Reinforcement learning for ramp metring on highways

Heni WALHA, Yasmine DALY, Rabeb SDIRI, Anis BOUHAMED, Feryel DRIDI, Mariem Mazouz

A report submitted as part of the requirements for our Reinforcement Learning and Optimal Control Project
M2 SIA
at Gustave Eiffel Univeristy

.

December 2023

Supervisor Mr. Nadir FARHI

Abstract

The project aims to apply Q-learning and Deep Q-learning algorithms to learn ramp metering control on highways through numerical simulation. It utilizes the traffic simulator SUMO (Simulation of Urban Mobility) to simulate car-following and lane changing behaviors. The focus is on a highway stretch with a specified number of lanes and an entering ramp controlled by a traffic light. The Q-learning algorithm will control the traffic light at the ramp to optimize traffic flow on both the highway stretch and the ramp.

State variables in the project represent the traffic state on both the highway and the ramp. Action variables might include the proportion of the green light in a predefined traffic light cycle. The reward modeling is designed to reflect the optimization of traffic on both the highway and the ramp.

The project also considers varying scenarios by changing parameters such as the length of the highway stretch, speed limits, initial car density, car flow, and the number of lanes. This approach aims to cover a wide range of traffic conditions and optimize ramp metering effectively.

Acknowledgements

We express our deepest gratitude to Dr. Farhi Nadir for his invaluable guidance and profound expertise, which have been pivotal in shaping our ambitious project on reinforcement learning for ramp metering on highways. Dr. Nadir's comprehensive understanding and keen insights into this complex field have enlightened us, paving the way for groundbreaking work.

Additionally, our heartfelt appreciation extends to every member of our project team. Their unique skills, unwavering dedication, and collaborative spirit have created an environment that fostered innovation, creativity, and excellence. The synergistic effort of the team has been the cornerstone of our project, enabling us to achieve remarkable outcomes.

Our project stands as a testament to what can be accomplished when a group of dedicated and talented individuals unite towards a common goal. The collective effort, commitment, and passion of the team have been awe-inspiring. We thank each and every one of you for your invaluable contributions and support throughout this journey, which have been crucial in bringing our vision to life and making a meaningful impact in the field of reinforcement learning and traffic management.

Contents

List of Figures

List of Acronyms

- **SUMO:** Simulation of Urban Mobility
- **XML:** eXtensible Markup Language
- **DQN:** Deep Q-learning
- **API:** Application Programming Interface
- **TRACI:** Traffic Control Interface
- **AI:** Artificial Intelligence
- **DL:** Deep Learning
- **TF:** TensorFlow
- **M:** Movement (if this is a specific term used in your report)
- **O:** Open
- **S:** Stop
- **G:** Green
- **Y:** Yellow
- **R:** Red

Chapter 1

General Introduction

Welcome to our academic exploration in traffic management using reinforcement learning, guided by Dr. Nadir Farhi. This project delves into optimizing traffic flow on highways through ramp metering, employing Q-learning and Deep Q-learning algorithms. We focus on a stretch of highway with a designated number of lanes and an entrance ramp controlled by a traffic light. Our objective is to enhance the efficiency of this traffic light, optimizing the flow of vehicles on both the highway and the ramp.

To achieve this, we utilize SUMO (Simulation of Urban Mobility), a sophisticated traffic simulator that replicates real-world car-following and lane-changing behaviors. Our approach also draws inspiration from existing research in this domain, such as the work of Romain Ducrocq and projects on intelligent traffic signal control.

The state variables in our model will represent the traffic conditions on the highway and the ramp. The action will involve regulating the duration of the green light phase in the traffic light's cycle. The reward system is designed to reinforce optimal traffic flow, encouraging efficiency on both the highway and the ramp.

Chapter 2

Technologies Used

2.1 Introduction

This chapter delves into the various technologies that have been pivotal in the success of our project on ramp metering control using reinforcement learning. The core of our technological approach hinges on the seamless integration of Python, the SUMO simulator, the TRACI library, and the deep learning frameworks TensorFlow and Keras. These technologies collectively facilitated the simulation and optimization of traffic flow, playing a crucial role in advancing our project. Emphasis is placed on the strategic use of these tools to create a robust, efficient, and scalable solution for traffic management.

2.2 Technological Framework

The project's technological backbone is built upon the strategic integration of simulation tools, programming languages, and deep learning technologies.

2.2.1 SUMO: Advanced Traffic Simulation

SUMO (Simulation of Urban Mobility), an advanced traffic simulation tool, serves as a fundamental element of our project. It provides a dynamic and realistic environment for modeling traffic flow, essential for testing and refining our reinforcement learning algorithms.

2.2.2 Python and the TRACI Library

Python, renowned for its versatility and ease of use, is the primary programming language utilized in our project. The integration of the TRACI library within Python

enables real-time interaction with SUMO, allowing for the effective implementation of our control strategies.

2.2.3 Deep Learning with TensorFlow and Keras

To harness the capabilities of Deep Q-learning algorithms, our project leverages TensorFlow and Keras. TensorFlow provides a scalable and robust platform for machine learning models, while Keras simplifies the construction and training of neural networks, essential for optimizing our traffic management solutions.

2.3 Conclusion

This chapter has highlighted the integral technologies that underpin our project, emphasizing the significant role of Python, SUMO, TRACI, TensorFlow, and Keras. These technologies have synergized to form a comprehensive framework, enabling advanced research and development in the field of traffic management using reinforcement learning. Their integration represents the technological foundation upon which the subsequent chapters build, exploring the intricacies and outcomes of our project in detail.

Chapter 3

Implementation

3.1 Introduction

In the initial phase of our project, we utilized Netedit, a graphical tool within the SUMO framework, to design a virtual highway network. The network is defined by an XML (.net.xml file) configuration that outlines various elements crucial for traffic simulations. Here are the key components of the network:

3.2 Network Building

3.2.1 Internal Edges

Internal edges represent lanes within the network. Each edge consists of lanes with specific speed limits, lengths, and shapes, contributing to the overall structure. The internal edges include junctions such as J1, J6, J7, and J8. These junctions house a total of six internal edges, each consisting of multiple lanes with distinct speed limits and lengths.

3.2.2 External Edges

External edges connect junctions, defining entry and exit points of the network. Priority and lane details are specified for seamless traffic flow. Our project featured the E0, E3, E4, E5, and E6 external edges, each with specific priority settings and lane configurations.

Figure 3.1: Junction and Lane Definitions

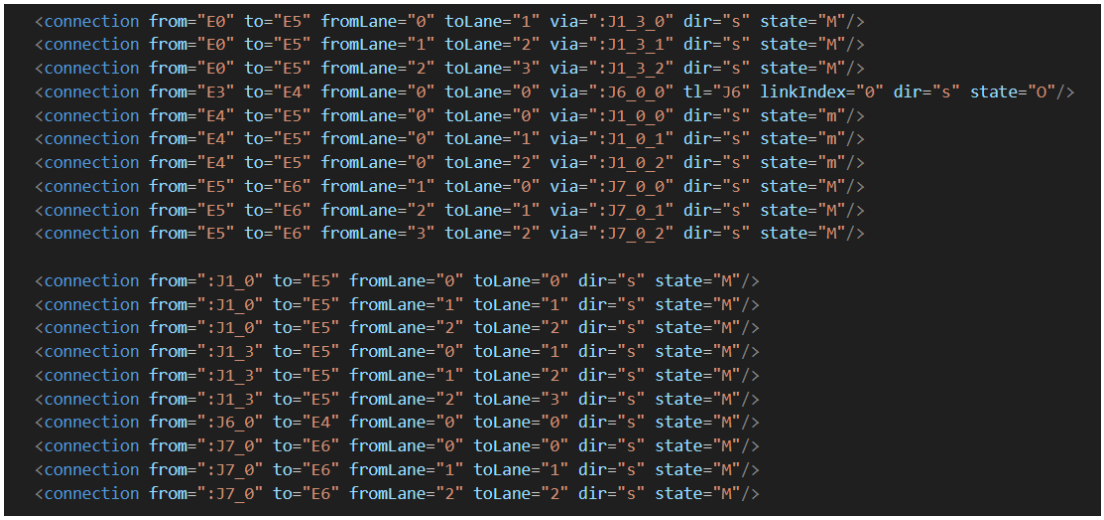

Figure 3.2: Connection Definitions

3.2.3 Traffic Light Logic

Traffic light control is implemented at specific junctions. Phases with durations and corresponding states are defined, influencing the flow of traffic through the network.

3.2.4 Junction and Lane Definitions

Junctions are strategically placed with priority or dead-end types. Lanes are associated with specific edges, forming the infrastructure for vehicle movement.

3.2.5 Connection Definitions

Connections establish links between edges, lanes, and junctions, defining permissible traffic flow and influencing the simulation dynamics.

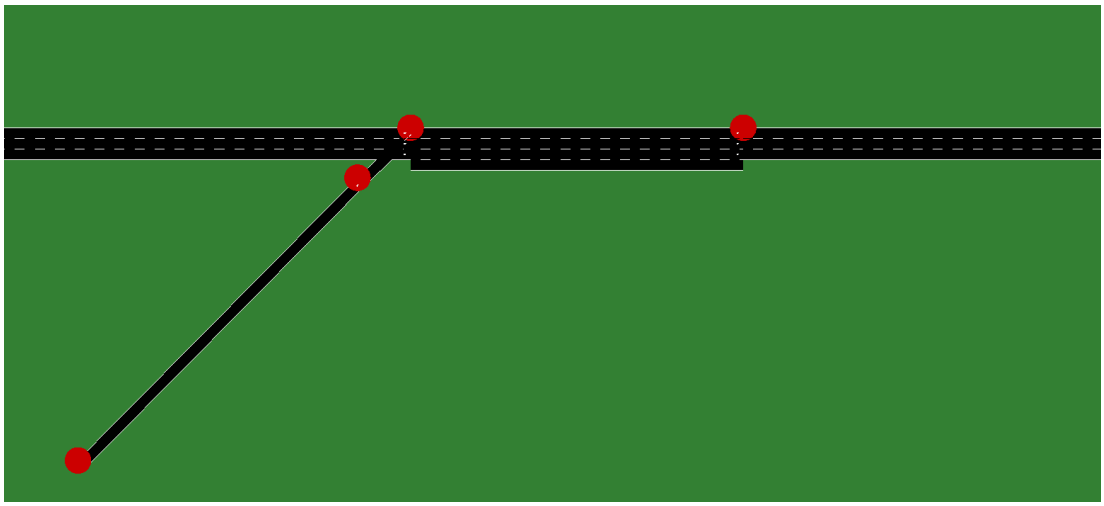

Figure 3.3: Location Configuration

3.2.6 Location Configuration

The network spans a defined area, and the XML configuration includes details such as offset, boundaries, and projection parameters.

3.3 Q-Learning

3.3.1 Introduction to Q-Learning

Q-learning is a model-free reinforcement learning algorithm used to find the optimal action-selection policy for any given finite Markov decision process. It operates by learning the value of an action in a particular state and seeks to maximize these values over time, leading to the best decision for each situation.

3.3.2 Algorithm Basics

The algorithm uses a Q-table, a matrix-like structure, where rows represent states of the environment, and columns represent possible actions. The Q-value for a state-action pair is updated as:
Q(s, a) $\leftarrow Q(s,a) + \alpha[R(s,a) + \gamma \max Q(s',a') - Q(s,a)]$

3.3.3 State-Action-Reward Structure in SUMO

In the context of SUMO traffic simulations, the state might include various parameters like vehicle count, waiting time, and traffic density. The actions could be changing traffic light phases. Rewards are given based on the efficiency of traffic flow – lower

wait times and smoother traffic might yield higher rewards.

3.3.4 Project Architecture :

QLearningAgent Class :

Attributes:

The Q-learning agent is initialized with the following parameters:

- `state_size`: The size of the state space.
- `action_size`: The size of the action space.
- `learning_rate`: The learning rate for the Q-learning algorithm.
- `discount_factor`: The discount factor for future rewards.
- `exploration_rate`: Initial exploration rate.
- `exploration_decay`: Decay rate for the exploration.

Methods

The Q-learning agent includes the following methods:

__init__(self, state_size, action_size, learning_rate=0.1, discount_factor=0.95, exploration_rate=1.0, exploration_decay=0.995)

Initializes the Q-learning agent with the specified parameters and initializes the Q-table with zeros.

choose_action(self, state) -¿ action

Chooses an action based on the current state. With probability `exploration_rate`, chooses a random action. Otherwise, exploits the learned Q-values.

learn(self, state, action, reward, next_state, done)

Updates the Q-table using the Q-learning algorithm. Calculates the Q-value update and decays the exploration rate.

save(self, filename)

Saves the trained Q-table to a file.

load(self, filename)

Loads a trained Q-table from a file.

SumoEnvironment Class :

Attributes

The `SumoEnvironment` class is initialized with the following attributes:

- `sumo_cfg_file`: Path to the SUMO configuration file.
- `highway_lanes`: List of lane IDs for the highway lanes.
- `ramp_lane`: Lane ID for the ramp lane.
- `traffic_light_id`: ID of the traffic light controlling the ramp.

Methods

The `SumoEnvironment` class includes the following methods:

__init__(self, sumo_cfg_file)

Initializes the `SumoEnvironment` with the path to the SUMO configuration file and sets up necessary variables.

_setup_sumo_home(self)

Checks and sets the SumoHOME environment variable.

define_state_size(self) and define_action_size(self)

Defines the size of the state and action spaces for the reinforcement learning problem.

reset(self)

Resets the SUMO simulation and returns the initial state.

step(self, action)

Advances the simulation by applying an action and returns the next state, reward, and done flag.

get_state(self)

Retrieves real-time data about the lanes and constructs the current state.

calculate_reward(self)

Calculates the reward based on the waiting time of vehicles.

apply_action(self, action)

Controls the traffic light based on the given action.

close(self)

Closes the SUMO simulation.

Explanation

The `SumoEnvironment` class abstracts the interactions with the SUMO simulator. It is designed for integration with reinforcement learning algorithms, providing methods for simulation control and interaction.

DQNAgent Class :

This class represents the Deep Q-Network (DQN) agent, which is a key component in reinforcement learning, particularly in environments with high-dimensional state spaces.

Attributes

The `DQNAgent` class is initialized with the following attributes:

- `state_size`: The size of the state space.
- `action_size`: The size of the action space.
- `learning_rate`: The learning rate for the agent, set to 0.001.
- `discount_factor`: The discount factor (gamma) for the Q-learning algorithm, set to 0.95.
- `exploration_rate`: The initial exploration rate (epsilon) for the epsilon-greedy policy, set to 1.0.
- `exploration_decay`: The decay rate for the exploration rate, set to 0.995.
- `memory_size`: The size of the memory buffer (replay buffer), set to 2000.

Methods

The `DQNAgent` class includes the following methods:

__init__

The constructor initializes the `DQNAgent` with the specified attributes and sets up the neural network model and memory buffer.

build_model

Builds the neural network model for approximating the Q-function.

remember

Stores experiences in the replay buffer.

act

Chooses an action based on the current state using an epsilon-greedy policy.

train

Trains the neural network model using experiences sampled from the memory buffer.

update_target_model

Updates the target network, a common practice in DQN to stabilize learning.

load

Loads a pre-trained model.

save

Saves the trained model.

General Overview

The `DQNAgent` class is a typical implementation of a Deep Q-Network agent. It uses a neural network to approximate the Q-function and employs experience replay for efficient learning. The specific details like the architecture of the neural network and the mechanics of the experience replay would require a closer look at the actual implementation.

Main class :

The `main.py` file is a Python script designed to integrate a Deep Q-Network (DQN) agent, as defined in `dqn_agent.py`, with a SUMO (Simulation of Urban Mobility) environment. This analysis outlines the key components and functionalities of the script.

Imports and Setup

The script includes the following key imports and setup:

- Standard modules like `os`, `sys`.
- The `traci` module for interfacing with SUMO.
- Custom modules: `SumoEnvironment` and `DQNAgent`.

Environment Variable Check

The script verifies if the `SUMO_HOME` environment variable is set, which is crucial for running SUMO simulations.

Initialization of Environment and Agent

3.3.5 Environment (`env`)

An instance of `SumoEnvironment` is created, which encapsulates the simulation setup and state.

Agent (`agent`)

An instance of `DQNAgent` is initialized, with its state and action sizes derived from the `env` object.

Additional Variables

- `average_wait_time`: Initialized to 0, likely tracking a performance metric.

Main Functions and Logic

- A function to run the simulation, including interactions with the environment and updates to the agent.
- A training loop for iterative learning.
- Logic for calculating and displaying performance metrics.
- Functions for saving and loading the trained model.

Integration with SUMO

The script integrates with SUMO, a robust traffic simulation package, enabling the DQN agent to operate in a realistic traffic environment.

Purpose and Application

The primary purpose of the script is to establish a reinforcement learning setup for the DQN agent in traffic simulations. This can be applied in areas like traffic management and autonomous vehicle behavior studies.

3.4 Deployment :

3.4.1 Traffic Light Control Logic

The first screenshot (Figure 3.4) shows the implementation of the `apply_action` function within the Q-learning agent. This function controls the traffic light by sending commands through the SUMO interface, using the `traci` library.

```
def apply_action(self, action):
        # Control the traffic light based on the action
        if action == 1:  # Assuming 1 is green and 0 is red
            traci.trafficlight.setPhase(self.traffic_light_id, 0)  # Green phase
        else:
            traci.trafficlight.setPhase(self.traffic_light_id, 2)  # Red phase
```

Figure 3.4: The apply_action function determining traffic light phases.

The code snippet demonstrates how actions taken by the agent (0 for red, 1 for green) correspond to different traffic light phases in the simulation. This direct interaction is a critical part of the reinforcement learning loop, allowing the agent to affect the environment.

3.4.2 Traffic Light Phase Configuration

The second screenshot (Figure 3.5) displays the XML configuration for a traffic light in SUMO. The phases indicate the duration and state of the traffic light, with 'g', 'y', and 'r' representing green, yellow, and red, respectively.

```
<tlLogic id="J6" type="static" programID="0" offset="0">
    <phase duration="82" state="G"/>
    <phase duration="3"  state="y"/>
    <phase duration="5"  state="r"/>
</tlLogic>
```

Figure 3.5: SUMO traffic light phase configuration in XML format.

This configuration is essential for setting up the simulation environment and ensures that the agent's decisions correspond to meaningful changes in the simulation.

3.4.3 Reinforcement Learning Loop Output

In the third screenshot (Figure 3.6), we observe the output of the reinforcement learning loop. Each line in the output represents a state-action-reward sequence, showcasing the agent's decision-making process over discrete time steps.

```
State: [ 2.         14.23263449  0.         13.89        0.         13.89
  6.          0.21264744  1.          0.          0.         13.89
  2.         13.0067084   1.         14.7615263   0.         13.89
  4.         13.18826791  0.         13.89        0.         13.89
  6.          0.21264744], Action: 0, Reward: -37.0
State: [ 1.         15.41749165  0.         13.89        0.         13.89
  6.          0.02452329  1.          0.          0.         13.89
  3.         12.75505676  0.         13.89        0.         13.89
  4.         12.87982521  0.         13.89        0.         13.89
  6.          0.02452329], Action: 1, Reward: -44.0
State: [ 0.         13.89        0.         13.89        0.         13.89
  6.          0.02338284  0.         13.89        0.         13.89
  3.         14.36146887  0.         13.89        0.         13.89
  4.         12.08757902  1.         14.91419463  0.         13.89
  6.          0.02338284], Action: 1, Reward: -35.0
State: [ 0.         13.89        0.         13.89        0.         13.89
  5.          1.11316754  0.         13.89        0.         13.89
  3.          9.26176173  0.         13.89        0.         13.89
  4.         12.19279204  1.         15.01025536  0.         13.89
  5.          1.11316754], Action: 0, Reward: -9.0
State: [ 0.         13.89        0.         13.89        0.         13.89
  5.          1.28179579  0.         13.89        0.         13.89
  2.         13.54548036  0.         13.89        0.         13.89
  6.         12.69383149  1.         15.07095384  0.         13.89
  5.          1.28179579], Action: 1, Reward: -0.0
```

Figure 3.6: Output from the reinforcement learning loop.

The displayed states, actions taken, and rewards received are crucial for debugging and understanding the agent's learning progression.

3.4.4 Simulation Environment

The final screenshot (Figure 3.7) illustrates the SUMO simulation environment. It highlights the intersection where the agent controls the traffic lights, impacting the traffic flow.

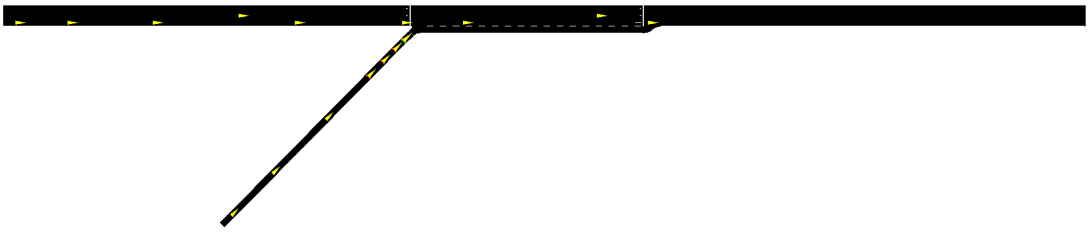

Figure 3.7: The SUMO simulation environment with the controlled intersection.

This visual representation is vital for verifying the correct implementation of the simulation and provides a visual context for the agent's actions.

3.4.5 Discussion

Throughout the Q-learning deployment, these screenshots represent key components of the system. From the direct control of traffic lights to the configuration of the simulation environment, each element plays a role in creating a comprehensive learning experience for the DQN agent. The loop output is particularly valuable for monitoring the agent's learning process and tuning the parameters for optimal performance.

3.5 Conclusion

This XML configuration serves as the foundation for our subsequent simulations, allowing us to explore traffic scenarios, assess system performance, and optimize transportation logistics within the virtual environment. The meticulous design and parameterization of the network contribute to the accuracy and realism of our simulations, providing valuable insights into traffic behavior and system dynamics.

Chapter 4

Challenges Faced

4.1 Introduction

In this chapter, we delve into the various challenges encountered during the execution of our project on ramp metering control using reinforcement learning. The journey of implementing a complex system like this is fraught with obstacles, ranging from the intricacies of the algorithms used to the practicalities of their application in real-world scenarios. These challenges not only tested our technical acumen but also pushed us to innovate and think critically. We discuss each of these hurdles in detail, shedding light on how we navigated these complexities to advance our project.

4.2 Challenges

Technical Complexity of Reinforcement Learning

Implementing Q-learning and Deep Q-learning algorithms presented a significant challenge due to their intrinsic complexity. The process of fine-tuning these algorithms to suit the specifics of highway traffic and ramp metering required a deep understanding of both traffic dynamics and reinforcement learning principles.

Simulation Accuracy with SUMO

While SUMO is a powerful tool for simulating traffic scenarios, ensuring the accuracy and realism of these simulations was a challenge. Replicating real-world traffic patterns, vehicle behaviors, and various traffic conditions in a virtual environment demanded meticulous calibration and testing.

Integration of Different Technologies

Seamlessly integrating SUMO, Python, the TRACI library, TensorFlow, and Keras posed significant integration challenges. Each technology has its own set of complexities, and ensuring they worked harmoniously to simulate, control, and learn from the traffic environment was a considerable task.

Real-Time Decision Making

Developing a system capable of making real-time decisions in a dynamic and constantly changing traffic environment was a complex challenge. The algorithms had to be robust and adaptive to ensure effective ramp metering control in varying traffic conditions.

Balancing Theoretical Concepts and Practical Application

Bridging the gap between theoretical reinforcement learning concepts and their practical application in traffic management required innovative thinking and practical problem-solving skills. Translating complex theoretical ideas into workable solutions that could be implemented in a real-world scenario was a demanding task.

Ensuring Safety and Reliability

Given the real-world implications of ramp metering control, ensuring the safety and reliability of the proposed solutions was paramount. This involved rigorous testing and validation to ensure that the control strategies would not lead to unintended consequences in real traffic scenarios.

Conclusion

The journey through the challenges faced in this project has been a testament to the resilience and adaptability required in the field of intelligent transportation systems. Each challenge, from the technical complexity of reinforcement learning algorithms to the rigorous demands of ensuring safety and reliability, required a careful balance of theoretical understanding and practical application. Overcoming these obstacles has not only enhanced our expertise but also contributed significantly to the robustness and efficacy of our traffic management solutions. This experience reinforces the notion that the path to innovation is often paved with challenges, each providing valuable lessons and opportunities for growth.

Chapter 5

Conclusion

The journey of developing an optimized traffic management system using reinforcement learning for ramp metering on highways marks a significant advancement in the field of intelligent transportation systems. This report has illuminated the complexities involved in such an integration and underscored the effectiveness of combining simulation, advanced algorithms, and deep learning to address real-world traffic challenges.

5.1 Achievements and Insights

- **Effective Traffic Optimization:** The successful application of Q-learning and Deep Q-learning algorithms in ramp metering has demonstrated their effectiveness in optimizing traffic flow. This achievement highlights the potential of reinforcement learning in managing and improving highway traffic conditions.
- **Advanced Simulation with SUMO:** Utilizing SUMO for simulating traffic scenarios proved invaluable, offering a realistic and dynamic platform to test and refine our algorithms. This tool's versatility in modeling various traffic conditions was crucial in developing and validating our approach.
- **Technological Synergy:** The integration of Python, SUMO, TRACI, TensorFlow, and Keras created a powerful technological synergy. This combination facilitated a comprehensive approach to traffic management, from simulating complex traffic scenarios to implementing and refining sophisticated control algorithms.
- **Continuous Learning and Improvement:** The project's iterative process of development, testing, and refinement was key in adapting our strategies to the dynamic nature of traffic flow. These efforts have significantly contributed to the accuracy and efficiency of our traffic management system.

5.2 Concluding Thoughts

This exploration into the use of reinforcement learning for ramp metering, leveraging the strengths of simulation tools and deep learning frameworks, highlights the vast potential of these technologies in transforming traffic management. The insights and methodologies developed through this project lay a solid foundation for future innovations in intelligent transportation systems, promising to enhance both the efficiency and safety of highway traffic management.