

Chapter 1

Introduction

1.1 Context

Nowadays, humans are required to drive vehicles reliably. However, if one can fully automate vehicle driving, it can provide a competitive advantage to the industries that use it. “Automated Guided Vehicles (AGVs) are mobile robots which are extensively used in the industry to transport goods from A to B” [6]. They can be automobiles, trucks, drones, forklifts, or other industrial vehicles. Developing AGVs is a complex project that Deloitte Central Services, SA is currently embracing. One of this project’s components is related to managing conflicts at intersections.

With the recent advances in computing technology, e.g. Graphic Processing Units (GPUs) [16] and Deep Learning (DL) [10], the potential of Reinforcement Learning (RL) has drawn the attention of many scholars. It is often used in decision-making systems, mainly after AlphaGo developed a model based on a deep Q-Network, capable of defeating the human Go champion [4]. However, when we talk about AGVs, there is still a long way to go until they learn to drive fully autonomously.

> “For learning research to make progress, important subproblems have to be isolated and studied, but they should be subproblems that play clear roles in complete, interactive, goal-seeking agents, even if all the details of the complete agent cannot yet be filled in.” [23, chap.1 Introduction]

According to this quote related to RL research, we must not address the problem of AGVs as a whole, as it would be too difficult to find significant results. A good and isolated subproblem is the control of AGV traffic light intersections. The intersections can have different complexity levels. One possibility is to vary the number of roads that meet at each intersection or the number of lanes on each street. Combining more complex intersections, like roundabouts or multiple junctions, is possible. However, they all have in common that multiple vehicles might want to cross the same area simultaneously. So it is desirable to develop a system capable of training models to handle any intersection, assuming that every vehicle on the road is autonomous (driven by a computer).

1.2 Objectives

The hypothesis in this study is whether RL is a suitable methodology for representing control systems in dynamic environments such as AGV traffic light intersections. To check that hypothesis, we must study and implement state-of-the-art RL algorithms. The agent of the RL models must be an intersection that changes the lights of the semaphores between green and red. If the traffic light is green, the vehicle must pass; if it is red, the vehicle must stop. The best system is the one capable of achieving the highest traffic flow while keeping collisions from happening.

1.3 Document structure

Additionally to this introduction, this thesis contains five more chapters. Chapter 2 references some valuable knowledge about Reinforcement Learning to allow a better understanding of the problem and solution found. Then, Chapter 3 presents previous work developed in this field. Chapter 4 formalises the problem, proposes a solution, and explains the solution's methodology. Afterwards, in Chapter 5, there is a presentation and discussion about the results obtained. Finally, Chapter 6 makes some conclusions about the results and findings of this project.

Chapter 2

Reinforcement Learning

2.1 Introduction

According to Sutton et al. [23], **Reinforcement Learning (RL)** is a machine learning paradigm in which agents learn what to do by interacting with an environment by trial and error. This idea comes from the concept of human behaviour: when a person acts, the environment reacts. The authors state that "exercising this connection produces information about cause and effect, consequences of actions, and what to do to achieve goals" (Figure 2.1).

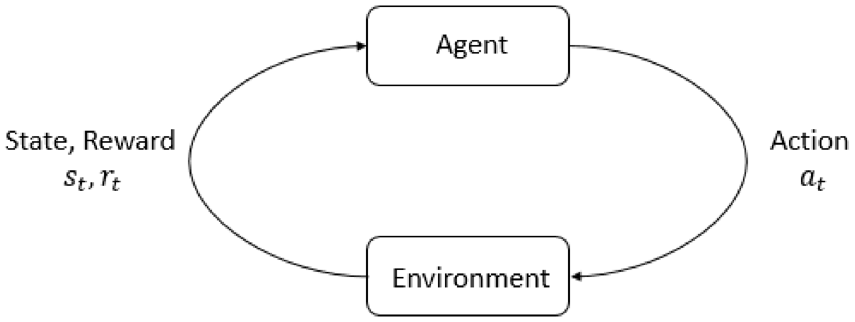


Figure 2.1: Agent-environment interaction loop, from [1].

How do agents know what to do and whether or not they are achieving their goals? The agents "learn what to do so as to maximise a numerical reward signal." The agent must try new actions to discover which ones result in the most significant reward. Those actions might influence not only the immediate reward but also the rewards of all future actions. RL's two most important features are **trial and error** and **delayed rewards**.

In machine learning, there are two other paradigms (supervised and unsupervised learning) that must not be taken as being the same as RL. Both supervised and unsupervised learning do not care about achieving goals through trial and error. Because of that, one of the challenges unique to RL is the trade-off between **exploration** and **exploitation**. In order to get more significant and accurate rewards, the agent must exploit the best actions taken, but to find them out before, it also has to explore new actions. Mathematicians have intensively studied the exploration-exploitation dilemma for many decades, yet it still needs to be solved.

2.2 Main elements of an RL system

Besides the **agent** and the **environment**, there are four main elements of an RL system:

- **Policy** - Consists of mapping from perceived **states** of the environment to **actions**. This is an RL agent's core, sufficient to determine its behaviour.
- **Reward signal** - Defines the goal of the RL problem. In each step, the environment sends the agent a numerical value called *reward*. The agent's goal is to maximise its total rewards over time. The reward signal is the main reason to update the policy during training; if a low reward value follows an action, then the agent should do it less often and vice versa.
- **Value function** - Values are associated with states: a high-value state is more likely to result in higher rewards later than a low-value state. It defines the desirability of each state and can be challenging to accurately estimate and re-estimate them during the training phase of the agent.
- **Model** of the environment - The model is an optional element of an RL system. It consists of a set of rules defined in the environment so that agents can predict how the environment will behave. The presence of a model lets the agent **plan** a course of action without actually experiencing all possible states and actions. Methods that use models are called **model-based** methods, opposing **model-free** methods that are entirely trial and error learners.

These elements will be further detailed in Section 2.4 while explaining Markov Decision Processes.

2.3 Kinds of RL algorithms

Achiam [1] has done a "non-exhaustive, but useful taxonomy of algorithms in modern RL" that one can use to compare multiple RL algorithms (Figure 2.2). According to him, "one of the most important branching points in an RL algorithm is the question of whether the agent has access to (or learns) a model of the environment." This access to a model divides the algorithms into two groups called **model-based** and **model-free**, as seen in Section 2.2.

The main upside of model-based algorithms is that the agent can plan its actions, knowing the multiple options and what will happen by choosing each one. When the agent fully knows the model of the environment, it can result in considerable improvements in sample efficiency.

The main downside of model-based algorithms is that the whole model of the environment is not available to the agent. If an agent wants to use a model, it has to learn it from experience, which creates several challenges. The biggest challenge is that the agent can perform well in the

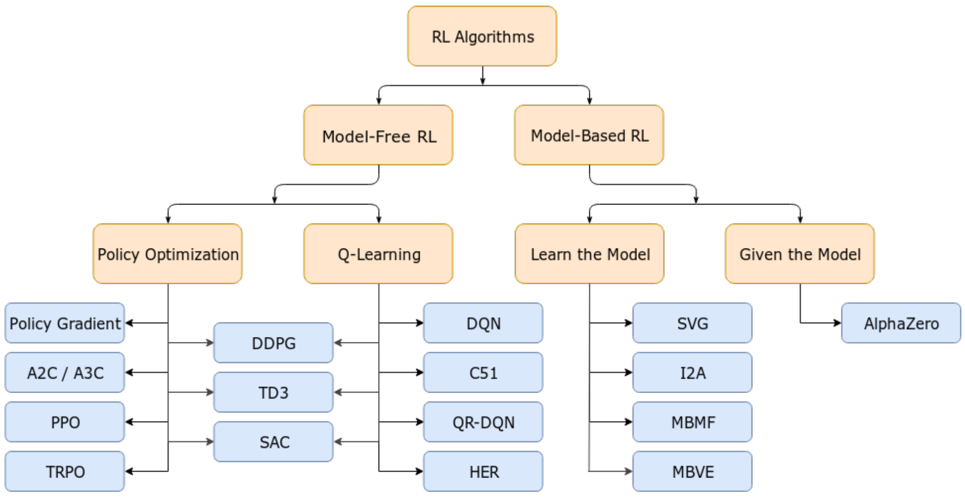


Figure 2.2: Taxonomy of algorithms in modern RL, from [1].

learned model of the environment but terribly in the real world by exploiting bias on the learned model.

Regarding model-free methods, despite losing the sample efficiency from not knowing a model of the environment, these methods are usually easier to implement and tune. There are two main approaches for model-free methods, **policy optimisation** and **q-learning**. Without going deeper into details about each of them for now, one must know the main trade-offs between policy optimisation and q-learning.

On the one hand, policy optimisation methods are known to be more stable and reliable as they only use the most recent actions to actively update the policy used by the agent. This way, the agent is always acting according to the latest policy. On the other hand, q-learning methods are known to be substantially more sample efficient as they do not directly update the agent's policy. Each update can reuse data collected at any time, regardless of how the agent was exploring the environment then. According to [20], policy optimisation methods are better suited for problems with continuous action spaces and q-learning methods for problems with discrete action spaces.

In the following sections (2.5 and 2.6), we will see further details about q-learning and deep q-networks methods, as those are the ones that will be used later in Chapter 4.

2.4 Markov Decision Process

According to Ding et al. [7] and Strook [22], **Markov Processes (MP)** follow the assumption of Markov chains where one state s_{t+1} is only dependent on the current state s_t and the action a taken (it is also common to represent the current state as s and the next state as s'). Equations 2.1 and 2.2 describe this property represented by the tuple of $< S,P >$, given a finite **state set** (S) and a **state transition matrix** (P). In Figure 2.3, one can see the graphical model of MP where t represents the time step and $p(s_{t+1}|s_t)$ represents the state transition probability from s_t to s_{t+1}.

$$p(s_{t+1}|s_t) = p(s_{t+1}|s_0, s_1, s_2, ..., s_t) \tag{2.1}$$

$$p(s_{t+2} = s'|s_{t+1} = s) = p(s_{t+1} = s'|s_t = s) \tag{2.2}$$

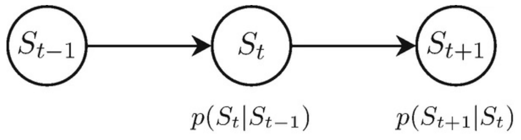


Figure 2.3: Graphical model of Markov Processes, from [7].

Using MP, there is no way for the agent to get feedback from the environment. In order to do so, **Markov Reward Processes (MRP)** extend MP from $< S,P >$ to $< S,P,R,\gamma >$, with R and γ representing the **reward function** and the **reward discount factor**, respectively. In this configuration, the reward function depends only on the current state, according to Equation 2.3. Once again, in Figure 2.4, one can see the graphical model of MRP.

$$R_t = R(S_t) \tag{2.3}$$

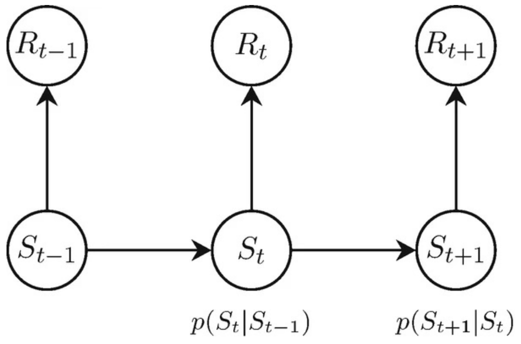


Figure 2.4: Graphical model of Markov Reward Processes, from [7].

Defining a **trajectory (τ)** as a set of steps the agent follows in sequence, the **return (G or $R(\tau)$)** of a finite process with T time steps is the sum of all rewards obtained in that trajectory (Equation 2.4).

$$G_{t=0:T} = R(\tau) = \sum_{t=0}^{T} R_t \tag{2.4}$$

However, steps closer to the end of the trajectory usually have more impact than distant ones. To represent that different impact is introduced the concept of **discounted return** (Equation 2.5), where a **reward discount factor** ($\gamma \in [0,1]$) is applied to reduce the impact of previous steps.

A discount factor is essential when dealing with infinite MRP so that the return value does not increase infinitely.

$$G_{t=0:T} = R(\tau) = \sum_{t=0}^{T} \gamma^t R_t \tag{2.5}$$

To achieve higher return values is helpful for the agent to know which states have higher expected return values. The **value function** ($V(s)$) defined by Equation 2.6 uses the estimated rewards obtained following the current policy (π) starting in the current state (s).

$$V(s) = \mathbb{E}[R_t | S_0 = s] \tag{2.6}$$

Markov Decision Processes (MDP) [11] complement the MRP with the concept of choosing an action (Figure 2.5). In this case, MDP is defined as a tuple of $< S, A, P, R, \gamma >$, and the state transition matrix (P) is defined by Equation 2.7. A represents the set of actions, and the immediate reward obtained at each step becomes defined by Equation 2.8.

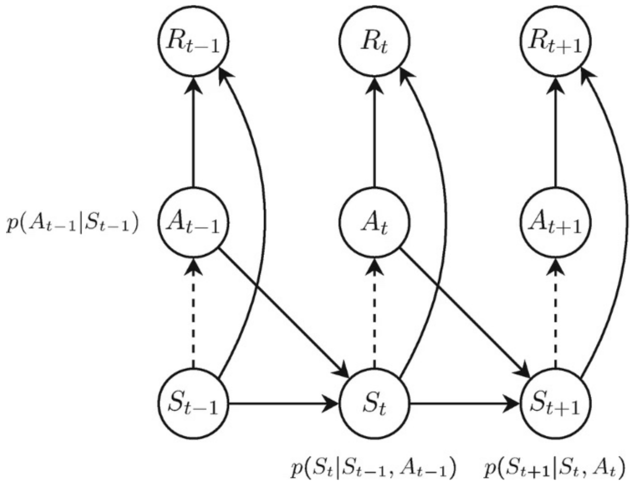


Figure 2.5: Graphical model of Markov Decision Processes, from [7]: the dashed lines and $p(A_t|S_t)$ indicate the action choice/decision process, also known as agent's policy (π).

$$p(s'|s,a) = p(S_{t+1} = s' | S_t = s, A_t = a) \tag{2.7}$$

$$R_t = R(S_t, A_t) \tag{2.8}$$

2.5 Q-Learning

According to Jang et al.[9] and to Vilarinho et al. [26], Q-Learning is a model-free RL method that allows agents to act optimally in Markovian domains, defined by the Markov Decision Process

(MDP). Q-learning is an off-policy method because it separates the learning policy from the acting policy using Equation 2.9 to calculate the Q-values of state-action pairs. The agent uses these values to select the best action to take at each state. In the same equation, $\alpha \in [0,1]$ is the learning rate used by the agent in each iteration of the q-values update, R is the reward obtained by taking action a at state s, and γ is the reward discount factor. This process must be repeated several times until it converges to optimal values used to solve the problem at hand.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.9}$$

2.6 Deep Q-Networks

With the evolution of Deep Learning, **Deep Q-Networks (DQNs)** emerged. They can extract essential features from the data without human handcrafting domain-specific features [19]. According to [9], DQNs combine Convolution Neural Networks (CNNs) with Q-learning. The key idea behind this method is to use experience replay to reduce the correlations between different states, actions, and rewards, that the CNN could exploit. A replay buffer stores samples from the states, actions, and rewards and every time the CNN learns, random samples are extracted from the replay buffer to the CNN. However, remember that using more samples at each step means it takes longer for the CNN to update its values.

In combination with the experience replay technique, a target network ($Q_{\theta'}$) is used separately from the Q network (Q_θ) to increase training stability. The first one must only be updated periodically, while the second must be continuously updated. Thus, we end up with a new formula for the updates of the Q-values (Equation 2.10), where we use the target network $Q_{\theta'}$ to get subsequent Q-values for the next state-actions and the current network Q_θ to get the current Q-value for the current state-action.

$$Q_\theta(s,a) \leftarrow Q_\theta(s,a) + \alpha[R + \gamma \max_{a'} Q_{\theta'}(s',a') - Q_\theta(s,a)] \tag{2.10}$$

However, suppose the size of possible states S is considerable. In that case, before the agent can learn enough information from the environment, it may get stuck to exploiting the already explored environment, even if better options have yet to be explored. A solution to this problem is to use **Double DQN (DDQN)** [19]. This method still has a target $Q_{\theta'}$ and a current Q_θ network. The difference is that we select which action is best to take from the Q network Q_θ, but we use the value from the target network ($Q_{\theta'}$) to update the Q network (Q_θ), as seen in Equation 2.11.

$$Q_\theta(s_t,a_t) \leftarrow (1-\alpha)Q_\theta(s_t,a_t) + \alpha(R_{t+1} + \gamma Q_{\theta'}(s', argmax_{a'} Q_\theta(s',a'))) \tag{2.11}$$

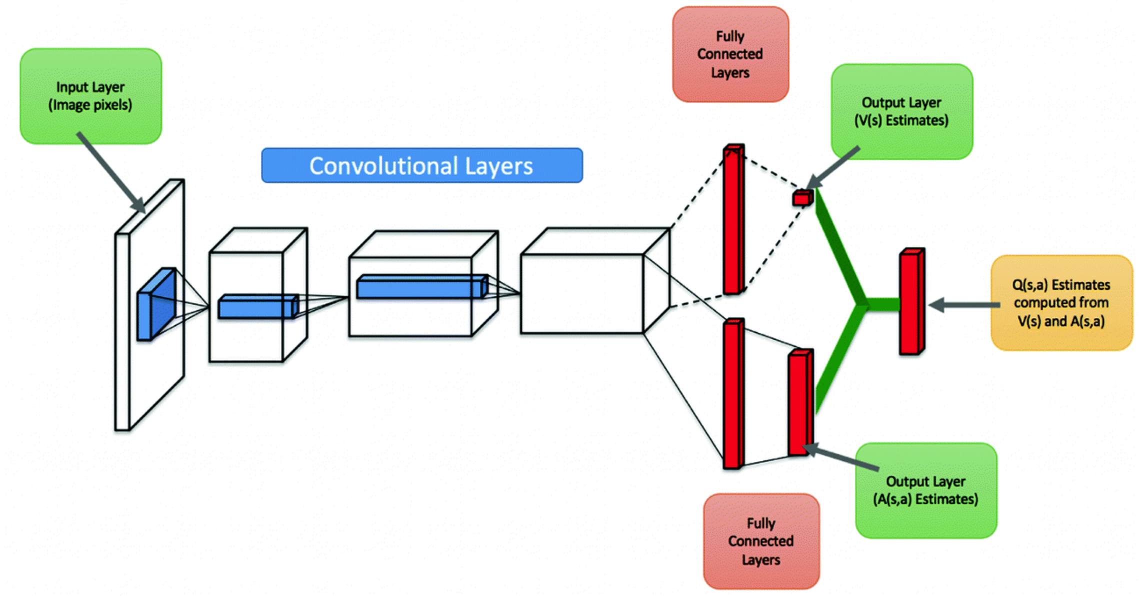

Input Layer (Image pixels)
Convolutional Layers
Fully Connected Layers
Output Layer (V(s) Estimates)
Q(s,a) Estimates computed from V(s) and A(s,a)
Output Layer (A(s,a) Estimates)
Fully Connected Layers

Figure 2.6: Schema of a Dueling Q-Network, from [19].

Finally, **Dueling DQN** [30] branches the neural network into two sub-networks (Figure 2.6). The first one corresponds to the "**Value**" function $V(s)$ of each state, and the second one to the "**Advantage**" function $A(s,a)$ that computes the advantage of each action over the base value of being in state s. This separation does training much quicker: it has a global value for each state, updated upon any action. To get the Q-values for each state action, we must use Equation 2.12. Being α the parameter vector of the "Advantage" sub-network, β the parameter vector of the "Value" sub-network, and θ the parameter vector of the convolutional layer that is common to both networks [19].

$$Q_{(\theta,\alpha,\beta)}(s,a) = V_{(\theta,\beta)}(s) + \left(A_{(\theta,\alpha)}(s,a) - \frac{1}{|A|}\sum_{a} A_{(\theta,\alpha)}(s,a) \right) \tag{2.12}$$

The previous Equation (2.12) could be as simple as $Q(s,a) = V(s) + A(s,a)$; however, identifiability would be lost. This means that the opposite would not be true despite being able to get the Q-value given the values of s and a. One could not retrieve the values of s and a from the given value of Q. The solution is to subtract the mean value of the advantage values for state s, given by $\frac{1}{|A|}\sum_a A_{(\theta,\alpha)}(s,a)$, from the advantage value of taking action a in state s.

2.7 Summary

The current chapter introduces and explains the concept of Reinforcement Learning (RL). In the first section, we saw that RL's essential and distinguishable features are trial and error and delayed rewards. One of the unique challenges to RL methods is the trade-off between exploration and exploitation. Then, the main elements of an RL system (agents, environment, policy, reward signal, value function, and model) were introduced in Section 2.2, followed by a small comparison between model-based and model-free methods and policy optimisation (or on-policy) and q-learning (or off-policy) methods in Section 2.3. Afterwards, in Section 2.4, we explained Markov Decision Processes (MDP) as they are helpful to formalise RL problems. Finally, in Section 2.5, we briefly explained what q-learning methods are, and in Section 2.6, one can understand how deep learning is used in combination with q-learning to get better results.

Chapter 3

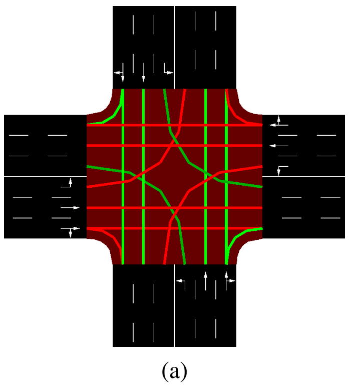
Related Work

3.1 Introduction

There are multiple articles available covering the problem of **Traffic Signal Control (TSC)** using RL. This chapter will cover some of them, looking at the algorithms, state representation, action space, reward functions, and metrics used to evaluate such systems.

The main difference between these solutions and the one developed in this dissertation is that we consider only AGVs, while others believe that human drivers are behind the wheel. The main difference between these two approaches is in the action space – conflicting green lights turned on simultaneously are not desirable when humans drive vehicles. However, when talking about AGVs, because their movement is predictable, we can have conflicting green lights turned on simultaneously without causing accidents and ideally providing higher traffic flow.

Before going into further details about some projects, it is essential to understand the concept of traffic light phases commonly used in TSC problems. A phase is what we call the state of a traffic light. If a traffic light, for example, is said to have two phases, it means that there are two possible combinations of green and red lights in that intersection. Following this example of a two-phase traffic light (Figure 3.1), one phase (a) would be green light in the north and south directions and red light in the east and west directions. The second phase (b) would be the opposite.


(a)

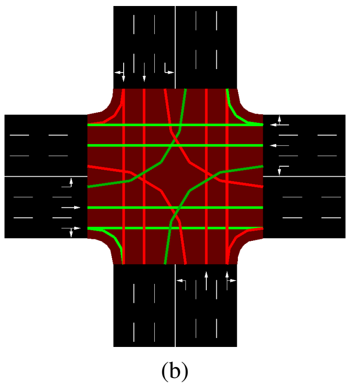

(b)

Figure 3.1: Traffic light with 2 phases.

3.2 The Alan Turing Institute's TSC

The authors of [2] developed a TSC system using the leading RL approaches to date. Their goal was to compare those approaches to the ones used by commercial systems such as Fixed-Cycle [13] (seen in most traffic signals), MOVA [27], and SURTRAC [21] in two scenarios:

- "Cross straight", where there is an intersection with four two-way one-lane roads where vehicles can only go straight through the intersection (Figure 3.2);
- "Cross triple", where there is an intersection with four two-way three-lane roads with one dedicated left turn lane (Figure 3.3).

From now on, the authors will be referred to as *the Alan Turing Institute*, as this is one of their funding institutions.

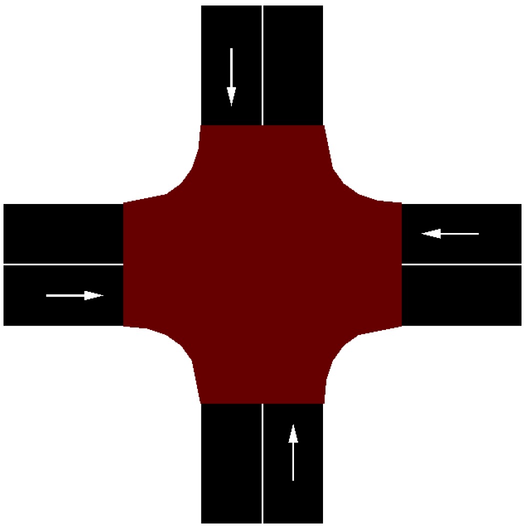


Figure 3.2: Intersection with four two-way one-lane roads where vehicles can only go straight through the intersection.

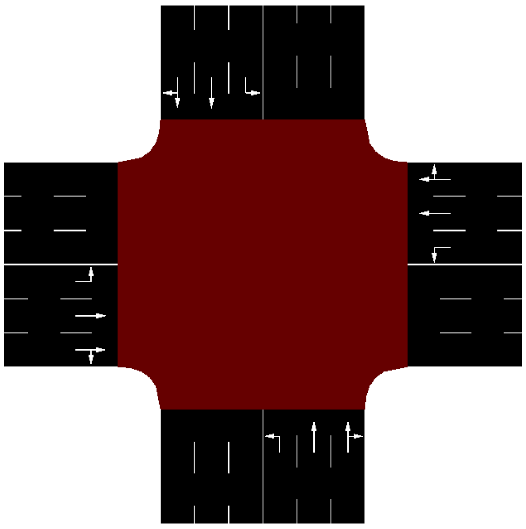


Figure 3.3: Intersection with four two-way three-lane roads with dedicated left turn lane.

3.2.1 Algorithms

The Alan Turing Institute's TSC (ATI-TSC) proposed solution uses three different methods: Double DQN (DDQN), Dueling Double DQN (DDDQN) and Advantage Actor Critic (A2C). The first two algorithms were used in combination with Prioritised Experience Replay, a form of experience replay (seen in Section 2.6) that optimises the sample selection from the replay buffer by prioritising the temporal-difference error of the samples [29]. All the implementations use the same state representation, action spaces, and reward functions that we will see in the following subsections.

3.2.2 State Representation

ATI-TSC used a state vector with a dimension equal to the number of incoming lanes of the intersection plus one. Each vector component represents the queue length in meters of each lane. The last component is the current light's phase. For example, an intersection with four two-way three-lane roads will have a vector with size 13 $(3lanes * 4roads + 1)$.

3.2.3 Action Space

The action space used by the ATI-TSC varies according to the complexity of the intersection. The main idea is that each action corresponds to the light phase of the traffic signal.

In the "cross straight" scenario, there are two possible phases: green for north and south traffic (and red for the other ones) or green for the east and west traffic (and red for the other ones).

Two experiments were done in the "cross triple" scenario, one with four phases and the other with eight. In the four phases experiment, each phase allows traffic to flow from one road at a time, and the other three get red light traffic signals. In the eight phases experiment, four additional stages allow for non-conflicting cross traffic (for example, green lights to go straight through and to turn right on north and south roads, and red lights for all other lanes, including to turn left on the same roads).

The change between phases does not have to follow any particular order. At each step, the agent can select any stage it desires.

3.2.4 Reward Function

According to ATI-TSC, the same value of the queue length in meters used for the states is a reasonable choice for the reward function. It can transmit information about the mean rate of delay of the system. The following Equation (3.1) is the reward function used, where the reward of each step is the negative sum of the length in meters of the lanes' queues.

$$R = -\sum_{lane} queue\_length_{lane} \tag{3.1}$$

3.2.5 Evaluation Metrics

To evaluate and compare the performance of the algorithms, the Alan Turing Institute selected three metrics:

- The global accumulated delay of all vehicles (in seconds) during the simulation;
- The average sum of queue lengths (in meters) from all roads at all simulation steps;
- The average of the maximum queue length (in meters) at all simulation steps.

3.2.6 Overview

Table 3.1 is an overview of the previous subsections, where one can see the algorithms, state representation, action space, reward function, and evaluation metrics ATI-TSC used in their experiences.

Table 3.1: Overview of the Alan Turing Institute's work.

	Cross straight	**Cross triple**
Algorithms	DDQN, DDDQN, A2C	
State Representation	queue lengths, current phase	
Action Space	2 phases	4 or 8 phases
Reward Function	$R = -\sum_{lane} queue\_length_{lane}$	
Evaluation Metrics	global accumulated delay (s) average sum of queue lengths (m) average max queue length (m)	

3.2.7 Results

According to the Alan Turing Institute, every RL approach (DDQN, DDDQN, and A2C) outperformed the commercial systems (Fixed-Cycle and MOVA). In the "cross straight" experiment, the gap between the solutions was not so obvious. Still, in the "cross triple" experiments, the RL approaches reduce the vehicles' cumulative delay to less than half. However, some things must be considered. For the MOVA system to achieve its full potential, it should be fine-tuned by a traffic engineer for each intersection and traffic pattern. The RL approaches were also not fine-tuned. This means that both commercial and RL techniques can achieve better results. That being said, DDDQN was the algorithm that performed the best.

3.3 IntelliLight

According to [32], IntelliLight is "a Reinforcement Learning Approach for Intelligent Traffic Light Control". The authors developed an RL method to control traffic lights, tested it against real-world traffic data gathered from surveillance cameras and showed case studies of policies learned from that data. It defends that in real-world scenarios, the reward obtained by the RL agents is not enough to produce the best policy. Different policies can create the same reward. However one can be more suitable than the other if, for example, it has fewer phase changes.

3.3.1 Algorithms

To train the agent, IntelliLight uses a Deep Q-Network (as seen in Section 2.6), represented in Figure 3.4. To improve the results, they also used two techniques complementing DQN, Memory Palace and Phase Gate.

Memory Palace (MP) is a form of experience replay (Section 2.6) where they use different palaces to store samples of states, actions, and rewards from different phase-action combinations. As we will see in the following subsections, two phases and two actions are available, so we get four palaces to store the samples. Then, the agent is trained with an equal number of samples from each palace.

Phase Gate (PG) is a technique that separates the fully connected (FC) layer in two, creating one Q-Network for each phase. This technique then uses a selector to tell the agent which network to use when getting the Q-value of an action.

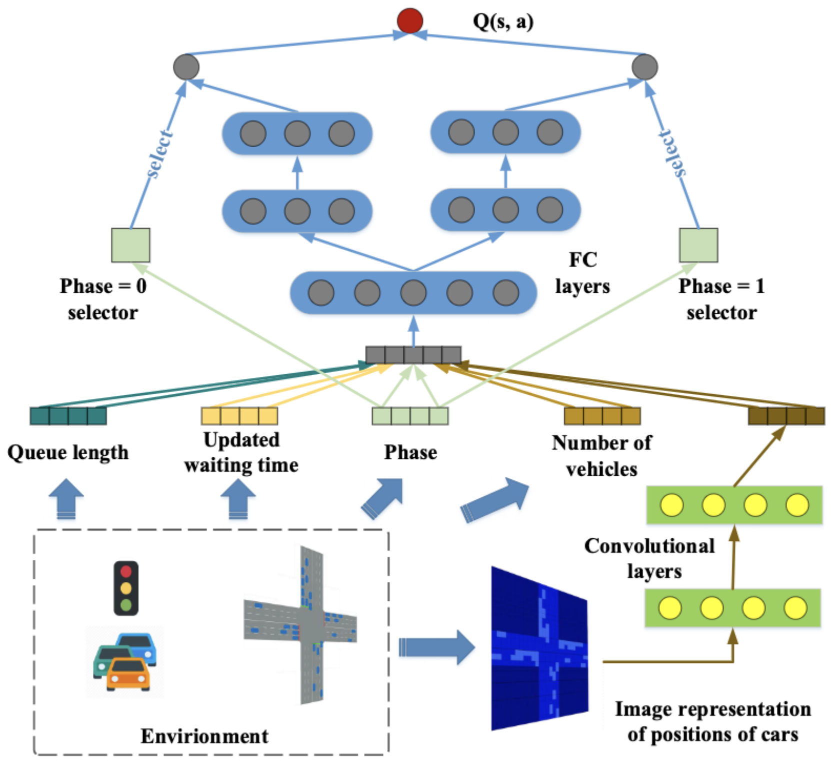


Figure 3.4: IntelliLight's Q-network representation, from [32].

3.3.2 State Representation

For each lane i at an intersection, the state is defined by:

- L_i - queue length;
- V_i - number of vehicles;
- W_i - waiting time of the vehicles.

In addition to these, the state includes a bird-view image of the intersection (M), the current light phase (P_c) and the next light phase (P_n). A graphical representation of the state is seen as the input for the Q-network in Figure 3.4.

3.3.3 Action Space

The experiments conducted by IntelliLight only consider traffic lights with two phases available: green light for traffic from north and south (and red for the other ones) and green light for traffic from east and west (and red for the other ones). Two possible actions are available:

- $a = 1$ (change phase);
- $a = 0$ (keep the same phase).

3.3.4 Reward Function

The reward function, as shown in Equation 3.2, is composed of a weighted sum of six factors:

- $\boldsymbol{L_i}$ is the total number of waiting vehicles on lane i. A vehicle is waiting if its speed is under 0.1 m/s.
- $\boldsymbol{D_i}$ is the delay of each lane. The delay is defined in Equation 3.3, where the lane speed is the average speed of the vehicles, and the speed limit is the maximum speed allowed on that lane.
- $\boldsymbol{W_i}$ is the sum of the waiting times of all vehicles $j \in J_i$ in each lane (Equation 3.4). The waiting time of each vehicle at each time step ($W_j(t)$) is equal to the waiting time of the previous time step plus one if the vehicle is standing still (speed under 0.1 m/s) or zero if the vehicle is moving at least at 0.1 m/s (Equation 3.5).
- $\boldsymbol{C}$ indicates if the light phase changed or not. $C = 0$ for keeping the same phase; $C = 1$ for changing the phase.
- $\boldsymbol{N}$ is the number of vehicles that passed the intersection since the last agent action a.
- $\boldsymbol{T}$ is the total travel time (in minutes) of the vehicles that passed the intersection since the last agent action a.

The reward coefficients follow the values in Table 3.2.

$$R = w_1 * \sum_{i \in I} L_i + w_2 * \sum_{i \in I} D_i + w_3 * \sum_{i \in I} W_i + w_4 * C + w_5 * N + w_6 * T \tag{3.2}$$

$$D_i = 1 - \frac{lane\ speed}{speed\ limit} \tag{3.3}$$

$$W_i(t) = \sum_{j \in J_i} W_j(t) \tag{3.4}$$

$$W_j(t) = \begin{cases} W_j(t-1)+1, & \textit{vehicle speed} < 0.1 \\ 0, & \textit{vehicle speed} \geq 0.1 \end{cases} \tag{3.5}$$

Table 3.2: Reward coefficients.

w_1	w_2	w_3	w_4	w_5	w_6
-0.25	-0.25	-0.25	-5	1	1

3.3.5 Evaluation Metrics

To evaluate and compare the performance of the different methods, the authors of IntelliLight use the following [32]:

- **Reward** - average reward value over time (higher is better);
- **Queue length** - average sum of all lanes' queue lengths over time (lower is better);
- **Delay** - average sum of all lanes' delay over time (lower is better);
- **Duration** - average time vehicles spend (in seconds) on approaching lanes (lower is better).

3.3.6 Overview

Table 3.3 is an overview of the previous subsections, where one can see the algorithms, state representation, action space, reward function, and evaluation metrics that the authors of IntelliLight used in their experiences.

Table 3.3: Overview of IntelliLight's work.

Algorithms	DQN + MP + PG
State Representation	queue length waiting time of the vehicles number of vehicles current and next phases bird-view image of the intersection
Action Space	2 actions: keep/change phase
Reward Function	$R = w_1 * \sum_{i\in I} L_i + w_2 * \sum_{i\in I} D_i + w_3 * \sum_{i\in I} W_i +$ $+ w_4 * C + w_5 * N + w_6 * T$
Evaluation Metrics	reward, queue length, delay, duration

3.3.7 Results

To test their system, the authors of IntelliLight use three baseline methods in addition to the ones mentioned in Subsection 3.3.1:

- Fixed-time Control (FT): uses pre-determined cycles and phase time plans [14].
- Self-Organizing Traffic Light Control (SOTL): changes the traffic light phase according to the number of vehicles waiting at the red lights [5].
- Deep Reinforcement Learning (DRL): applies DQN relying only on images of the traffic state [24].

According to all the experiments conducted, IntelliLight significantly reduced queue lengths and delays of traffic and the time needed to clear all traffic (duration of the experiment).

3.4 CTLCM

The authors of [18] conducted a "**C**omparison of Q-Learning based **T**raffic **L**ight **C**ontrol **M**ethods and Objective Functions" (CTLCM). They analyse the performance of DQN, DDQN, Dueling DQN and DDDQN methods to control traffic lights in a single intersection environment. The intersection is composed of four one-lane two-way roads where traffic can go straight through or turn right (Figure 3.5).

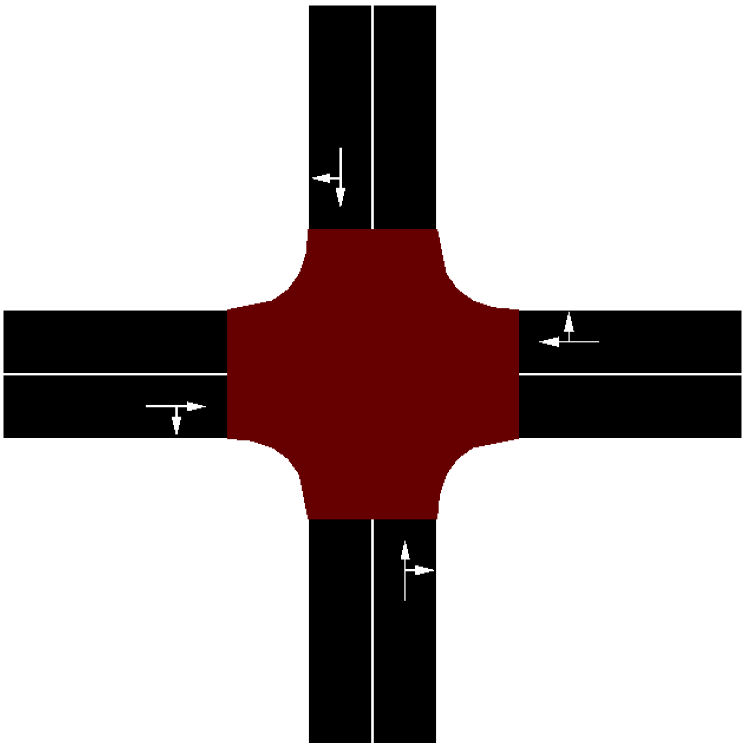


Figure 3.5: Intersection with four one-lane two-way roads without left turn.

3.4.1 Algorithms

The authors of CTLCM selected four algorithms to conduct their comparisons: DQN, DDQN, Dueling DQN and DDDQN. More details about these algorithms can be found in Section 2.6, except for DDDQN, which is a combination of DDQN and Dueling DQN methods.

3.4.2 State Representation

The authors of CTLCM placed detectors 50 meters behind the traffic light on each road and retrieved three kinds of information from the simulator:

- Number of vehicles in the detector area;
- Percentage of the area occupied by vehicles;
- The average speed of the vehicles in the area.

This information is combined with the current phase of the traffic light to represent each state.

3.4.3 Action Space

There are two possible actions in the scenario of Figure 3.5: green for north/south, red for east/west directions, and vice-versa.

3.4.4 Reward Function

The authors of CTLCM developed six different reward functions:

- **WTMin**: Waiting Time Minimization. A vehicle is waiting if travelling at speed under 0.1 m/s.
- **ASMax**: Average Speed Maximization.
- **TTMin**: Travel Time Minimization.
- **CTMax**: Completed Trip Maximisation. A vehicle completes its trip when it leaves the simulation at the end of its route.
- **WCTMax**: Weighted Completed Trip Maximization. Similar to the previous with the addition of weights according to the vehicle's route length.
- **TMQMax**: Throughput Minus Queue Minimization. It is the difference between the number of vehicles joining the simulation and the number of vehicles waiting at each step.

3.4.5 Evaluation Metrics

To evaluate the performance of each model, the authors of CTLCM used four metrics:

- Rewards earn during training;
- Overall waiting time;
- The overall number of waiting vehicles;
- Overall CO2 emissions during testing.

3.4.6 Overview

Table 3.4 is an overview of the previous subsections, where one can see the algorithms, state representation, action space, reward function, and evaluation metrics that the authors of CTLCM used in their experiences.

Table 3.4: Overview of CTLCM's work.

Algorithms	DQN, DDQN, Dueling DQN, DDDQN
State Representation	number of vehicles in the detector area percentage of area occupied by vehicles average speed of vehicles traffic light phase
Action Space	2 actions
Reward Function	WTMin, ASMax, TTMin, CTMax, WCTMax, TMQMax
Evaluation Metrics	reward, queue length, delay, duration

3.4.7 Results

After analysing the results carefully, the best method (on average) is DDDQN, followed by DDQN. Concerning the reward functions, it is possible to take two conclusions: WCTMax results in less CO2 emissions and fewer waiting vehicles, and although TMQMax is the one with the most emissions, it is by far the one with the less waiting time.

3.5 Summary

In this chapter, we have seen three approaches to solving the TSC problem. The main difference between this problem and the one formalised in the next chapter is that here we consider that human drivers are behind the wheel. So the action spaces only assume phases without conflicting green lights (usually between two and eight actions are available, depending on the complexity of the intersection and the method used to change between phases).

After briefly analysing ATI-TSC, IntelliLight, and CTLCM approaches to solve the same problem (Table 3.5), there are two things they all agree on. The first is that state representation must have the traffic queue lengths and the current phase of the traffic lights. The second is that the delay of vehicles and the sum of queue lengths in each step are good metrics to evaluate the performance of the systems. Another important fact is that they all use an action space as little as possible. They do that because smaller action spaces are faster for agents to train by requiring fewer samples.

Regarding the algorithms used, Convolutional Neural Networks, used in DQN-based methods, are the way to obtain the best results. When analysing the state representation of each solution, only IntelliLight used bird-view images of the intersections. It increases the memory used by the algorithms but provides more information to the agents. Combined with other features like queue length and the current traffic light phase, it optimises the policy learned by the agent. Finally, when looking at the reward functions of the methods, there is no right way to go. It depends on the main goals of the project.

It is impossible to state that one approach is better than the others; each has its own goals, and each has accomplished them. The important value to take from these projects is that DQN methods can make real improvements in traffic management when compared to traditional approaches.

Table 3.5: Overview of Related Work.

		ATI-TSC	**IntelliLight**	**CTLCM**
Algorithms	DQN		X	X
	DDQN	X		X
	Dueling DQN			X
	DDDQN	X		X
	A2C	X		
	MP		X	
	PG		X	
State	number vehicles		X	X
	average speed			X
	queue length	X	X	X[a]
	waiting time		X	
	image		X	
	current phase	X	X	X
Action Space	2 actions	X	X	X
	4 actions	X		
	8 actions	X		
Reward	queue length	X		
	average speed			X
	waiting vehicles		X	X
	waiting time		X	X
	delay		X	
	vehicles throughput		X	X
	travel time		X	X
	phase change		X	
	completed trips			X
Evaluation	reward		X	X
	delay	X	X	X
	queue length	X	X	X
	simulation duration		X	X

[a]CTLCM does not use queue length in its state representation but uses the percentage of area occupied by vehicles which is a similar indicator.

Chapter 4

Intelligent Traffic Light Management for AGVs

This project aims to achieve Intelligent Traffic Light Management for AGV intersections (Figure 4.1). When replacing human drivers with computers controlling vehicles, new opportunities arise. There is no longer the need to use the typical yellow lights to let humans know they must slow down because the red light is coming soon. If the AGV receives a red signal (can be visual or through direct communication with the intersection controller), it will slow down; if it gets a green signal, it will proceed within its path. Another advantage of using AGVs is lower reaction times (humans need hundreds of milliseconds to start their reaction to a visual stimulus [28], compared to computer processing times in the nanoseconds range), so it is possible to change between light phases more often. Finally, computers are more predictable drivers than humans.

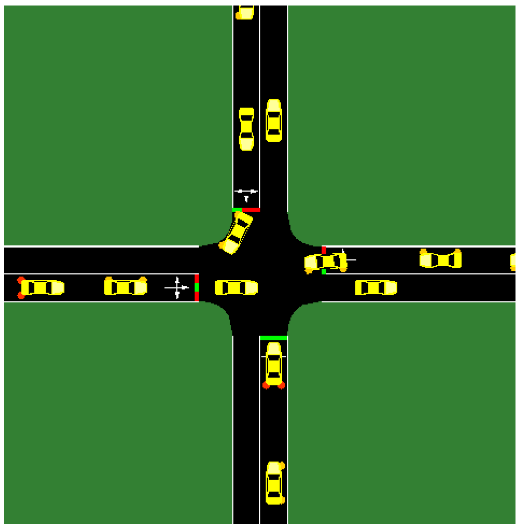


Figure 4.1: Example of an AGV intersection from SUMO [12].

4.1 Problem Formalisation

The selected approach to solving the problem of Intelligent Traffic Light Management for AGVs is to use DQN-based methods like those seen in Chapter 2 and used successfully by the authors of the projects seen in Chapter 3.

In the first place, DQN is an RL method, and therefore the problem must be defined as an MDP (Section 2.4). To do so, the tuple $< S, A, P, R, \gamma >$ is defined by the following attributes:

- **Environment States** (S) — Because we control a traffic intersection, the number of possible different states is so large that we can consider it infinite. It takes just one vehicle to be positioned a couple of centimetres away to be regarded as a different state of the environment. To represent such states, we use the following features:
  - An RGB bird view image of the intersection with 256x256 pixels corresponding to 50x50 meters of the simulator centred on the middle point of the intersection (Figure 4.2).

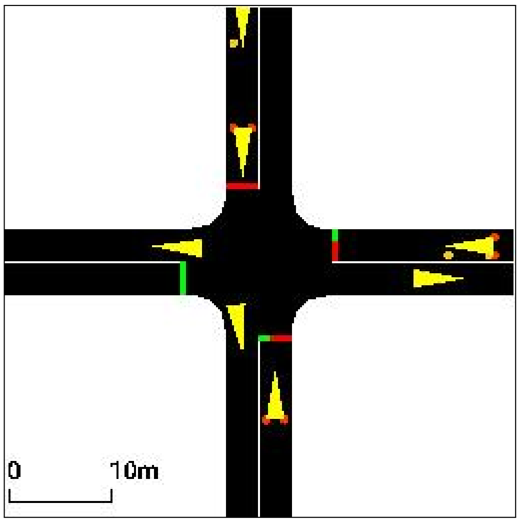
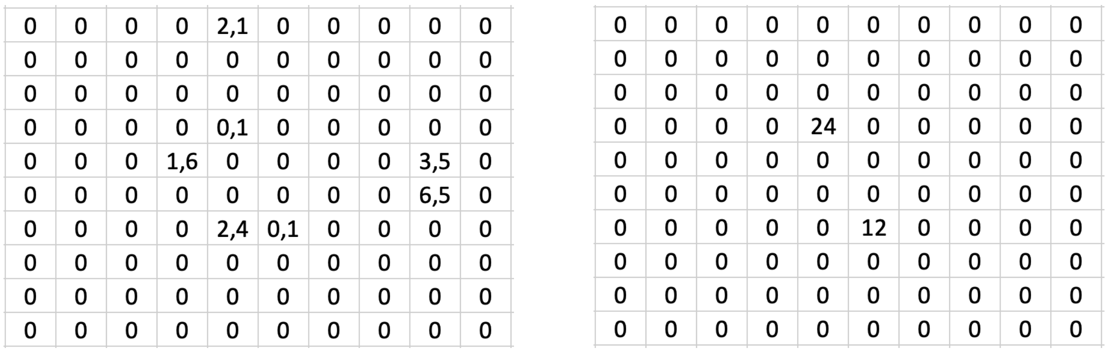


Figure 4.2: RGB image of an intersection with 256x256 pixels.

  - That image is complemented with two 256x256 matrices:
    * The first contains information about the speed of each vehicle. That information is positioned in the matrix in the position of the front of the vehicle in the RGB image (Figure 4.3 a).
    * The second contains information about the waiting time of each vehicle. The position of that information follows the same logic as the speed matrix (Figure 4.3 b). A vehicle is considered to be waiting if its speed is under 0,1 m/s.
  - Finally, four arrays, each one with information about the following features:
    * The percentage of space occupied for each lane.
    * The number of vehicles halting (with speed under 0,1 m/s) for each lane.



0	0	0	0	2,1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0,1	0	0	0	0	0
0	0	0	1,6	0	0	0	0	3,5	0
0	0	0	0	0	0	0	0	6,5	0
0	0	0	0	2,4	0,1	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(a)

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	24	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	12	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

(b)

Figure 4.3: Representative example of speed and waiting times matrices.

  * The sum of the vehicle's halting times for each lane.
  * The number of vehicles on each lane.

- **Action Space** (A) — For the action space, we use an integer value indicating the next phase of the traffic light. The phases are the different possible combinations for the green/red lights of the traffic signals and are predefined for each experiment.

- **State Transition Matrix** (P) — This project uses a traffic simulator named SUMO [12]. We do not directly control the environment; that simulator controls every transition between states. The code that is running on that simulator is what indirectly defines the state transition matrix. It follows the assumption of Markov chains, where one state s_{t+1} is only dependent on the current state s_t and the action a taken (Section 2.4).

- **Reward Function** (R) — The main goal of the reward function is to penalise collisions between vehicles while keeping a high traffic flow. To achieve the goal, six parameters define the reward function (Equation 4.1 and Table 4.1):

  – ***S***: The average speed in m/s of all vehicles in oncoming lanes (that have not passed through the intersection yet). This is one of the most critical factors of the reward function because higher speeds directly lead to higher traffic flow.

  – ***L***: The number of vehicles that left the intersection in that time step. It measures the throughput of the intersection. Higher throughput leads to more increased traffic flow.

  – ***W***: The sum of waiting times from all vehicles in oncoming lanes. A vehicle is waiting if its speed is below 0.1 m/s. It prevents the agent from allowing only one lane to pass through the intersection at high rates while keeping all the others at a standstill.

  – ***I***: The number of vehicles in oncoming lanes. If many vehicles accumulate in oncoming lanes, the traffic signal must allow more vehicles to go through.

  – ***P***: Has no value if the same traffic light phase is kept between the previous and the current steps. Otherwise has the value of 1 (that is multiplied by w_5, Equation 4.2).

Although not directly related to traffic flow, keeping the same phase between steps allows vehicles to achieve higher speeds and traffic to keep moving.

- ***C***: It has no value if there are no collisions in that step. Otherwise has the same value as S (average speed of the vehicles, Equation 4.3). In the case of collision, the value of C depends on the value of S because of two factors. If $w_6 * C$ had a constant value like -100, as long as vehicles kept going at high speeds (usually between 7 and 9 m/s), the final reward would still be positive, and that would be considered a good state (which is not). A solution would be to set $w_6 * C$ to a higher value, like -1000. But then accidents at lower speeds (between 1 and 3 m/s) would be brutally penalised, impacting not only that state but also a few dozen states before (thanks to the reward discount factor in use). At the same time, accidents at high speeds would have a slight negative reward. The second factor is that accidents at higher speed cause more damage than accidents at lower speeds, so the first ones should be more penalised. In the end, making this penalty grow linearly with the average speed of the vehicles is a sensed choice.

$$R = w_1 * S + w_2 * L + w_3 * W + w_4 * I + w_5 * P + w_6 * C \tag{4.1}$$

$$P = \begin{cases} 1, & last\_phase \neq current\_phase \\ 0, & last\_phase = last\_phase \end{cases} \tag{4.2}$$

$$C = \begin{cases} S, & collision = True \\ 0, & collision = False \end{cases} \tag{4.3}$$

Table 4.1: Reward coefficients.

w_1	w_2	w_3	w_4	w_5	w_6
10	10	-1	-1	-10	-100

- **Reward Discount Factor** (γ) — The selection of the reward discount factor is complex. As seen in Section 2.4, it is a value between 0 and 1, and the purpose is to specify how much previous actions influenced the reward of the current state. Changing the phase of the traffic light will have a direct impact on immediate future rewards, but not distant ones. The best example to prove it is when collisions between vehicles happen. The traffic light phase (or agent's action) that influenced a collision is the one that occurred between three and five steps before the crash. The actions taken five to ten steps before can still have some influence, but all the actions taken ten steps or more before the collision have little to no impact on that event. After carefully looking at Table 4.2, **0.90** is the value chosen for all our experiments. Values higher than that would give too much importance to earlier events (t-7, t-8, ...) and values lower than that would not give enough importance to later events (t-5, t-4, ...).

Table 4.2: How much rewards from past action-states (t-n) influence the current action-state (t) depending on the value of γ and according to Equation 2.5.

γ	t	t-1	t-2	t-3	t-4	t-5	t-6	t-7	t-8	t-9	t-10
0.95	1.00	0.90	0.86	0.81	0.77	0.74	0.70	0.66	0.63	0.60	0.57
0.90	1.00	0.81	0.73	0.66	0.59	0.53	0.48	0.43	0.39	0.35	0.31
0.85	1.00	0.72	0.61	0.52	0.44	0.38	0.32	0.27	0.23	0.20	0.17

Finally, the solutions are evaluated by letting the agents act in a traffic setup they have never seen before and measuring the obtained rewards, the number of accidents, and the throughput of the intersection during 3,600 steps (each step corresponds to one second of simulation, totalling one hour).

4.2 Methodology

The proposed solution to check whether RL is or not a suitable methodology to control AGV traffic light intersections consists in developing three different DQN-based agents, using **DQN**, **Double DQN**, and **Dueling DQN** (Section 2.6). Those agents are trained for 100,000 steps in 6 different scenarios (Subsection 4.2.1), following the Training Pipeline Loop (Subsection 4.2.2). Afterwards, a final testing loop of 3,600 steps is performed to evaluate the trained agents (Subsection 4.2.3).

4.2.1 Scenarios

A total of 6 different scenarios were developed to train and test each agent. All the scenarios consist of an intersection with four two-way roads controlled by a traffic light where only AGVs with the same characteristics are allowed to go around. The differences between the scenarios are divided into two groups:

- **Number of lanes**: Regarding the number of lanes on the roads, there are two possibilities (figures 4.4 and 4.5). In Figure 4.4, there is an intersection where all the roads have **one lane** in each direction. That one lane is responsible for traffic going in all three possible directions (left, front, and right). Figure 4.5 shows an intersection where all roads have **three lanes** in each direction. The left lane handles traffic turning left, the middle lane only handles traffic going straight, and the right lane handles traffic going straight and turning right.

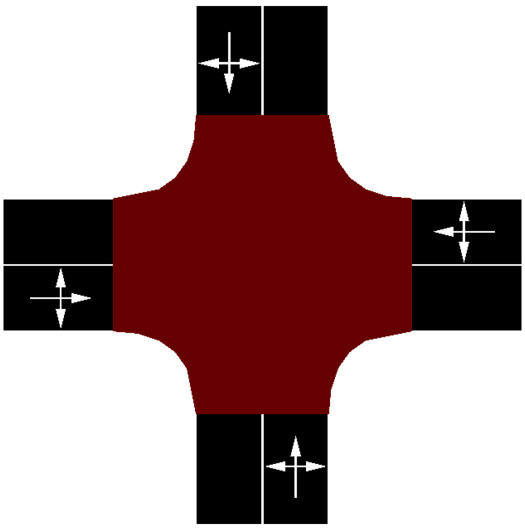


Figure 4.4: Intersection with four two-way one-lane roads.

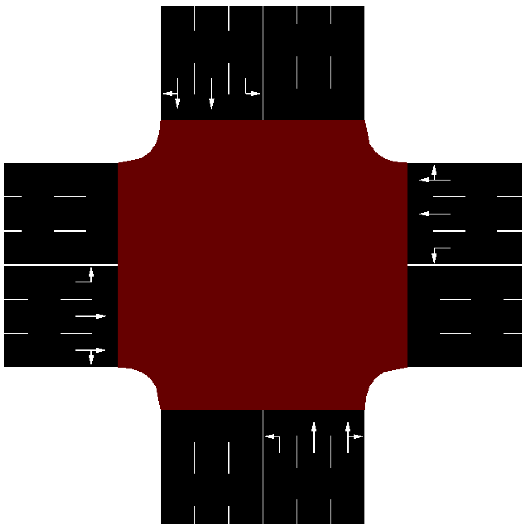


Figure 4.5: Intersection with four two-way three-lane roads.

- **Number of phases**: There are three different phase setups for each intersection in the previous point (figures 4.6 and 4.7). In Figure 4.6, there is a setup with **2 phases**. The first one allows traffic to flow from north and south directions, while the second allows traffic to flow from east and west directions. In this setup there is actually a third phase that blocks traffic from all directions to let the agent clear the intersection if it wants to. Figure 4.7 shows a traffic light setup with **4 phases**. Going from (a) to (d), each phase allows traffic to flow from north, east, south, or west, respectively. As seen in the 2 phases setup, the 4 phases setup also has a fifth phase that blocks traffic from all directions. Finally, the intersection with one-lane roads has a setup with **4,096 phases**, and the intersection with 3-lane roads has a setup with **65,536 phases**. Both these setups allow the agent to control each line of movement individually (represented by the green and red lines in figures 4.6 and 4.7). Knowing that lights for each line of movement can be green or red, the number of phases in those cases is given by two to the power of the number of existing lines of movement (Equation 4.4). The representation of each phase is a number from 0 to *nr. phases*, and its binary representation has as many bits as there are lights in the traffic light. Each bit directly controls one light, where 0 represents a red light and 1 a green light.

Although figures 4.6 and 4.7 have intersections with three-lane roads, the same logic is applied to the intersections with one-lane roads.

$$nr.\ phases = 2^{nr.\ lines\ of\ movement} \tag{4.4}$$

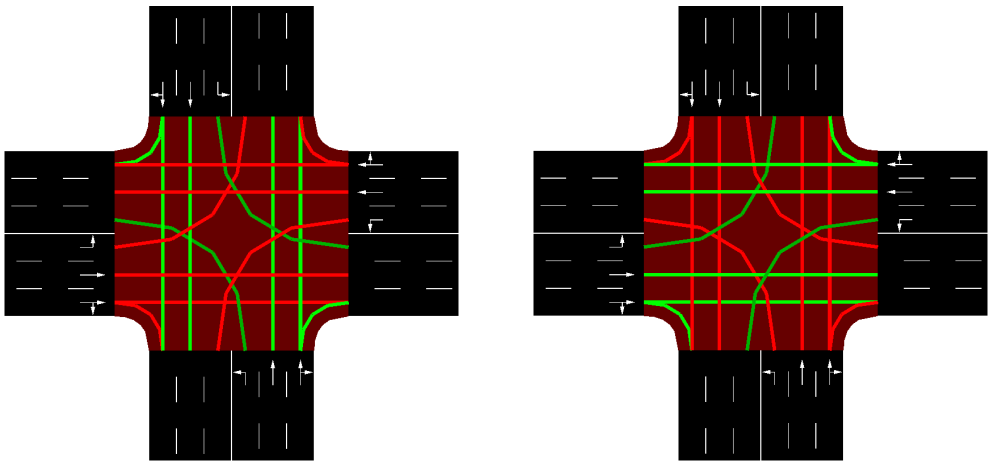

(a) (b)

Figure 4.6: Traffic light with 2 phases.

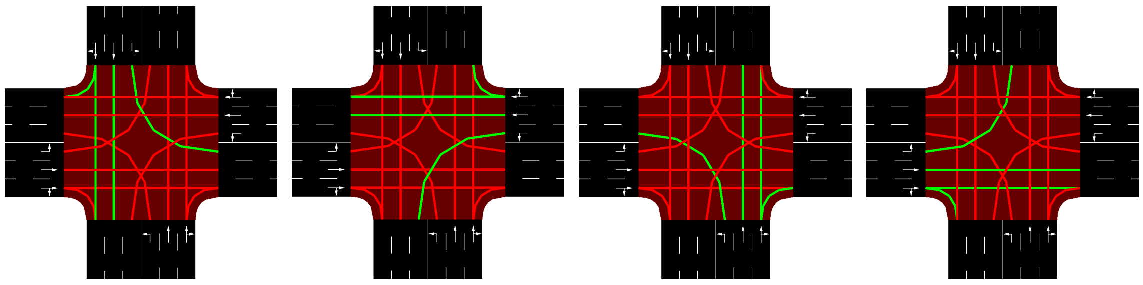

(a) (b) (c) (d)

Figure 4.7: Traffic light with 4 phases.

4.2.2 Training Pipeline Loop

Before the DQN Agent gets into the Training Pipeline Loop (Figure 4.10), it receives an initial representation of the starting state of the simulation from the Environment module, and then the loop starts (steps 1 to 4).

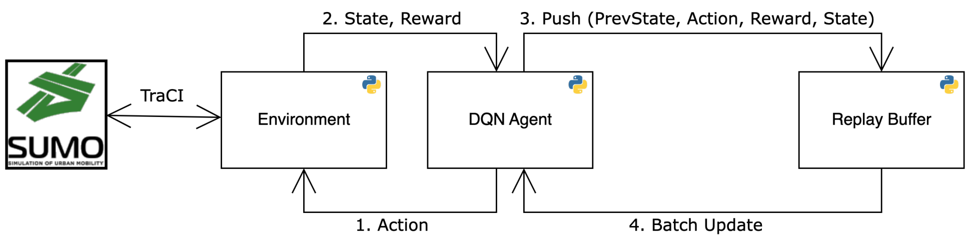


Figure 4.8: Training pipeline for DQN-based agents.

- **Step 1**: The Agent tells the Environment which action it chooses (an action corresponds to setting a phase at the traffic light, as seen in Section 4.1). Selecting an action depends on some randomness defined by the coefficient between exploration and exploitation (ε). Every time the Agent has to select an action, a random number between 0 and 1 is generated. If that number is smaller than ε, an arbitrary action is selected (exploration); otherwise, the Agent decides the best action according to the policy being used (exploitation). Because the action space is really high in two of the scenarios (4,096 and 65,536 different possible actions) and the environment can generate an infinite number of states in any scenario, it is important to start with a high value of ε (set to 0.50) and use an exponential function (Equation 4.5 and Figure 4.9) to decrease that value until a minimum of 0.10 during training.

$$\varepsilon(s) = \begin{cases} 0.5 * 0.99998^s, & 0.5 * 0.99998^s \geq 0.1 \\ 0.1, & 0.5 * 0.99998^s < 0.1 \end{cases} \tag{4.5}$$

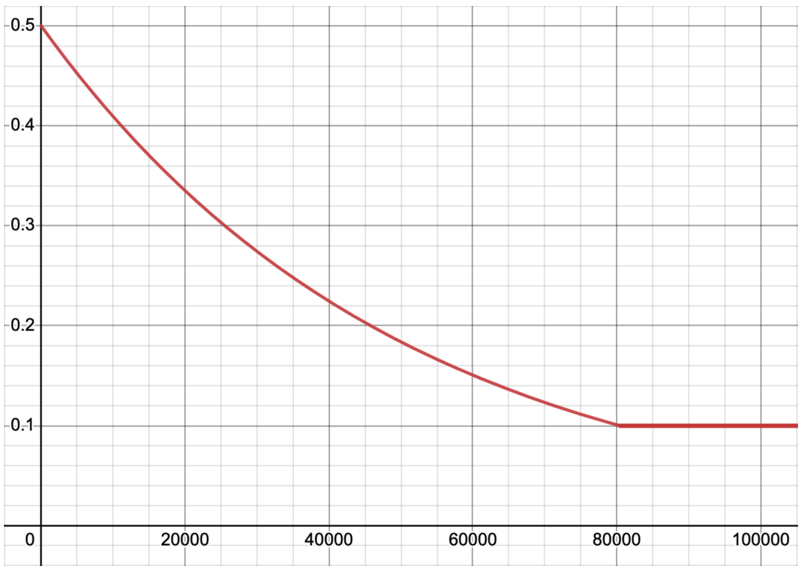


Figure 4.9: Graphical representation of Equation 4.5.

- **Step 2**: The Environment, directly connected to SUMO via TraCI [12], performs the chosen action. Upon exchanging information with SUMO, the Environment builds the new state representation, detailed in the previous section, calculates the value/reward of that state, and sends that information (state and reward) back to the Agent.

- **Step 3**: The tuple (Previous State, Action, Reward, State) is pushed to a Replay Buffer that stores up to 5,000 tuples, acting as the Agent's memory.

- **Step 4**: A batch of 64 random tuples is sampled from the Replay Buffer to train the Agent's neural network. This step is only processed after 64 steps of simulation to have enough tuples to perform the batch training of the Agent. Despite only performing 100,000 steps of simulation, the use of the Replay Buffer to train the agent allows it to repeat past experiences and update the neural network almost 6,400,000 times.

4.2.3 Testing Pipeline Loop

The Testing Pipeline Loop is based on the training one but without the training features. Only one hour of simulation is performed (3,600 steps). All actions are chosen using the Agent's policy (no more exploration is needed), and no updates are done in the Agent's neural network, so there is no need to have the Replay Buffer. This loop aims to collect data about the performance of the multiple agents in different scenarios.

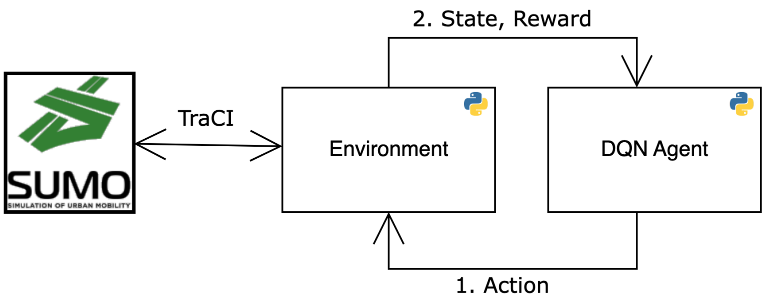


Figure 4.10: Testing pipeline for DQN-based agents.

4.3 Simulation architecture

4.3.1 Agent's Neural Network

As stated at the beginning of Section 4.2, three different methods are used to program the agents. Despite the differences between DQN, Double DQN, and Dueling DQN agents, stated in Section 2.6, they all use the same Convolutional Neural Network schema, built with PyTorch [17] and represented in Figure 4.11. In Section 4.1, we see that the state representation is composed of an RGB image of the intersection, the speed matrix, the waiting time matrix, and for arrays for the percentage of space occupied, the number of waiting vehicles, the sum of waiting times, and the number of vehicles for each lane. Figure 4.11 shows that the image and the two arrays are combined in five two-dimensional tensors [17] (one tensor for each colour component of the image, red, green, and blue, and an extra dimension for each matrix). Then, the five tensors go through three two-dimensional convolutional layers, and the result is a one-dimensional tensor with 50,176 features. That tensor is combined with the features from the four other arrays, resulting in a tensor with $50,176 + nr.lanes$ features (50,192 in the one-lane scenario and 50,224 in the three-lane scenario). Finally, that tensor passes through three fully connected layers, resulting in a final tensor with a size equal to the number of possible actions. The values in the last tensor are the Q-values for each action in the given state. The Agent then selects the action with the biggest value.

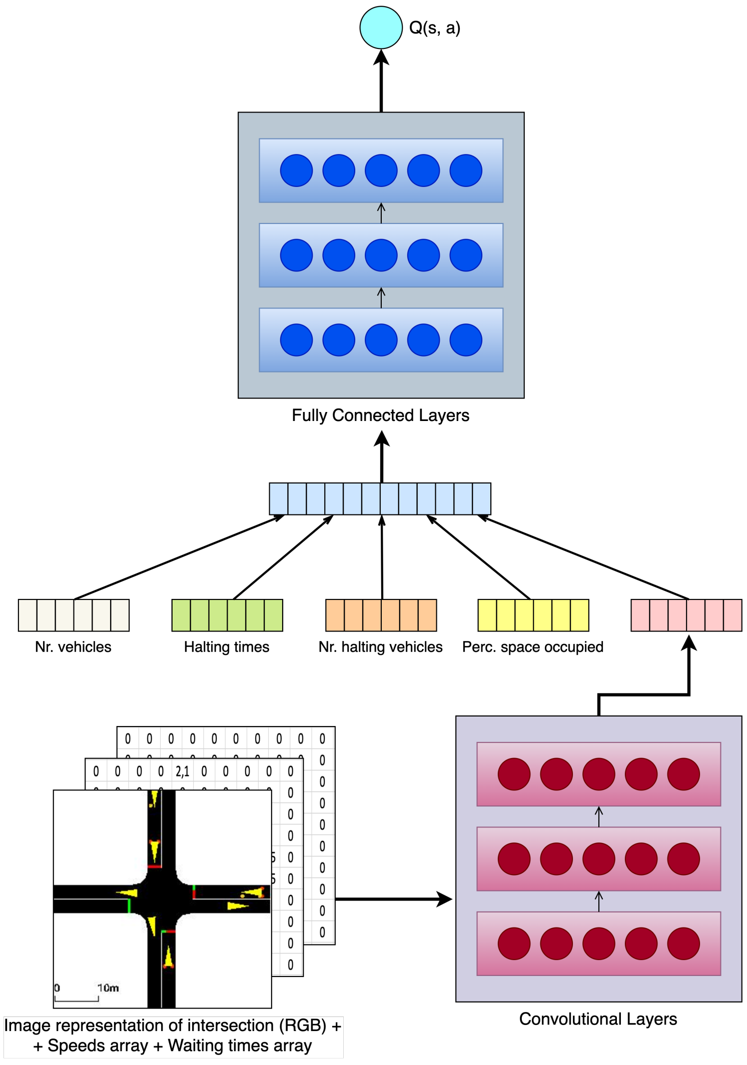


Figure 4.11: State representation going through agent's neural network.

4.3.2 Simulator

A traffic simulator is essential to train and test the agents in an environment as close as possible to the real world. After taking a deep look into multiple projects that use RL to control traffic (e.g. the ones in Chapter 2), a widely used traffic simulator is SUMO [12] due to SUMO's Traffic Control Interface (TraCI). TraCI helps exchange information about the environment with the agents and allows a Graphical User Interface (GUI) to be used simultaneously.

The GUI is the component that allows us to collect images from the intersections at every step to build our state representation (Section 4.1). It is possible to interact with the GUI through TraCI, creating a *view* centred in the middle of the intersection and taking screenshots of that *view* at every step. The screenshots can be stored at our desired location and used by the program to build the state representation. In couple with the images, TraCI allows us to collect a significant amount of data about the simulation and to change the simulation state as we desire (extensive documentation is provided on SUMO's TraCI documentation web page[1]).

However, the control of the simulation with TraCI is only possible after defining the scenarios. Two files are needed to determine a scenario, a network file (<name>.net.xml) and a routes file (<name>.rou.xml). We defined two network files for our scenarios, one for the one-lane scenario and another for the three-lane scenario. They both consist of four perpendicular roads with 100 meters that meet at a traffic light. About the routes files, we created two files for each scenario, one for training and another for testing. The training files contain *flows* (a *flow* repeatedly generates vehicles in the same path with different departure times) from every road to every other road. According to it, our scenarios have four roads and 12 *flows*, three from each road. In each *flow*, we defined a probability of 0.05 for a new vehicle joining the simulation in each step. The testing files were generated with a different method to remove randomness and compare all agents with the same traffic demands. SUMO also has a tool called *randomTrips*[2] to generate a random (but unique) set of trips given a network file. When used with the option *–route-file*, a routes file with only valid routes is generated that can be used to let all the agents experience the same traffic demand. SUMO's *netedit* tool[3] provides a GUI to quickly generate the networks and routes.

4.4 Summary

In this chapter, we formalised the problem of Intelligent Traffic Light Management as an MDP. The environment states have information about the number of vehicles on each lane, the waiting times, the number of waiting vehicles, the percentage of space occupied, the speeds of each vehicle, and an image of the intersection. The action space consists of the possible traffic light phases. The state transition matrix is defined by SUMO [12]. The reward function combines the average speed and waiting times of the vehicles, the number of vehicles in oncoming lanes and leaving the intersection, and the fact that there was a phase change or a collision on each step. Regarding the reward discount factor, there is an analysis of the best value for our models to use.

Then, we propose a solution consisting of three RL methods, DQN, Double DQN and Dueling DQN. Each of those methods is trained for 100,000 steps in six different scenarios. Between scenarios, we change the number of lanes of the roads and the number of phases of the traffic lights. Afterwards, each model is tested in a 3,600-step simulation with the same traffic demand.

Regarding the architecture of our system, further details about the agent's neural network and the SUMO simulator are provided. Our agent's neural network is divided into three two-dimensional convolutional layers to process the images and the matrices and three fully connected layers to process the result combined with the other arrays that form the state representation. The neural network's output is the Q-values for each action according to the current state of the environment. The simulator chosen is SUMO because of its multiple capabilities for integration with RL agents.

[1]`https://sumo.dlr.de/docs/TraCI.html`

[2]`https://sumo.dlr.de/docs/Tools/Trip.html`

[3]`https://sumo.dlr.de/docs/Netedit/index.html`

Chapter 5

Results and Discussion

This chapter is divided into two main sections: Section 5.1 discusses the results from the training pipeline loop (Subsection 4.2.2), and Section 5.2 discusses the results from the testing pipeline loop (Subsection 4.2.3).

5.1 Training

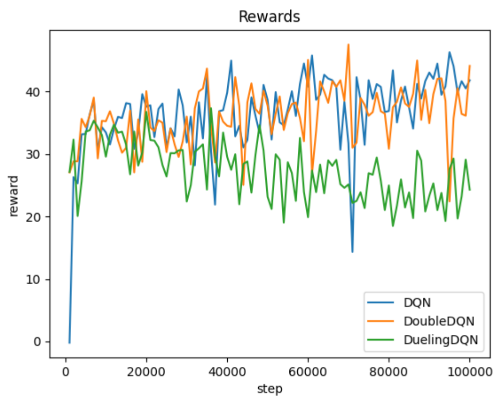
For each of the training scenarios from Subsection 4.2.1, we created three agents (DQN, Double DQN, and Dueling DQN) and trained each one for 100,000 steps. During training, we collected live information about the rewards obtained, the waiting times and speeds of the vehicles, and the number of collisions in the intersection. Regarding the scenarios with the most number of phases, we also collected information about the agents' actions distribution. We analyse all that information in the following subsections, scenario by scenario.

Figures 5.1, 5.2, 5.4, 5.5, 5.6, and 5.8 have four graphs (*a*, *b*, *c*, and *d*) that help us to analyse tendencies in the training process. To better visualise the information obtained, graphs *a*, *b*, and *c* only display one point per 1,000 steps. That point is valued as the average of the 1,000 steps before it. In the case of graph *d*, one point per 1,000 steps would still be hard to understand visually, so we display one point per 5,000 steps. That point is also the average of the 5,000 steps before it.

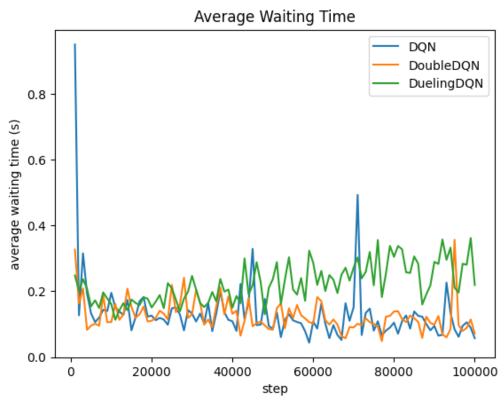
5.1.1 One Lane

5.1.1.1 2 Phases

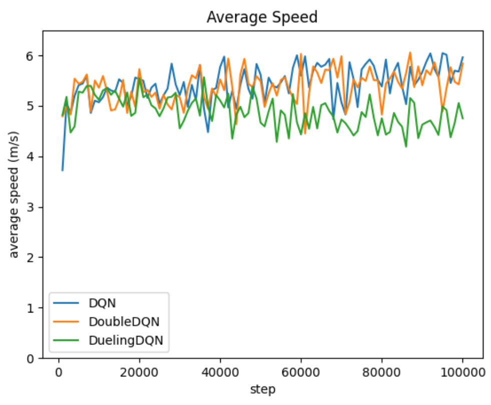
Here we are analysing the scenario where we have one-lane roads and a two-phased traffic light. The most obvious conclusion we can take by generally looking at the graphs from Figure 5.1 is that the agent with the worst performance during training is Dueling DQN. It shows the lowest values for rewards and average speeds and the highest values for average waiting times and the number of collisions. However, despite having the worst results, Dueling DQN shows to be the most stable approach.



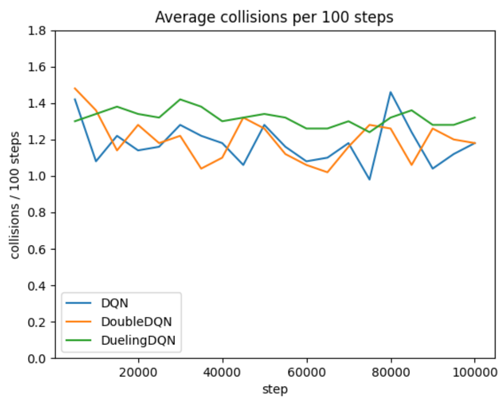
(a)



(b)



(c)



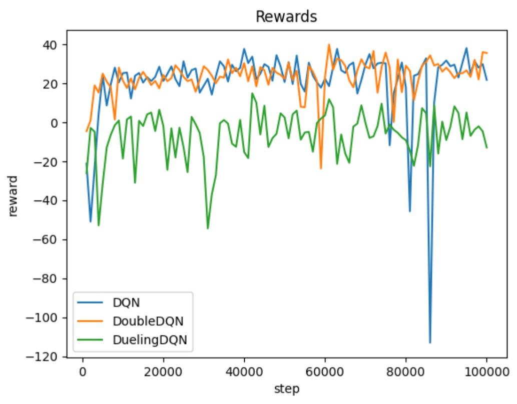
(d)

Figure 5.1: Graphs for rewards (a), average waiting times (b), average speeds (c), and average number of collisions (d) during training in scenarios with one-lane roads and two phases.

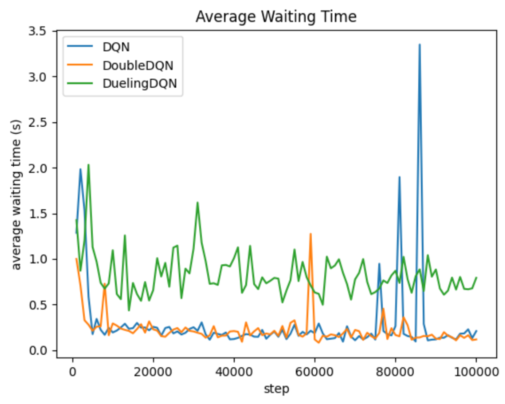
Comparing DQN with Double DQN, we see that DQN can achieve marginally better results. However, the results from DQN and Double DQN are not stable. This shows that these agents might be overfitting to some experiences, and whenever it faces a new situation, the agents do not know what the best action is.

5.1.1.2 4 Phases

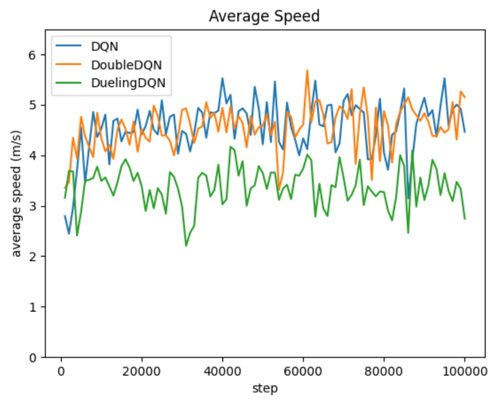
Now, analysing the scenario where we have roads with one lane and the traffic light has four phases (Figure 5.2), the Dueling DQN agent gets the worst overall results once again. However, it still manages to be the more stable agent in every metric. The main contributing factor to being more stable is using two separate networks (Value and Advantage, seen in Section 2.6) to obtain the Q-values. One of the networks has the value of each state, and the second has the advantage of taking one action over the others in that state. Even if the agent faces a state-action pair it has never faced before, it might have faced that same state with another action. So despite no value being stored in the Advantage network for that action, there will be a stored value in the Value network



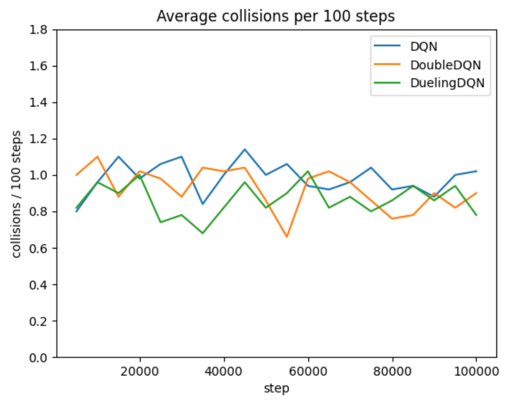
(a)



(b)



(c)



(d)

Figure 5.2: Graphs for rewards (a), average waiting times (b), average speeds (c), and average number of collisions (d) during training in scenarios with one-lane roads and four phases.

for that state, stabilising the training. If the DQN and Double DQN agents have never faced that state-action pair, they will have no idea of its value and will be in an entirely new situation. This makes training less stable while giving better results for known situations (known as overfitting to known situations).

5.1.1.3 4,096 Phases

The scenario with one-lane roads and a traffic light with 4,096 phases is the most demanding for the agents as the action space increases substantially. Figure 5.3 shows the distribution of the agent's actions. Double DQN explored the action space more evenly because the action is selected from the current network, which is only updated every 1,000 steps. In comparison, DQN and Dueling DQN agents select the following action from the target network, which is continually updated, resulting in less exploration of the action space and more exploitation of the already explored actions (Section 2.6).

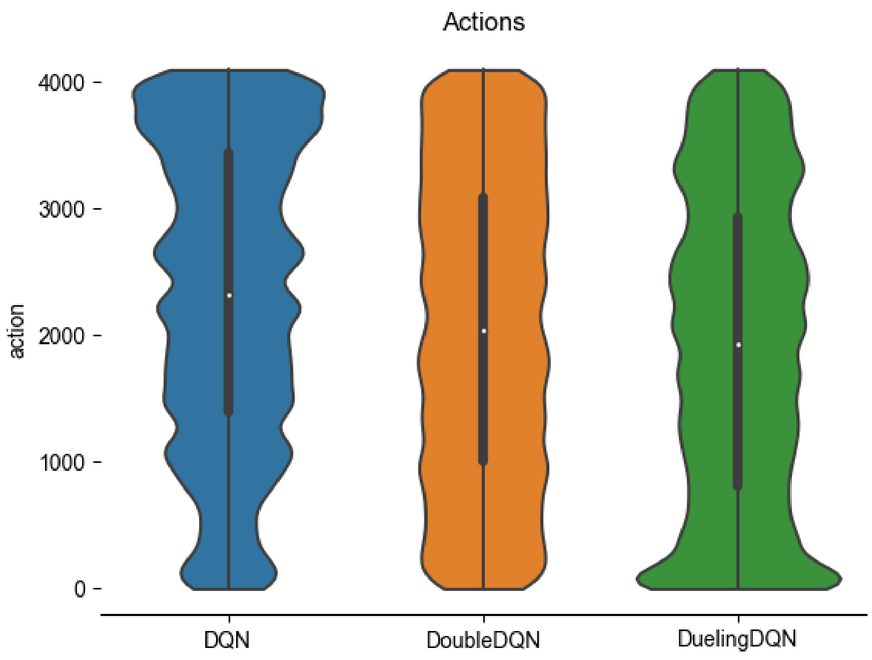


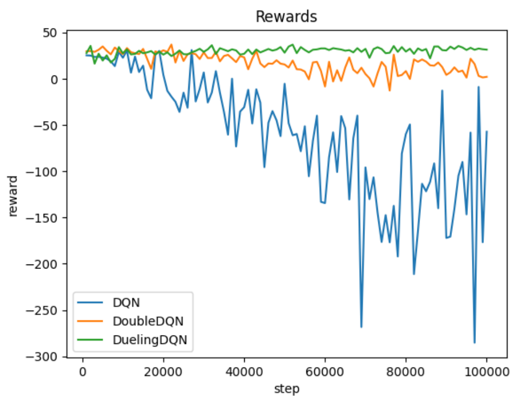
Figure 5.3: Violin bars of the distribution of actions taken during training in scenarios with one-lane roads and 4,096 phases.

Section 4.2.1 states that each bit of the binary representation of the action corresponds to one of the twelve lights. According to this information and Figure 5.3, the DQN agent prefers actions with most lights green (actions higher than 3,800 or 1110 1101 1000 in binary), while the Dueling DQN agent prefers actions with most lights red (actions below 200 or 0000 1100 1000 in binary).

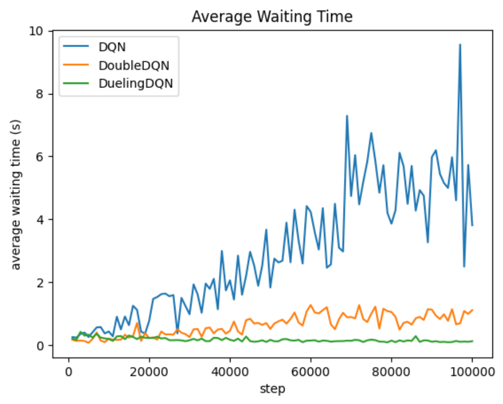
Regarding the results from Figure 5.4, we can see significant differences between the agents when the action space increases. Dueling DQN continues to be the most stable agent and is now the top performer among the three excluding the number of collisions. This shows that collisions should have more impact on the reward function, as keeping the system collision-free is more important than having higher speeds or lower waiting times.

The stability issues from DQN and Double DQN agents stand out in this scenario, with a higher impact on the DQN agent. Although being able to keep the number of collisions lower than the other agents, that is due to the higher number of standing vehicles (with higher waiting times).

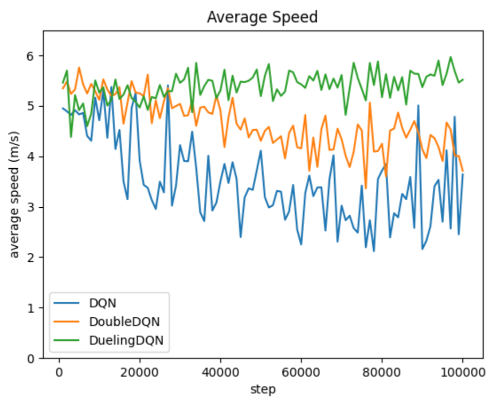
Despite the Double DQN agent being a lot more stable than the DQN agent, when compared to the Dueling DQN agent, it was revealed not to be as stable nor as good in terms of performance. As we have seen in the previous scenario, the architectural differences in the agents' neural networks once more provide a stability advantage to the Dueling DQN agent.



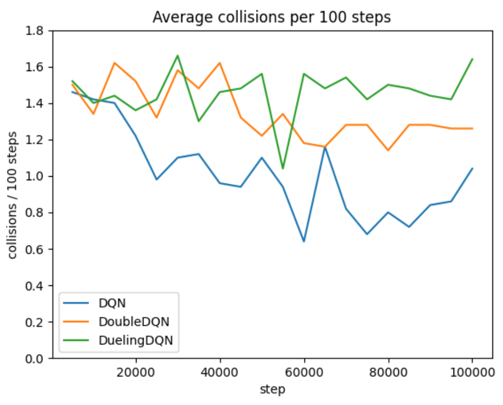
(a)



(b)



(c)



(d)

Figure 5.4: Graphs for rewards (a), average waiting times (b), average speeds (c), and average number of collisions (d) during training in scenarios with one-lane roads and 4,096 phases.

5.1.2 Three Lanes

5.1.2.1 2 Phases

Looking at the results from the scenario with three lanes and two phases (Figure 5.5) is possible to see that they follow the same tendency as the ones from the scenario with one lane and two phases (Figure 5.1, page 36), except in the collisions graph. The absence of collisions is due to the way SUMO's vehicles handle this intersection and not due to agents' actions.

Once more, Dueling DQN was the worst performer agent. Also, this time, there were more stable agents. Having an action space with only three possibilities (the two phases plus all lights red) puts most of the responsibility on the agent's Value network. Adding two lanes on each road makes the observation space grow considerably, destabilising the Value network.

Comparing DQN with Double DQN, this time, DQN appears to be more stable, which is expected as we have a low number of possible actions. The advantages of Double DQN over DQN are best noticed in problems with large action spaces.

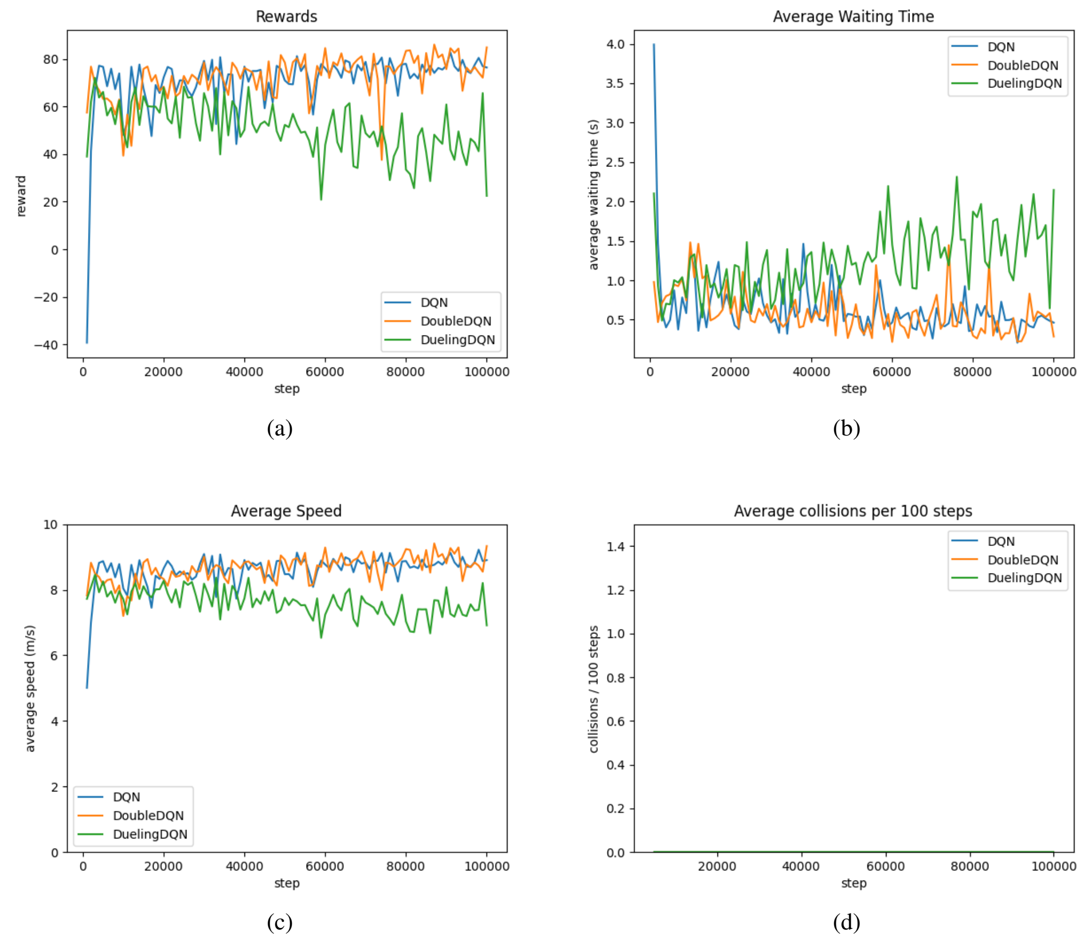


Figure 5.5: Graphs for rewards (a), average waiting times (b), average speeds (c), and average number of collisions (d) during training in scenarios with three-lane roads and two phases.

5.1.2.2 4 Phases

When we have the same three-lane roads, but with four-phase traffic lights, the training of all agents stabilises (Figure 5.6). As in the previous scenario, Dueling DQN is the worst performer and the less stable agent. The differences between DQN and Double DQN are negligible, and they are expected to be the top performers in this scenario.

Comparing this scenario with the one-lane scenario from Figure 5.2 (page 37), all agents managed to be more stable and achieve better results. This is expected as, in the four phases scenarios, there are no conflicts between the green lights. Figure 4.7 from page 29 shows that only traffic from one road can move at a time. In the three-lane scenario, this means that three vehicles can pass through the intersection simultaneously (one on each lane), while in the one-lane scenario, only one vehicle can go through at each time. The result of those differences is that the three-lane scenario can achieve higher speeds, lower waiting limes, and, consequently, higher rewards, no matter what action they choose.

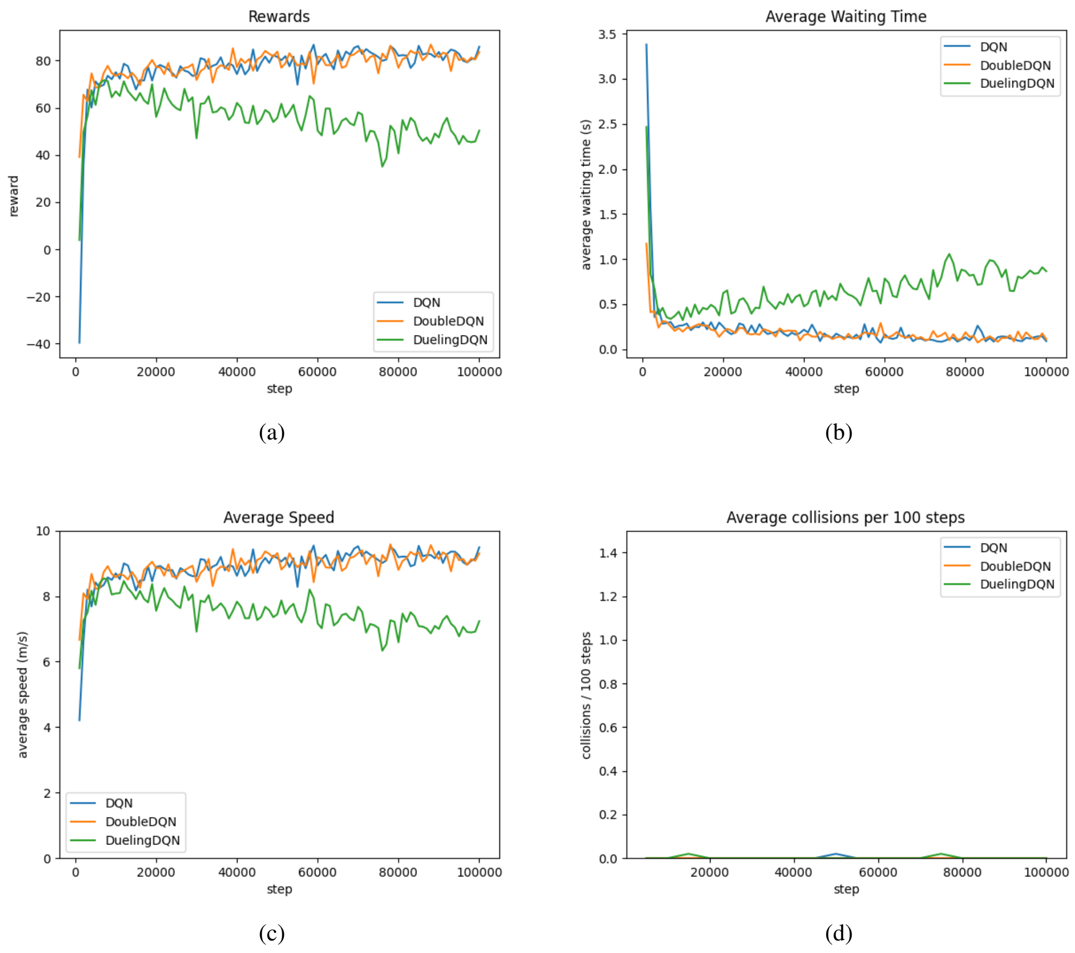


Figure 5.6: Graphs for rewards (a), average waiting times (b), average speeds (c), and average number of collisions (d) during training in scenarios with three-lane roads and four phases.

5.1.2.3 65,536 Phases

The final scenario is the one with three-lane roads and 65,536-phase traffic lights. Once more, due to the size of the action space in this scenario, Figure 5.7 shows the agent's actions distribution during training. As seen in the equivalent scenario with one-lane roads (Figure 5.3 from page 38), Double DQN explores more actions than DQN because it chooses actions from the network that is not constantly updated. The DQN distribution shows that the agent exploited the actions it found to be the best, which can lead to overfitting. The Dueling DQN distribution shows that while having some preference for some actions over others, it still performed a fair exploration of the action space.

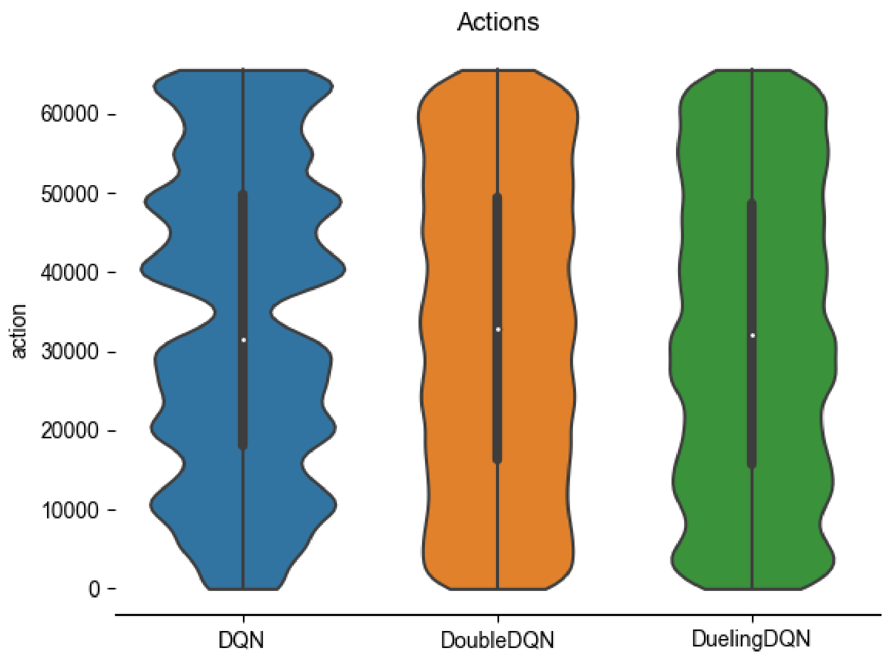


Figure 5.7: Violin bars of the distribution of actions taken during training in scenarios with three-lane roads and 65,536 phases.

Looking at the graphs from Figure 5.8, it becomes clear that the DQN agent overfitted to the initial experiences it faced. From step 60,000 forward, its performance becomes almost random except for the number of collisions that somehow managed to lower in that training phase.

The Double DQN agent managed to perform a lot better than the DQN. This shows that when the action space grows, choosing actions from a network that is not constantly updated can stabilise training and achieve better results.

The best performer in almost every metric was the Dueling DQN. This is the way to go when the action and observation spaces have big dimensions. The training results were stable while keeping low waiting times and high speeds. The problem with this agent is that it shows a high number of collisions, which indicates that the reward function might need some adjustments.

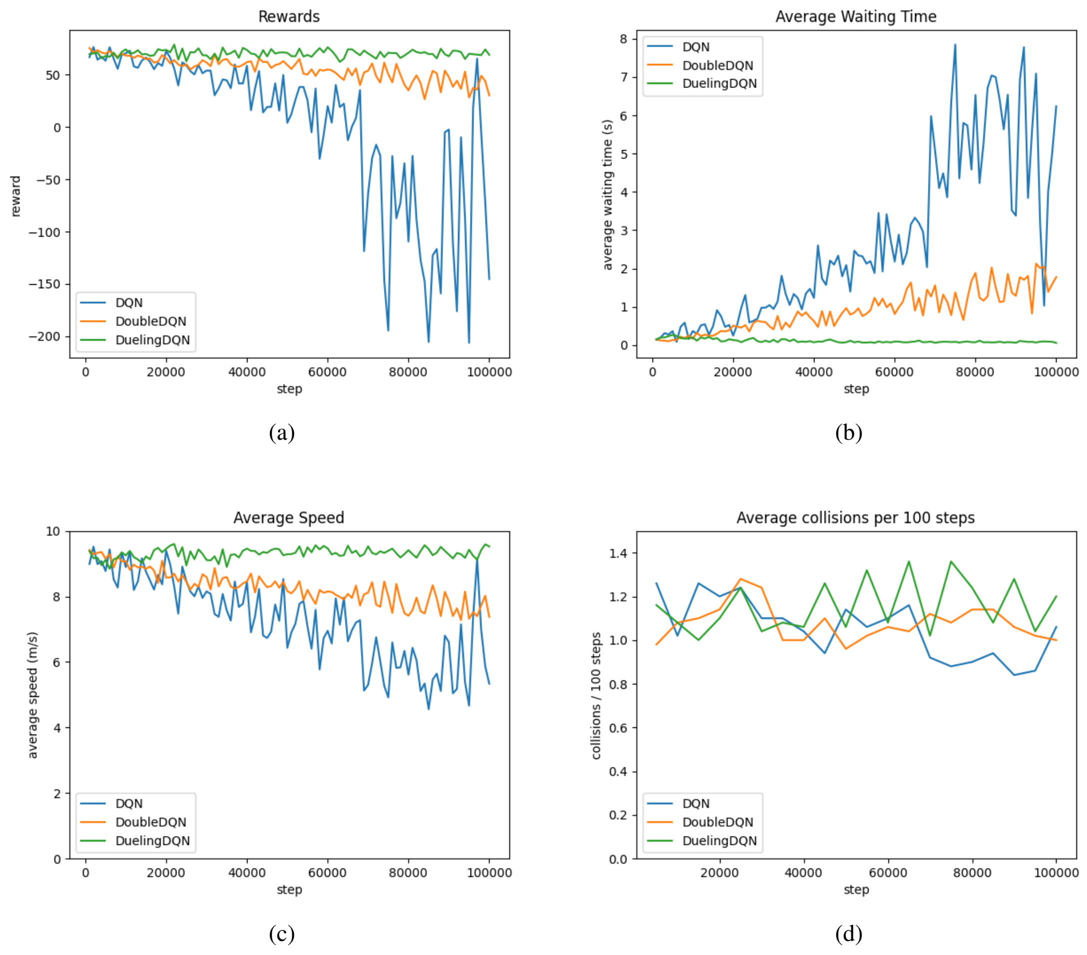


Figure 5.8: Graphs for rewards (a), average waiting times (b), average speeds (c), and average number of collisions (d) during training in scenarios with three-lane roads and 65,536 phases.

5.2 Testing

We tested the trained agents in the same traffic demand for each scenario, as explained in Subsection 4.3.2 on page 32. This means that the same number of vehicles enter the simulation at the same time steps and follow the same routes for all the agents. During testing, we analyse three metrics: the average rewards, the total number of accidents, and the throughput (number of vehicles that passed through the intersection) of each agent for 3,600 steps (equivalent to one hour).

5.2.1 One Lane

DQN is a good agent in scenarios with one-lane roads. However, if we remember Subsection 5.1.1, it was the least stable agent and suffered the most from overfitting. That becomes clear in the 4 and 4,096 phases scenarios from Table 5.1, where it obtained a highly negative reward

Table 5.1: Performance of DQN, Double DQN, and Dueling DQN agents in scenarios with one-lane roads.

Nr. Phases	Metric	DQN	Double DQN	Dueling DQN
2	average reward	9	-2690	**10**
	number of accidents	51	**37**	50
	throughput	**1533**	1497	1422
4	average reward	-1347	**8**	-6
	number of accidents	**18**	47	43
	throughput	**1550**	1493	1371
4,096	average reward	-837	-14	**20**
	number of accidents	**0**	51	66
	throughput	702	1438	**1449**

despite having a low number of accidents and a high throughput. In the 4,096 testing simulation, it is clear that the DQN agent exploited one action where only cars from one road can go through, keeping high speeds for that road and high throughput for the overall system. However, all the other roads stayed at a standstill for most of the simulation, lowering the reward value. This is a methodology to avoid in a real-life scenario due to its stability and overfitting issues.

Comparing Double DQN with Dueling DQN, selecting which one is better is not straightforward. Dueling DQN is more consistent than Double DQN. However, it tends to generate more collisions, which can be a problem related to the reward function used and not the method itself. The Double DQN still manages to be a good choice, especially in scenarios with large action spaces.

5.2.2 Three Lanes

In the three-lane scenarios, the results were different (Table 5.2). All agents managed to keep the intersection collision-free with similar throughputs in scenarios with two or four phases, which is excellent. The rewards of the Dueling DQN are lower than the other agents due to the speeds and waiting times of the vehicles. Both DQN and Double DQN agents managed to keep the intersection free of long queues of vehicles (consistently below five vehicles), while the Dueling DQN agent occasionally formed a big queue on one of the roads, but only for short periods.

When we increase the action space to 65,536 phases, all agents struggle to keep up with the demand. DQN agent, once more, found that always keeping the same state would keep traffic flowing, and that was its strategy. However, that leads to highly negative rewards and is not desirable in a real-world scenario. Double DQN followed the same strategy of always keeping the same phase but with a different combination of green and red lights. That led to a better evaluation but is also not desirable in a real-world scenario. The Dueling DQN agent achieved a

Table 5.2: Performance of DQN, Double DQN, and Dueling DQN agents in scenarios with three-lane roads.

Nr. Phases	Metric	DQN	Double DQN	Dueling DQN
2	average reward	52	**53**	-81
	number of accidents	0	0	0
	throughput	**2700**	2699	2682
4	average reward	**67**	**67**	1
	number of accidents	0	0	0
	throughput	**2699**	2697	2691
65,536	average reward	-6917	-2137	**60**
	number of accidents	8	**1**	65
	throughput	**1832**	964	1829

higher number of accidents but used multiple phases of the traffic light, allowing traffic to flow from every direction. So this is the only solution suitable for the real world.

Finally, Table 5.2 shows that increasing the traffic light's number of phases is not adequate to keep traffic flowing.

5.3 Summary

This chapter discusses the results from both the training and testing loops from subsections 4.2.2 and 4.2.3 with DQN, Double DQN, and Dueling DQN agents.

In the training loop (Subsection 5.1), we analyse the rewards, waiting times, speeds and the number of collisions each agent obtains in each scenario. The DQN agent shows to be the least stable one and suffers from overfitting to the initial observations of the training. The Double DQN agent shows a fair exploration of the action space, leading to more stable and better results. The Dueling DQN agent does not have the best results. However, it is highly stable, especially when the action space increases.

In the testing loop (Subsection 5.2), we got a confirmation of those ideas. The DQN agent is unsuitable for real-world scenarios due to overfitting and the probability of always choosing the same action (leaving some lanes with a constant red light). The Dueling DQN agent is not always the best but has consistent results.

Finally, increasing the number of lanes on the roads leads to better results, while increasing the number of phases on the traffic lights leads to worse results.

Chapter 6

Conclusions

6.1 Main Contributions

The main contributions of this work are identified in three main dimensions: scientific, technological, and applicational. Regarding the scientific contribution, the work detailed in this document proposes a pipeline based on Deep Reinforcement Learning to solve a relatively recent problem: the control of AGV traffic. It uses DQN-based agents in a technique that combines two-dimensional convolutional layers to process images and matrices, with fully connected layers to process the result and arrays of extra information.

Talking about the applicational contribution, we do not solve the AGV traffic control as a whole. Our focus is to automatically control traffic intersections where the fact that no humans are driving our vehicles introduces new communication abilities to the system. Better communication skills bring more information about the environment to the traffic light controller and better decisions. The second applicational contribution is related to the number of phases of the traffic lights. Standard traffic lights only have between two and four phases. However, by having only AGVs in our systems, more phases can be introduced thanks to the predictable behaviour of AGVs.

Finally, the technological contribution of this work relies on integrating our pipeline with an open-source microscopic traffic simulator, namely SUMO [12]. The code developed during this project can be used to develop and test other RL architectures to control AGV intersections.

6.2 Future Work

While reviewing other works related to traffic light management using RL, multiple techniques are available to improve the results. This section suggests some of them and describes how they could impact the results.

6.2.1 State Representation

As seen in Chapter 3, different authors adopt different state representations in their methods. It is not straightforward to understand which features are more important to be represented.

The current traffic light phase is one feature they all agreed to use that is out of our state's representation. Our reward function is even influenced by whether or not the phase changes between steps. However, our agents have no way of knowing if they are keeping the same phase or not because the environment does not provide that information to the agents. Including this information is simple to do and can improve the results.

One feature that we did include is an image representation of the intersection. Although images bring many features with low programming effort, most of the pixels of those images are irrelevant. After a rough analysis of Figure 4.2 (p. 24), we can see that around 60% of the image represents the road surroundings, which is irrelevant to control the traffic light. Removing this information from the images is complex, but it can improve the agents' training performance.

6.2.2 Flipping and Rotating Traffic

Zheng et al. [33] conducted a traffic light control experiment exploring the rotation and flipping of traffic in intersections with four roads. Looking at Figure 6.1, it is possible to understand that when we rotate an intersection 90, 180, or 270 degrees, we get a similar situation. It is also possible to understand that for each of those four situations if we flip the east road with the west road, we once again get four more similar situations. In the open world (where humans are the vehicles' drivers), these flipping situations are typical when most people go from home to work in the morning (e.g. east to west) and from work to home in the evening (e.g. west to east).

The utility of flipping and rotating traffic situations is that from one single observation of the environment is possible to generate eight different (but similar) states that our RL agents must be able to handle. This technique can make our agents perform better in scenarios they have never faced.

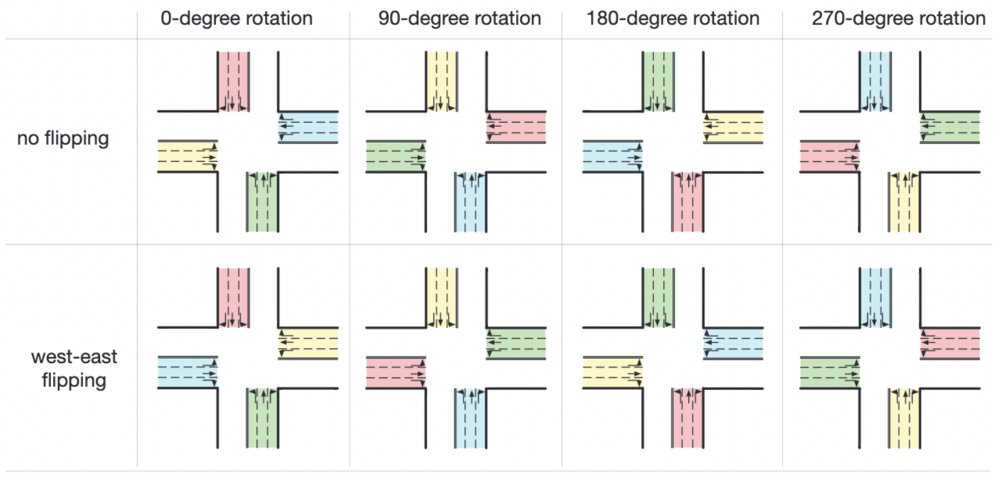


Figure 6.1: Possible variations based on flipping and rotation of the top-left case, from [33].

6.2.3 Multi-agent Scenarios

Until now, we have only explored scenarios with a single intersection. From real-world scenarios, we know that when we have multiple traffic lights in a more or less complex road network, the state of one traffic light can influence the traffic of the surrounding ones. In these scenarios, the goal is not to optimise the traffic flow of each intersection individually but to optimise the traffic flow of the entire road network. Multi-Agent Reinforcement Learning (MARL) is a field of RL where the actions of one agent influence the rewards received by all agents [8].

A step forward in improving this work is exploring MARL methods in controlling multiple AGV traffic intersections. However, these methods come with new challenges that Nguyen et al. [15] explored. The five main challenges are partial observability of each agent, non-stationarity of the environment, dealing with continuous action spaces, multi-agent training schemes so that all agents train together, and transfer learning in multi-agent systems.

Some authors have already explored the use of MARL approaches to control multiple traffic lights. One of the early approaches by Pol et al. [25] uses multi-agent DQN with transfer learning, but the training is not always stable. PressLight, from Wei et al. [31], is another work that introduces the concept of *pressure* of intersections to coordinate multiple DQN agents. One more recent work by Chen et al., MPLight [3], improves the work done by PressLight by using agents with the FRAP architecture explored by Zheng et al. [33]. According to the author, FRAP is an architecture that “is invariant to symmetric operations like Flipping and Rotation and considers All Phase configurations”.

Using multi-agent methods allows us to complement the work already developed with the capacity to handle scenarios with a higher degree of complexity.

References

[1] Joshua Achiam. Spinning up in deep reinforcement learning. *GitHub repository*, 2018.

[2] Alvaro Cabrejas-Egea, Raymond Zhang, and Neil Walton. Reinforcement learning for traffic signal control: Comparison with commercial systems. *Transportation Research Procedia*, 58:638–645, 2021.

[3] Hua Wei Chacha Chen, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *Proceeding of the Thirty-fourth AAAI Conference on Artificial Intelligence (AAAI'20). New York, NY*, 2020.

[4] Jim X. Chen. The evolution of computing: Alphago. *Computing in Science Engineering*, 18(4):4–7, 2016.

[5] Seung-Bae Cools, Carlos Gershenson, and Bart D'Hooghe. Self-organizing traffic lights: A realistic simulation. In *Advances in Applied Self-organizing Systems*, pages 41–50. Springer London, nov 2007.

[6] M. De Ryck, M. Versteyhe, and F. Debrouwere. Automated guided vehicle systems, state-of-the-art control algorithms and techniques. *Journal of Manufacturing Systems*, 54:152–173, 2020.

[7] Zihan Ding, Yanhua Huang, Hang Yuan, and Hao Dong. *Introduction to Reinforcement Learning*, pages 47–123. Springer Singapore, Singapore, 2020.

[8] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[9] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019.

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

[11] M.L. Littman. Markov decision processes. In Neil J. Smelser and Paul B. Baltes, editors, *International Encyclopedia of the Social Behavioral Sciences*, pages 9240–9242. Pergamon, Oxford, 2001.

[12] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[13] Alan J. Miller. Settings for fixed-cycle traffic signals. *Journal of the Operational Research Society*, 14(4):373–386, 1963.

[14] Alan J. Miller. Settings for fixed-cycle traffic signals. *Journal of the Operational Research Society*, 14:373–386, 1963.

[15] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions on Cybernetics*, 50(9):3826–3839, 2020.

[16] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2):56–69, 2010.

[17] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[18] Péter Pálos and Árpád Huszák. Comparison of q-learning based traffic light control methods and objective functions. In *2020 International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, pages 1–6, 2020.

[19] Mohit Sewak. *Deep Q Network (DQN), Double DQN, and Dueling DQN*, pages 95–108. Springer Singapore, Singapore, 2019.

[20] Mohit Sewak. *Policy-Based Reinforcement Learning Approaches*, pages 127–140. Springer Singapore, Singapore, 2019.

[21] Stephen F. Smith, Gregory J. Barlow, Xiao-Feng Xie, and Zachary B. Rubinstein. Surtrac: Scalable urban traffic control. 2013.

[22] Daniel W Stroock. *An introduction to Markov processes*, volume 230. Springer Science & Business Media, 2013.

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

[24] Elise van der Pol and Frans A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. 2016.

[25] Elise Van der Pol and Frans A Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of learning, inference and control of multi-agent systems (at NIPS 2016)*, 1, 2016.

[26] Cristina Vilarinho, José Pedro Tavares, and Rosaldo J. F. Rossetti. Intelligent traffic lights: Green time period negotiaton. *Transportation Research Procedia*, (22):325–334, 2017.

[27] R. A. Vincent and J. R. Peirce. 'mova': Traffic responsive, self-optimising signal control for isolated intersections. *TRRL RESEARCH REPORT*, 1988.

[28] Iris FA Vis. Survey of research in the design and control of automated guided vehicle systems. *European Journal of Operational Research*, 170(3):677–709, 2006.

[29] Xuesong Wang, Haopeng Xiang, Yuhu Cheng, and Qiang Yu. Prioritised experience replay based on sample optimisation. *The Journal of Engineering*, 2020(13):298–302, 2020.

[30] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2015.

[31] Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. Presslight: Learning max pressure control to coordinate traffic signals in arterial network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, KDD '19, page 1290–1298, New York, NY, USA, 2019. Association for Computing Machinery.

[32] Hua Wei, Guanjie Zheng, Huaxiu Yao, and Zhenhui Li. Intellilight: A reinforcement learning approach for intelligent traffic light control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18, page 2496–2505, New York, NY, USA, 2018. Association for Computing Machinery.

[33] Guanjie Zheng, Yuanhao Xiong, Xinshi Zang, Jie Feng, Hua Wei, Huichu Zhang, Yong Li, Kai Xu, and Zhenhui Li. Learning phase competition for traffic signal control. *CoRR*, abs/1905.04722, 2019.