Tunisian Republic
Ministry of Higher Education and Scientific Research
Carthage University - Engineering School of Statistics and Information Analysis

Graduation Project presented for the obtention of

National Engineering Diploma in Statistics and Information Analysis

Submitted by

Mohamed Safouen Markou

Multi-agent Reinforcement Learning for Traffic Signal Control

---

Defended on 20/06/2024 in front of the committee composed of:

Ms Sellami Hajer	President
Ms Hamdeni Tasnime	Reviewer
Pr. Amari Kaïs	Supervisor
M. Bel Mufti Ghazi	Mentor

A Graduation Project made at

Analysis and control of PDEs laboratory

Year: 2023/2024

Dedication

Dear my mom in heaven «Saïda»:

This report is firstly dedicated to the loving memory of you, whose guidance, love, and inspiration continue to light my path. Though you are no longer with us, your values and principles have been a constant source of strength for me.

Dear my father «Bassem»:

I dedicate this report to you, whose wisdom, patience, and unwavering support have been my foundation. Your dedication and love have provided me with the strength and motivation to pursue and accomplish my aspirations.

Dear my mother «Sihem»:

I would also like to dedicate this work to my stepmother also, whose support, kindness, and encouragement have been unvaluable. Your presence in my life has been a blessing, and your unwavering belief in my abilities has helped me achieve my goals.

To my sister «Elaa»:

I have never felt that I am alone in this world and I am sure that you will always be by my side. I dedicate this work to you with all my wishes for joy, health and success.

To my friend «Mohamed»:

Thanks to your encouragement and support, I managed to move forward and complete my project who truly proved the true meaning of friendship. As a testimony to the friendship that unites us and the memories of all the moments we spent together, I dedicate this work to you and I wish you a life full of health and joy.

Acknowledgements

I could never have realized this project without the precious help and support of many people.

First, I would like to thank my academic supervisor, M. Bel Mufti Ghazi, I would like to express my gratitude to Pr. Amari Kaïs, for giving me the desire to do this project within Analysis and control of PDEs laboratory . I also thank him for his welcome.

I would also like to say to president how honored I am. I am very grateful to Ms Sellami Hajer for stimulating me, I am deeply grateful to Ms Hamdeni Tasnime for the interest she has shown in this project by committing to be a reporter.

My gratitude goes to those who have provided emotional support for this work: my family and friends.

Abstract

The present work is part of a graduation project carried out within the company Analysis and control of PDEs laboratory in order to obtain the national diploma of engineer at the Engineering School of Statistics and Information Analysis . This project's objective is to design and implement a multi-agent system with agents learning independently via reinforcement learning and more precise, deep reinforcement learning algorithms as options to control traffic signals in real-time and then to compare the final output. Our results show that PPO algorithm is the most effective as it reduces the average waiting time in the function reward function but also increases the average speed of vehicles in a traffic network.

Keywords— Traffic Signal Control, Deep Reinforcement Learning, Multi-Agent, PPO, A2C, DQN

Contents

List of Figures

List of Algorithms

List of Tables

Introduction

Modern society relies on its many transportation systems for the movement of individuals, goods and services. Ensuring vehicles can move efficiently from their origin to destination is desirable by all. However, increasing population and subsequent vehicle ownership, have increased the demand for road infrastructure often beyond its capacity, resulting in congestion, travel delays and unnecessary vehicle emissions.

Traffic congestion is a phenomenon caused by too many vehicles trying to use the same infrastructure at the same time. The total time lost owing to traffic congestion in Japan in 2012 was approximately 5 billion hours per year. The economic loss in terms of total lost time is estimated to be approximately 10 trillion yen per year, and it is calculated by converting cash wages in 2012 to time [11]. Therefore, because the economic benefits of reducing traffic congestion are significant, it is necessary to tackle this issue.

One way to accomplish this is by control techniques, notably the dynamic control of traffic signal controllers. A major challenge of optimizing such a controller, is that the problem is very constrained, and the control policy needs to be fair to all traffic directions.

Several studies have been conducted on the dynamic control of traffic signal duration using reinforcement learning to reduce the traffic congestion and realize a better traffic light control system.

We compare different Reinforcement learning approaches for traffic signal control at urban intersections. Specifically, we use PPO (Proximal Policy Optimization), A2C (Advantage Actor Critic) and DQN (Deep Q-Network) to deal with the complex traffic signal control problem.

Chapter 1

Generalisation

1.1 Host Organization

We completed our internship at the Analysis and Control of PDEs laboratory, L22ES03, a research laboratory focused on the analysis and control of partial differential equations, which brings together teacher-researchers and researchers. It is a unit of the Department of Mathematics, Faculty of Sciences of Monastir.

Figure 1.1: Hosting laboratory Logo

It has many members: permanent teacher-researchers (tenured professors, doctors, assistant lecturers) from various schools in Tunisia and around the world, researchers, doctoral students, master's students, and interns. Its main mission is research in mathematics.

Research Theme

The research theme focuses on the mathematical analysis and control of linear and non-linear partial differential equations and their applications to engineering problems. The topics covered include:

- Control and stabilization of systems governed by PDEs
- Inverse Problems and Optimization
- Linear and nonlinear evolution problems, asymptotic behavior
- Data Science
- Homogenization
- Spectral analysis and mathematical physics

1.2 Problematic

Our project focuses on the huge challenges of improving traffic signal control to make city traffic flow better. Urban traffic systems are really complicated because they have lots of cars that go up and down, follow unpredictable patterns, and have all kinds of different drivers. The usual ways of controlling traffic signals struggle to keep up with these changes, which makes traffic move slowly and causes more congestion.

On top of that, it's hard to add new technology to the current traffic systems because there are technical and logistical problems to solve, like collecting data, processing it in real-time, and making it work with the old systems.

The challenge of integrating new technologies with existing infrastructure goes beyond just technical issues, it's also a financial hurdle. Upgrading traffic control systems requires a significant investment, which can be a major barrier for many municipalities. It's crucial that the new system seamlessly work with the old ones to avoid disruptions and maximize efficiency during the transition.

An additional layer of complexity is introduced by the partially observable and stochastic nature of traffic conditions. Uncertainty is introduced by elements like emergency vehicles priority and pedestrian movements, this must be effectively managed.
The implementation process is further complicated by the need for real-time decision-making and the scalability of solutions across a network of interconnected intersections.

These factors combine to create a complex problem that demands innovative solutions and careful planning to effectively address. This project seeks to address these issues through advanced techniques and a comprehensive approach to modernizing traffic signal control.

1.3 Proposed solution

To deal with the problem of integrating this technology in real world infrastructure problem, we suggest using a traffic signal control simulator called SUMO.

This simulator lets us create and study different traffic situations in a controlled setting, allowing us to test and improve our methods without causing disruptions to real traffic. By using the simulator, we can collect important data and insights into traffic patterns and behaviors, which are crucial for developing effective solutions.

Next, instead of using the traditional optimization or simple rule-based approach, our project looks at using advanced reinforcement learning techniques. These special algorithms have the potential to improve traffic signal control by learning from what's happening in real-time and making smart decisions. But using these techniques brings more challenges, like needing a lot of computer power, making sure the algorithms work well in changing situations, and training models to work with different traffic scenarios.

Solving these problems is really important if we want to make traffic signals smarter and create smoother, safer, and more efficient city traffic systems.

Chapter 2

Reinforcement Learning

Reinforcement Learning is one of the three **machine learning** paradigms added to supervised learning and unsupervised learning. it is a framework for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and receiving rewards (positive or negative) as unique feedback.

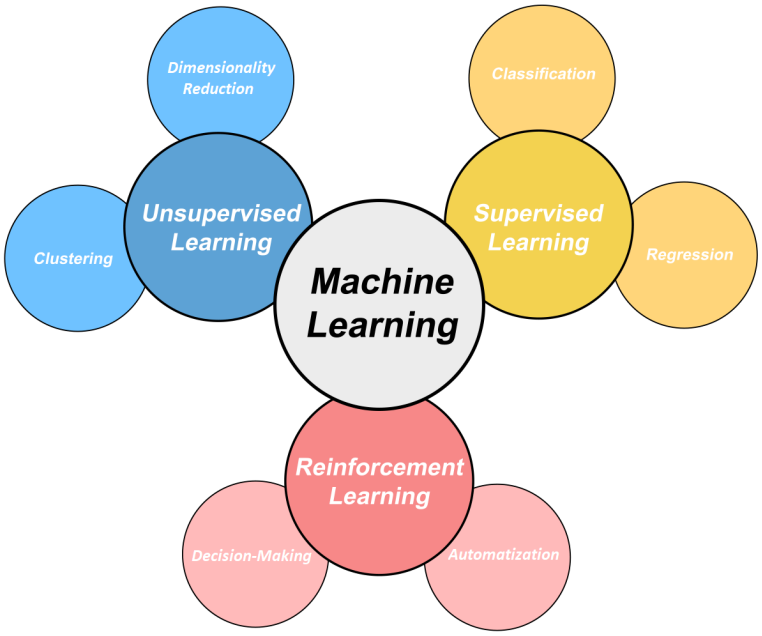


Figure 2.1: Machine learning approaches

To simplify, the idea behind Reinforcement Learning is that an **agent** (an AI) will learn from the **environment** by interacting with it (through trial and error) and receiving **rewards** (negative or positive) as feedback for performing actions.

2.1 Action space

In a specific reinforcement learning environment, the set of all effective actions of the agent is called **action space** and noted A. The action space must have two properties: **completeness** and **validity**.

- **Completeness**: guarantees the possibility that the agent can achieve the expected goal. If a certain action is missing, the agent has a high probability of failing to complete the task.
  - An example of this is that a car must have the function of accelerating, decelerating, turning and braking to ensure safety.
- **Validity**: requires that the actions in the action space must be legal,
  - you can never add the function of flying to the action space of driving car.

Reinforcement learning action space may be divided into three main categories which are discrete action space, continuous action space and discrete-continuous hybrid action space.

- For **discrete actions**, the number of actions is countable. So, we can use one-hot vectors to indicate whether an action is executed. Example :
  - In Super Mario, you have 4 possible actions: moving up, down, left and right
- In the **continuous action space** problem, the number of actions is uncountable, but we can describe their range.
  - In auto-driving cars, the action of acceleration is continuous.
- For **discrete-continuous hybrid action spaces**, it requires the algorithm to output discrete actions and the continuous parameters performed by the discrete actions.

2.2 State/Observation space

In reinforcement learning, the observation space is the set of all available information and problem features that are useful for making a decision and noted S. An observation is a partial description of a state, which may omit information.

There is a differentiation to make between observation and state:

- **State**: is a **complete** description of the state of the world (there is no hidden information). In a fully observed environment. example:
  - In a chess game, we have access to the whole board's information.
- **Observation**: is a **partial** description of the state. In a partially observed environment. example:
  - In Super Mario, we only see the part of the level close to the player. So, we receive an observation.

To resume the reinforcement learning system, at each time t:

1. The agent receives state $s_t \in S$ from the environment
2. Based on that state s_t, the Agent takes action $a_t \in A$
3. The environment gives some reward r_t to the agent
4. The environment goes to a new state $s_{t+1} \in S$

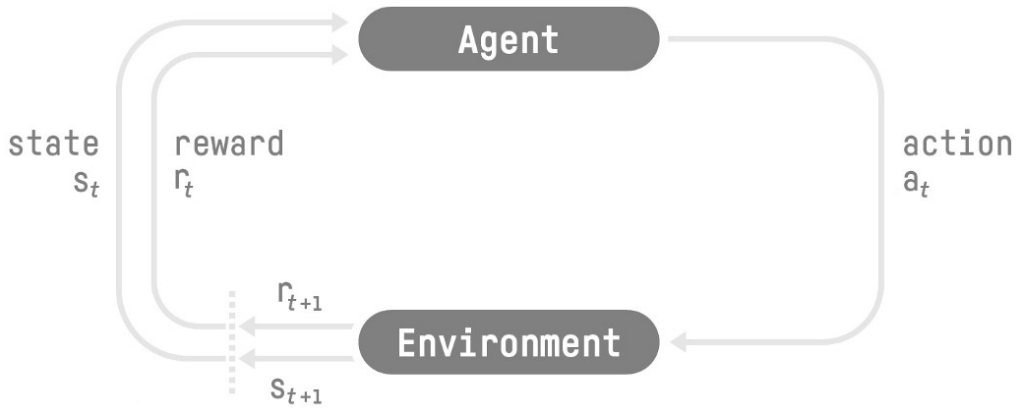


Figure 2.2: Reinforcement Learning System

This Reinforcement Learning loop outputs a sequence of state, action, reward and next state (s_t, a_t, r_t, s_{t+1}).

2.3 Reward

The reward function is a **fundamental** part that defines the goal or objective of the agent. It quantifies the immediate feedback that the agent receives after taking an **action** in a certain state. Given a sequence $\boldsymbol{r_{t+1}}$, $\boldsymbol{r_{t+2}}$, ... of rewards received while following a policy, then the expected reward function R_t from time t is given by the Eq. 2.1:

$$R_t = \ r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.1}$$

- $\gamma \in [0, 1]$ is the **discount rate**, that determines the present value of future rewards:
  - $\gamma \to 0$: the agent maximizes immediate rewards. so the future rewards are less and less likely to happen.
  - $\gamma \to 1$: the agent cares more about the long-term reward.

The agent receives rewards for performing actions and uses them to measure the action's success or failure. The Reward R_t can be expressed in different forms, as a function of the action $R_t(a)$, or as a function of action-state pairs $R_t(a, s)$.
The reward is granted by adding all the rewards generated from executing an episode. The episode *(trajectory)* represents a finite number of actions and ends when the agent achieves a final state,

2.4 Environment and agents

An **environment** in artificial intelligence is the surroundings of the agent. It is a physical or virtual world whose state evolves and influenced by the actions of the agents that exist within the environment. The environment specifies the actions that agents can take at any point in time as well as the observations that individual agents receive about the state of the environment.
The **agent** takes **input** from the environment and delivers the **output** to the environment through actuators.
The types of environments can vary significantly depending on their characteristics and complexity. Below are the primary types of environments in reinforcement learning:

- Single-agent vs Multi-agent
- Competitive vs Collaborative
- Deterministic vs Stochastic
- Discrete vs Continuous

These types define the characteristics of the environment. So, we will explain the most crucial and important types:

2.5 Single-agent Vs Multi-agent

A **multi-agent system (MAS)** consists of a population of decision-making agents that operate within a shared environment, each pursuing their objectives. These agents observe the state of the environment and may communicate with one another. The communication among agents can be constrained by decentralization.

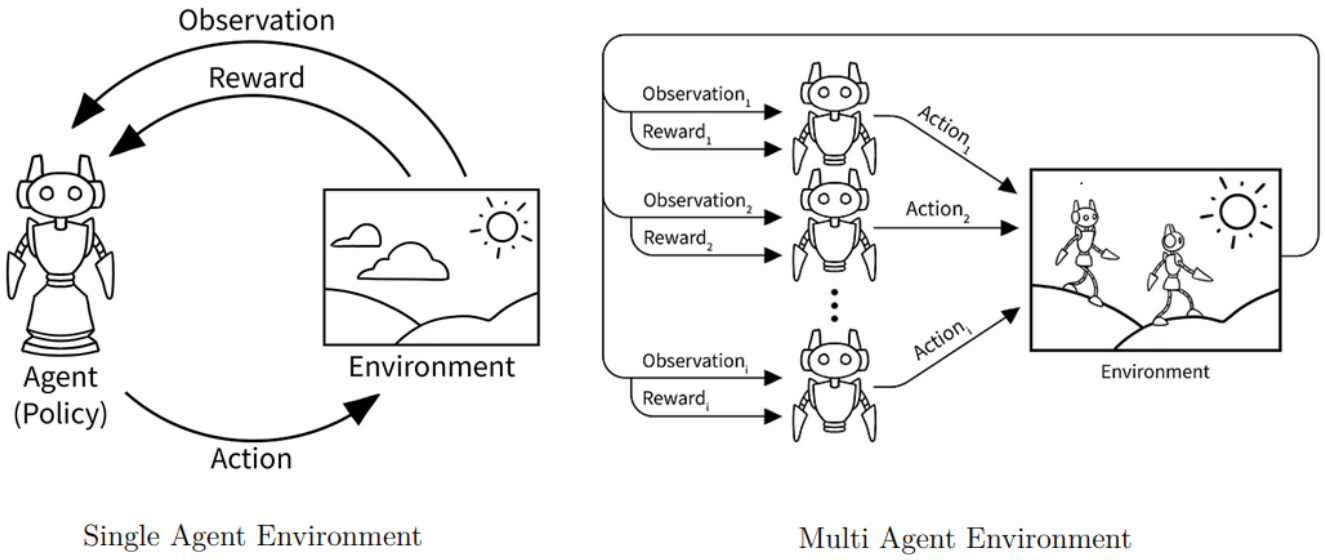


Figure 2.3: Agent type environments [14]

- For **single-agent** environments, there is only **one agent** (actor) who can perform actions in the environment.
- For the **multi-agent** environments, there are **at least two agents** (actors) and these agents could compete or cooperate to achieve common goals or individual goals depending on the system.

2.5.1 Nature of Interaction

Given that the reward functions play a central role in shaping agents' behavior, an important factor to consider when defining the reward function is how the agents' actions and decisions affect each other and the nature of their interactions.

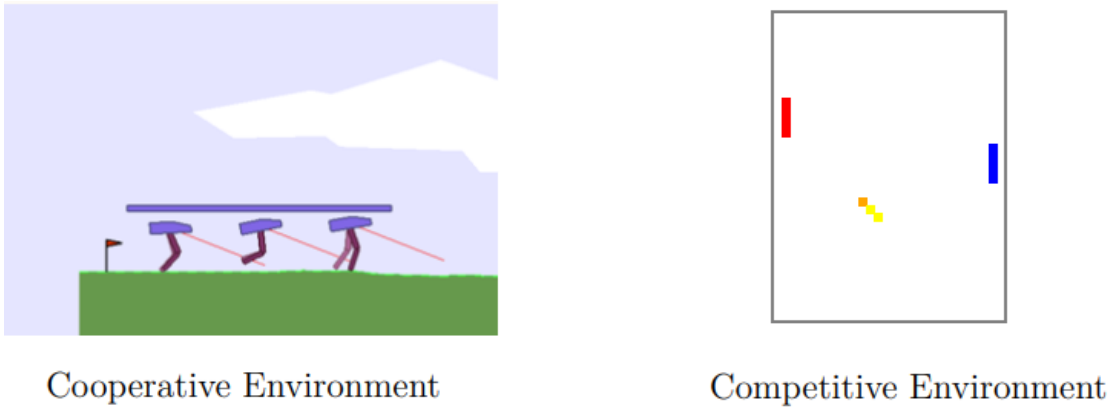


Figure 2.4: Nature of Interaction

- In **cooperative** environments, multiple agents work **together** to achieve a **common** goal.
- In **competitive** environments, agents **compete** against each other to **maximize** their **individual** rewards.

2.5.2 Training Schemes

In order to train the agents in the environment, there are 3 different approaches based on how to share information between the agents and the environment.

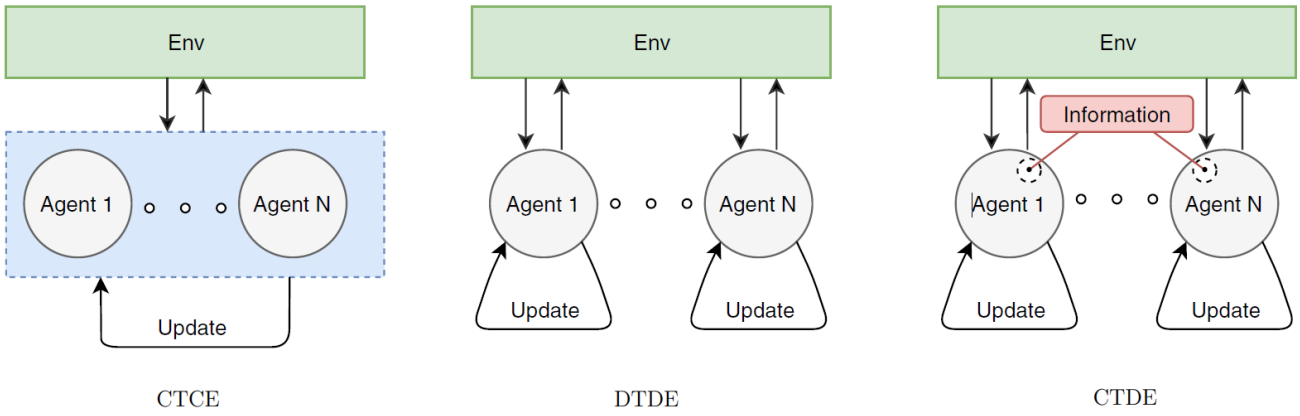


Figure 2.5: Training schemes [6]

- **CTCE** (Centralized Training, Centralized Execution): In the first approach of the figure 2.5, all agents have access to global information, including the observations of other agents. and for the execution phase, the system behaves as a single agent.
- **DTDE** (Distributed Training, Decentralized Execution): For the next approach, each agent learns its policy based on its local observations and rewards without access to global information or the states and actions of other agents. Similarly, for execution each agent makes decisions based solely on its local observations and the policy it has learned during the decentralized training phase.
- **CTDE** (Centralized Training, Decentralized Execution): Lastly, during training, agents collaborate in a centralized manner, sharing information such as the global state, other agents' status, and rewards. However, during execution, agents act independently without communication or coordination.

2.6 Bellman equation

Before Bellman equation, let's define the Markov Property, as it states that :

"Future is Independent of the past given the present"

Mathematically we can express this statement as the Eq. 2.2 :

$$P[s_{t+1} \mid s_t] = P[s_{t+1} \mid s_1, s_2, ..., s_t] \tag{2.2}$$

where s_t is the current state and s_{t+1} is the next state.
This equation means that the transition from state s_t to s_{t+1} is entirely **independent** of the past.

The **value function**, in the framework of reinforcement learning, defines how **good** a particular state is. The measure of how good a given state is, is based on the expected future rewards that will be achieved from that state.
The value of a state s is the expected future return of following policy $\pi(s, a)$ starting from s, and can be written as:

$$V_\pi(s) = E_\pi[R_t \mid s_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s] \tag{2.3}$$

The **action-value function** is defined in terms of the expected summation of the discounted rewards and represents the target Q-value 2.4:

$$Q_\pi(s, a) = E_\pi\ [\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a] \tag{2.4}$$

We define the **optimal** state-value function as:

$$V_{\pi}^{*}(s) = \max_{\pi} V_{\pi}(s) \ \forall s \in S \tag{2.5}$$

And similarly, the **optimal** state-action value function is given as:

$$Q_{\pi}^{*}(s,a) = \max_{\pi} Q_{\pi}(s,a) \ \forall s \in S, \ \forall a \in A \tag{2.6}$$

The idea of the **Bellman equation** is that instead of calculating each value as the sum of the expected return, which is a long process, we calculate the value as the sum of the immediate reward r_{t+1} + the discounted value of the state that follows $(\gamma * V(s_{t+1}))$.

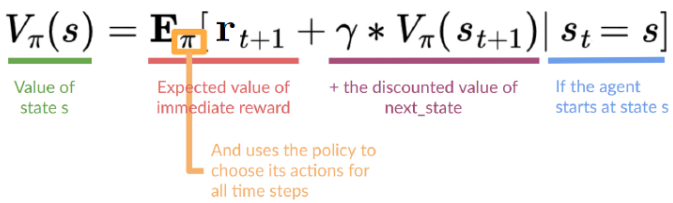


Figure 2.6: Bellman equation

2.7 Exploration/Exploitation

The agent does not initially know how its actions affect the environment, hence it has to learn this by trial and error (in an exploration phase). However, the agent should not only explore, to maximize the rewards of its action, it also has to exploit the gained knowledge (in an exploitation phase). Thus, there must be an **exploration-exploitation strategy** that is to be followed by the agent. One of these strategies is ϵ**-greedy**.
The **Epsilon-greedy strategy** is a policy that handles the exploration/exploitation trade-off.
The idea is that with an initial value of $\epsilon = 1.0$:

- With probability $1 - \epsilon$: we do exploitation (the agent selects the action with the highest state-action pair value).
- With probability ϵ: we do exploration (trying random action).

At the beginning of the training, the probability of doing exploration will be huge since ϵ is very high. So, most of the time, we'll explore. But as the training goes on, and consequently our policy gets better and better in its estimations, we progressively reduce the epsilon value since we will need less and less exploration and more exploitation.

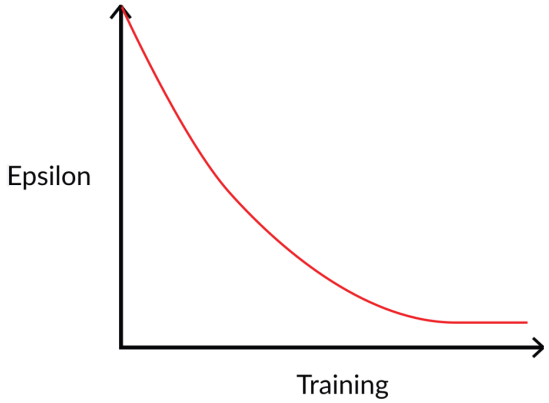


Figure 2.7: Epsilon Evolution [8]

2.8 Policy

The Policy is the **brain** of our Agent, it's the function that tells us what action to take, given the state we are in. So, it defines the agent's behavior at a given time step (state).
Our goal is to find the optimal policy π^*, the policy that maximizes expected return when the agent acts according to it. We find this π^* through training. There are two approaches to train our agent to find this optimal policy π^*:

- **Directly**, by teaching the agent to learn which **action** to take, given the current state: **Policy-Based Methods**.
- **Indirectly**, teach the agent to learn which **state** is more **valuable** and then take the action that leads to the more valuable states: **Value-Based Methods**.

Figure 2.8: Policy-based Vs Value-based method [8]

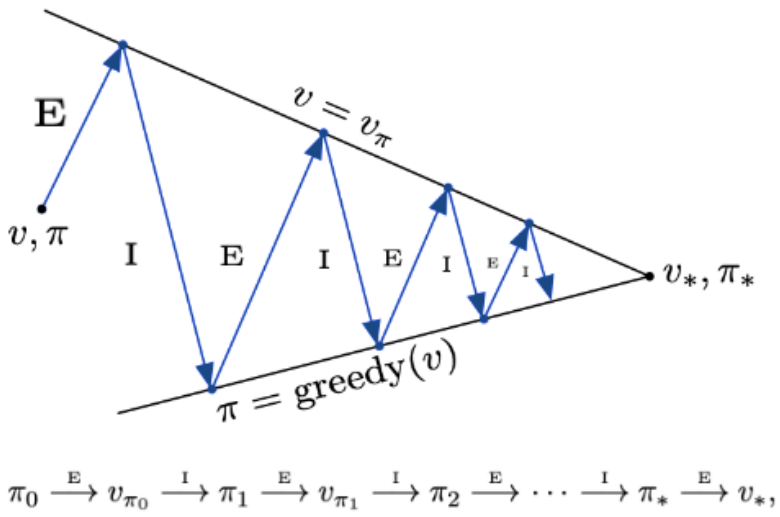


Figure 2.9: Policy evolution

Policy Iteration: The process of policy iteration consists of an iterative cycle of policy evaluation (shown as $\xrightarrow{E}$) and policy improvement (shown as $\xrightarrow{I}$).
Policy evaluation computes the value function for the current policy whereas policy improvement updates the current policy concerning the evaluated value function. The following figure was taken and modified from [19].

In terms of policy, several strategic approaches can be considered to address the challenges and opportunities in Reinforcement learning. we outline the primary policy options below:

2.8.1 Q-Learning

Q-learning [9] is a **value-based** method that trains its action-value function (Q-Table). the algorithm that we use to train our Q-function, an action-value function that determines the value of being at a particular state and taking a specific action at that state.
Q-learning updates the action selection using the Bellman optimal equations and the ϵ-greedy policy. Unlike other reinforcement learning algorithms, Q-learning has simple Q-functions, hence it has become the foundation of many other reinforcement learning algorithms.
Given a state and action, our Q Function outputs a state-action value (also called Q-value).
Internally, our Q-function is encoded by a **Q-table**, a table where each cell corresponds to a state-action pair value. Think of this Q-table as the memory or cheat sheet of our Q-function.

Figure 2.10: Q-learning Diagram

The Q-learning algorithm:

1. We initialize the Q-table.
2. Choose an action a_t with the highest Q-value given the state s_t.
3. Perform action a_t, get reward r_t and next state s_{t+1}.
4. Update $Q(s_t, a_t)$ using **Bellman equation** :

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[\, r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

New Q-value estimation — Former Q-value estimation — Learning Rate — Immediate Reward — Discounted Estimate optimal Q-value of next state — Former Q-value estimation — Target — Error

Figure 2.11: Q-learning update equation

The initialization of the Q-table is a crucial step that can influence the learning speed and the effectiveness of the policy learned by the agent. there is various possibilities for it:
There is the zero initialization, All Q-values are initialized to zero, it is the easiest way but it is slower in convergence.
We have the Random Initialization where the Q-values are initialized to small random values and it encourages Exploration as the random values can lead the agent to explore various actions initially but the agent can face a problem of instability.
These are the most common initialization for the Q-table.

Here is the algorithm of Q-learning :

Algorithm 1 Q-learning

Input : policy π, positive integer *num_episodes*, small positive fraction α
Output: value function Q ($\approx q_\pi$ if *num_episodes* is large enough)

Initialize Q arbitrarily (e.g., $Q(s,a) = 0$ for all $s \in S$ and $a \in A(s)$, and $Q(\text{terminal-state}, \cdot) = 0$)
for $i \leftarrow 1$ ***to*** *num_episodes* **do**
  Observe S_0
  $t \leftarrow 0$
  repeat
    Choose action a_t using policy derived from Q (e.g., ϵ-greedy)
    Take action a_t and observe r_{t+1}, r_{t+1}
    $Q(s_t, s_t) \leftarrow Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$
    $t \leftarrow t + 1$
  until
  s_t is terminal
end
return Q

2.8.2 DQN

For our environment the state space is **gigantic**, due to this, creating and updating its Q-table would not be efficient. In this case, the best idea is to approximate the Q-values using a deep learning network.

Deep learning. It is the subset of machine learning methods based on **neural networks** with representation learning. The adjective "deep" refers to the use of multiple layers in the network. Methods used can be either supervised, semi-supervised or unsupervised.

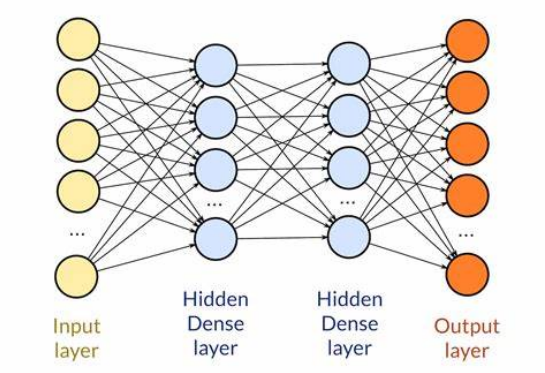


Figure 2.12: Deep learning diagram

Neural networks can be used for a variety of tasks, such as recognition images, machine translation, time series prediction, recommendation products, and much more.

Deep learning networks are appreciated for their ability to model complex relationships in data and make accurate predictions about new examples. In the Reinforcement Learning, they are used to overcome memory space problems.

Deep Reinforcement Learning [16]: It is the combination of reinforcement learning and deep learning. Deep reinforcement learning algorithms are capable of handling very large amounts of data, something that was not possible with simple reinforcement learning.
DQN was one of the first algorithms to demonstrate that deep learning can be successfully applied to reinforcement learning tasks

Deep Q learning architecture: This neural network will approximate, given a **state**, the different Q-values for each possible action at that state. And that's exactly what Deep Q-Learning does.

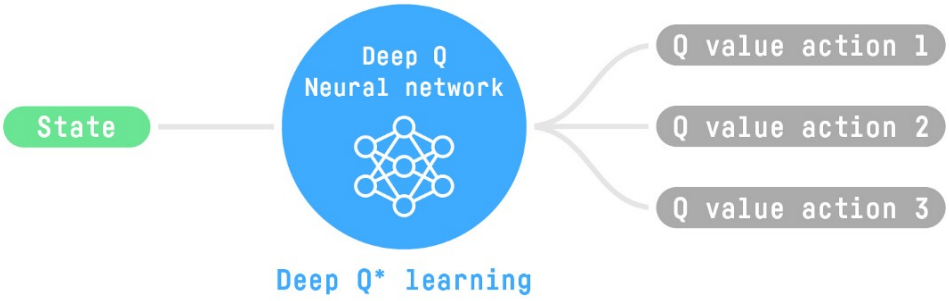


Figure 2.13: Deep Q-Network diagram

In Deep Q-Learning, we create a **loss** function that compares our Q-value prediction and the Q-target (same as the Q-learning target) and uses gradient descent to update the weights of Deep Q-Network to approximate our Q-values better.

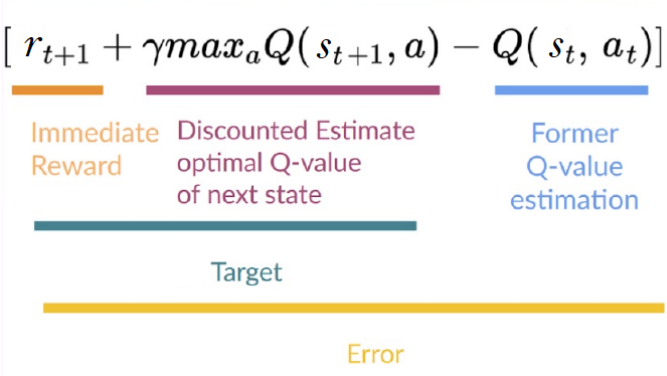


Figure 2.14: DQN equation

Experience Replay: In Reinforcement Learning, an experience e can be described as the **knowledge** produced by the agent performing an action a_t in a state s_t causing a new state s_{t+1} and a generated reward r.

The experience can be expressed as a tuple $e(s_t, a_t, r_t, s_{t+1})$. So, we get the proposed technique called Experience Replay, where experiences are stored in a replay memory D and used to train the agent. Since experiences are stored in the memory, and some experiences might be paramount, they can repeatedly be reused to train the agent which improves convergence.

Although experience replay should help the agent theoretically learn from previous important experiences, it entails sampling experiences uniformly from the replay memory D regardless of their significance.

The Deep Q-learning training algorithm has two phases:

- **Sampling**: We perform actions and store the observed experience tuples in a replay memory (Experience replay).
- **Training**: Select a small batch of tuples randomly and train it with gradient descent.

Algorithm 2 Deep Q-Learning with Experience Replay algorithm

Input : Replay memory D with capacity N, discount factor γ, action-value function Q and target action-value function $\hat{Q}$

Output: Optimized value function Q

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta' = \theta$
for *episode = 1, M* **do**
  Observe State s_1
  for $t = 1, T$ **do**
    % Sampling
    With probability ϵ select a random action a_t, otherwise select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t in the environment and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    % Training
    Sample random mini-batch of transitions (s_j, a_j, r_j, s_{j+1}) from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_a \hat{Q}(s_{j+1}, a) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $(y_j - Q_\theta(s_j, a_j))^2$ with respect to the network parameters θ
    every C steps reset $\hat{Q} = Q$
  end
end
return Q

2.8.3 A2C

The A2C method [13], a **hybrid** architecture combining value-based and policy-based methods, it is an evolution of the Actor-Critic method and widely used in training agents for complex tasks:

- An **Actor** that controls how the agent behaves (policy-based method).
- A **Critic** that measures how well is the taken action (value-based method).

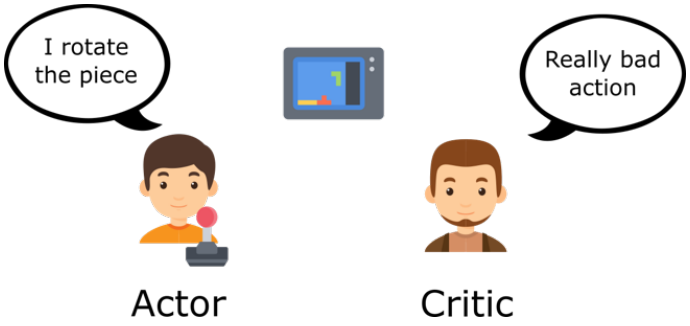


Figure 2.15: A2C illustration

Here we have the A2C algorithm:

1. At t, we get the state $\boldsymbol{s_t}$ and pass it as input through our Actor and Critic.
2. Our Policy (Actor) takes the state and outputs an action $\boldsymbol{a_t}$.
3. The Critic takes that action and using $\boldsymbol{s_t}$ and $\boldsymbol{a_t}$, computes the value of taking that action at that state.
4. The action $\boldsymbol{a_t}$ performed outputs a reward $\boldsymbol{r_t}$ and a new state $\boldsymbol{s_{t+1}}$.
5. The Actor updates its policy parameters using the Q-value.
6. The Critic then updates its value parameters.
7. The Actor produces the next action to take at $\boldsymbol{a_{t+1}}$ given the new state $\boldsymbol{s_{t+1}}$.

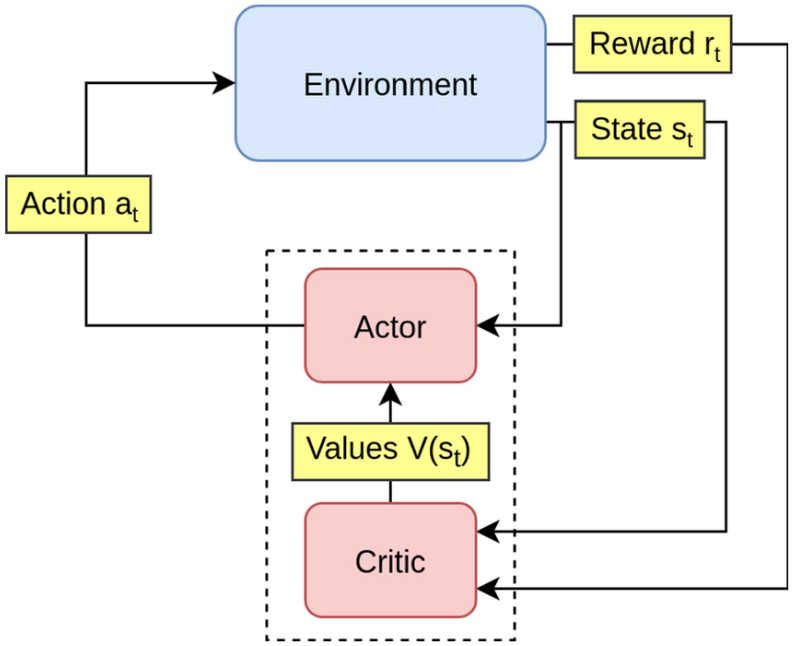


Figure 2.16: Simplified A2C Diagram [12]

We can **stabilize** learning further by using the **Advantage function** as Critic instead of the Action value function. More precisely, the function given by 2.7 calculates the extra reward we get if we take this action at that state compared to the mean reward we get at that state.

$$A(s,a) = Q(s,a) - V(s) \tag{2.7}$$

Where:

- $\boldsymbol{Q(s,a)}$: Q value for action a in state s
- $\boldsymbol{V(s)}$: average value of that state

The extra reward is what's beyond the expected value of that state.

- If $\boldsymbol{A(s,a) > 0}$: The gradient is pushed in that direction.
- If $\boldsymbol{A(s,a) < 0}$: The chosen action does worse than the average value of that state. So, the gradient is pushed in the opposite direction.

The advantage function is essential for enhancing the stability and efficiency of the learning process. By centering the updates around zero, it effectively reduces the variance of the policy gradient updates, leading to a more stable and reliable learning environment.

The full process of A2C is shown below:

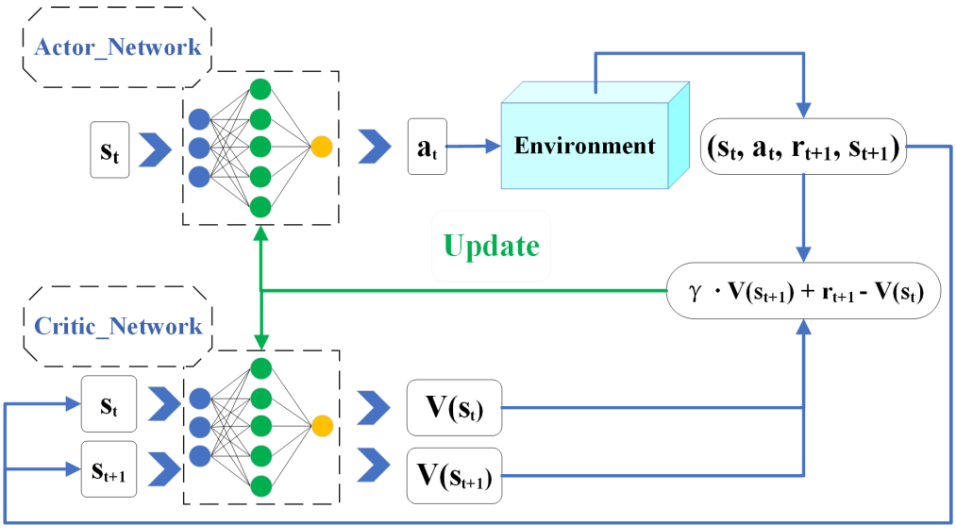


Figure 2.17: A2C flow chart

Algorithm 3 Advantage Actor Critic algorithm

Input : Maximum number of steps S_{max}, Actor-New network A_{new} , Actor-Old network A_{old} , Critic network C, Number of actor network updates N_A, Number of critic network updates N_C, Replay Memory M

Output: The optimal Actor-New network A_{new} and Critic network C

Initialize replay memory with capacity L
Initialize the parameters of A_{new} and A_{old}
for *episode = 1, M* **do**
    Observe s_1
    for *t= 1, S_{max}* **do**
        Choose action a_t based on the output distribution of the A_{new}
        Take action a_t, observe reward r_t and next state s_{t+1}
        Store transition(s_t, a_t, r_t, s_{t+1}) in M
    end
    reach S_{max}
    for *t=1, T* **do**
        Obtain value from the Critic network $v = \gamma V(s_{t+1}) + r_{t+1} - V(s)$
        Obtain advantage value A_t from Critic network C
        if *the replay memory size exceeds L* **then**
            Train and update the parameters of A_{new} for N_A times
            Train and update the parameters of C for N_C times
            Replace the parameter of A_{new} with A_{old}
            Clear replay memory M
        end
    end
end
return A_{new} , C

2.8.4 PPO

The idea with Proximal Policy Optimization (PPO) is that we want to improve the training stability of the policy by limiting the change you make to the policy at each training epoch, we want to avoid having too large of a policy update. For two reasons:

- We know empirically that smaller policy updates during training are more likely to converge to an optimal solution.
- A too-big step in a policy update can result in falling "off the cliff" (getting a bad policy) and taking a long time or even having no possibility to recover as shown below in figure 2.18.

Figure 2.18: PPO Illustration

With PPO, the idea is to constrain our policy update with a new objective function called the **Clipped surrogate objective** function that will constrain the policy change in a small range using a clip as shown in Eq. 2.8.

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[\ min(r_t(\theta)\ \hat{A}_t,\ clip(r_t(\theta),\ 1-\epsilon,\ 1+\epsilon)\ \hat{A}_t] \tag{2.8}$$

This new function is designed to avoid destructively large weights updates.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.9}$$

The clipping rate Eq. 2.9 is the probability of taking action at state in the current policy, divided by the same for the previous policy.

- If $r_t(\theta) > 1$, the action at state is **more** likely in the current policy than the old policy.
- If $0 < r_t(\theta) < 1$, the action is **less** likely for the current policy than for the old one.

	$p_t(\theta) > 0$	A_t	**Return Value of** *min*	**Objective is Clipped**	**Sign of Objective**	**Gradient**
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	+	$p_t(\theta)A_t$	no	+	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	−	$p_t(\theta)A_t$	no	−	✓
3	$p_t(\theta) < 1-\epsilon$	+	$p_t(\theta)A_t$	no	+	✓
4	$p_t(\theta) < 1-\epsilon$	−	$(1-\epsilon)A_t$	yes	−	**0**
5	$p_t(\theta) > 1+\epsilon$	+	$(1+\epsilon)A_t$	yes	+	**0**
6	$p_t(\theta) > 1+\epsilon$	−	$p_t(\theta)A_t$	no	−	✓

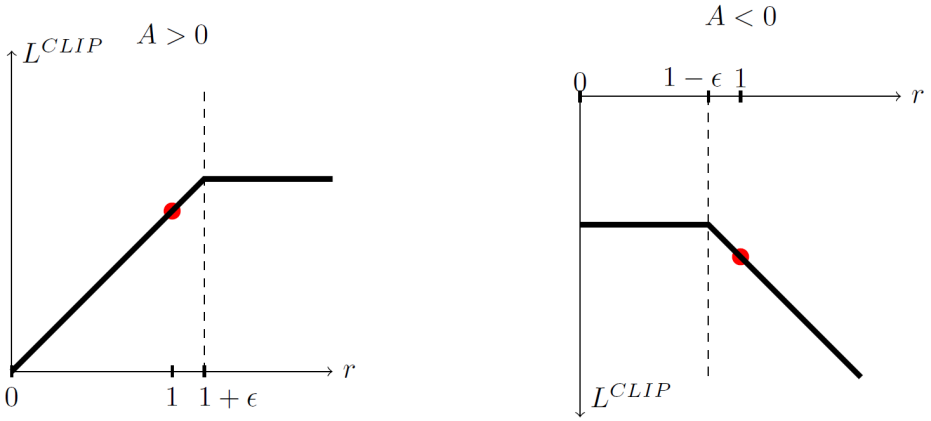


Figure 2.19: Different clips possibilities
[18]

Based on 2.19:

1. In situations 1 and 2, the clipping does not apply since the $r_t(\theta)$ is between the range $[1-\epsilon, 1-\epsilon]$.
2. In situations 3 and 4, if $r_t(\theta) < 1-\epsilon$, the probability of taking that action at that state is much lower than with the old policy.
   (a) If $A > 0$: then we increase the probability of taking that action at that state.
   (b) If $A < 0$: we don't want to decrease further the probability of taking that action. Therefore, the gradient $= 0$ (since we're on a flat line), so we don't update our weights.

3. In situations 5 and 6, if $r_t(\theta) > 1 + \epsilon$, the probability of taking that action at that state in the current policy is much higher than in the former policy.

   (a) If $A > 0$: we don't want to get too greedy. We already have a higher probability of taking that action at that state than the former policy. Therefore, the gradient = 0 (since we're on a flat line), so we don't update our weights.

   (b) If $A < 0$: we want to decrease the probability of taking that action at that state.

To recap, we only update the policy with the unclipped objective part.
When the minimum is the clipped objective part, we don't update our policy weights since the gradient will equal 0.

We update our policy only if:

- Our ratio is in the range $[1 - \epsilon, 1 - \epsilon]$.
- Our ratio is outside the range, but the advantage leads to getting closer to the range:
  - Being below the ratio but the advantage is > 0
  - Being above the ratio but the advantage is < 0

To summarize, using this clipped surrogate objective, we restrict the range that the current policy can vary from the old one. Because we remove the incentive for the probability ratio to move outside of the interval since the clip forces the gradient to be zero.
If the ratio is $> 1 + \epsilon$ or $< 1 - \epsilon$ the gradient will be equal to 0.

Algorithm 4 PPO pseudo-code

Input: Initialize θ_0 initial policy parameters, set clip parameter ϵ

for *k=0,1,2,...* **do**
  Collect a set of trajectories (s, a, r, s) according to the policy $\pi_k = \pi(\theta)$ and store them in a memory buffer D_k
  Estimate the advantage function $\hat{A}_t^{\pi_k}$
  Update the policy $\theta_{k+1} = arg \max_{\theta} L_{\theta_k}^{\text{CLIP}}(\theta)$ using Eq 2.8
end

Chapter 3

Traffic Signal Control Application

In this chapter, we apply reinforcement learning system highlighted above in the Traffic Signal Control in order to have an automated agent for traffic signal control.

3.1 Tools

3.1.1 SUMO

SUMO (**S**imulation of **U**rban **MO**bility) is an open source, highly portable, microscopic and continuous traffic simulation package designed to handle large networks. It allows for inter-modal simulation including pedestrians and comes with a large set of tools for scenario creation. It is mainly developed by employees of the Institute of Transportation Systems at the German Aerospace Center [5].

3.1.2 TraCI

TraCI (Traffic Control Interface) connects to a SUMO simulation in a programming language (in this case Python) to allow for feeding inputs and receiving outputs. it gives access to a running road traffic simulation, enabling the control of the behavior of multiple simulation objects during a live simulation. It allows for external scripts to interact with the simulation and its vehicles, pedestrians, and infrastructure.

3.1.3 SUMO-RL

SUMO-RL is a python package that provides a simple interface to instantiate Reinforcement Learning (RL) environments with **SUMO** for Traffic Signal Control and it Provides a simple interface to work with Reinforcement Learning [1].

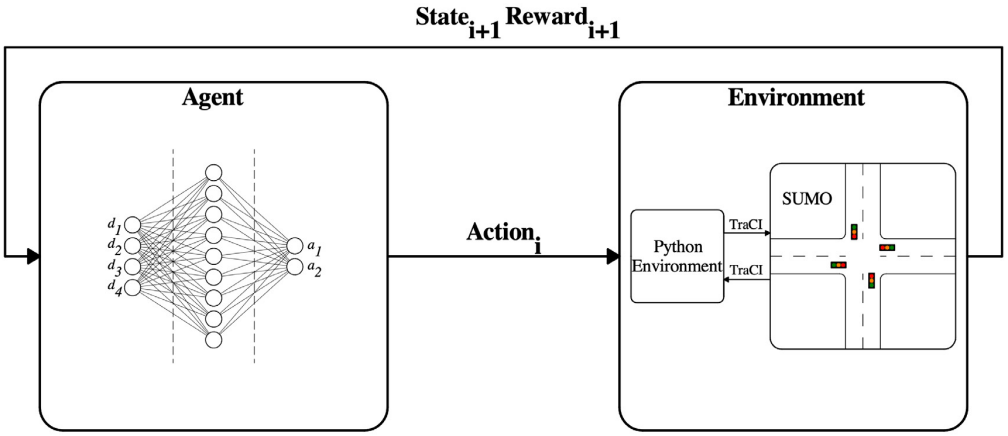


Figure 3.1: SUMO-RL Environment [3]

The Python training loop is designed according to the OpenAI gym standards. Hence the decisions of the agent are channeled into TraCI commands by the Python environment. Then the TraCI commands trigger changes in the SUMO infrastructure. The change's measures are acquired by TraCI and formulated appropriately by the Python environment to represent the infrastructure's state to the agent.

3.1.4 Gymnasium

Gymnasium is an open source Python library for developing and comparing reinforcement learning algorithms by providing a standard API to communicate among learning algorithms and environments, as well as a standard set of environments compliant with that API. This is a fork of OpenAI's Gym library by its maintainers, and is where future maintenance will occur going forward. [Towers et al.]

Gymnasium

Figure 3.2: Gymnasium logo

3.1.5 Stable baselines3

Stable Baselines3 (SB3) is a set of reliable implementations of reinforcement learning algorithms in PyTorch. It is the next major version of Stable Baselines. [15]

Figure 3.3: Stable Baselines3 logo

3.2 Environment and agents

- Traffic signal control is commonly applied on **intersections** composed of multiple entry points, also known as approaches (arrows on Figure 3.4). These approaches meet on the crossing area of the intersection, on which multiple traffic streams can cross (gray zone).
- A traffic stream can engage on the crossing area when it has a right of way over the intersection, it is given by a traffic light controller.
- As shown in Figure 3.5, there is a left turn lane, a straight lane, and a right turn lane in the direction of each intersection entrance.
- Our environment (Figure 3.6) is a **4x4** traffic signal grid, with **16 traffic signal controllers** (one in each intersection), which are the learning agents.
- Traffic signals in our scenario have a minimum and maximum time they must remain green. They are referred to as ***min_green*** and ***max_green***.
- When the traffic signal phase switches, the **yellow light** is turned on for a certain period between the green and red signal phases for safety purposes.
- The signal control of intersection is mainly for motor vehicles, and it does not consider the setting of pedestrian exclusive phase since the pedestrian crossing behaviour is often accompanied by the movement of a vehicle.

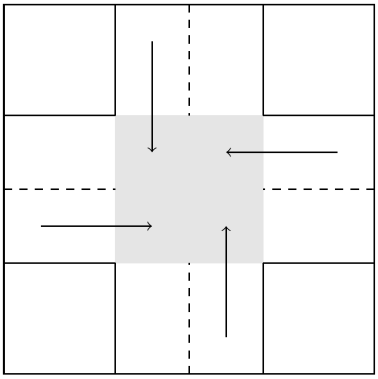


Figure 3.4: An example 4-way intersection [21]

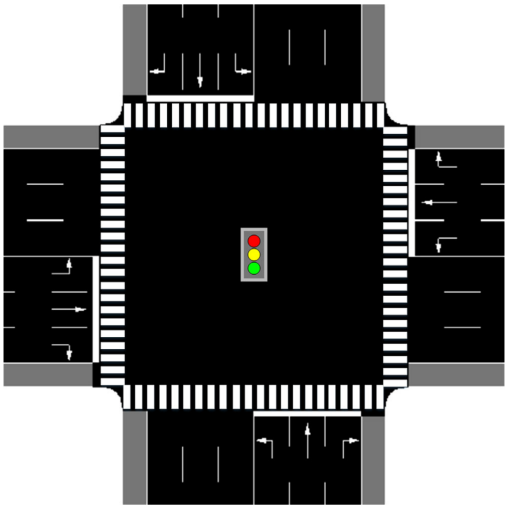


Figure 3.5: One intersection [7]

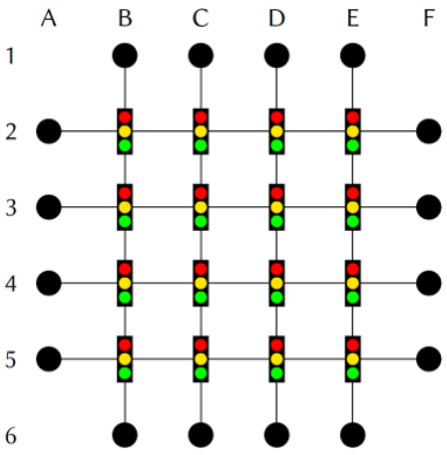


Figure 3.6: 4x4 Grid map [4]

3.3 Parameters

In this section, we mention some of the most important parameters related to the SUMO environment.

Parameter	Description	Used value
delta_time	Simulation seconds between actions	2s
min_green	Minimum green time in a phase	5s
max_green	Maximum green time in a phase	300s
yellow_time	Time in seconds for the yellow phase before green phase	2s
reward_fn	Defines the reward function	x

Table 3.1: SUMO parameters

3.4 Action Space

The action space is ***discrete(8)***. Every ***delta_time*** seconds, each traffic signal agent can choose the **next** green phase configuration.

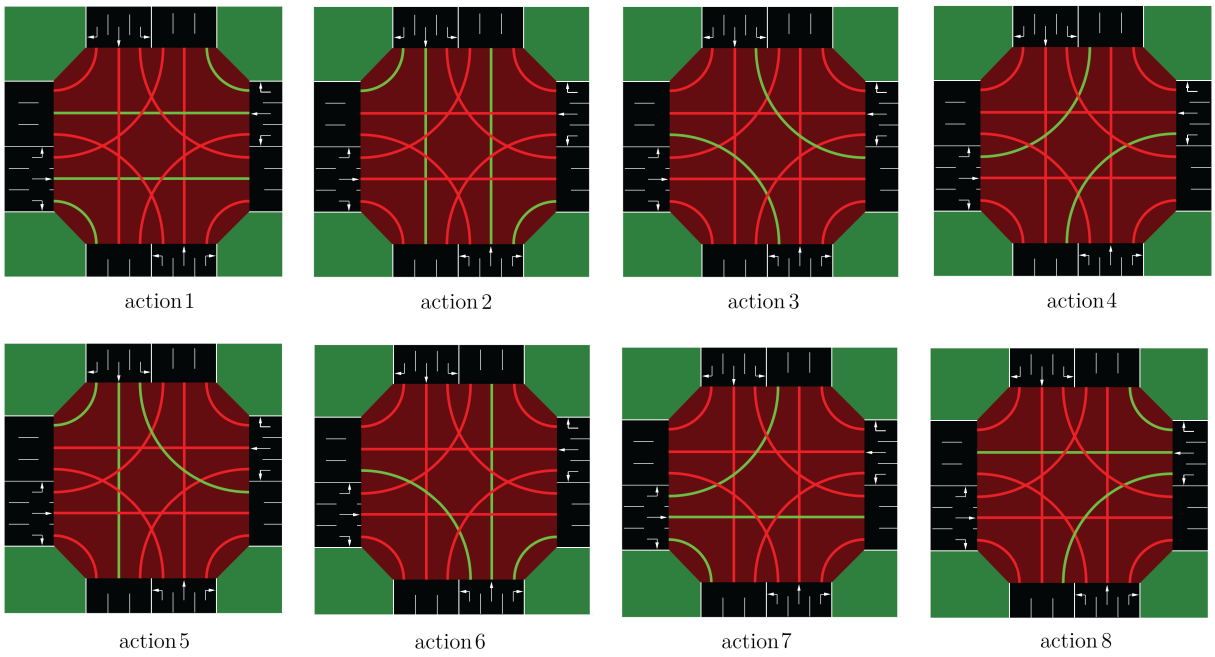


Figure 3.7: Action space A for each agent

The figure 3.7 shows the available actions for each agent, in other words, **each agent** of the **16** traffic lights agents have the same set of this 8 available actions.

Example

To understand how that work does, we set an example from SUMO:

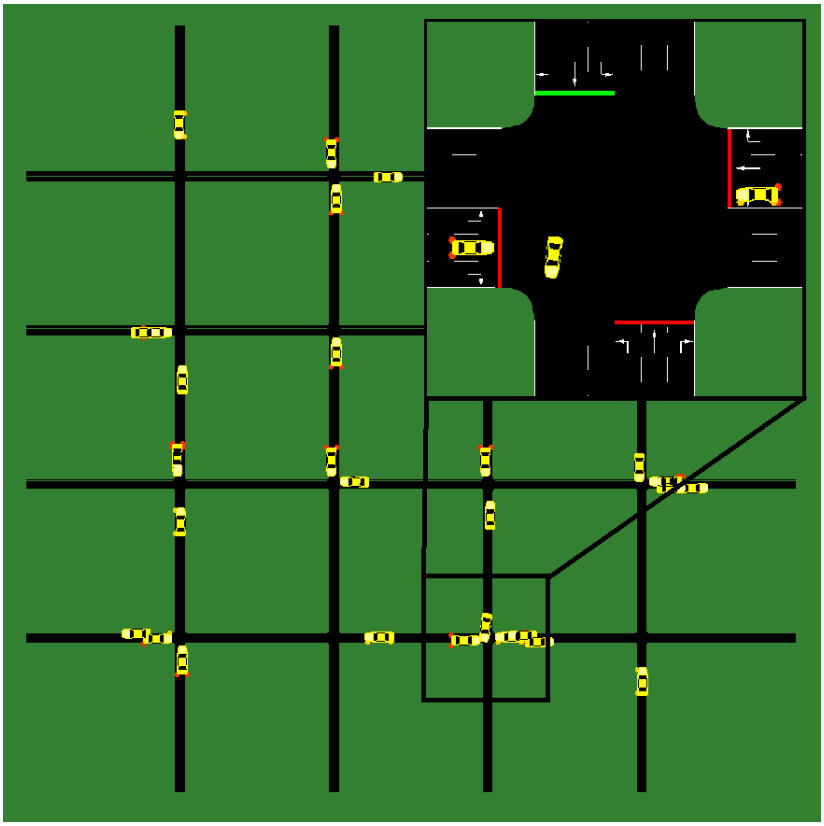


Figure 3.8: SUMO example

The figure 3.8 shows the **full 4x4 grid map** and the spawned cars, noting that each intersection is a learning agent.

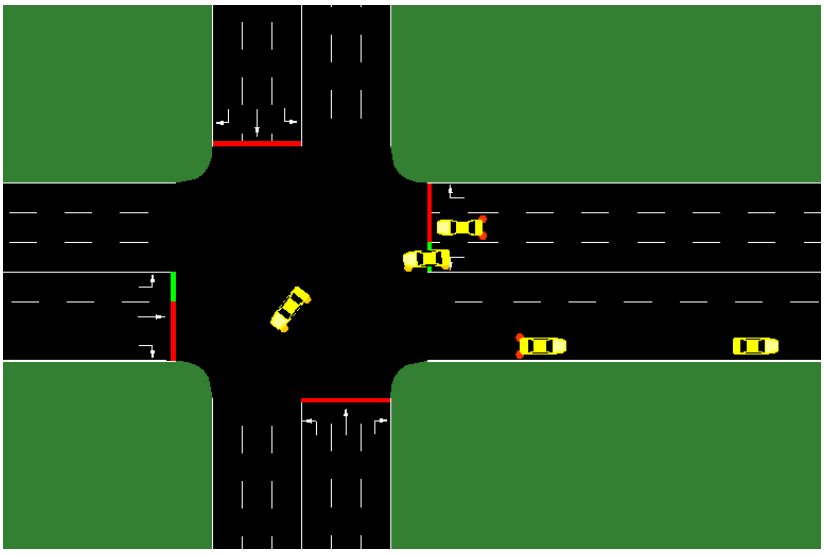


Figure 3.9: Scenario 1 of an intersection

In the intersection of figure 3.9, with the light configuration and given the action space above, we can conclude that the **action 4** is active at the moment.

The green and red lines in the figure represent traffic signal lights for each lane.

In this case, only cars coming from EAST to SOUTH (E-S) or WEST to NORTH (W-N) of the intersection have green signals, allowing them to proceed.

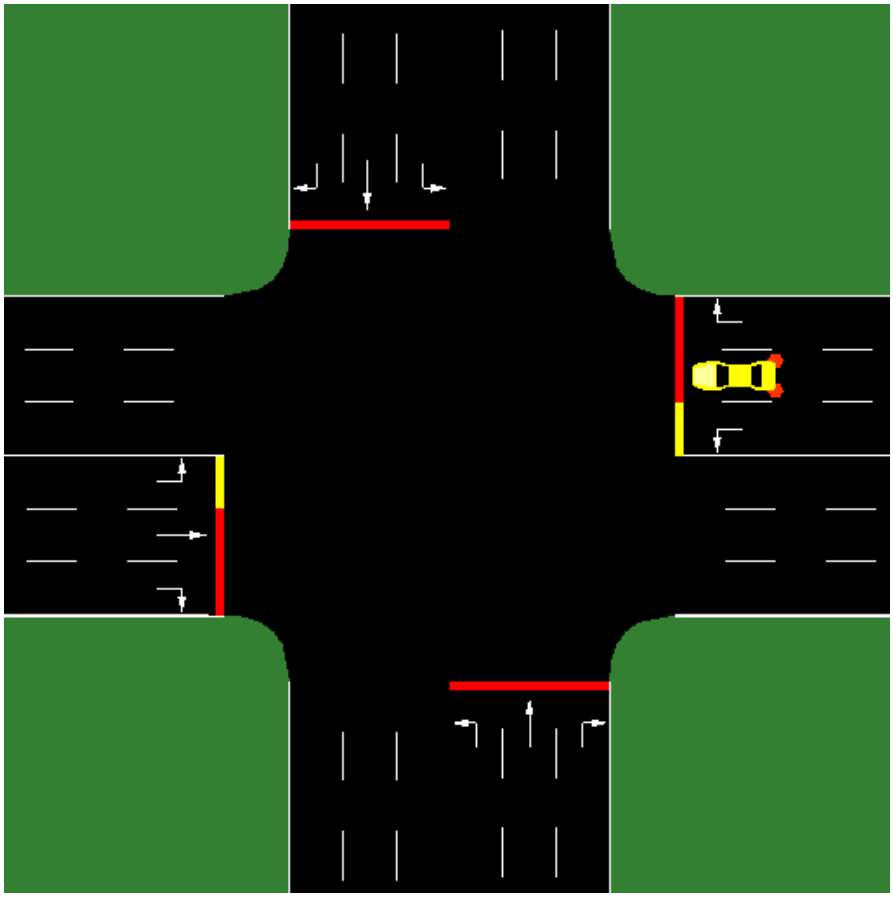


Figure 3.10: Scenario 2 of an intersection

Next, the second intersection 3.10 shows **no green** light but there are the yellow lights, which are applied before applying the chosen green phase configuration (in this case **action 4** too).

The figure 3.10 is the phase just before the one in 3.9 as they have the same traffic light configuration, so it starts with yellow lights (for safety purposes) for ***yellow_time*** seconds, then it becomes green for at least ***min_green*** seconds.

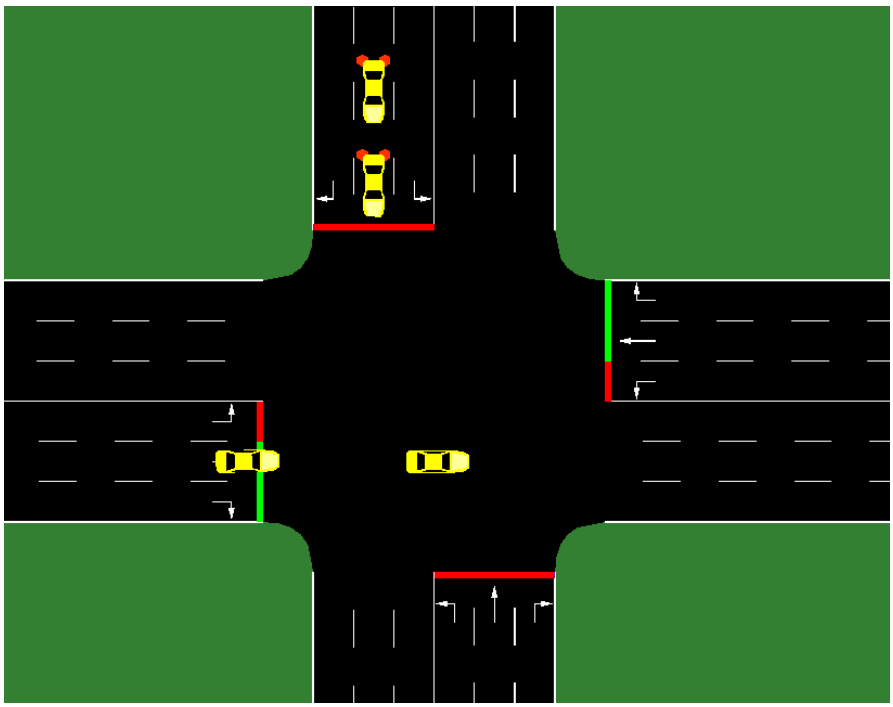


Figure 3.11: Scenario 3 of an intersection

Lastly, this intersection shows another possible action from the action set A which is the **action 1** and it gives the right only to the cars coming from EAST and going WEST (E-W) or NORTH (E-N), or coming from WEST and going to EAST (W-E) or SOUTH (W-S).

3.5 Observation Space

The observation space for each traffic signal agent is a vector composed of:

Obs=[phase_one_hot, min_green_binary, lane_1_density ,..., lane_n_density, lane_1_queue, ..., lane_n_queue]

- ***phase_one_hot***: one-hot encoded vector indicating the **current** active green phase.
- ***min_green_binary***: is a binary variable indicating whether the current green phase time have passed the **min_green** (explained above) seconds or no.
- ***lane_i_density***: is the number of vehicles in incoming lane i divided by the total capacity of the lane.
- ***lane_i_queue***: is the number of queued (speed below 0.1 m/s) vehicles in incoming lane i divided by the total capacity of the lane.

For our environment, the observation vector is 33 in length as follows:

- ***phase_one_hot*** vector has a length of 8 as there is 8 available actions in the action space.
- The ***min_green_binary*** variable is used to prohibit the agent from changing the green configuration before it reaches the ***min_green*** seconds. To simplify:
  - If ***min_green_binary = 0*** : (means current green phase time $<$ ***min_green***) The agent is not allowed to switch the current green phase configuration until it reaches the ***min_green***.
  - If ***min_green_binary = 1*** : (means current green phase time $\geq$ ***min_green***) The agent can switch the green light configuration.
- ***lane_i_density*** and ***lane_i_queue*** have a length of 12 for each one as there is 3 lanes in each one of the 4 entrances in the intersection.
- Noting also that the total capacity of the lane is calculated as length of the lane divided by the length of the vehicle + the gap between cars in the lane.

3.6 Reward

In the context of a traffic signal control system, **various** possibilities exist for defining the **reward** function dependently on the usage, such as:

- **Waiting time**: Minimization of accumulated vehicles waiting time in each time-steps batch (the vehicle in state of waiting if it has a speed less than 0.1 m/s).

$$R_t = -\sum_i W_i \tag{3.1}$$

Or

$$R_t = -\sum_i W_i^2 \tag{3.2}$$

Or

$$R_t = -\sum_i W_i - \gamma \sum_e W_e \tag{3.3}$$

Where:

- W_i is the waiting time of vehicle i.

- W_e is the waiting time of emergency vehicle e.

- γ is a large weighting factor to prioritize emergency vehicles.

  – The reward function 3.2 assigns higher penalties to vehicles with longer waiting times.

  – For 3.3, the agent penalizes more emergency vehicles in order to prioritize them.

- **Queue length**: minimization of number of vehicles waiting in the queue.

$$R_t = -\sum_j Q_j \tag{3.4}$$

Where:

  – Q_j is the number of vehicles in queue j.

- **Average speed**: Maximization of vehicles average speed.

$$R_t = \frac{1}{N}\sum_i S_i \tag{3.5}$$

Or

$$R_t = \frac{1}{N}\sum_i S_i^2 \tag{3.6}$$

Or

$$R_t = \frac{1}{N}\sum_i S_i - \lambda \sum_i max(0, V_{threshold} - S_i) \tag{3.7}$$

Where:

- S_i is the speed of vehicle i.

- N is the total number of vehicles.

- $V_{threshold}$ is the speed threshold below which penalties are applied.

- λ is a weighting factor for the penalty.

  – The reward function 3.6 gives a higher reward for higher average speeds, encouraging faster traffic flow.

  – This version 3.7 adds a penalty for vehicles traveling below a certain speed threshold.

- **Pressure**: Minimization of the difference between the number of vehicles approaching an intersection and the number of vehicles leaving it.

$$R_t = -\sum_j P_j \tag{3.8}$$

Or

$$R_t = -\sum_j \mid P_j \mid \tag{3.9}$$

Or

$$R_t = -\sum_j P_j^2 \tag{3.10}$$

Where:

- $P_j = N_j^{in} - N_j^{out}$ is the pressure for lane j.
- N_j^{in} is the number of vehicles approaching lane j.
- N_j^{out} is the number of vehicles leaving lane j.
  - This reward function 3.9 aims to balance the pressure across all approaches to avoid high congestion in any single direction.
  - This version 3.10 prioritizes reducing pressure in lanes with higher pressure to prevent congestion buildup.

- Or any custom reward function (the possibility to combine different reward functions).

3.7 State-Of-Art

In this section, we mention the results collected from various papers.

In [7], Huang and al. discuss the significant role of traffic signal control in urban traffic management. The existing systems often use fixed signal timing schemes, which are inefficient and inflexible, leading to difficulties in adapting to varying traffic flows. So, they introduce the LSTM-PPO single-point signal control model, a novel method that uses Proximal Policy Optimization (PPO), combined with Long Short-Term Memory (LSTM) networks to enhance the efficiency of traffic signal control operations. This method encodes traffic state information into a format suitable for an LSTM network, which then determines the optimal signal phases.

But first, let's define the LSTM network [17] , Long Short-Term Memory (LSTM) is a recurrent neural network (RNN) architecture that has been designed to address the vanishing and exploding gradient problems of conventional RNNs. RNNs have cyclic connections making them powerful for modeling

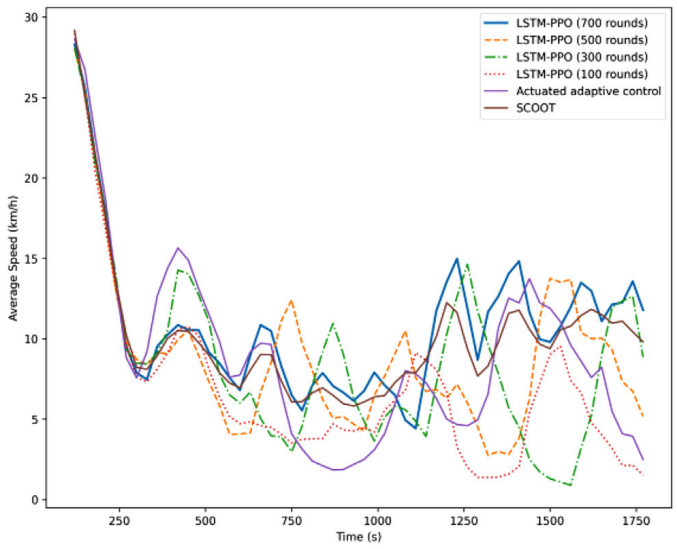
sequences. However, in contrast to the deep neural networks, the use of RNNs in speech recognition has been limited to phone recognition in small scale tasks. LSTM architectures makes more effective use of model parameters to train acoustic models for large vocabulary speech recognition.
The iterative training of the adaptive intelligent strategy adjusted the signal control strategy. It uses the SUMO environment and compares this method with the LSTM-PPO model to verify the model's control performance, self-learning and self-adaptive capabilities.

Parameter name	100 rounds	300 rounds	500 rounds	700 rounds
Average speed (km/h)	8.232	9.945	10.029	11.944
Average share (%)	23.956	21.624	19.392	18.834
Average queue length (pcu)	6.608	6.017	5.166	4.899
Average waiting time (s)	4.042	4.012	3.767	3.519
Average traffic (pcu/h)	169.831	178.475	182.542	211.525
Average training time (min)	0.862	0.865	0.861	0.863
Average decision time (ms)	2.319	2.285	2.307	2.301

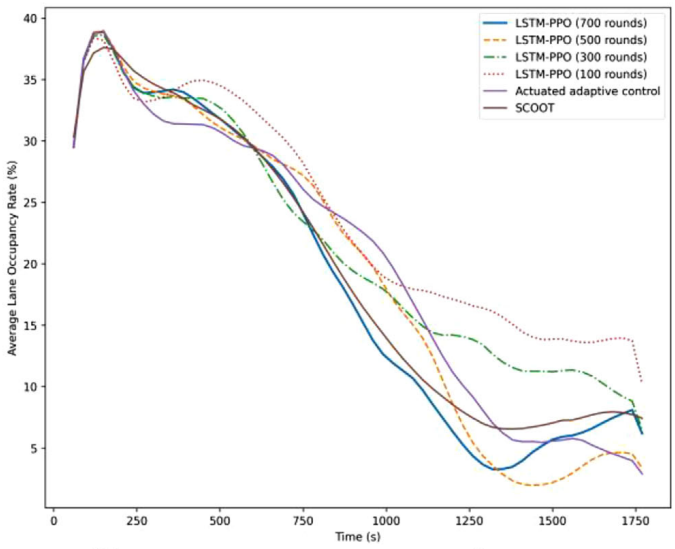
Figure 3.12: Various results of PPO-LSTM

In the study, the proposed PPO-LSTM algorithm for traffic signal control is bench-marked against other traditional and advanced methods to evaluate its performance such as the fixed-time control, actuated adaptive control method and SCOOT which is a real-time, online, dynamic, and stable traffic system that continuously monitors the traffic demand in the approach lanes of each intersection of the road network, and it is one of the adaptive signal control methods.

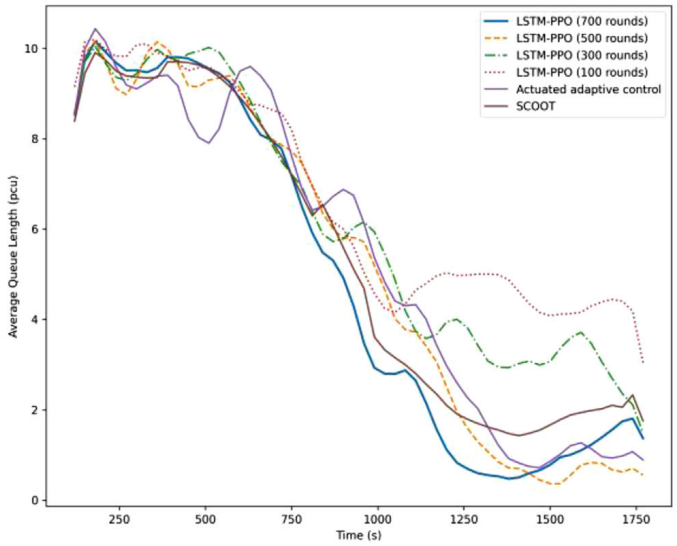
- **Fixed-Time Control**: A conventional approach where traffic signal timings are predetermined and do not change based on real-time traffic conditions. This method is often used as a baseline for comparison.
- **Actuated Adaptive Control**: Aims to improve efficiency at individual intersections, reducing wait times and responding to immediate demand.
- **Adaptive Signal Control Technologies** : Aims to optimize overall traffic flow across larger areas, reducing congestion, and improving travel times on a network scale.



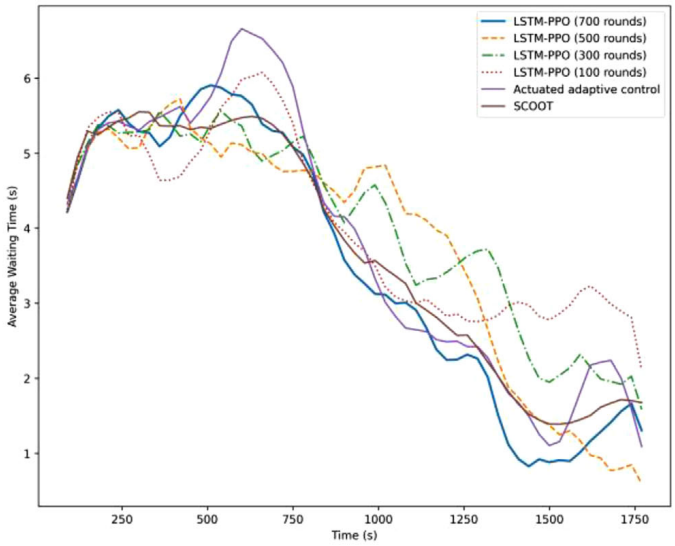
(a) Speed change curve



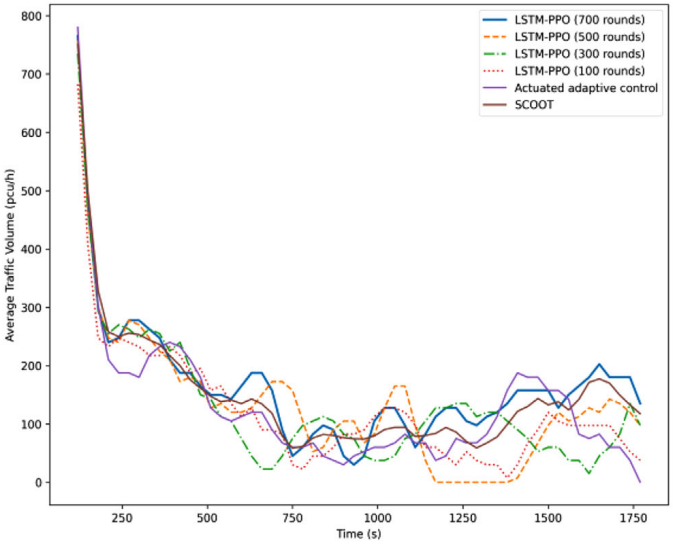
(b) Average occupancy rate change curve



(c) Average queuing length change curve



(d) Average waiting time change curve



(e) Average traffic volume change curve

As shown in Figure 3.7, under the control of the newly-introduced model, the average speed and the average traffic flow have been greatly improved, while on parallel the average lane occupancy rate, the average queue length, and the average waiting time were decreased in different magnitudes, and the operation efficiency of the intersection has been in a state of continuous improvement.

When comparing the control effects of each model, the LSTM-PPO model with 700 rounds of iterative simulation training outperformed the other models.

Specifically, after 700 rounds of simulation training the average running speed of vehicles at the intersection gradually increased from 8.232 km/h to 11.944 km/h, which corresponds to an increase of 45.09%. The average lane occupancy rate, the average queue length, and the average parking waiting time were decreased by 21.38%, 25.86%, and 12.94%, respectively, while the average traffic volume was increased by 24.55%.

It is obvious from the changing of the trend of these five evaluation indicators that the operating efficiency of the traffic flow at intersections has been significantly improved, showing that with the increase of the number of simulation rounds.

However, the LSTM-PPO model still has room for further improvement in terms of convergence and stability. In the case of single intersection signal control, the proposed model only needs to sense the change of traffic flow state at the intersection, while in the case of multi-intersection signal control, a multi-agent signal control system needs to be established. In order to realize signal phase coordination between intersections, it is necessary to further improve the expression of traffic state space and realize multi-scale perception of traffic flow characteristics.

Kodama and al. [11] propose a traffic signal control system that can function without direct communication or transfer cost, and aims to improve the learning performance of the traffic signal control system. First, the signal control system in this study perceives the change in congestion on a road within the intersection every hour. This observation helps to predict the behavior of traffic signals in the vicinity without the need for information transfer between the intersections. However, this problem becomes a multi-agent problem, and when dealing with reinforcement learning, the learning performance may be adversely affected by the simultaneous learning problem. Therefore, Kodama and al. use the dual targeting algorithm (DTA) [10] to reduce the impact of the simultaneous learning problem. Compared to DQN, the DTA is a deep reinforcement learning method that places a higher emphasis on the success experience.

It has been experimentally confirmed that a method that strongly reinforces the success experience, such as Profit Sharing, can be effectively used to address the simultaneous learning problem, and this study aims to achieve this effect through DTA. And the authors have compared this model with Traffic signal control systems that use the existing deep reinforcement learning methods.

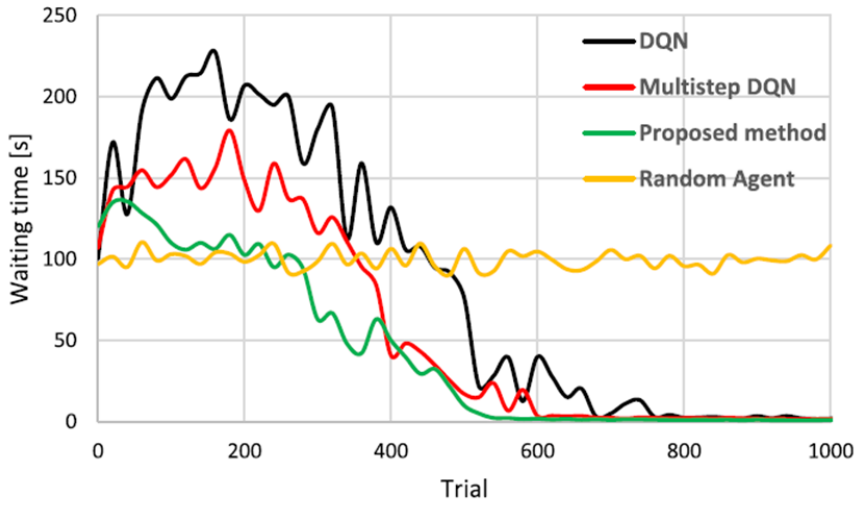


Figure 3.13: Proposed method comparison

Traffic flow simulation results show that the proposed method reduces the waiting time by more than 33% compared with similar TSCSs using DQN and multi-step - DQN. The learning time to convergence is also shorter than that of the existing methods, indicating that the method is effective even when real time learning is considered.

Almeida and al. [4] introduce a multi-agent system with agents learning independently based on k-nearest neighbors, it is presented as an option to control traffic signals in real-time.
The proposed method estimates the Q-values of the current state by calculating the weighted average of the Q-value estimates of the k-nearest states, based on the euclidean distance metric. The closer a neighbor state is, the greater the impact its Q-value estimates have on the Q-values of the current state. It then selects an action based on an exploration strategy like Epsilon-greedy, transitions to a new state and receives a reward.
Subsequently, it calculates the error based on the reward and the expected value of the last and new state, and uses this error to update the Q-value estimates of all the k-nearest states that contributed to estimate the Q-values of the previous state.
There are two different traffic contexts, which correspond to two different vehicle flow rates:

- **Context 1 (NS = WE)**: one vehicle is inserted in all 8 Origin-Destination pairs every 3 seconds.
- **Context 2 (NS < WE)**: one vehicle is inserted every 6 seconds in the 4 OD pairs in the North-South direction and one vehicle is inserted every 2 seconds in the 4 OD pairs in the West-East direction.

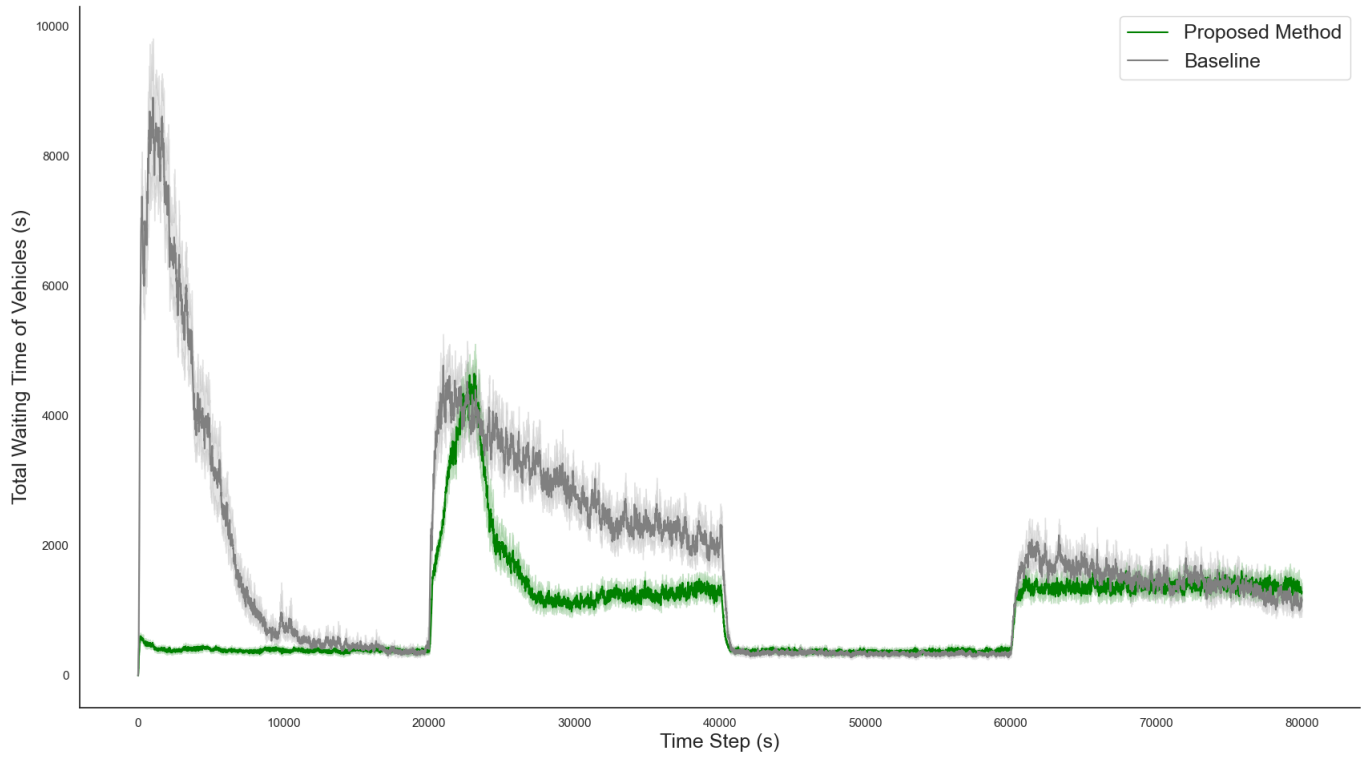


Figure 3.14: k-NN model evolution

Each simulation runs for 80,000 seconds and switches contexts every 20,000 seconds; thus there are three changes in context (Context 1 → Context 2 → Context 1 → Context 2).

Figure 3.14 shows the average waiting time (over all vehicles) for this method and the baseline [2], along the 80,000 seconds of simulated real-world traffic dynamics, which represent 16,000 actions taken (since the agent takes an action every 5 seconds). The vehicle flow also changed at 20,000 seconds, 40,000 seconds and 60,000 seconds

Comparing the proposed method with the baseline [2], it is clear that this approach outperforms the baseline, converging sooner. The first time the agents explore each of the two contexts (from 0 to 20,000 seconds for Context 1, and from 20,000 seconds to 40,000 seconds for Context 2), in both of them, the proposed method converged much faster than the baseline.

In the first paper [7], the authors have trained the agent using the LSTM-PPO algorithm only on a single-point signal but the tried various reward functions (average speed, average share, average queue length, etc..) so there is no comparison between different algorithms.

The second paper [11], the authors have created smaller environment with 9 intersections only and focused only on the waiting time as a reward function.

Lastly, in paper [4] and same as the previous paper, Almeida, Bazzan and Abdoos have used just the total waiting time as reward function in two different contexts but they haven't tried their proposed method with other reward functions.

Chapter 4

Practical Section

In order to provide a balanced work for our practical portion, we attempt to integrate the various contributions made in the aforementioned works. We next apply the described algorithms to various reward functions, comparing the various outcomes to obtain a complete understanding of the training procedure.
The three applied algorithms (PPO, A2C, DQN) are trained using reward functions for **average speed** and **vehicle waiting time** in addition to a comparison with a **random agent** as it serves as a baseline, showing the expected poor performance without any learning or strategy, maintaining a low average speed and high total waiting time throughout the training.

4.1 Settings

Although carefully selecting the hyper-parameters is essential for improved training outcomes, we were unable to tune every one of them in our work due to the lengthy tuning process. We did, however, select the optimal parameters for each of the three algorithms.

4.1.1 PPO

For PPO, there is various parameters so that we chose the most important and affecting ones. Starting with:

Hyper-parameter	Description	Value range	Used value
learning_rate	Corresponds to the strength of each gradient descent update step	[0, 1]	0.00062
n_steps	The number of steps to run per update	[32, 5000]	256
batch_size	Mini-batch size	[4, 4096]	512
gamma	Discount factor	[0, 1]	0.95
clip_range	Clipping range parameter (ϵ)	[0.1, 0.3]	0.3

Table 4.1: PPO parameters

4.1.2 A2C

A2C requires few less hyper-parameters as it is almost the same as PPO but without the clipping function part.

Hyper-parameter	Description	Value range	Used value
learning_rate	Corresponds to the strength of each gradient descent update step	[0, 1]	0.0007
gamma	Discount factor	[0, 1]	0.99
vf_coef	Determines how much importance the algorithm places on minimizing the value function loss	[0.25, 1]	0.5
ent_coef	A measure of randomness in the policy distribution	[0, 0.1]	0.05

Table 4.2: A2C parameters

Noting that the total loss is calculated as 4.1.

$$Total\ Loss = Policy\ Loss + vf\_coef * Value\ Loss - ent\_coef * Entropy \tag{4.1}$$

The entropy term is used to encourage exploration by adding a penalty for being too certain about the action to take.

4.1.3 DQN

DQN policy could be considered as the main and the general algorithm and from which the rest of the algorithms are inherited and improved.

Hyper-parameter	Description	Value range	Used value
buffer_size	Size of the replay buffer	[10 000, 1 000 000]	1 000 000
exploration_initial_eps	Initial value of random action probability (ϵ)	[0.9, 1]	1.0
exploration_final_eps	Final value of random action probability (ϵ)	[0.01, 0.1]	0.05
exploration_fraction	Fraction of entire training period over which the exploration rate is reduced	[0.1, 0.5]	0.1

Table 4.3: DQN parameters

4.2 Average speed reward function

The goal of the agents, with this reward function, is to **maximize** the average speed of the vehicles in the environment, which is a metric commonly used.

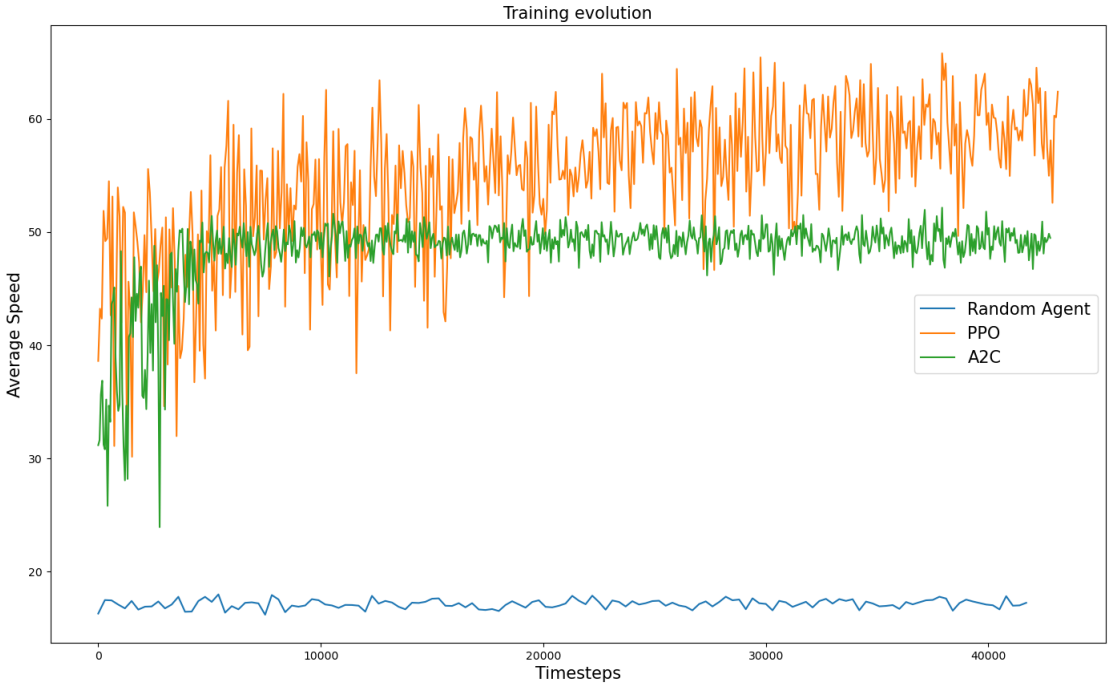


Figure 4.1: Average speed optimization evolution

As the figure 4.1 shows, the **random agent** has average speed of **10 m/s** and does not improve over time and maintains consistently poor performance.

While the **PPO** agent has the best average speed that reaches around **55 to 60 m/s** within the first 5,000 time-steps. but doesn't stabilize even after 50.000 time-steps and it exhibits high variability.

Lastly, the **A2C** agent has stabilized with less than 10.000 time-steps but it has almost **50 m/s** as an average speed. it does not achieve the same high peaks, indicating that PPO consistently outperforms A2C.

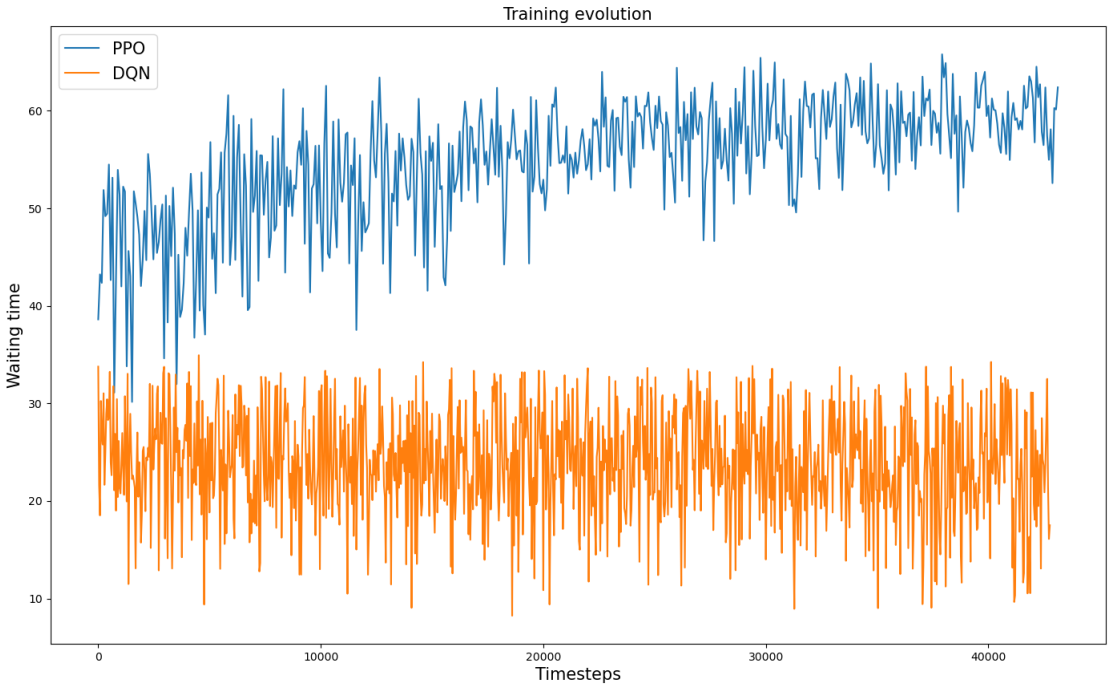


Figure 4.2: PPO and DQN comparison for average speed

Due to the huge instability of DQN agent, we were obliged to split the graphics and compare PPO and DQN only. the final result 4.2 shows that the **DQN** algorithm has limited improvement in average speed. The average speed stabilizes in a range between **20 and 30 m/s**. the DQN agent shows high variability and does not achieve the same level of performance as PPO or even A2C.

4.3 Waiting time reward function

Unlike the goal of the agents for the average speed, this reward function aims to **minimize** the vehicles waiting time in different lanes of the map.

As mentioned previously, a vehicle whose speed is less than 0.1 m/s is considered to be waiting. Reducing the average waiting time of vehicles is an important objective for traffic signal control, as it reduces vehicle queue sizes and improves traffic flow.

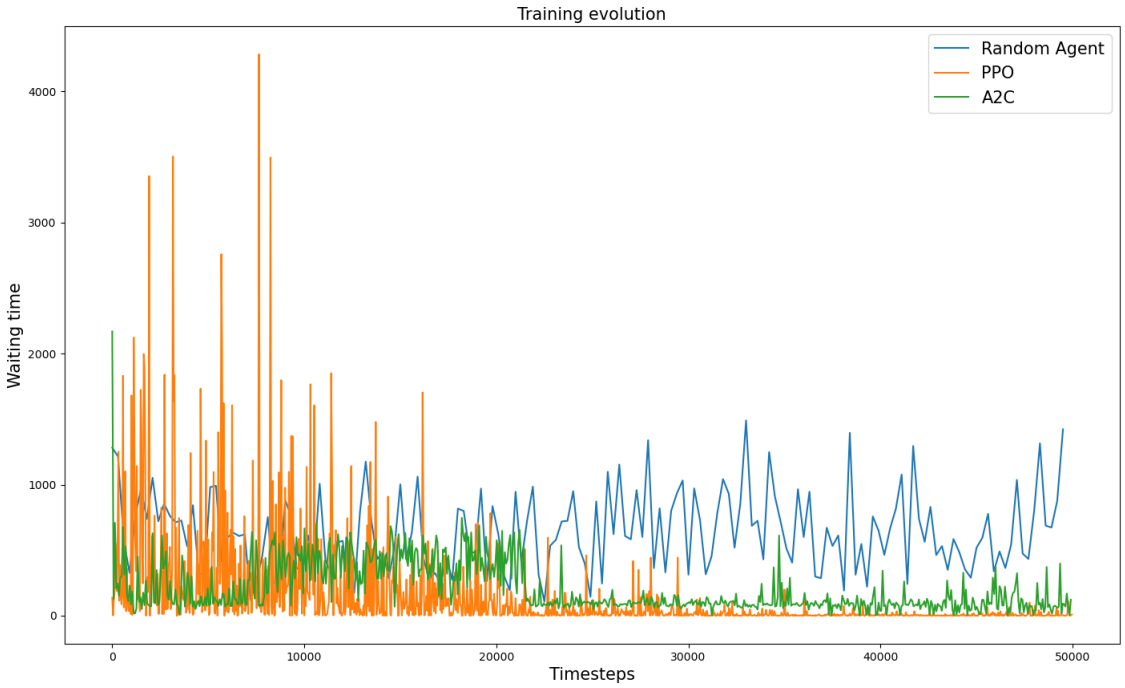


Figure 4.3: Waiting time training evolution

Based on figure 4.3, the **random agent** shows that the waiting time for the random agent remains relatively high and variable throughout the training period. It exhibits significant fluctuations, indicating inconsistent performance.

For **PPO**, it shows a high variability in waiting time, especially in the initial stages of training, but it started stabilizing after 21.000 time-steps and it reaches almost a **0s of waiting time** which is really good result.

Lastly, for **A2C** and same as the previous reward function, at the beginning it was more stable than PPO and after 22.000 time-steps it got better, however **PPO** agent has a **better final** average waiting time.

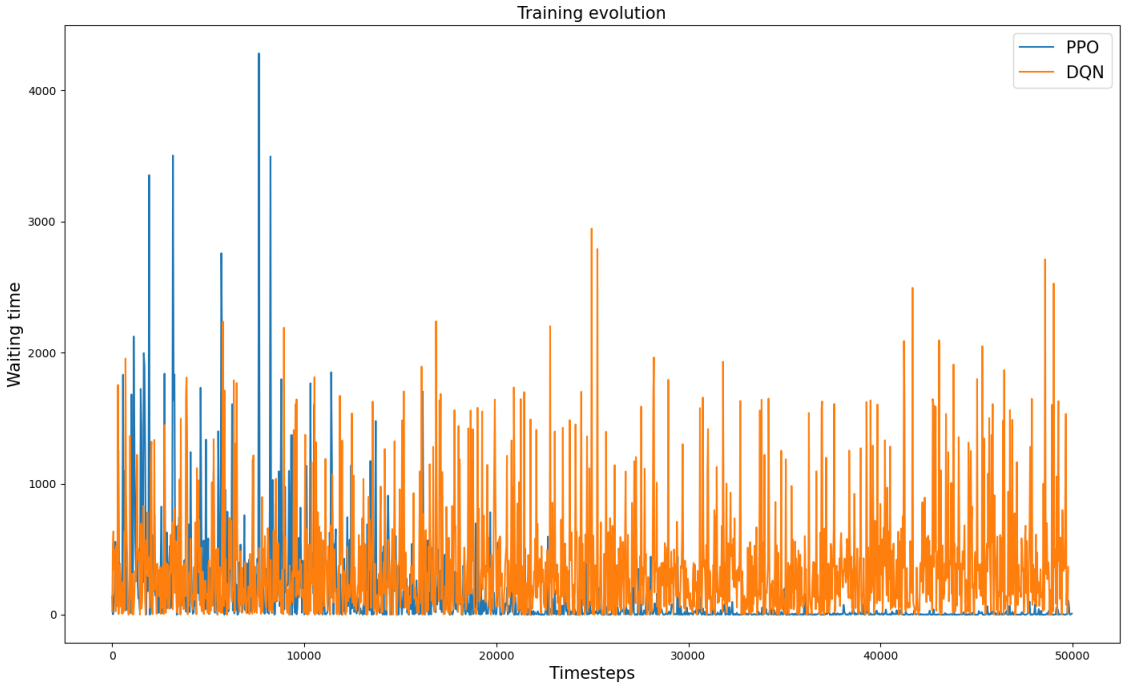


Figure 4.4: PPO and DQN comparison for waiting time

Same case for DQN with the average speed reward function. So, we split graphs and take PPO and DQN agents together, the figure shows that DQN algorithm faces huge problem of instability and divergence even after 50.000 time-steps.

4.4 Results

Reward	PPO	A2C	DQN	Random agent
Waiting time(s)	145.372	203.309	374.498	635.282
Average speed (m/s)	55.637	48.494	23.930	17.155

Table 4.4: Average values table

As the table 4.4 shows, confirming the results above, the **PPO agent** has the **best results** among all other agents for both reward functions with average waiting time of **145.372 s** over all training time-steps and mean value of the average speed of **55.637 m/s**. The second place goes to **A2C agent** with average waiting time **203.309 s** and **48.5 m/s** as average speed. And lastly, the **DQN agent** got the worst results among all the agents with **375s** of average waiting time and **24 m/s** as average speed.

Conclusion

In this project, reinforcement learning proved to be the most effective solution to the primary issue, despite numerous challenges in setting up the environment and implementing the multi-agent algorithms. Our main objective was to minimize average waiting time and maximize average vehicles speed by optimizing traffic flow through dynamic signal modifications.

Nevertheless, we achieved favorable outcomes, particularly with regard to the PPO agent, who was shown to be the most efficient, constantly generating the shortest wait times and the greatest average speed, exhibiting exceptional flexibility in response to changing traffic circumstances. This, despite the agent's initial volatility during each reward function, produced the best overall outcomes.

Although it trailed PPO in terms of convergence speed and efficiency, A2C also demonstrated strong performance. DQN, on the other hand, produced the least desirable outcomes, frequently convergent to sub-optimal policies and unable to successfully adapt to the multi-agent environment.

But we have encountered numerous challenges and barriers in order to arrive at this outcome. Initially, we faced difficulties obtaining sufficient materials because this topic is relatively new and the publications we have used are all current. Therefore, the primary issue is that papers and other learning resources don't include a section on coding.

Gym demands a specific format for the environment, the action space, the state space, etc., therefore some expertise with SUMO is necessary to make maps and integrate them in Python. We have also encountered issues implementing the process of mapping the simulation to utilize it as an environment.

We were also forced to use the default values after building the environment in Gym due to a lack of guidance on altering some parameters, such as the vehicle's spawning rate or speed restriction, etc... so that, we were obliged to read and understand the whole source code of SUMO-RL in order to get more details about these parameters. Moreover, when applying reinforcement learning methods to the used traffic signal control system, factors such as pedestrians, non-motorized vehicles, and road

conditions may need to be taken into consideration to improve the robustness of the model.

Even though we achieved good results, we were forced to employ single-agent versions of the multi-agent methods due to their difficulties in applying in the multi-agent environment. This brings us to our final big issue.

Even with these results, the work is still ongoing, with plans to investigate methods to improve the scalability of multi-agent algorithms to handle larger and more complex traffic networks, as well as hybrid models that combine the strengths of different multi-agent algorithms or integrate traditional traffic engineering methods to create more comprehensive and effective solutions. And finally, instead of the current map, which is overly simplified and fails to project real-life roads, a more complex and realistic map should be created.

Bibliography

[1] Alegre, L. N. (2019). SUMO-RL. https://github.com/LucasAlegre/sumo-rl.

[2] Alegre LN, Bazzan ALC, d. S. B. (2021). Quantifying the impact of non-stationarity in reinforcement learning-based traffic signal control. *PeerJ Computer Science*.

[3] Bálint, K., Tamás, T., and Tamás, B. (2022). Deep reinforcement learning based approach for traffic signal control. *Transportation Research Procedia*, 62:278–285. 24th Euro Working Group on Transportation Meeting.

[4] de Almeida, V. N., Bazzan, A. L. C., and Abdoos, M. (2022). Multiagent reinforcement learning for traffic signal control: a k-nearest neighbors based approach. In *ATT@IJCAI*.

[5] DLR (2024). Sumo documentation.

[6] Gronauer, S. and Dieopold, K. (2022). Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55:1–49.

[7] Huang, L. and Qu, X. (2023). Improving traffic signal control operations using proximal policy optimization. *IET Intelligent Transport Systems*, 17(3):592–605.

[8] HuggingFace (2022). Deep reinforcement learning course.

[9] Jang, B., Kim, M., Harerimana, G., and Kim, J. (2019). Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, PP:1–1.

[10] Kodama, N., Harada, T., and Miyazaki, K. (2019). Deep reinforcement learning with dual targeting algorithm. pages 1–6.

[11] Kodama, N., Harada, T., and Miyazaki, K. (2022). Traffic signal control system using deep reinforcement learning with emphasis on reinforcing successful experiences. *IEEE Access*, 10:128943–128950.

[12] Le, N., Rathour, V., Yamazaki, K., Luu, K., and Savvides, M. (2021). Deep reinforcement learning in computer vision: a comprehensive survey. *Artificial Intelligence Review*, 55.

[13] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning.

[14] Muñoz, V., Prados, A., and Barber, R. (2023). An introduction to apprenticeship learning by teaching an agent how to play a video game.

[15] Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8.

[16] Roderick, M., MacGlashan, J., and Tellex, S. (2017). Implementing the deep q-network.

[17] Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition.

[18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

[19] Sutton, R. and Barto, A. (2018). *Reinforcement Learning, second edition: An Introduction.* Adaptive Computation and Machine Learning series. MIT Press.

[Towers et al.] Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., de Cola, G., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, A. J. S., and Younis, O. G. Gymnasium.

[21] Tréca, M. (2022). *Designing traffic signal control systems using reinforcement learning.* PhD thesis.

Scan me:

Resume

This report details the outcomes of a graduation project conducted at the Analysis and Control of PDEs laboratory. The project's main goal is to design and implement a sophisticated multi-agent system where agents autonomously learn to control traffic signals in real-time through the application of advanced deep reinforcement learning algorithms. The research focuses on comparing the effectiveness of three distinct algorithms: Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep Q-Network (DQN) in managing urban traffic signals. The results indicate that the PPO algorithm significantly outperforms the others, achieving substantial reductions in average waiting times and notable increases in the average speed of vehicles within the traffic network. The findings underscore the critical role of dynamic traffic signal control in alleviating congestion and enhancing the flow of urban mobility. The project also highlights the potential of machine learning techniques in optimizing complex urban systems, providing a promising direction for future research and practical applications in smart city initiatives.

Résumé

Ce rapport présente les résultats d'un projet de fin d'études réalisé au sein du laboratoire d'Analyse et de Contrôle des PDE. L'objectif principal du projet est de concevoir et de mettre en œuvre un système multi-agents sophistiqué où les agents apprennent de manière autonome à contrôler les feux de signalisation en temps réel grâce à l'application d'algorithmes avancés d'apprentissage profond par renforcement. La recherche se concentre sur la comparaison de l'efficacité de trois algorithmes distincts : l'Optimisation de Politique Proximale (PPO), l'Advantage Actor-Critic (A2C) et le Deep Q-Network (DQN) dans la gestion des feux de signalisation urbains. Les résultats montrent que l'algorithme PPO surpasse significativement les autres, en réduisant de manière substantielle les temps d'attente moyens et en augmentant notablement la vitesse moyenne des véhicules dans le réseau de circulation. Les conclusions soulignent le rôle crucial du contrôle dynamique des feux de signalisation pour atténuer la congestion et améliorer la fluidité de la mobilité urbaine. Le projet met également en avant le potentiel des techniques d'apprentissage automatique pour optimiser les systèmes urbains complexes, offrant ainsi une direction prometteuse pour les recherches futures et les applications pratiques dans les initiatives de villes intelligentes.