Arab American University

Faculty of Graduate Studies

Using Deep Reinforcement Learning Model to Design Sustainable Bicycle Mobility Infrastructure

By
Rabee Adel Al-Qasem

Supervisor
Prof. Giovanni Stilo

This Thesis was Submitted in Partial Fulfillments of the Requirements for the Master's Degree in Data Science and Business Analytics

January / 2022

Using Deep Reinforcement Learning Model to Design Sustainable Bicycle Mobility Infrastructure

By

Rabee Adel Al-Qasem

This thesis was defended successfully on 29/ 1 /2022 and approved by:

Committee Members	**Signature**
1. Dr. Prof. Giovanni Stilo (Supervisor)	....
2. Majdi Owda (Internal Examiner)	....
3. Prof. Rushdi Hamamreh (External Examiner)	....

Declaration

I, **Rabee Adel Al-Qasem**, one of the students of the Faculty of Graduate Studies at the Arab American University hereby declare that this thesis entitled "**Using Deep Reinforcement Learning Model to Design Sustainable Bicycle Mobility Infrastructure",** is all by my own work and the resources that are used in this thesis (including the internet resources) have been referred to and properly acknowledged as required.

I declare that I have fully understood the concept of plagiarism and I acknowledge that my thesis will be immediately rejected in case of including any type of plagiarism.

Rabee Adel Al-Qasem

Signature: Rabee Al-Qasem

Date: 29 /1 /2022

Acknowledgments

First and foremost, I'd like to express my gratitude to my supervisor, Dr. Giovanni Stilo[1], for allowing me to work with him on my thesis. Without his help, this project would not have been possible. He was always willing to give me advice on my work and recommendations when I was stuck on something. He was always open to meeting with me on short notice if I needed assistance. Next, I would like to thank both my parents, my dear sister Lubna and my two brothers Imad and Ahmed, for their support and love throughout my life. And finally, I would like to thank my university for opening the Data science major, which was a dream that came true to be one of the first students who enroll in this major.

[1] Dr. Stilo is partially supported by Territori Aperti a project funded by Fondo Territori Lavoro e Conoscenza CGIL CISL UIL.

Abstract

Using Deep Reinforcement Learning Model to Design Sustainable Bicycle Mobility Infrastructure

Nowadays, the whole world is concerned with the increasing environmental issues, and many countries are working towards reducing the impact of humans on the environment by adopting various sustainable development strategies. One of the promoted actions to face this issue is encouraging the use of bicycles as the primary mean of transportation. If cycling becomes the primary mean of transportation, there will be a need for new and suitable routes and paths that suit the needs of the bicycles' riders. In this thesis, we will tackle the problems and propose solutions to the issues that cyclists may face concerning the city's topography (e.g., types of road, road surface, and their slope).

This thesis proposes a solution that promotes using an AI agent that utilizes reinforcement learning and neural network to find the best path in a way that is customized by user preferences. We first presented the data collection process and how these data will be used in a readily available way by the agent. Then, we tested several reinforcement learning algorithms to find the most suit- able method to be used in our challenging scenario. We have also converted the map into a graph which represents the deep reinforcement learning environment, and converted each feature into a sub-reward in our complex reward system. Finally, we trained multiple reinforcement learning models.

The results show that Dual Deep Q Network has the best outcome; we achieved 7500 cumulative rewards in less than 5 hours of training time, and our agent was able to design a route based on the end-user specification and overpass all the roads that do not meet the criteria.

Table of Contents

List of Tables

List of Figures

Chapter One

Introduction

Chapter One

Introduction

The core of the thesis is to explore the possible solutions to create a support system for Mobility managers which allow designing a sustainable mobility infrastructure for bike lanes. The proposed solution uses an Artificial Intelligent approach called reinforcement learning to consider the many conditions from a real-world environment. The Agent learns to achieve a goal in an uncertain environment. The shortest path algorithm, also known as Dijkstra's algorithm, allows finding the fastest path among two places; it has been used successfully in many mobility apps. Many research works on adding several variables to the considered model, for example, taking into consideration the traffic and the road's vibrations, to compute the most comfortable route. Our research will use the virtual Agent to determine the most convenient path between two locations. The route generated by the Agent will take into consideration a lot of variables like: the characteristics of road networks, geographical features of the territory (e.g., altitudes, presence of shade), information of all kinds of traffics, the population of an area, the characteristics of the used vehicle (e.g., battery-powered vehicles), the energy expenditure of the driver, and other information that we are going to collect. Our thesis will answer the following questions:

(1) Are there publicly available sources of information that contains rich geographical data?

(2) The publicly available data are sufficient for training and testing a virtual agent able to compute personalized smart paths?

(3) Which can be the best mechanism to integrate all the aforementioned geographical information?

(4) Which is a possible reward modulization which encompasses all the collected rich geographical information?

(5) Which is the best Neural Network architecture that must be employed by our Agent is the best deep reinforcement learning algorithm that can be used in this specific application scenario?

The above questions have driven us during this of this project.

The shortest path approach focuses only on one variable, for example, traffic. Using reinforcement learning in our project will help us find a new way to choose the best path between two locations based on multiple variables that can interest the cyclist. The main goal of Reinforcement Learning (RL) is to give the Agent the wisdom and the knowledge to react like a human being in an unknown environment by obtaining the optimal strategy by maximizing long-term compensation by providing the current state reward (Liu & Chen,2019). Moreover, few researchers used RL for route planning in general and not only in Bike Lane planning. Most of the research conducted on this subject mainly uses the shortest path algorithm. The research enhances it by using one or two prediction models considering very few factors; the drivers' or the cyclists' faces on the road. On the other hand, our project focuses on adding as many variables as possible and hopes that the Deep Reinforcement Learning agent can handle it.

Lastly, this thesis will help the municipalities implement the routes (or their variations) that are best suitable to promote hybrid mobility infrastructure. Also, this study will help the researchers take a new approach in route planning for different sectors (disabled people, water supply network for agriculture areas, pipeline transport for oil and gas, etc.) and consider many cities' variables.

1.1 Problem Definition

Nowadays, due to the pandemic diffusion and the raising interest of many countries to reduce the pollution that is caused by gas emissions from cars, countries now encourage their citizens to use bicycles for mobility, which raises the interest and need for cycling routes (bike lanes). This adoption allows mitigating the pollution and exposure to the virus. One problem for the public administration officers related to this rising interest in designing the new bike lanes fast and thoughtfully is that the mobility manager (officer) cannot simply design the bike lane considering the shortest route, but several other factors must be evaluated. For example, cyclists may prefer to use a route with shadows where the traffic jam is low. Sometimes the cyclist uses an e-bike, thus considering its power and battery capacity can be an additional factor to consider at the designing phase. From the mathematical standpoint, the number of considered variables and the problem's size can be in- tractable to be tackled with classic optimization techniques. For this reason, we propose to build an artificial intelligence agent based on the city's data. The virtual cyclist (the agent) learns to balance all the variables and draws the city's bike routes. To solve the problem mentioned above, we need to:

- Understand which are the involved variables
- Obtain the required data
- Model the problem in terms of a Reinforcement Learning based Agent
- Test the proposed solution on actual data

1.2 Research Goal

The thesis investigates the advantages of applying DRL algorithms to high-dimensional systems and complex reward systems. Therefore, the following research questions are addressed:

- Is there a suitable source to extract geographical data with many features, especially for roads, that we can use in our reward system?
- Can the Neural Network handle the complex state representation with the many features used in the reward system and find an optimal policy and best action in each state?

1.3 Contribution

This thesis will help researchers take a new route planning approach for different sectors, like; located disabled people facilities, water supply network for agriculture areas, pipeline transport for oil and gas in consideration of a huge number of variables by using the deep reinforcement learning method, which is customized based on each sector's needs. Another contribution is defining how we can collect different GIS data sources and combine them into one data-set that can be used in many machine learning applications.

1.4 Motivation

One of the primary motivations for working on this thesis is to provide intelligent decision support on planning of roads in all global regions. Road planning does not only depend on human decisions, but also on the support of Artificial intelligence that aids in planning for better routes that satisfy all individuals. Another personal motivation is to help the Palestinian government to plan for safer roads to avoid any danger that the citizens might face due to political reasons.

1.5 Thesis Outline

The remaining sections of the thesis are organized as follows:

- Chapter 2 provides a formal introduction to Neural Networks by explaining the mathematical preliminaries. Also, it discusses Reinforcement Learning, Deep Reinforcement Learning components, model-free methods, and policy-based methods.
- Chapter 3 provides some previous studies which implemented DRL in their projects. Also, it introduces widely used algorithms and discusses the code for each one of them.
- Chapter 4 discusses the process starting from searching for the best GIS data source to collecting the data and finally integrating it into one data set.
- Chapter 5 discusses the environment of our agent, state representation and the rewards system.
- Chapter 6 is the core of the thesis which presents and discusses the modeling and implementation of our code.
- Chapter 7 provides an overview and conclusion of the work done in the thesis, as well as future research directions.

1.6 Conclusion

In this chapter, we discussed the main idea of the thesis and how we aim to explore the possible options to create a support system for mobility managers. Moreover, the proposed solution uses an Artificial Intelligent approach by using reinforcement learning methods to generate routes based on the end-user's preferences. The route generated by the agent will consider many variables, like; the characteristics of road networks, the geographical features, and traffic patterns. We also discussed how this project will help the city municipalities to plan bicycle routes which need for them is increasing due to the massive increase of citizens using bicycles as the primary means of transport. We also addressed the research goal, and we aim to investigate the advantages of applying DRL algorithms on the high-dimensional environment and complex reward systems. Finally, we explained how the thesis contributes in helping researchers take a new route planning approach for different sectors, like; the located disabled people facilities, water supply network for agriculture areas, and pipeline transport for oil and gas. Another contribution is defining how we can collect different GIS data sources and combine them into one data set.

Chapter Two

Background

Chapter Two

Background

2.1 Deep Neural Networks

2.1.1 Introduction

Deep Neural Networks (DNN) is now at the root of some of the most significant advances in AI and machine learning in recent years. They're at the heart of some of the most innovative technologies, including self-driving cars, image recognition systems, speech recognition systems, and self-driving robotics. They have reached state-of-the-art results in these tasks.

DNN is a general framework for estimating non-linear functions based on training data. Real value, discrete-valued, and vector-valued functions are all possible. DNN's success is due to its capacity to learn from data and extract key features without manual work.

This section will provide a quick overview of neural network design, training techniques, and regularization.

2.1.2 Building Units

Artificial neural networks are based on the workings of the human brain. The human brain is a complex web of interconnected neurons that can extract critical information from inputs and produce an output signal (signal). On the other hand, an artificial neural network is made up of a series of interconnected neurons organized in layers, each of which takes real-valued inputs and outputs real-valued results (Rashid,2016).

Artificial Neuron

The following is how the neural network works: given a vector of inputs *x*, the neuron calculates the weighted sum of the inputs, with the weights denoted by *w* (Rashid,2016). Then, the weighted sum is added to a bias term, *b*, and then passed through an activation function, f and finally it calculates the output based on the following equation1.

$$y_i = f\left(\sum_j x_j\, w_{ij} + b_i\right) \tag{1}$$

Activation Functions

The activation function provides non-linearity in a neuron's output. This helps the network's learning of non-linear representations from training data. Some activation functions, such as Sigmoid, Hyperbolic Tangent, and a Rectified linear unit, have been proposed in the literature. The following are the most widely used activation functions:

- Sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

- Hyperbolic Tangent:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

- Rectified Linear Unit (ReLU):

$$\boldsymbol{f(x) = MAX(0, x)} \tag{4}$$

In modern neural networks, the default recommendation is to use the rectified linear unit or ReLU due to its performance (Goodfellow, Bengio, & Courville,2016).

2.1.3 Feed Forward Networks

The Feed-Forward neural network is the most popular DNN architecture (Rashid,2016). The neurons are arranged in layers in this framework. This architecture has three types of layers: an input layer, a few hidden layers, and an output layer. Information flows from the input layer to the hidden levels and then to the output layer to calculate the output. this process is illustrated in the following graph 2.1.

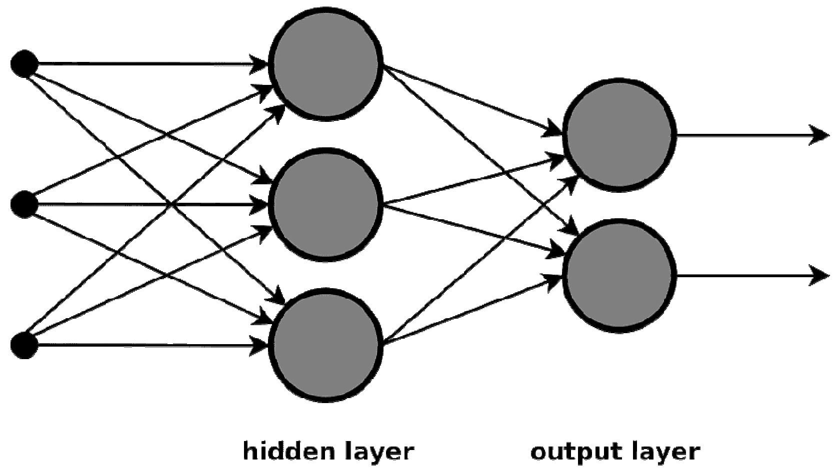


Figure 2.1: Feed Forward Network

Training The Network

Backpropagation is a technique used to train the weights of the neural network. The idea behind this method is to start with a random weight initialization and then calculate the output for a particular input. Using a gradient descent approach, the difference between the generated and actual output is used to update the network's weights. The technique is called backpropagation because the output layer weights are updated first and then back propagated into the network.

In recent years, stochastic gradient descent has become a popular method for training neural network weights. A few better stochastic gradient descent variations, such as ADAM (Kingma & Ba,2015), have also been proposed. These approaches have an advantage over traditional gradient descent in that they can adjust learning rates dependent on the distribution of the training data. This eliminates the need for precise learning rate selection, allowing the algorithm to converge faster.

Regularization

Regularization is a machine learning technique that helps to avoid the problem of over-fitting statistical models. The purpose of regularization is to increase the network's generalizability to unseen data. L1 or L2 regularization is the traditional way at tackling the problem of over-fitting. The primary idea behind this strategy is to incorporate a penalty element in the cost function for the weights.

2.1.4 Loss

Loss functions are related to model accuracy and are a key component of AI/ML governance. Loss functions are a way to assess how well the models perform on the data. Your loss function will show a higher value if model predictions are completely wrong. It will give a lower number if they're pretty decent. In a way, the loss function will inform you if the model is making progress after you tweak parts of your algorithm to enhance your model. Multiple famous loss functions are used in DRL, like mean square error and Gaussian loss, but (Co-Reyes et al.,2021) suggested two new loss DQNClipped and DQNReg which we will implement and use later in 6.

DQNClipped equation.

$$\mathrm{L}_{\text{DQNClipped}} = max[Q(s_t, a_t), \delta^2 + Y_t] + max\left[Q(s_t, a_t) - Y_t, \gamma \left(\underset{a}{max} Q_{targ}(s_t, a)\right)^2\right] \quad (5)$$

- Where $Y_t = r_t + \gamma * \underset{a}{max} Q_{\text{targ}}(s_t, a)$
- $\delta = Q(s_t, a_t) - Y_t$

DQNreg equation.

$$L_{\text{DQNReg}} = 0.1 * Q(s_t, a_t) + \delta^2 \quad (6)$$

2.2 Deep Reinforcement Learning

2.2.1 Introduction

Reinforcement Learning (RL) is an area of machine learning (ML) that deals with sequential decision making by using an Artificial intelligence (AI) agent that interacts with an environment to get rewards which indicate how good the action that the agent took in the environment is (Sutton & Barto,2018a). The primary purpose of RL is to learn how to map each state in the environment to action and maximize the expected sum of rewards, which is defined as Policy. In contrast with other ML supervised and unsupervised techniques, the AI agent is not told the right action to take in a given state. Instead, the agent explores the environment to achieve the optimal Policy. The path to success in Reinforcement Learning isn't as straightforward: the algorithms contain a lot of moving components that are difficult to debug, and they take a lot of adjusting to produce decent results.

In this chapter, we will discuss the RL problem, the components of RL, such as; the Environment, Policy, value function, the bellman equation and finally Markov Decision Process (MDP)

2.2.2 Environment

An environment is a virtual world that the agent interacts with to learn. It could be a hole 3D space or 2D space, like the Chessboard. It can represent the whole world through maps or represent a board game, like Tic Tac Toe or chess or even an Atari games. An environment mathematically consists of a set of States $\boldsymbol{S_t}$that represent the state space of the environment $\boldsymbol{D}$ 7.

$$S_t \subset D \tag{7}$$

Where S_t represents a State S in time step t which is a subset of State-space D.

2.2.3 Return

As we mentioned before, the main purpose of the RL framework is to make the agent learn how to maximize its future rewards by taking the sequences of action that lead to positive rewards and avoiding the penalties to maximize its return which is formulated according to the following equation 8:

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_{N-1} \tag{8}$$

Where R_t represents the return in a particular time step t and N represents the end of the episode. For stochastic environment, we can rewrite the equation as the following 9:

$$\sum_{\boldsymbol{k=0}}^{\infty} \boldsymbol{r_{t+k}} \tag{9}$$

2.2.4 Discount Factor

The discount factor γ determines how much the agent cares about the rewards in the near and late future. We ca n consider the discount factor as a learning parameter, its value varies from 0 ≤γ ≤1 (Sutton & Barto, 2018a). If γ is close to zero, the agent will consider the closest rewards for the present state. On the other hand, if the γ is close to one, it will be considered as a future reward and all future rewards will be equally important and the return equation will look like this:

$$\sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{10}$$

2.2.5 Policy

A policy denoted by π is a map determining what action should be taken in a specific state (Sutton & Barto,2018a). The policy that collects the most significant amount of rewards for an environment is called an optimal policy, and is denoted by π^*. There are two types of policies: deterministic and stochastic. The deterministic policy is when the agent execution of action a_t is guaranteed, and it can be formulated as the following equation:

$$\pi(s_t) = a_t, s_t \subset D a_t \subset A \tag{11}$$

On the other hand, in the stochastic policy, the action a_t is considered as a certain probability, and it can be formulated as follows:

$$\pi(a_t \mid s_t) = P_i \quad , \quad s_t \subset D \quad , \quad a_t \subset A \quad , \qquad 0 \le P_i \le 1 \tag{12}$$

2.2.6 Value Function

Value function represents how good it is for the agent to act in a given state. The value function depends on the policy where the agent picks the action to perform in a given state (Sutton & Barto, 2018a). Therefore, value functions are represented as follows:

$$V: V^{\pi} \rightarrow R, V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = s\right\} \tag{13}$$

where *E{}* denotes the expected value the agent follows π , and is any *t*.

Action value function, also known as Q-function, is defined as the sum or rewards expected to occur while taking action a in state s, by policy, and is formulated as follows:

$$Q^{\pi}(s, a) = E_{\pi}\{R_t \mid s_t = sa_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s_t = sa_t = a\right\} \tag{14}$$

2.2.7 Advantage Function

The Advantage function, denoted as $A(s, a)$, measures how good or bad a particular action is given a particular state. In other words: *what is the advantage of choosing a specific action from a specific condition?* The Advantage is mathematically defined as:

$$A(s, a) = E[r(s|a) - r(s)] \tag{15}$$

- where $r(s|a)$ is the reward of a state given a certain action
- $r(s)$ the reward of the current state

and also, it can be viewed as the following:

$$A(s,a) = Q(s,a) - V(s) \tag{16}$$

2.2.8 Bellman Equations

The Bellman equations express maximizing the expected return in terms of the value function. It helps in policy comprising. Bellman The Policy π is considered better than π' if the expected return π is more significant than π' for all $s \in S$ (Divyam,2017). Therefore, the optimal value function denoted by $V^{\star}(s)$ which is known as the Bellman optimality equation can be mathematically formulated as:

$$V^{\star}(s) = \underset{a}{MAX} \sum_{s'} \{p(s' \mid s,a)[R(s,a,s') + \gamma V_{\star}(s')]\} \tag{17}$$

2.2.9 Markov Decision Process

Markov Decision Process (MDP) is a mathematical framework used in decision-making that helps the agent in making decisions (Sutton & Barto, 2018a). MDP is 4-tuple (S, A, Pa, Ra) where:

- S is a set of states called in the state space
- A is the set of actions in the action space
- P is the probability that action a in states at time t will lead to state s' at time $t+1$,
- R is the expected immediate reward that the agent received after transitioning from one state to another due to the action.

2.2.10 Epsilon Greedy

Epsilon-Greedy is a simple strategy for balancing exploration and exploitation by the agent by randomly choosing between random action or action based on the neural network.

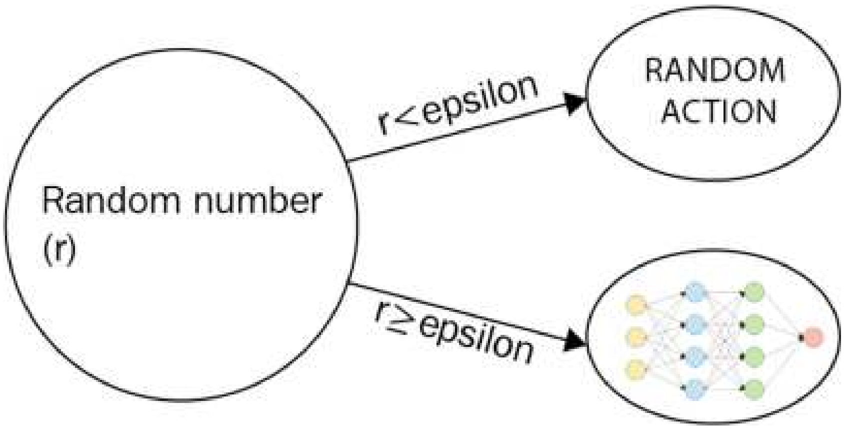


Figure 2.2: Epsilon greedy

2.2.11 Model-free Methods

We can apply all RL problems using model-free methods since they don't need any environment model. There are two Model-free methods; the first is the **Value-based methods**, which try to learn the value function and infer an optimal policy. The second approach is called **Policy search methods** that search in the policy parameters' state-space to find an optimal policy.

We can also classify the model-free methods as **on-policy** or **off-policy**. Off-policy use exploratory policy to generate actions as compared to the policy which is being updated and save its experience or appended it to the replay buffer *D*, which will represent samples from $\pi_0, \pi_1, \ldots \pi_k$,and all the data that is stored will be used to train the new policy $\pi_k + 1$ as presented in 2.3.

In contrast, the On-policy method uses the current policy to generate actions and update the cur-rent policy. In other words, the policy π_k is updated with data collected by π_k itself. By optimizing the current policy, we can determine which spaces and actions to do next 2.4.

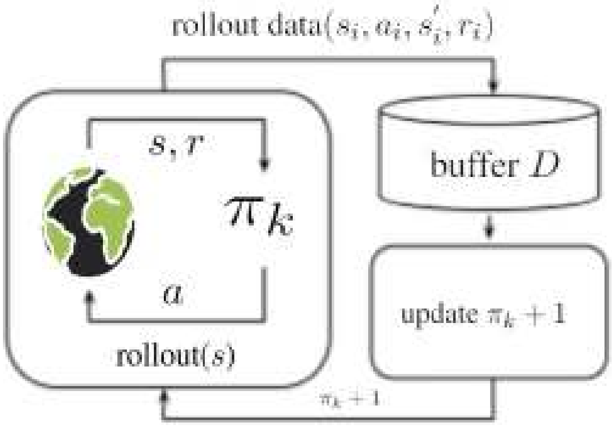


Figure 2.3: Off Policy

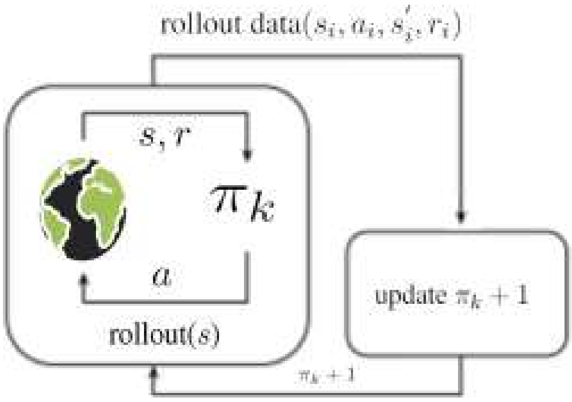


Figure 2.4: On Policy

2.2.12 Value Function Based Methods

Monte Carlo Method

The Monte Carlo method (MCM) is a learning method for estimating value function and dis- covering optimal policies. MC requires only experience from states, actions, and rewards from the agent's interaction with the environment. MCM solves reinforcement learning problems based on averaging sample returns. To ensure good returns are available, we assume experience is divided into episodes, and all episodes terminate at the end no matter what actions are selected (Sutton & Barto,2018b). MCM can be incremental an episode-by-episode, but not in a step-by-step. MCM uses the idea of generalized policy iteration (GPI). The GPI is composed of two steps. The first step is called the policy evaluation step, which builds an approximation value function based on the current policy. The second step is called the policy improvement step. It improves the existing pol- icy based on the current value function. Despite the easy implementation of the method, the main disadvantages of this method are that it takes a huge iteration.

Temporal Difference

Temporal Difference (TD) uses the same idea as the GPI. It uses the temporal error rather than accumulated reward, like what is used in the MCM. Temporal error calculates the difference between the new estimate and the old estimate of the value function. It also considers the current reward, and uses it to update the Value function. This method helps in reducing the variance but increases the bias in the estimation of the value function. The equation from the value function will be as the follows 18:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \tag{18}$$

Where α is the learning rate, r is the reward received at the current time, s' is the new state, and $s$is the old state

2.2.13 Actor Critic Methods

Actor Critic Methods are TD methods but with two components, which are: the actor representing the policy that provides the action in a given state and the value function that acts as the critic, and it helps in evaluating the policy based on the temporal difference error. One of the main advantages of Actor-Critic Methods is that it provides better convergence than most TD methods—faster action computations, especially for continuous tasks. One of the most used algorithms that massively solve complex problems in RL problems is the A3C algorithm which can use multiple CPU cores to run the reinforcement learning and distribute the agent training. We will discuss this algorithm later in chapter 3.

2.2.14 Policy Search Methods

The Policy Search Method (PSM) is RL algorithm that use parameters policies $\pi\theta^{\pi}$, with θ^{π} being the parameter vector. To determine the reward, the policy is evaluated by executing rollouts from the existing policy. Gradient descent is then used to update the policy in the direction of increasing expected return. The equation for the update rule for the parameters of the policy can be formulated as follows:

$$\theta_{t+1}^{\pi} = \theta_t^{\pi} + \alpha \nabla_{\theta^{\pi}} J \qquad , J = \mathbb{E}_{\pi}\left(\sum_{k=0}^{\infty} \gamma^k r_k\right) \tag{19}$$

This method has well convergence properties and can learn stochastic policies, which are not possible with value-based approaches (Divyam,2017)

The major drawback of PSM is their policy evaluation step, which suffers from a significant variance and can slow the process of learning good policies. This can happen in a variety of inter- actions with the environment, making it unsuitable for tasks involving actual robots.

2.3 Conclusion

In this chapter, we introduced some of the basic concepts for both Reinforcement learning and Neural Network where we mentioned the primary learning methods for deep reinforcement learning, which are three. First, the value-based method, which learns the value function and infers an optimal policy. Second, the policy-based method, which uses the vector parameter θ to determine the probability of taking action a when at state s. Lastly, the actor critics method that has two components; the critic estimates the value function, and the actor updates the policy distribution in the direction suggested by the critic. We also mentioned the concept of Epsilon-greedy, which helps the agent balance the exploration-exploitation which helps the agent have a better understanding of the environment. Finally, we introduced some of the essential elements for the neural network and its training process.

Chapter Three

State Of the Art

Chapter Three

State Of the Art

3.1 Literature Review

The last decade is characterized by the increasing availability of public Geographical information. These data can be publicly accessed on the internet due to the efforts made by Volunteered Geographic Information(VGI), geosocial media platforms such as OpenStreetMap (OpenStreetMap contributors,2017) or Earth Engine (earthengine,2021), and the raised interest in such data ex- pressed by the mainstream social media platforms (e.g. Twitter and Facebook). Due to the afore- mentioned reasons, route planning received more interest from many researchers who would like to find a feasible way to embed this data in the routing planning process. Unfortunately, we could not find previous studies on using deep reinforcement learning for planning bikes routes, however, most of the researches used different approaches to plan roads or paths based on the current routing algorithms. The study conducted by (Wang & Zipf,2020) provides a quiet routing service using a new Dijkstra-based routing algorithm that minimizes the exposure of pedestrians to traffic noise pollution by maintaining the route distance constraint. In order to build the traffic noise model on the base of their algorithm, there will be a need to combine volunteered geographic information, official socio-economic data, and open access GPS trajectory data. The approach was tested on the road network of Heidelberg (Germany) showing a great capability in generating quiet routes.

Despite its experiments' success, this approach has several limitations: i) the model that is used to estimate the traffic volume is rough; ii) the 'residential' zones were not considered. Another study by (Lamouik & Sabri,2018) presents a study conducted on behalf of the University of Sidi Mohamed Ben Abdallah, where they introduce a dynamic routing system for traffic in intersections based on real-time traffic conditions (individual vehicle speed, destination, and traffic light status). The approach uses deep convolutional neural networks to estimate and recommend the fastest path for vehicles in an intersection. The model's result has been tested using simulation tools that show that the recommended path had lower travel time and fewer red lights which helps in avoiding the long queues of vehicles at red lights and as a result the agent will favor using the roads with lite traffic. This result was achieved by predicting the future state of traffic lights from an intersection away. (Liu & Chen,2019) proposed a new path selection method for an intelligent driving vehicle that solves path planning in case of traffic jams, restricted driving, and accidents. The approach is based on prior knowledge in applying reinforcement learning, to enhance the shortest path algorithm. The simulations show that the algorithm has advantages in terms of path length. Finally, (Zhiguang Cao,2020) experimented on both artificial and real large road networks using the Q-learning approach to solve the probability tail model-based stochastic shortest path problem targets to minimize the probability of delay occurrence. By using Q-values, they represent the probability of reaching the destination before the deadline. Their models show an accuracy of 97.5% and are ready to be applied in the real-world scenario.

3.2 Modern Deep Reinforcement Learning Algorithms

In this chapter, we will discuss multiple algorithms that are used in the field. We will start with Value-based Algorithms like DQN (Mnih et al.,2013). We will discuss the actor-critic method, which we will go through over A3C, the most hyped algorithm.

Finally, we will discuss Proximal Policy Optimization, which is one of the policy gradient algorithms.

3.2.1 Deep Q-Network

The core principle behind Deep Q-Network (***DQN*** **)** (Mnih et al.,2013) is to adapt the Temporal Difference method based on this formula 20 which is similar to the gradient descent process while training a neural network to handle a specific regression problem.

$$Q^*(s, a, \theta) = \theta * s, a \tag{20}$$

Where all θ form a vector of parameters $\theta \in R^{|s||a|}$

First, the algorithm supposes y as the target of our regression task, i. e. the quantity that our model is trying to predict is:

$$y(s, a) \coloneqq r(s') + \gamma \max_{a'} Q^*(s', a', \theta) \tag{21}$$

Where s' is a sample from $p(s'|s, a)$and s, a is input data. In this notation 20 is equivalent to:

$$\theta_{t+1} = \theta_t + \alpha_t[y(s, a) - Q^*(s, a, \theta_t)]e^{s,a} \tag{22}$$

Where we multiplied scalar value $\alpha_t[y(s, a) - Q^*(s, a, \theta_t)]$ on the following vector:

$$e_{i,j}^{s,a} := \begin{cases} 1 & (i, j) = (s, a) \\ 0 & (i, j) \neq (s, a) \end{cases} \tag{23}$$

According to (Mnih et al.,2013) it is important that dependence of y from θ is ignored during gradient computation. On each step of the temporal difference algorithm, a new target y is con- structed using the current Q-function approximation, and a new regression task with this target is set. For this fixed target one Mean squared error

(MSE) optimization step is done, and on the next step a new regression task is defined.

We now suppose that the appropriate Q-function may be approximated using the neural network $Q * \theta(s, a)$ with parameters θ. Note that in the discrete action space case, this network can only take s as input and output $|A|$ integers representing $Q^*_\theta(s, a_1) \dots Q^*_\theta(s, a_{|A|})$, allowing for a single forward pass through the net to discover an optimal action in a given state s. As a result, goal y for a particular transition $(s, a, r', s', \text{done})$ can be computed in one forward pass, and the optimization step in one additional forward and one backward pass.

The only minor drawback to this simple strategy is that training neural networks with batches of size one is unrealistic. (Mnih et al.,2013) propose using experience replay to save all collected transitions $(s, a, r0, s0, done)$ as data samples and a batch of standard for neural network training size for each iteration sample. The loss function is assumed to represent an average of losses for each batch transition. Because the TD method is an off-policy algorithm, it can work with arbitrary transitions obtained from any agent's interaction experience. This use of previously experienced transitions is legitimate. Another significant advantage of experience replay is sample decorrelation, which occurs when successive transitions from an interaction are generally identical to one another since the agent is usually located at a specific portion of the MDP.

Though the empirical results of the presented algorithm were impressive, the behavior of Q^*_θvalues revealed that the learning process was unstable. Reconstruction of the target after each optimization step resulted in a so-called compound error, which occurred when approximation error spread in an avalanche fashion from close to-terminal states to the starting point, resulting in a guess 10^6 times larger than the genuine Q^* value. To overcome this issue, (Mnih et al.,2013) proposed the target network, which core idea is to handle a fixed regression problem for $K > 1$ steps, i.e., recompute target every $K - th$ step instead of each.

We get the classic DQN method by combining everything and adding the greedy strategy to make exploration easier as represented in 1 by (Mnih et al.,2013).

Algorithm 1 Deep Q-Learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with two random sets of weights θ, θ'
for $episode = 1, M$ **do**
  for $t = 1, T$ **do**
    Select a random action a_t with probability ε.
    Otherwise, select $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t, collect reward r_{t+1} and observe next state s_{t+1}
    Store the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in $\mathcal{D}$
    Sample mini-batch of transitions $(s_j, a_j, r_{j+1}, s_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_{j+1}, & \text{if } s_{j+1} \text{ is terminal} \\ r_{j+1} + \gamma \max_{a'} Q(s_{j+1}, a'; \theta'), & \text{otherwise} \end{cases}$
    Perform a gradient descent step using targets y_j with respect to the online parameters θ
    Every C steps, set $\theta' \leftarrow \theta$
  end for
end for

3.2.2 Dueling DQN

Dueling DQN acts the same as DQN, but the difference is that Dueling DQN aims to build a network that computes the advantage A and value functions V separately as figure 3.1 and then only combines them into a single q-function at the final layer based on the equation:

$$Q(s, a) = V(s) + A(s|a) \tag{24}$$

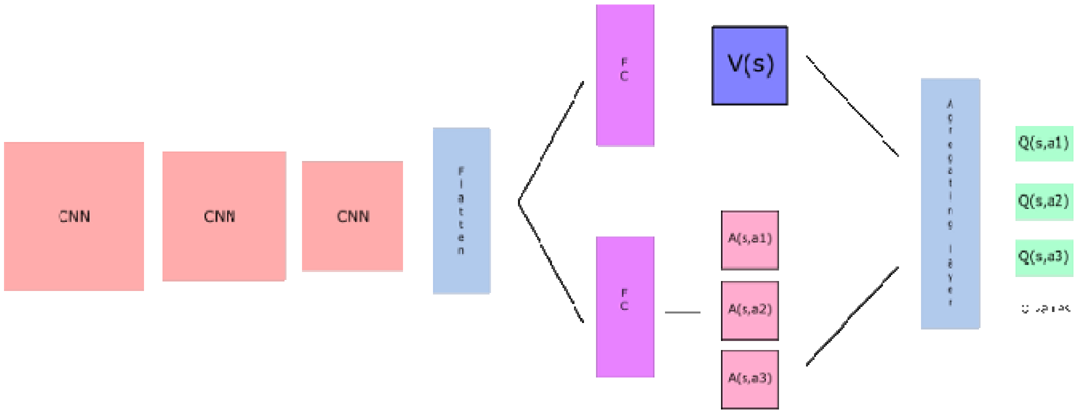


Figure 3.1: Dueling Architecture

3.2.3 Asynchronous Advantage Actor Critic - A3C

Where DQN uses a single agent associated with a single neural network that interacts with the environment, A3C learns more effectively by combining many above iterations with the Actor critic method that we mentioned in 2. There is a global network and many worker agents with their local Neural network in A3C. At the same time, other agents engage with their environments. Each of these agents interacts with its own copy of the environment. One of the reasons this works better than having a single agent is because each agent's experience is separate from the experience of the others. As a result, the entire training experience available becomes more diverse and faster (Mnih et al.,2016).

A3C updates both the policy and the value function in the forward view, using a combination of n step returns. The policy and value function are modified after each action's $Tmax$ value or when a terminal state is reached as represented in 2 as mentioned in (Mnih et al.,2016) paper. The updated equation 25 is as the following:

$$\nabla_{\theta'} \log \pi(a_t \mid s_t; \theta') A(s_t, a_t; \theta, \theta_v) \tag{25}$$

Where $\boldsymbol{A(s_t, a_t; \theta, \theta_v)}$ is an estimate of the advantage function given by:

$$\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v) \tag{26}$$

Algorithm 2 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

//Assume global shared parameter vectors θ and θ_v and counter $T = 0$

//Assume thread-specific parameter vectors θ' and θ'_v

Initialize thread step counter $t \leftarrow 1$

repeat

  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$

  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$

  $t_{start} = t$

  Get state s_t

  repeat

    Perform action a_t according to policy $\pi(a_t|s_t, \theta')$

    Receive reward r_t and new state s_{t+1}

    $t \leftarrow t + 1$

    $T \leftarrow T + 1$

  until terminal **or** $t - t_{start} == t_{max}$

  if s_t is terminal **then**

    $R = 0$

  else

    $R = V(s_t, \theta'_v)$ *//bootstrap from last state*

  end if

  for $i \in \{t-1, ..., t_{start}\}$ **do**

    $R \leftarrow r_i + \gamma R$

    Accumulate gradients wrt θ':

    $d\theta \leftarrow d\theta + \nabla_{\theta'} \log\pi(a_i|s_i, \theta')(R - V(s_i, \theta'_v))$

    Accumulate gradients wrt θ'_v:

    $d\theta_v \leftarrow d\theta_v + \partial(R - V(s_i, \theta'_v))^2 / \partial\theta'_v$

  end for

  Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$

until $T > T_{max}$

3.2.4 Policy Gradient Methods-PPO

Policy Gradient Methods (PPO) are fundamental to recent breakthroughs in using deep neural networks for control, from video games, to 3D locomotion, to GO, (Schulman,2020). However, attaining acceptable results using policy gradient approaches is difficult since they are sensitive to the step size chosen; if it's too small, then the signal is drowned by the noise, or catastrophic de- clines in performance may happen. They also have a low sample efficiency which requires millions (or billions) of time steps to learn basic tasks. We can quickly implement the cost function, run gradient descent on it, and be confident that we'll obtain fantastic results with minimal hyperparameters modification thanks to supervised learning. PPO tries to find a compromise between ease of implementation, sample complexity, and tuning ease by computing an update at each step that minimizes the cost function while ensuring a modest divergence from the preceding policy. PPO uses a novel objective function not typically found in other algorithms 27 (Schulman,2020):

$$L^{CLIP}(\theta) = E_t[min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t)] \tag{27}$$

- $\boldsymbol{\theta}$ is the policy parameter
- $\boldsymbol{E_t}$ denotes the empirical expectation over timesteps
- $\boldsymbol{r_t}$ is the ratio of the probability under the new and old policies, respectively
- $\boldsymbol{A_t}$ is the estimated advantage at time $\boldsymbol{t}$

Algorithm 3 PPO-Clip

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0

2: **for** $k = 0, 1, 2, ...$ **do**

3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

4: Compute rewards-to-go $\hat{R}_t$.

5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

8: **end for**

3.3 Conclusion

In this chapter, we introduced previous work of multiple studies that used machine learning techniques to find a new routing method for different routing preferences. We also introduced two algorithms that are used in deep reinforcement learning, firstly; DQN, that uses two Neural networks to calculate the Q-Values for both the current state and future state. Secondly, the duel DQN, which uses the same method as DQN but it differs in the neural network structure. The difference is that in the duel DQN we calculate the A and V in the last hidden layer A3C that uses Actor critic method and in how we can use multithreading training using CPU to train our DRL agent faster. Finally, we introduced the PPO algorithm that uses the Policy learning method that uses vector parameter θ to determine the probability of taking action at state S_i.

Chapter Four

Data Gathering and Collection Process

Chapter Four

Data Gathering and Collection Process

4.1 Introduction

In this chapter of the thesis, we will discuss the procedures and the tools we used to gather all the necessary data for our model. This section will also discuss data transformation that we applied to a particular aspect of the collected data to help us in developing our model.

4.2 Data Sources

Since our project focuses on smart city planning and most of our data are geographical data, the leading search criteria were to find an excellent geographic database to extract data from it. We can define a geographic database as a repository that stores data that are spatially referenced. The collected data are related to each other through location, data structure, or type. (Geography,2021)

There are two types of geographic databases Graphic and Nongraphic. Nongraphic data does not describe graphic map features, but instead describes a particular map feature or is linked to graphic elements through geocodes identifiers. It takes the form of a geographic index or is used to describe a spatial relationship (Geography,2021).

On the other hand, graphic data contain points, lines, polygons, and other map features such as projections, coordinates systems, and cartographic symbols—this type of data is stored in two ways vector or raster. Vector is represented by coordinates of longitude and latitude of specific nodes and lines or rules to connect the area. In comparison, raster data is stored as a set of a uniform grid of cells representing the continuous surface.

These databases can be accessed by Application Programming Interface (API), which helps developers create applications from it or use for research studies, as we are doing in our thesis.

So, we collected data from the internet on the API that has a comprehensive geographical database that we can extract information from that is useful for our project. Our ideal API must cover the following:

(1) The API must cover all the globe since a lot of GIS APIs cover only Europe and the USA, but we are also concerned about other continents.

(2) The API must contain information about Natural terrain since we are interested in finding the best path for a cyclist in areas outside the cities and villages.

(3) The API must contain information on the roads in each city or village, like the main roads, cycle path, footpath, unclassified roads, living street, etc.

(4) We must extract information on physical facilities in a particular area, like; cafes, bathrooms, shops, parks, etc.

(5) The API should help us in getting information on the traffic and the roads' conditions.

Table 4.1 illustrates all possible APIs that can be used to extract data from it.

Each one of these APIs has its pros and cons, but we try to use most of them to extracts data from and transform it into a unified format that we can use in our model.

Table 4.1: Possible API

Application name	Owned by	API	API NAME	Pricing	Number of transaction
open-streetmap	Openstreetmap Organization	yes	Overpass api	Free	Unlimited
Google Earth engine	Google inc.	yes	Earth engine api	Free to use for research, teaching, and charitable purposes. They provide paid commercial licenses for commercial uses.	–
Tomtom developer	Tomtom	yes	Tomtom maps-api	0.42$ per 1000 transaction	Limited by the cost
HERE developer	Here	Yes	Rest API	Has a free version and pain version	The free version has 250K transaction and you can pay for more

4.3 Data Collection

In this section, we will describe how we used each API to extract the data and how we stored it into our unified JSON file.

4.3.1 OpenStreetMap

OpenStreetMap (OSM) is an open-source collaborative project founded in 2004 (OpenStreetMap contributors,2017). It collects data on houses, forests, road networks, and there are many features included in its database. OSM has an API called Overpass API which helps the developers fetch and save raw geodata from the OpenStreetMap database (Openstreetmap,2021b). Every feature on the ground, e.g., roads or places, is decorated using tags attached to its data structures.

OSM has three essential elements representing the physical world's conceptual data: nodes, ways, and relations. These elements have one or more associated tags (which describe a particular element's meaning) (Openstreetmap,2021c). A node represents a point on earth defined by its longitude and latitude, and each node has its unique id. For example, a node can represent a café, bathroom or crossroad, etc. A Way is an ordered list from 2 to 2000 nodes that define a polyline, and it represents any linear feature such as roads and rivers. Lastly, a relation is a data structure rep- resenting a relationship between two or more data elements, for example, a building and a parking spot.

Roads

To extract roads from the OSM database, we need to use the key **highway=***, which helps us extract any roads, streets, or paths, and the value of the key represents the highway's importance within the road network. Figure 4.1 shows the coverage of the tag worldwide and is represented by the red color on the map, and from the graph, we can see that it covers a massive proportion of the globe (Openstreetmap,2021d).

Figure 4.1: The distribution of nodes and ways with this key. Relations are not shown.

Furthermore, Table 4.2 below from OSM WIKI (Openstreetmap,2021d) shows us the multiple values we can assign to the highway tag.

Table 4.2: Commonly used values with highway tag

motorway	unclassified	secondary link	busway	track
trunk	residential	tertiary link	footway	bus guideway
primary	motorway link	living street	bridleway	escape
secondary	trunk link	service	steps	raceway
tertiary	primary link	pedestrian	corridor	road

Table 4.2 shows how much data we can export from the OSM database that is related to each road; for more information about each value, you can check the full table in the appendix A.2.1. Each road comes with a set of attributes based on the data that the users enter, and these attributes are:

- Access: describe the legal permission for using the road by people, animals, bicycles, and vehicles.
- Max height: describe the legal height of a vehicle that can use the road.
- Max width: describe the legal width of a vehicle that can use the road.
- Max speed: describe the legal speed of a vehicle or a motorcycle, for example.
- Max weight: describe the legal weight of a vehicle.
- Onaway: show if the road is one way or not
- Width: show us the width of the road in meters
- Road surface: show us the material or the structure of the road. For example, if the surface of the road is asphalt, concrete, dirt, paving stones, or others.
- Number of lanes in the road

Finally, we used python to extract the roads from the specific bounded box as shown in Figure 4.2 and converted it into a JSON file output 1 to use it later in our model, as shown in the example below.

Figure 4.2: Way selection in a specific bounding box

Listing 1 Representation of a single road in JSON file

```
10141263 {
            "type": "way",
            "id": 10141263,
            "nodes": [
                153591704,
                3135817806,
                4652784940,
                3135817819,
                83660660
            ],
            "tags": {
                "highway": "tertiary",
                "is_in:city": "Roma",
                "lanes": "2",
                "lit": "yes",
                "maxspeed": "50",
                "name": "Via Crescenzio",
                "source:maxspeed": "IT:urban",
                "surface": "sett"
            }
        }
```

Path/Road Distance

The road distance is one of the primary key elements in our project. There are two reasons that we need to compute the distance for:

To calculate the road slope, which will help the agent understand if there is an inclining or declining on the road in order to compute the road's distance. Although we are not interested in finding the shortest path in our project, we can use the distance of the road as one of the rewards that the agent will receive; for example, if we have two similar roads with the same attribute, we can help the agent choose the shortest road.

As we mentioned in the previous section, the Ways structure in OSM contains nodes; each node has a longitude and latitude and a unique ID as Figure 4.3 illustrates. We used a Python package called **Pyproj**, a great package to work with map projections. **Pyproj** takes longitude and latitude for two-point and returns the distance in meters. In order to calculate the distance for the whole way, we calculated the distance between each node and its neighbor, then added it to a list and finally summed all of the list. After we found the distance of the hole way, we added it as a key attribute in our JSON file. See code 2.

Figure 4.3: The selection of a single way onOpenStreetMap

Figure 4.4: Nodes that represents the way onOpenStreetMap

Listing 2 Representation of a single road with distance attribute in JSON file

```
22912146:{
  'id':22912146,
 'distance':125.11237377238788,
 'nodes':[246677458,246677459,246677460],
 'type':'way'}
```

Physical Facilities

OSM represents the physical facilities like restaurants, public bathrooms, banks, supermarkets, and other facilities as a single node with a unique ID. To extract these data, we need to use the key **amenity=***. Figure 4.5 shows the coverage of this key on the globe represented by the red dots. From the map, we can see that the amenity tag covers most of Europe, North America, and some of the middle east. However, on the other hand, some parts of north Africa and Australia do not have many amenities in them, which will cause an issue if we would like to test our model in these countries (Openstreetmap,2021e).

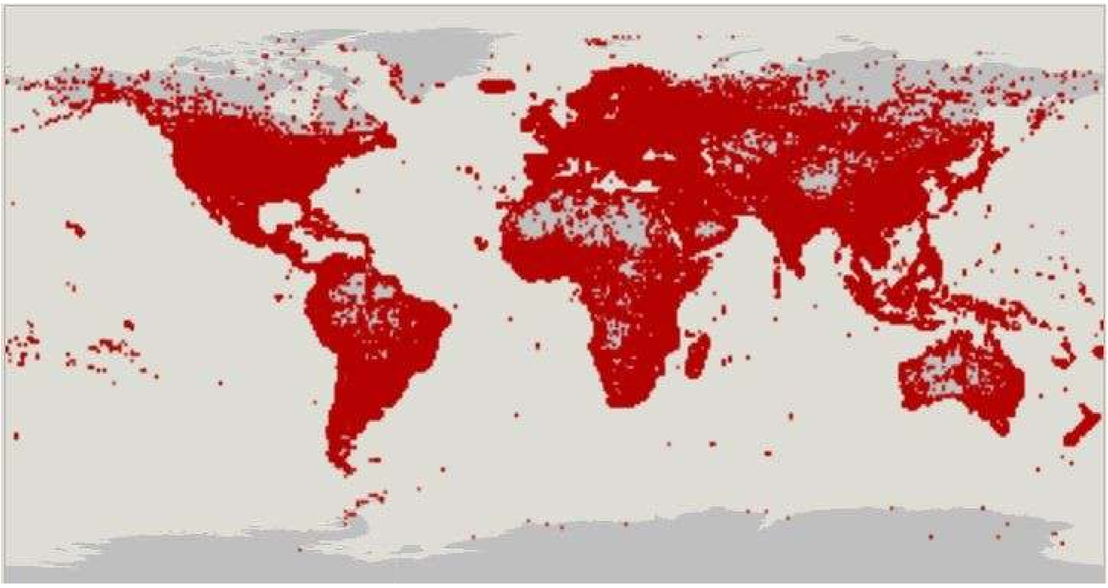


Figure 4.5: The distribution of nodes and ways with this key. Relations are not shown.

Table 4.3: Commonly used values that can be used with amenity tag

bar	public bookcase	grit bin	waste basket
biergarten	social center	motorcycle parking	waste disposal
cafe	stripclub	parking	waste transfer station
fast food	studio	parking entrance	animal boarding
food court	swingerclub	parking space	animal breeding
ice cream	theatre	taxi	animal shelter
pub	courthouse	atm	baking oven
restaurant	embassy	bank	childcare
college	fire station	bureau de change	clock
toy library	prison	hospital	gym
music school	ranger station	nursing home	hunting stand
school	townhall	pharmacy	internet cafe
university	bbq	social facility	kitchen
bicycle parking	bench	veterinary	kneipp water cure
bicycle repair station	dog toilet	arts center	lounger
bicycle rental	drinking water	brothel	marketplace
boat rental	give box	casino	monastery
boat sharing	parcel locker	cinema	photo booth
bus station	shelter	community center	place of mourning
car rental	shower	conference center	place of worship
car sharing	telephone	events venue	public bath
car wash	toilets	fountain	public building
vehicle inspection	water point	gambling	refugee site
charging station	watering place	love hotel	vending machine
ferry terminal	sanitary dump station	nightclub	user defined

Furthermore, Table 4.3 from OSM WIKI (Openstreetmap,2021a) shows us the multiple values we can assign to the amenity tag and more information about each value's description from the table all the needed information mentioned in the appendix A.2.

The main goal of using physical facilities in our project is to determine the best path that has the amenities suitable for the cyclist. For example, the road that has a supermarket and public bathroom is more suitable for the cyclist than the road that has multiple restaurants in it. Another example is if the cyclist is in Saudi Arabia or Algeria, he/she would be more interested in a water point on the road because of the weather in such countries is hotter, and a road that has a water point will help him/her more than one with a lot of fast-food restaurants. We can determine the importance of the amenities in the reward system in our model.

In the code's implementation, we computed the distance between nodes of the road and amenities near the road. If the distance is less than ten meters, we add the road's amenity as an attribute of the road. Figure 4.6 shows an example of how we added the amenity into our JSON file3. As we can see, the road with the blue line is surrounded by a café, bar, cloth shop, and drinking water point. All of these amenities have been added as a list of their id's in our dictionary with the key 'amenities' for future access in our model.

Figure 4.6: Single way and the amenities near it.

Listing 3 Representation of a single road with amenities attribute in JSON file

```
{'amenities':[1324874797,
  3781268652,
  4551731590,
  4875923521,
  1324874753,
  1324874797],
 'distance':125.11237377238788,
 'id':22912146,
 'nodes':[246677458,246677459,246677460],
 'slop':0.0,
 'slop_degrees':0.0,
 'tags':{'highway':'residential',
  'maxspeed':'30',
  'name':'Via Plauto',
  'oneway':'yes',
  'surface':'sett'},
 'type':'way'}
```

As we mentioned before in our reinforcement model, We will focus not only on the number of amenities in the road, but also on their type. We will furtherly discuss in our thesis the reward system.

4.3.2 Google Earth Engine

Google earth engine (GEE) combines a multi-petabyte catalog of satellite imagery and geospatial datasets with planetary-scale analysis capabilities and makes it available for scientists, re- searchers, and developers to detect changes, map trends, and quantify differences on the Earth's surface (earthengine,2021).

Elevation

In our project, we had to extract elevation for each node of the road to calculate the road's slope. The slop will help the agent determine if there is an incline or decline in the road, and we can use it in our reward system for our model. To extract elevation, we used an Elevation dataset called SRTM 90m Digital Elevation Database v4.1, produced by NASA originally, and it is a breakthrough in digital mapping of the world. This data set's main advantage is to provide high-quality elevation data for large portions of the globe. SRTM dataset can be integrated easily in GEE. By giving GEE the longitude and latitude for each node of the road, it will return the elevation for each point in meters, and then we add it as a value of the node, as we can see in code 4.

Listing 4 Elevation value in each node in the file

```
{'type':'node','id':6875735244,'lat':41.90542,

                    'lon':12.4611844,'elevation':26}

{'type':'node','id':3135817825,'lat':41.9058197,

                    'lon':12.4612333,'elevation':26}

{'type':'node','id':6875735241,'lat':41.9058536,

                    'lon':12.4612373,'elevation':26}

{'type':'node','id':246677429,'lat':41.9059068,

                    'lon':12.4612455,'elevation':31}
```

Slope Calculation

As we mentioned earlier, we need the slope to find the road's incline and decline because the cyclist prefers the road with a decline rather than the road with an incline in it, because it takes him/her less effort in the declining. To calculate the slope, we need to use the slope equation as Figure 28.

$$m = \frac{y_2 - y_1}{x_2 - x_1} \tag{28}$$

- m=slope
- $(\boldsymbol{x1}, \boldsymbol{y1})$ coordination of the first point in the line
- $(\boldsymbol{x2}, \boldsymbol{y2})$ coordination of the second point in the line

So, to calculate the slope for the whole road, we will consider the difference in $(y2 - y1)$ as the elevation difference between maximum elevation point and minimum elevation point in the same road, while the value of $(x2 - x1)$ is the distance of the road. See Figure 4.7, which was computed earlier then we added the value in the JSON file as shown in Code 5.

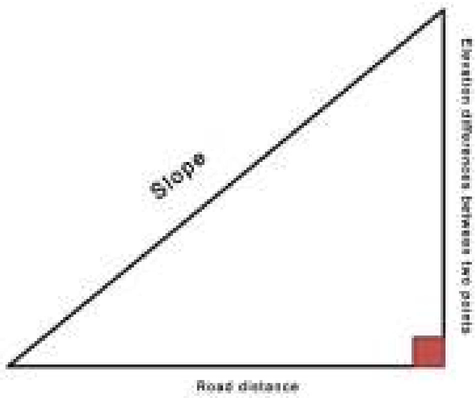


Figure 4.7: Calculation of the slope

Listing 5 Slope and slope degrees for a single way

```
{'distance':110.67147153782149,
 'id':734205333,
 'nodes':[6875735244,
  3135817825,
  6875735241,
  246677429,
  3135817807,
  83656707],
 'slop':0.04517876134222424,
 'slop_degrees':2.586793319868952,
 'tags':{'highway':'residential',
  'lanes':'1',
  'lit':'yes',
  'name':'Via
  Properzio',
  'oneway':'yes',
  'parking: lane: both’: ‘diagonal',
  'parking:lane:both:diagonal':'painted_area_only',
  'parking:lane:left:capacity':'10',
  'parking:lane:right:capacity':'8',
  'surface':'asphalt'},
```

4.4 Conclusion

In this chapter, we discussed the data gathering and collecting process where we searched for multiple GIS database sources, their availability and their data integrity. Then we compared each one of them and mentioned the advantages and disadvantages of each source. We also discussed the extracted feature from both Overpass API and Earth Engine API, and how we used these extracted features to calculate new features, like the slope of the road. Finally, we discussed integrating both data sources into one uniform data structure in the form of a JSON file that helped us later in building the Deep Reinforcement Learning agent environment. We believe that this integration method can be used in different machine learning applications and handle not only two GIS sources, but multiple and different ones.

Chapter Five

Modeling

Chapter Five

Modeling

5.1 Introduction

In this chapter, we will discuss the modeling phase of our project. First, we will discuss how we set up the environment. Second, we will discuss how we set up our agent's reward system. Finally, we will discuss all the methods we used to enhance the accuracy of the agent's policy to make the best action.

5.2 Setting up the Environment

As aforementioned in chapter 4, our data have been collected from multiple GIS data sources, like; OSM and Earth Engine. The data have been collected and formulated as a JSON file. The keys of the file are id's of the intersections, and the values of the key are the attributes of the road, like; max speed, road name, the distance between intersections, etc. Since our data contain a lot of information, we need a representation of the environment that will help the DRL models to get easy and fast access to each road. So, we decided to use Graphs denoted by G as a state representative for our DRL model. Graphs ideally have no assumptions about the size or topology that should be made to ensure their general applicability. Graph processing methods ought to be designed in the absence of known and fixed causal dependencies (medium,2020).

In our case, Vertex (nodes) denoted by V in our environment represents the intersection of the roads. As we mentioned before, each intersection has a specified ID, describing the node's ID in our graph. The edges of the graph, which are lines that connect two nodes and denoted by E, will represent the road between two intersections, and all the information of the road is stored in E. The figure 5.1 shows how we transformed the map of a small number of blocks in Rome and Italy into a graph representation.

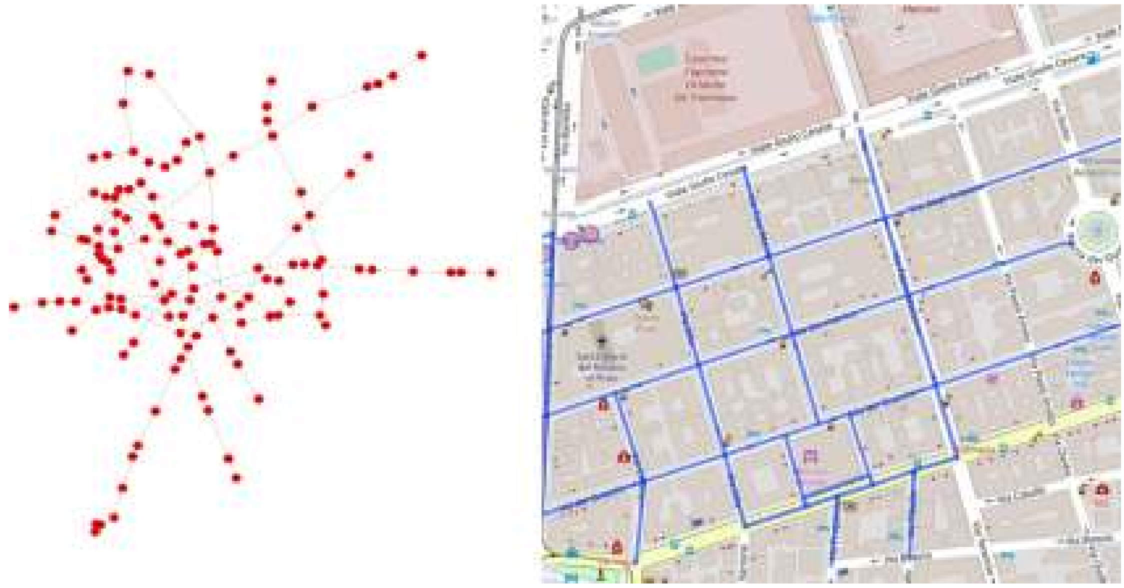


Figure 5.1: Representation of a small sector of the city in graph representation.

5.2.1 Action Space

Action space (*A*) is the set of all our agent's actions in a given state. Since we use Graphs as state representations, the agent will encounter each graph node to move from one point to another. In other meaning, the set of actions will be the total number of nodes in our graph. The agent should distinguish which node is linked to gathering and which of the nodes are not by specifying a reward and penalty for each case. We will discuss it later in this chapter. We can formulate the equation of as follows:

$$A = V \tag{29}$$

Where n is the total number of V in G and of the A.

5.2.2 State Space

In RL the agent comes across a state S, and then takes action A according to the state it's in.

In our project, the State Space (SB) is the set of all V that the agent will encounter from V_A (representing the starting node) to Node V_B (which is the goal that the agent will reach). So, the state space of $S_{A\rightarrow B}$ will be the total number of nodes of the same graph. For example, if we have a graph of 5 nodes and the nodes' id are from 1 to 5, and we need to find a path between node number 2 to node number 5, the state space will be the total number of nodes which is 5 and we can formulate it according to the following equation:

$$SB2 \rightarrow 5 \; = \; n \; = \; 5 \tag{30}$$

Where n is the total number of nodes in G

However, the user should pick any two-points on the map and find the optimal path based on their preference. So, to achieve this, the agent should be trained on all graph nodes; as starting node and destination node. Based on equation 30, if we take all the nodes of the graph, the state space will be the sum of N for all possible starting and destination nodes, which can be represented as the power of 2 for the number of V in G as following:

$$SB = N^2 \tag{31}$$

So, in our example of the 5 node graph, the State space of the graph is 25.

In our case, after we extracted a small section of the city of Rome, we get a total number of nodes of 3080. If we implemented equation 31, we would get 9486400 states, which is a massive number of states for a small section of the city.

5.2.3 State Encoding

Another aspect of our DRL models in any RL environment is that encoding the states should be in a way that shows the current node the agent is at and the goal destination it should reach. Equation32shows the encoding process for each state in our state-space.

$$S_c = c + n * g \tag{32}$$

Where $c \in V$ is the current node, $g \in V$ is the final node and n is the total number of V in G.

For example, if we use the 5 nodes graph case that we mentioned before and we suppose that the agent is in node number 4, and the goal is node number 1, the state will be as the following:

$$S = 5 + 5 * 1 = 10 \tag{33}$$

So, we will be in state number 10 in our state space which contains 25 states

5.2.4 State Decoding

For coding purposes, we needed to decode the state the agent is at and the next state that the agent will go to to get all the information stored in the edge between these two nodes. The equation below shows how we decoded the state. Equation 35 shows how we decoded the state to get the V

$$c = Sc \quad mod\, n \tag{34}$$

$$g = \frac{Sc - c}{n} \tag{35}$$

for example, if we have $S = 10$ based on the equation 35:

$$Vcurrent = 10 \quad mod\, 25 = 5 \tag{36}$$

5.2.5 Reward

The reward system for our DRL project is complex and has multi-features that need to be man- aged, and a reward equation needs to be set for each one of the features that we have. We will discuss how we construct each one of them, and lastly, we will discuss the whole equation that returns a single value for the step function that the agents take in each state.

Node Links

The agent action space, as we have aforementioned, represents all the nodes in G, therefore, the agent needs to learn to take the action that leads to going to another state that has an edge between their nodes. Accordingly, we decided to give the no-link between the two nodes a penalty of -1000, whereas, if there is a link, a penalty of 0 and other types of rewards will be given to the agent, like; distance, max speed, road type, etc., as the equation 37:

$$rew_{link}(e) = \begin{cases} 0 \Leftrightarrow e \in E \\ -1000 \Leftrightarrow e \notin E \end{cases} \tag{37}$$

Distance:

The distance between two nodes is the most fundamental aspect of any cyclist. The cyclists prefer to take the shortest distance of the road, so, the shorter the distance is, the better it is for the cyclist. Thus, the longer distance we have, the higher the penalty to be taken by the agent. So, the most convenient solution is to convert the distance into a negative value by multiplying it by -1 according to equation 41.

$$rew_{Distance} = -e_{Distance} \tag{38}$$

Figure 5.2 shows that the reward equation represents the linear decay. The longer the distance is, the higher the penalty will be. For example, the agent will avoid going by the highways and prefer to go by the shorter distances.

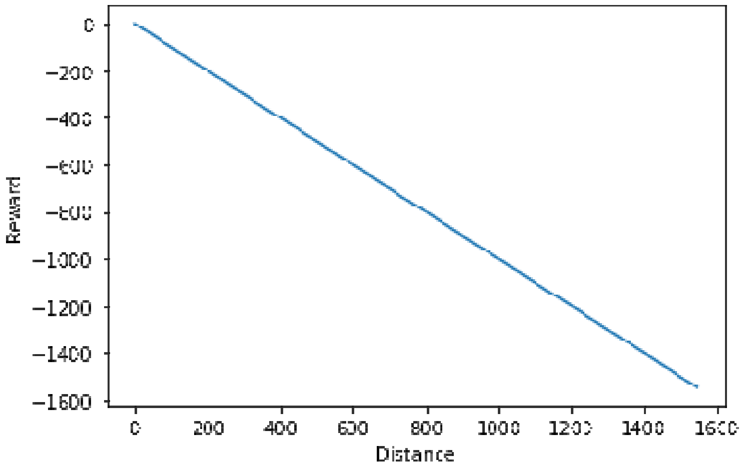


Figure 5.2: Reward for the distance

Slope

The slope of the road is also like distance in being one of the essential aspects of the road for any cyclist. The more incline the road has, the more effort the cyclist needs to put in. Therefore, the main idea here is to give the agent a penalty for the higher slope, which means the inclining. Additionally, this must take into consideration that when we have two slope values near each other, for example, slop1=10.123 and slop2=10.124, the reward for both of these two slopes should be the same; since they are relative values to each other. As a result, we decided to choose the power function base on equation 39, and this will help us in giving more minor penalties for lower slopes and higher penalties for large slopes, as figure 5.3 shows:

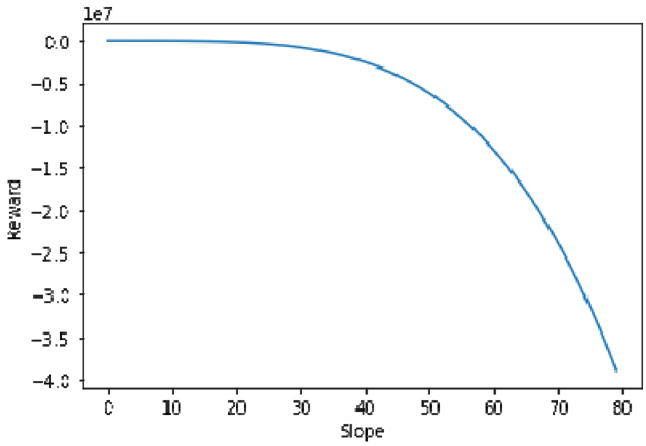


Figure 5.3: Reward for the slope of the road

$$\text{Rew}_{\text{ Slope}} = -\left(e_{\text{slope}}\right)^4 \tag{39}$$

Road Type

In this feature, we have many values for road type. As we mentioned in chapter 4, we have up to 50 values, so we decided to give the user the ability to decide which road type he/she prefers to use. For example, the user might like to use secondary, tertiary, and cycle path roads for the agent to have higher rewards, but in our case, since this project is for cyclists, any road that has a cycle-path or a cycle-way tag will have higher rewards as the following equation:

$$rew_{road_{type}}(e) = \begin{cases} 100 & e_{road_{type}} \in \{\text{cyclepath,cycleway}\} \\ 50 & e_{road_{type}} \in \{\text{road type user preferences}\} \\ -50 & else \end{cases} \tag{40}$$

Max Speed

Max speed of the road is one of the features that we have. Cyclists usually prefer to not use the roads that have high speed since it leads to more accidents. So, the higher the speed limits are, the higher the reward will be. We will use an equation like the one we used for distance equation 41

$$\text{rew}_{\ MaxSpeed} = -e_{MaxSpeed} \tag{41}$$

Road Surface Type

In the road surface, we are going to use the same method as we used in the road type. Some cyclists do not mind, for example; taking a dirty road or paving stones road that has shorter distance or users. So, we will give them the ability to choose which road surface to use, and we will use the same equation 40

Road Lighting

$$rew_{\text{rond\_surfacx}} = \begin{cases} 50 & \text{road surface user preferences} \\ -50 & \text{else} \end{cases} \tag{42}$$

As we mentioned in chapter 4 , road lights have different categories, such as; ”Yes”, ”24/7”, ”sunset-sunrise”, ”automatic”, ”operating times”, ”No”, and ”disused”. Accordingly, we decided that the roads that have lights on always ”yes” will have a reward of 50, ”24/7”,”sunset-sunrise”, ”automatic”, ”operating times” will have a small reward of 10 and finally ”No” and ”disused” categories will have a penalty of -20 and can be formulated as equation 43.

$$\text{rew}_{\text{road\_lights}} = \begin{cases} 50 \text{ yes} \\ 10 \quad 24/7, \text{sunset} - \text{sunrise, automatic, operating times} \\ -20 \quad \text{no, disused} \end{cases} \tag{43}$$

Tracer Number Reward

As we mentioned before, we used the trace number that we extracted from the OSM user tracking system to denote if the streets are used by many people or not. We prefer to use the roads that have less people in, so we decided to have a simple equation for this feature and only multiple the number of tracings for the users by -10, and this will help the agent to select the road that have fewer people in it.

$$rew_{\text{number of tracers}} = -10 * e_{\text{number of tracers}} \tag{44}$$

Total Reward

Since the agent needs to have a single reward in every action, we need to formulate an equation that can merge all of the rewards that we have into one final reward. We also need to take into consideration the interest of the user, and we need to give weight to each feature of the reward. The equation 5.2.5 explains how we combined all the rewards into one.

$$rew_{total} = \beta_1 * rew_{distance} + \beta_2 * rew_{road\_type} + \beta_2 * rew_{slop} + \beta_3 * rew_{\substack{max\_ \\ speed}} + \beta_4 * rew_{surface_{type}} + \beta_5 * rew_{lighting} + \beta_6 * rew_{tracer_{number}} \quad (45)$$

where:

$$\beta_0 + \beta_1 + \beta_2 + \beta_3 + \beta_4 + \beta_5 + \beta_6 = 1 \quad (46)$$

5.3 Model Evaluation and Validation

5.3.1 Evaluation protocol

In Deep Reinforcement Learning, we look into the two criteria to evaluate the performance of the trained models. First, we examine the cumulative rewards that the agent receives during the entire training, where we desire that the agent receive a positive cumulative reward in the early stages of the training, as shown in graph 5.3.1.The second metric is the loss where we desire to reach the minimum loss faster in the early stages, as shown in graph 5.3.1.

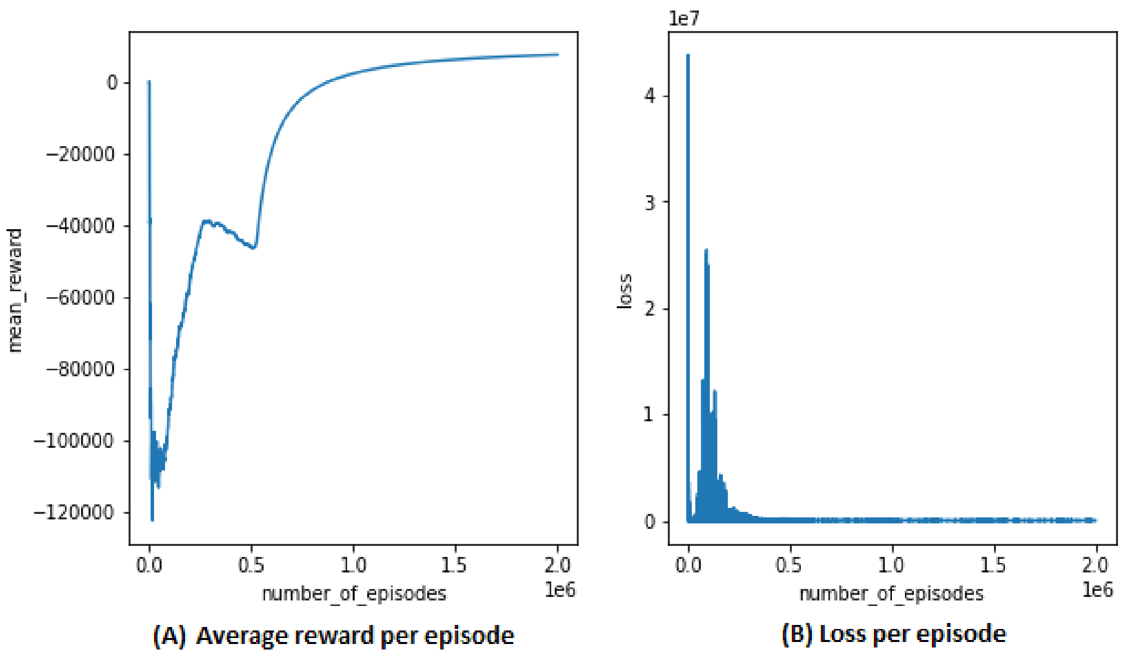


Figure 5.4: Evaluation Plotting examples

5.3.2 Validation protocol

In order to determine if the model results are valid or not, we need a way to validate the result of the agent output, and the standard way to validate the result of any DRL is to compare the cumulative agent rewards and maximum rewards with the actual maximum cumulative rewards of the optimal solution. Unfortunately, in our case, we do not have an optimal cumulative reward since we have different scenarios to calculate the maximum cumulative rewards. So, in order to validate our results, we decided to compare our results with the Dijkstra algorithm which finds the shortest path. In order to do so, we will give the $rew_{Distance}$ higher weight than the other variables, and if the agent was able to find the same path as the Dijkstra algorithm, then we can be sure that the agent performance is excellent.

5.3.3 Conclusion

In this chapter, we discussed the modeling part of our model, where we discussed the DRL environment and the JSON file which we converted into a graph representation. Also, we discussed the state encoding and decoding that will help the agent understand the current node that the agent is at, and the destination node and then convert the encoded state into vector representation that will represent the input of the neural network. Furthermore, we discussed our complex reward system that depends on the end-user preferences. We used each feature as a sup-reward and used a uniformed equation that calculates the final reward that the agent will receive.

Chapter Six

Implementation

Chapter Six

Implementation

6.1 Implementation using DQN

6.1.1 Experiment 1: testing the possible hyper-parameters

For the first implementation, we did choose a small sample of our main G, which contains 3080 nodes but, since this number needed much time to train the agent on, we sampled a node of 22 nodes, as figure 6.1.1 shows. We also decided to choose a fixed starting and ending node, to not have the need to train our model on each node as starting and ending nodes.

The neural network that we are using will take *s* and output a list of 22 nodes representing the number of actions that will be used in the *Argmax* to choose an action. Figure 6.2 shows the neural network's architecture. The NN architecture's input layer will take one single value and output of 128, the second layer will take 128 and output of 64, the third layer will take 64 nodes and output of 32, and the output layer will be 32 and the output will take 22 values.

Table 6.1 explains the results from training the DQN model with different hyper-parameters, like; changing the number of training episodes, learning rate, and batch size. We used the same hyper-parameters that are mentioned in (Mnih et al.,2013). A fixed list with a size of 10000 to save the transactions, and we used smooth L1 to calculate the loss between the *q-network* and the *target network*. Finally, we used linear epsilon decay for the epsilon-greedy method. However, the agent did not perform well in every situation, the maximum rewards did not surpass *0*, and the loss function did not converge as expected and it took so much time. As shown in figure 6.1.1 the reward curve of one of the training trials kept decreasing, and loss increased. This might be because of three reasons: First, the input of the neural network needs to be more than one and needs to be encoded into the one-hot vector. The second reason is that it might be because of the loss function, so, we need to explore more than one loss function. The final reason is that it might be because the agent is exploring more than exploiting the network is not getting enough optimization from the training loop.

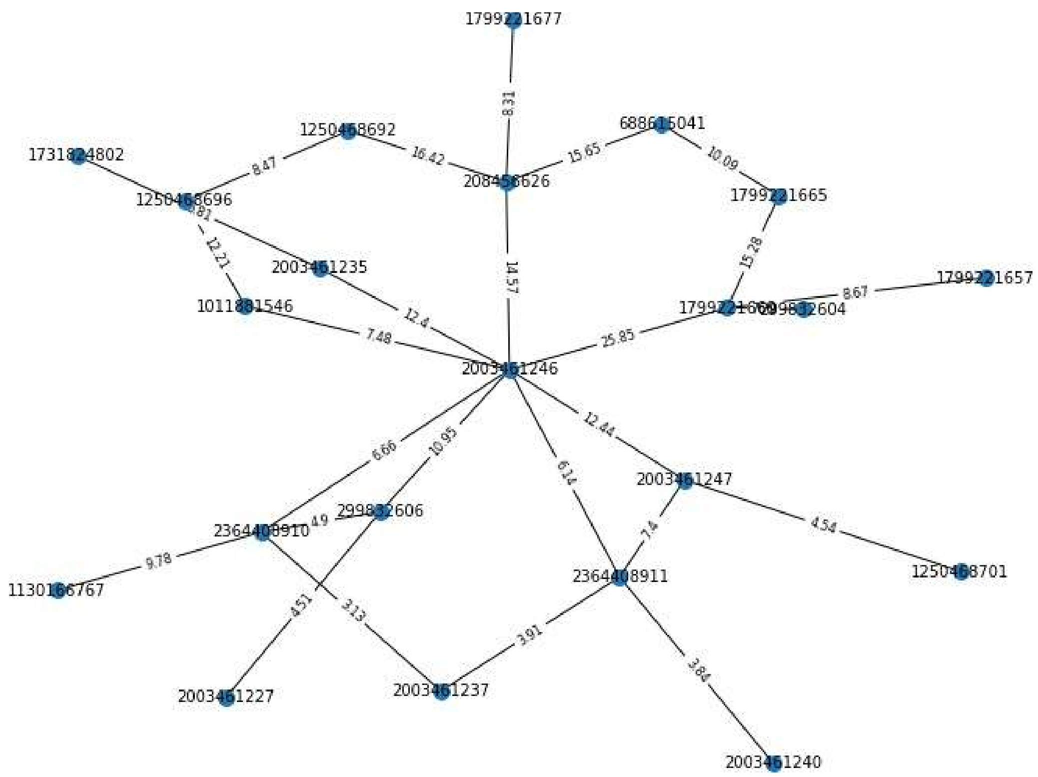


Figure 6.1: 22 Nodes Representation

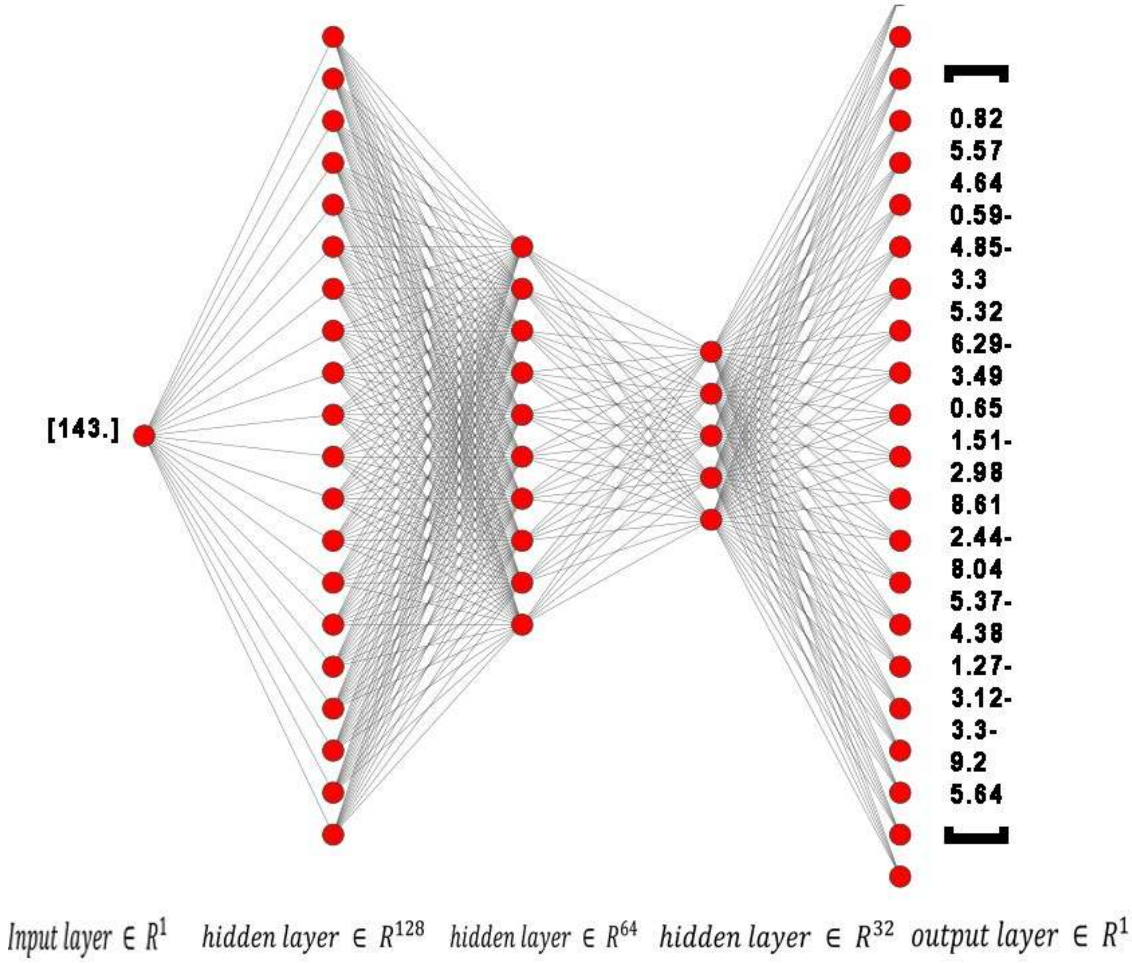


Figure 6.2: Neural Network's Architecture

Table 6.1: Training on multiple hyper-parameters

number of episodes	max reward	min reward	avg reward	loss for the last 100	explore	exploit	LR	time	batch size
100000	0	-1428437.078	-1017564.335	1097907.622	55056	44944	0.01	0.011215278	32
500000	0	-1040055.495	-828232.426	2024357.758	275123	224877	0.01	0.025798611	32
1000000	0	-960477.7964	-724461.2733	6395742.307	548904	451096	0.01	0.0396875	32
2000000	0	-1099749.413	-720137.615	2215640.349	1100345	899655	0.01	0.107743056	32
100000	0	-1473775.17	-973039.9728	938048.1829	54941	45059	0.01	0.003101852	64
500000	0	-1473775.17	-875211.284	44537.70373	274571	225429	0.01	0.015717593	64
1000000	0	-1166130.288	-824403.3387	2242083.187	550050	449950	0.01	0.031631944	64
100000	0	-1040437.401	-719450.822	206383.1865	55005	44995	0.001	0.286805556	32
500000	0	-1091517.954	-872580.9445	5364185.617	275583	224417	0.001	0.024097222	32
1000000	0	-915838.0495	-776889.3067	6438665.339	549899	450101	0.001	0.049548611	32
2000000	**0**	**-945932.3073**	**-369164.8433**	**2472125.135**	**1100220**	**899780**	**0.001**	**0.109467593**	**32**
100000	0	-1275453.819	-882831.7635	1039184.755	55159	44841	0.001	0.003171296	64
500000	0	-1049088.763	-935383.0827	69594.58524	274969	225031	0.001	0.015277778	64
1000000	0	-926163.3303	-757734.499	2048730.334	550096	449904	0.001	0.031388889	64
100000	0	-1409200.466	-984122.5152	246061.7164	55138	44862	0.0001	0.004965278	32
500000	0	-1659793.262	-1103674.093	308459.6945	274830	225170	0.0001	0.025138889	32
1000000	0	-1634936.503	-1046251.191	422429.4037	549339	450661	0.0001	0.107384259	32

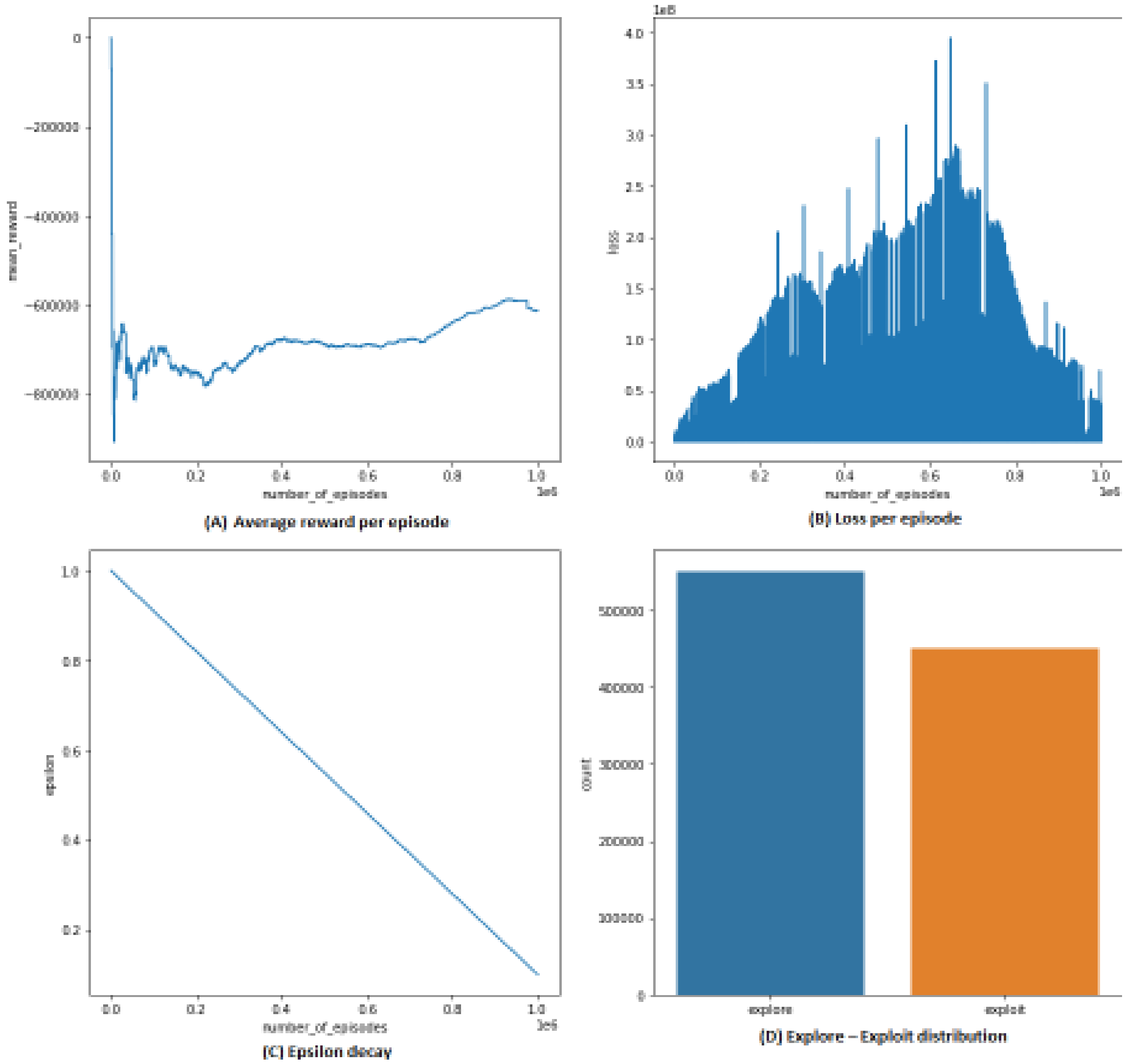


Figure 6.3: Training output for 1000000 episodes

Furthermore, we tried to change the state shape that the *NN* receives as an input and make it more complex, so, we changed it from one state number into a binary representation. The basic idea is to decode the state to get the current node in *G*, which the agent is at, and the terminal node that the agent attends to reach. Then encode both nodes into a list of zeros, and the node's index in the *G* will be one in the array of the One hot encoder. Figure 6.4 presents an example of how

We changed State number 148 into a binary representation with G = 8. First, we decoded the state using the state decoding function 35, giving us two outputs, the current and terminal nodes. Second, we assigned each node ID to index the node's location in G. Third, we converted each node into a list of zeros, and the node's location in G will be one. Finally, we concatenated the two lists into one.

Another change we did in the model is changing how the epsilon decay works. In the previous training, we made the epsilon decrements in a linear method that made the model explore more than exploit. So, we decided to make the exploitation part more than exploration. Therefore, we decided to decay the epsilon in an exponential method, as we can see in figure 6.1.1 which is one of the model's training we did before. In this figure, the agent starts exploiting two-thirds of the time, and this will do more optimization for the NN.

As we can see from table 6.1.1,the changes that we made in state representation and the exponential decay of the epsilon improved the agent performance, especially in the loss convergence values for the last 100 episodes, and the values of the loss are near zeros for all of the experiment with different hyper-parameters. However, as figure 6.6 shows, the cumulative rewards did not change much, and that might be because the NN did not find an optimal policy due to the in- sufficient convergence of the loss. Therefore, we need to try different loss equations on different hyper-parameters, which will be discussed in the following experiment.

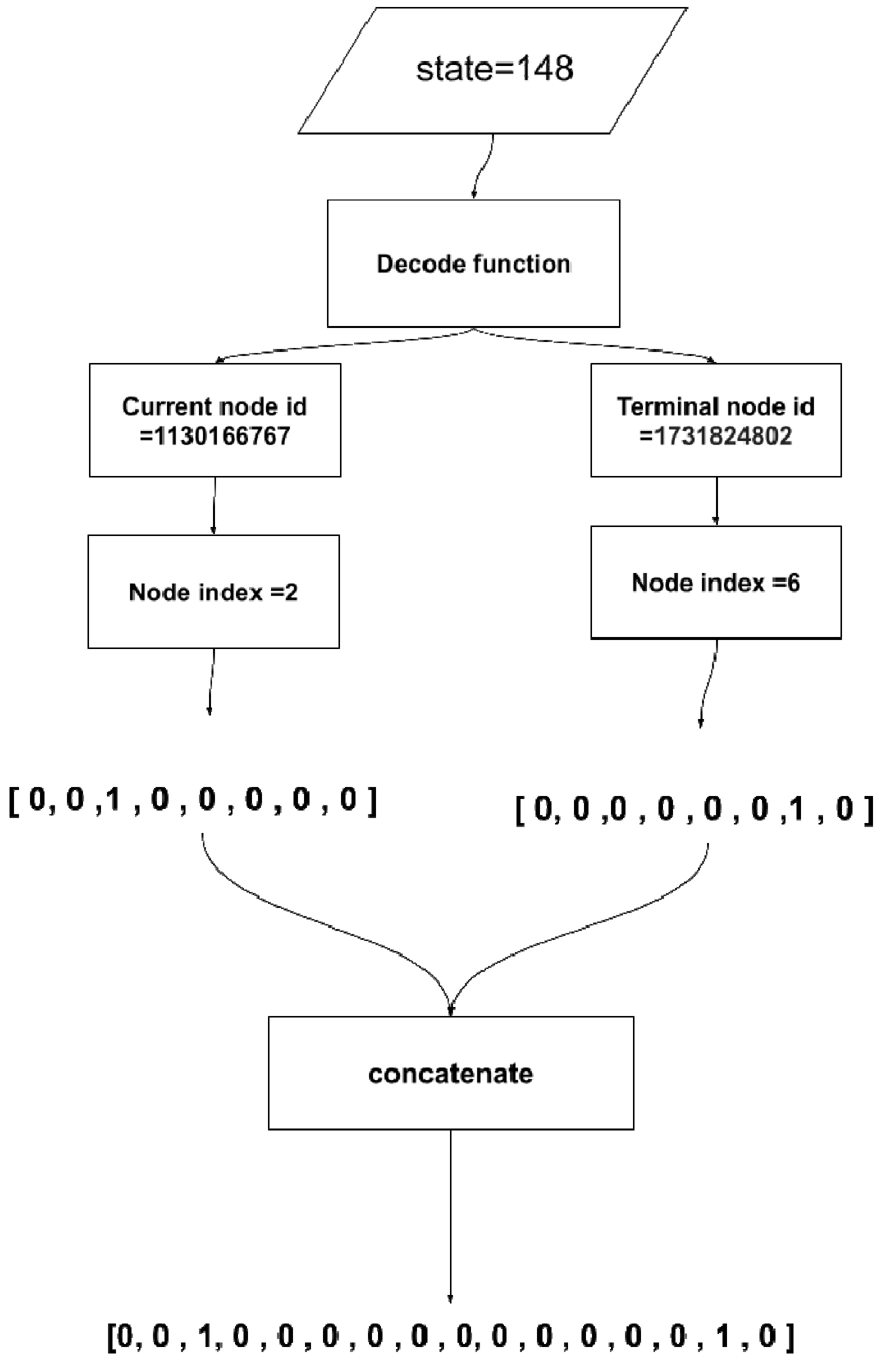


Figure 6.4: Convert State into Vector

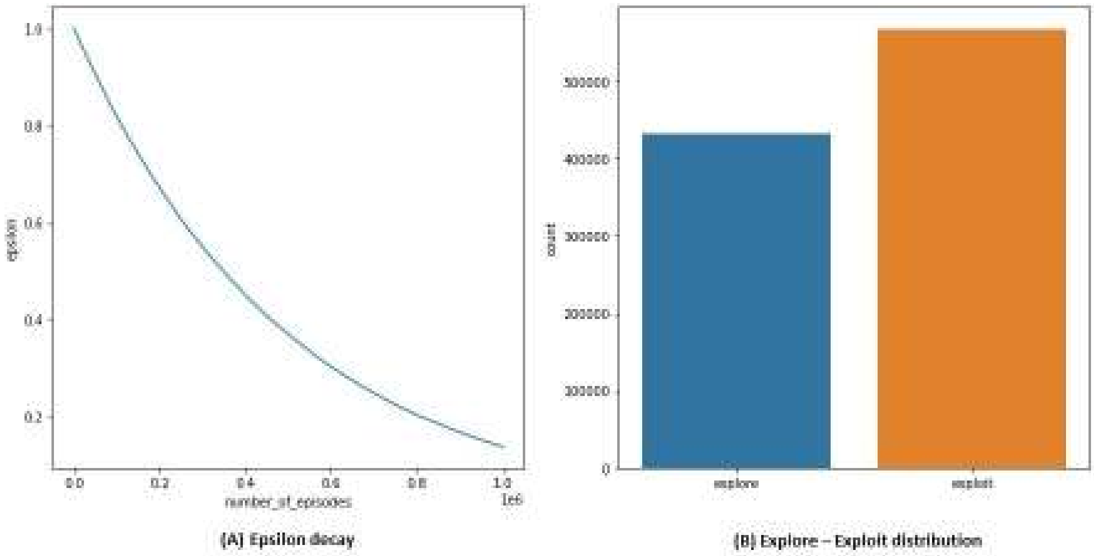


Figure 6.5: Epsilon Exponential Decay

Table 6.2: Training on multiple hyper-parameters

number of episodes	Max reward	Min reward	Avg reward	mean loss for the last 100	explor e	exploit	LR	Time	batch size
100000	0	-1339175.08	-1015476.229	160.537522	43053	56947	0.01	0.26597222	32
500000	0	-1482216.88	-1090516.242	287.06559	215667	284333	0.01	0.53125	32
1000000	0	-1325859.32	-1048380.321	195.447582	432637	567363	0.01	1.875	32
100000	0	-1166633.07	-939755.4216	216.48597	43174	56826	0.01	0.31736111	64
500000	**0**	**-947340.606**	**-853799.3743**	**601.834865**	**216086**	**283914**	**0.01**	**1.60763889**	**64**
100000	0	-1295025.58	-9490539368	173.624174	43244	56756	0.001	0.28125	32
500000	0	-1289354.32	-956076.9008	167.887373	216150	283850	0.001	1.20833333	32
1000000	0	-1384660.5	-1075206.933	110.250113	432197	567803	0.001	0.0453125	32
100000	0	-1185564.55	-912106.2395	74.0372241	43120	56880	0.0001	0.25069444	32
500000	0	-1272237.68	-948267.1576	50.8578429	216292	283708	0.0001	1.31666667	32

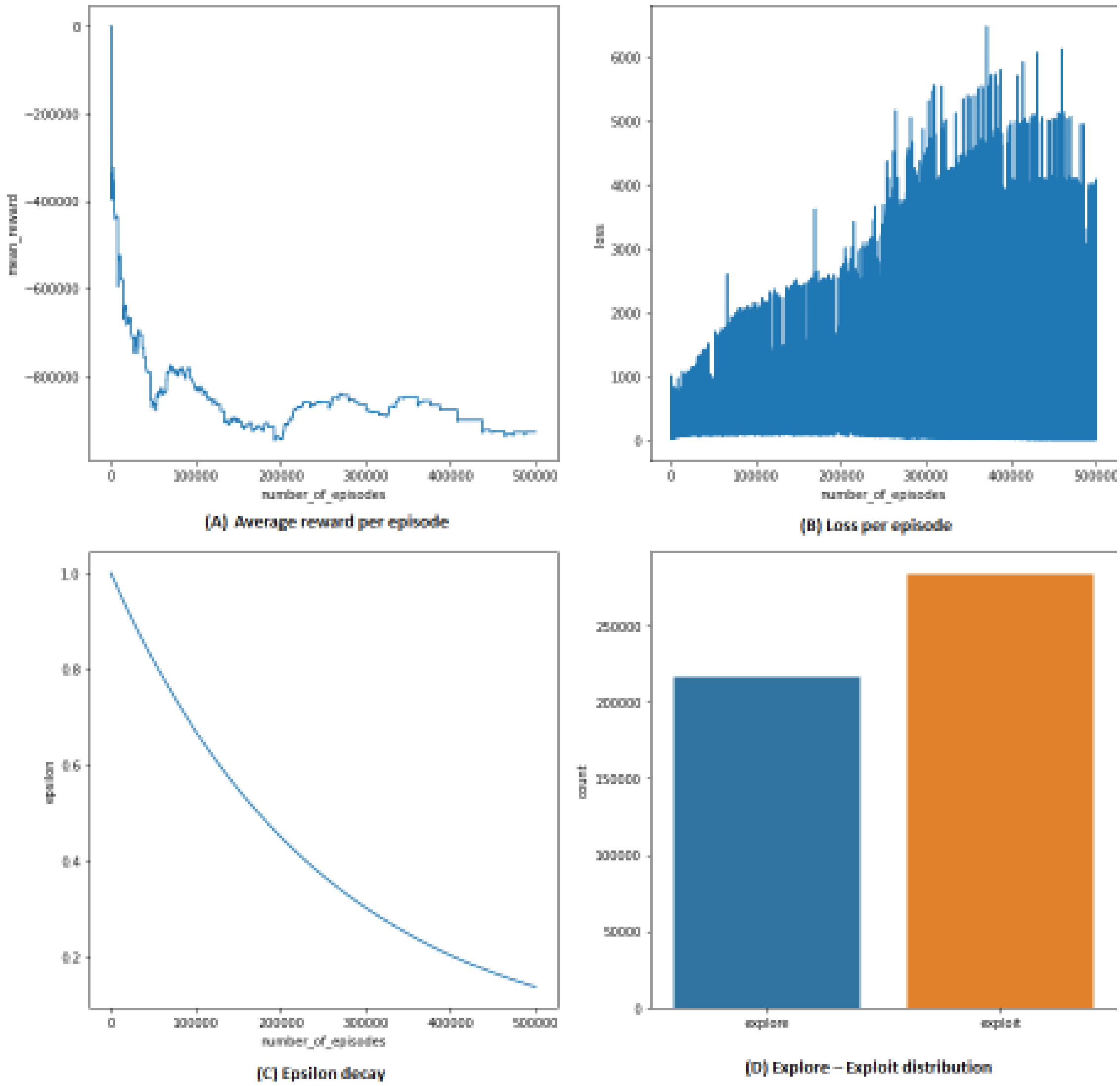


Figure 6.6: Training Output for 1000000 Episodes

6.1.2 Experiment 2: Testing the possible loss functions

In this section, we experimented different loss functions in order to solve the decreased reward issue, which might be related to SmothL1 loss that did not converge in each training we did be- fore. Therefore, we tried multiple loss functions, which are: Gaussian negative log-likelihood loss GaussianNLLLoss, Negative log-likelihood loss with a Poisson distribution of target PoissonNLL and Mean square error MSELOSS. We also tried using a new loss function, which was mentioned in Chapter 2, Clipped loss and Reg loss, then we compared it with the smoothL1 loss function. We ran the following function and did training with 32 batches: a learning rate of 0.01 and 100k games and 500K games. We can notice from the figures 6.1.2, 6.1.2 that, in regards to the rewards system, both MSE and Clipped loss performed more efficient than all the other loss functions we used. Furthermore, figure 6.1.2 shows that the convergence of the loss of MSE, Regloss, and GaussianLoss is faster than the other three. However, Regloss converges faster than MSE and GaussianLoss. As a result, that we will choose the Regloss for the implementation of our code.

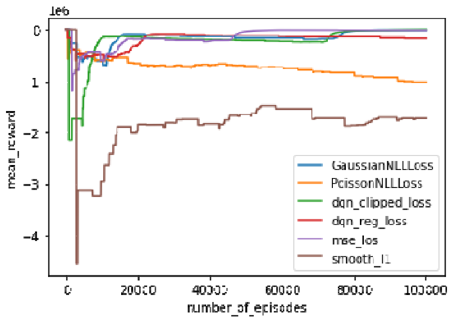


Figure 6.7: 100K Iterations

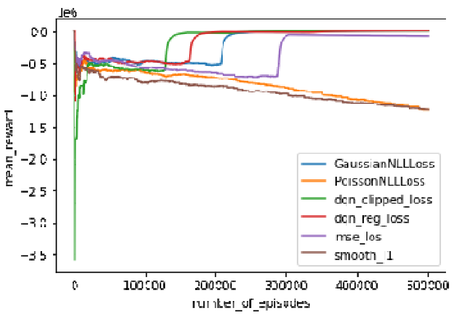
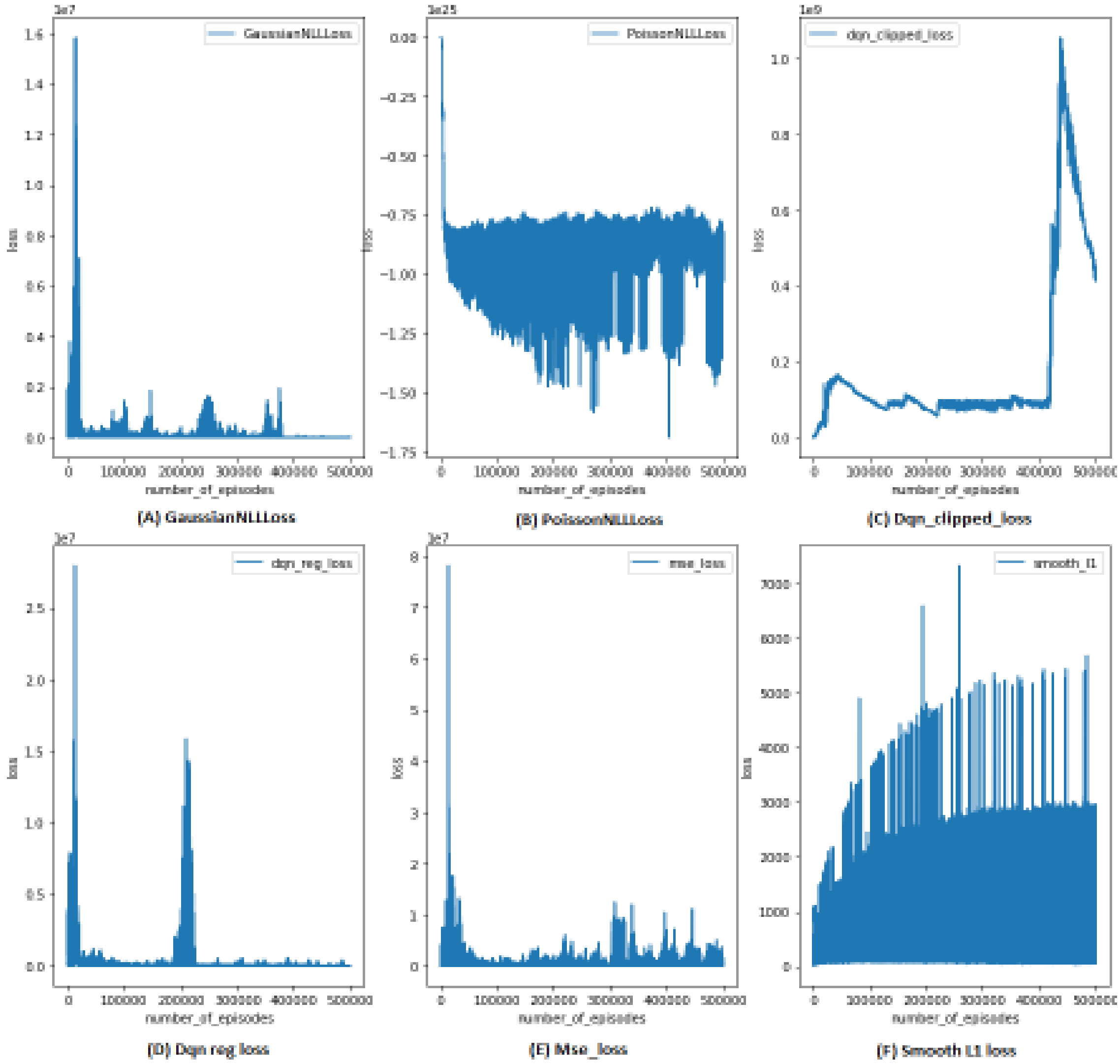


Figure 6.8: 500K Iterations



(A) GaussianNLLLoss (B) PoissonNLLLoss (C) Dqn_clipped_loss

(D) Dqn reg loss (E) Mse_loss (F) Smooth L1 loss

Figure 6.9: Loss functions for 500K iterations

6.1.3 Experiment 3: testing DQN on different Problem Sizes

G=22 nodes

In the first training, we used a G of 22 nodes, and we set the starting node at node number 1130166767 and the node goal at 1731824802. With these hyper-parameters in table 6.1.3, and we set the weights of the rewards system as the following table 6.1.3. As we can see, the distance and the slope have more weight than the others which means that the agent is more interested in finding a road with short distance and less slope.

Table 6.3: Hyper-parameters for 22 nodes

episodes	100000
lr	0.01
gamma	0.99
target update freq	0
loss	dqn reg loss
epsilone	expodinal decay
epsilone start	1
epsilone end	0.1
Buffer size	10KB

Table 6.4: Rewards wights

β_0	distance	24.93%
β_1	road type	21.47%
β_2	slop	6.29%
β_3	lane number	13.72%
β_4	lights	6.22%
β_5	max speed	20.30%
β_6	surface type	0.28%
β_7	number of traces	6.80%

Figure 6.10 shows that the agent performed exceptionally well and started getting positive re- wards after 20k episodes, which means that the agent started to distinguish the linked node quickly avoided the disconnected nodes and reached a maximum reward of 4700 in a training time of 7:03 minutes. Another point we can note from the figure is that the loss function also converged quickly. The output of this training shows the sequence of nodes that the cyclist should take to reach their destination. And since we are more interested in the distance because it has a higher weight than the other feature, we can see that the output is similar to the shortest path.

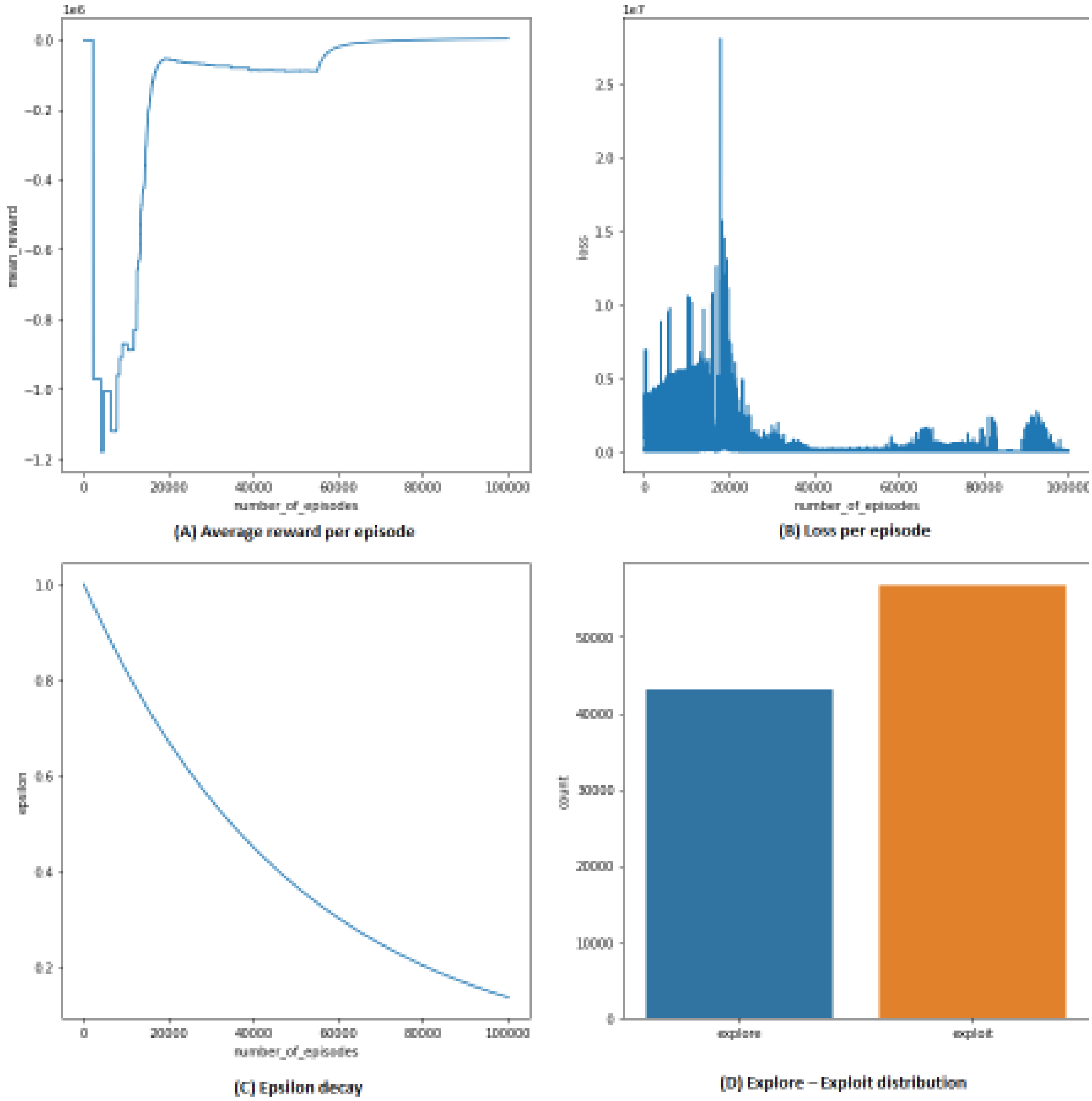


Figure 6.10: Training results

Agent path = 1130166767 → 2364408910 → 2003461246 → 2003461235 → 1731824802

Shortest path = 1130166767 → 2364408910 → 2003461246 → 2003461235 → 1731824802

G=47 nodes

Under this training, we wanted to increase the number of nodes to 47 to see if the training model works on a larger scale to construct the model on it. We used the same hyper-parameters, but we changed the episodes number and the weight's size, as you can see intable 6.1.3 ,6.1.3.

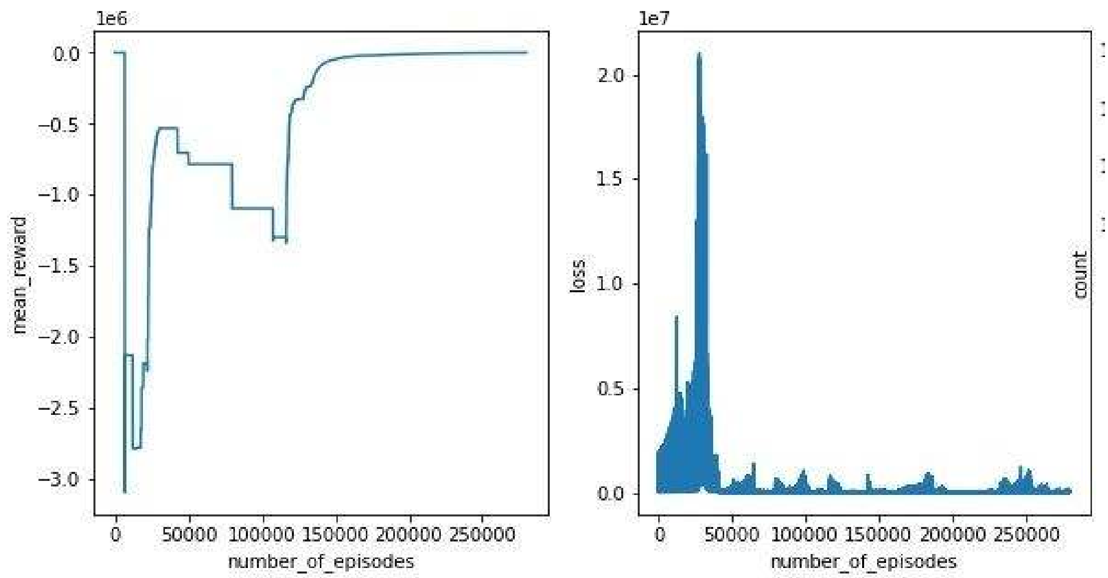
Furthermore, we can see from figure 6.1.3 that the agent performs well as we expected with maximum reward of 7450, but it took 20:04 minutes and that is expected because there are more episodes the agent needs to be trained in.

Table 6.5: hyper-parameters G=47

episodes	280000
lr	0.01
gamma	0.99
target update freq	0
loss	dqn reg loss
epsilone	expodinal decay
epsilone start	1
epsilone end	0.1
Buffer size	10KB

Table 6.6: rewards system weights

β_0	distance	20.00%
β_1	road type	7.69%
β_2	slop	20.91%
β_3	lane number	2.40%
β_4	lights	0.83%
β_5	max speed	6.71%
β_6	surface type	16.83%
β_7	number of traces	24.63%



(A) Average reward per episode **(B) Loss per episode**

Figure 6.11: Training results

G=147 nodes

In the previous two implementations that used G=22 and G=47, we could not see the effects of the rewards system for our DRL model since both graphs represent a tiny proportion of the city. So, we extended our graph into 147 nodes to observe the output of the training in the map. In figure 6.1.3, we observed the set of nodes that we will use in the training, and we set a starting node (id=153531392) in blue marker and red marker to present the destination node (id=5239133571). This small proportion of the map can test our model performance on a bigger scale.

Figure 6.12: OpenStreetMap view

For the weights in Table 6.1.3, we assigned a higher weight for the slope of the road (29% of the total weights) in order to observe if this will affect the path selection in larger areas and the number of traces, and to make the agent avoid the heavy traffic roads. So, we increased the road type weight to see if the agent will avoid the primary roads and take another type of road. We decided to increase the buffer size to 122KB for the hyper-parameters in table 6.1.3 because we have a larger area that will lead to many paths that the agent will explore. Another thing we increased the number of episodes to 1 million episodes and increased the batch size to 256.

However, the training took a massive time to finish (up to 9 hours) due to the large episode number and the large batch size. It did not give the anticipated results as the figure shows that the reward curve of our training kept decreasing exponentially, but as we can see, the curve slightly increased between episodes 200K and 400K. However, we can notice from the horizontal lines that the agent takes so much time to reach the terminal state, but at the end of the episode, the neural network did not perform well after we finished the exploration time. Additionally, we can see that the loss was performing well at the beginning of the training until mid of the training and started to increase massively at the peak of 1.3517703435770656e+17, which is a lot.

Table 6.7: hyper-parameters G=147

episodes	1000000
lr	0.01
gamma	0.99
target update freq	0
loss	dqn reg loss
epsilone	expodinal decay
epsilone start	1
epsilone end	0.1
Buffer size	122 KB

Table 6.8: Training Weights for 147 nodes

β_0	distance	8.42%
β_1	road type	9.08%
β_2	slop	29.38%
β_3	lane number	14.91%
β_4	lights	3.99%
β_5	max speed	1.24%
β_6	surface type	10.61%
β_7	number of traces	22.37%

Based on the previous result, we increased the episodes to 18e5 with the same hyper-parameters and weights.

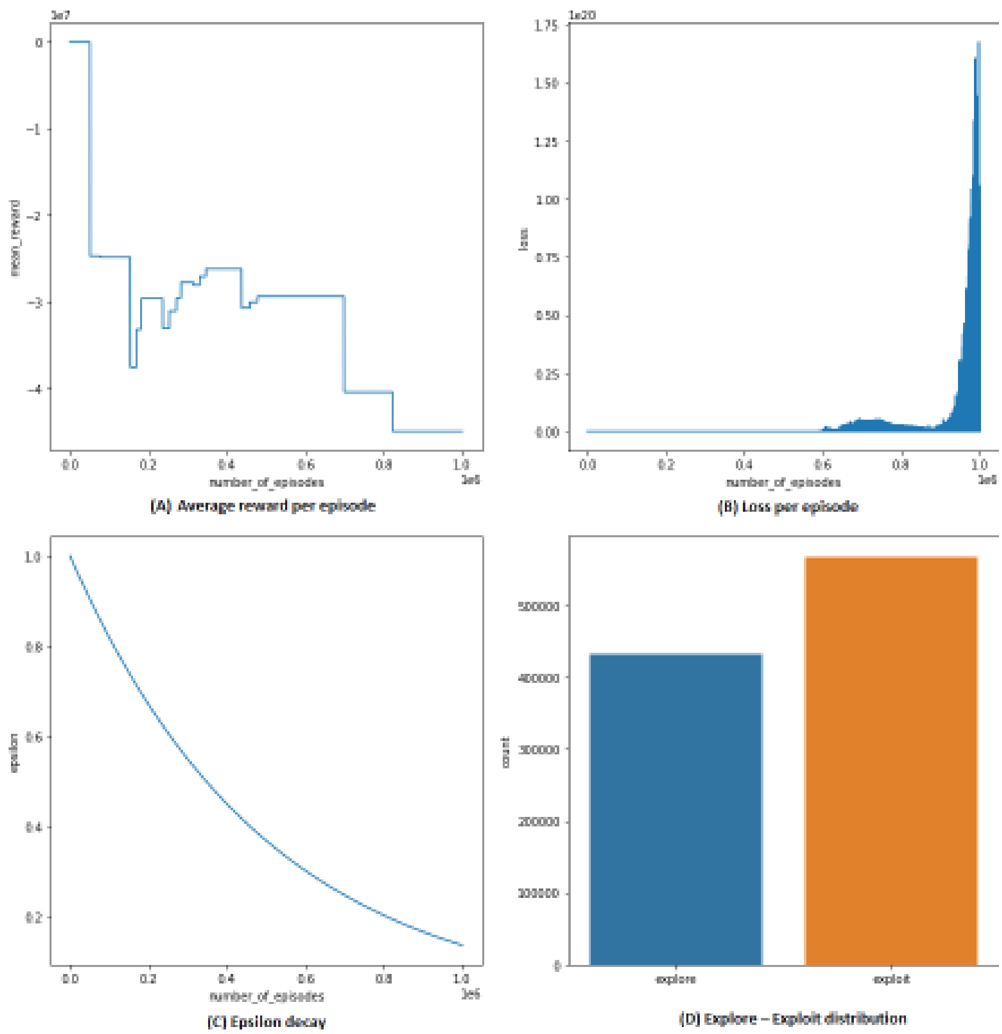


Figure 6.13: Training results

The training took 23 hours and 12 minutes. In figure 6.1.3, we can observe that the rewards kept decreasing as before, but on the other side, the loss function started to converge after 1m and 250K episodes with a mean loss for the last 100 episodes that equals to 46459, which is a massive difference between this training and the previous run. This means that if we double the time of the episode, it could help the loss to converge more.

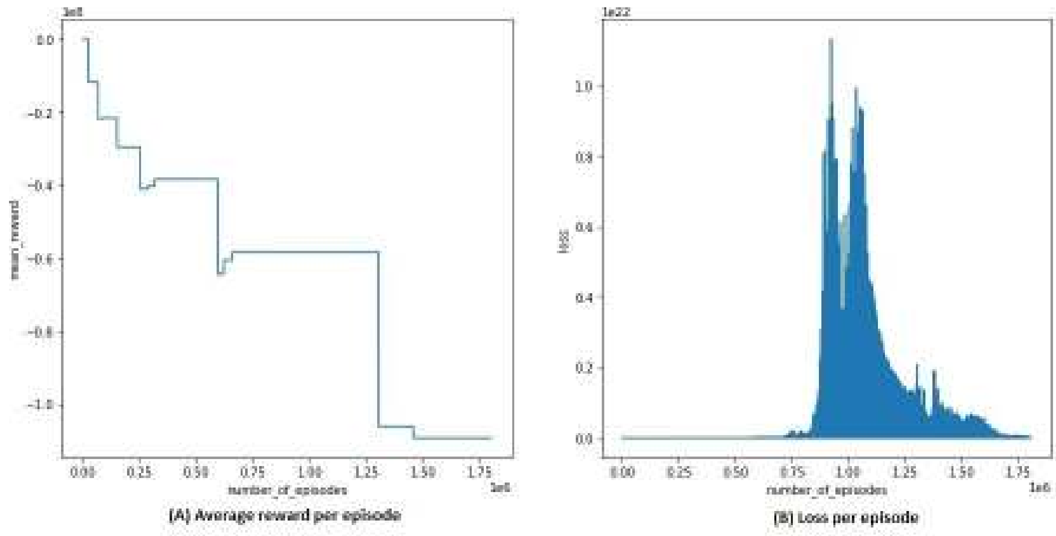


Figure 6.14: Reward and Loss results for 18e

6.1.4 Experiment 4: Comparison with A3C

In the previous implementations that used DQN, we noticed that a longer training time helps the agent find the maximum rewards for our problem based on our rewards system. However, unfortunately, DQN for a more significant number of nodes might take the agent two weeks or even more to find an optimal path, which is not efficient. We needed find a more efficient way to train our agents in a shorter time, so we decided to go with the A3C algorithm. A3C depends on the asynchronous method and uses multiple agents that interact with the same environment. While training, each agent shares their experience with the other agents, which will help solve the problem of training time. Chapter 3 discussed the algorithm, so we experienced different learning rates and the number of episodes, as shown in table 6.9. We decided to go with smaller episodes number because each agent in each episode keeps looping until it reaches the terminal state or a maximum number of steps, which is 500 steps, and we only changed the learning rate.

Table 6.9: Training on multiple hyper-parameters

Number of episodes	T max	Min reward	Max reward	Average reward	Mean loss for last 100	LR	Training time / hour
1000	10	-95875000	-9215172.982	-95043174.25	2.76162E+12	0.01	0.481161088
3000	10	-95875000	-220066.3139	-95217482.43	3.09885E+12	0.01	1.428449744
5000	10	-95875000	-95855022.95	-95874988.02	3.10685E+12	0.01	2.205047536
1000	10	-95875000	19956.20407	-83997064.09	8.41058E+12	0.001	0.441795206
3000	10	-95815117.24	19961.96181	-70488383.28	6.91288E+12	0.001	1.197373971
5000	10	-95875000	-95650531.33	-95874536.95	2.71697E+12	0.001	2.182557049
1000	10	-95875000	-90062.91339	-94208516.9	8.65978E+12	0.0001	0.465871826
3000	10	-95875000	19971.15828	-74277832.08	8.85677E+12	0.0001	1.212614794
5000	10	-95785102.63	-30038.03819	-74504542.92	7.83753E+12	0.0001	1.86164805
1000	10	-95710176.39	-45028.84172	-68718484.94	5.48005E+12	1.00E-05	0.384759826
3000	**10**	**-95755103.62**	**-110043.1704**	**-64387189.76**	**6.14386E+12**	**1.00E-05**	**1.130657925**
5000	10	-95855047.42	-30028.84172	-74148199.3	6.02149E+12	1.00E-05	1.816622731

In table 6.9, we can notice that the agent did not perform well in g=22 nodes in any of the experiments. For example, in learning rate 0.01, the average reward was -95217482, which is remarkably low, and the loss did not converge. The same thing was for learning rate 0.001 and 0.0001. The result was to use a learning rate of 0e−5, and it got an average reward of-64387189.76. We can see average reward was lower than the learning rate of 0.01, but not significant. The experience's best from the figure that the loss and the reward are not stable and increase and decrease in each episode, which might be from the way the agent calculates the loss. However, the DQN was faster in G=22 and reached positive rewards in less time. We also need to experiment more on using an asynchronous method for a value-based method, like; using asynchronous DQN for future enhancement of the model.

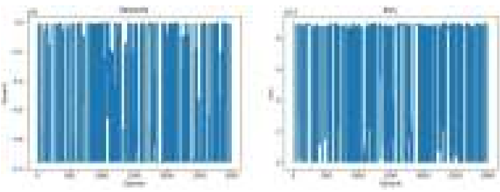


Figure 6.15: Reward and loss results for 18e

6.1.5 Experiment 5: Comparison with Dueling DQN

As a last try to get results from using 147 nodes, we decided to experiment using Dueling DQN, which acts the same as DQN but with different architecture in NN, as we mentioned in3. We used the location in the same city as we can see in 6.1.5 and we decided to choose the start and ending nodes that are far from each other to see a noticeable output.

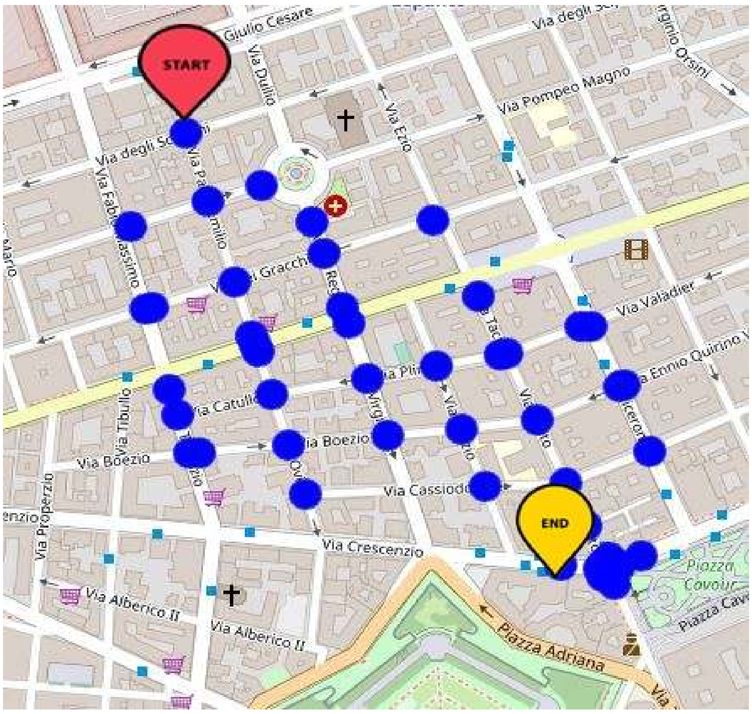


Figure 6.16: Training Area with 147 Nodes

So, in our training, we gave the slope, max speed, and surface type higher weights than the others and reduced the weight of our agent's distance. Also, we used the hyper-parameters in 6.11, like the following:

Table 6.10: Training Weights for 147 Nodes

β_0	distance	7.1%
β_1	road type	3.5%
β_2	slop	14.8%
β_3	lane number	6.2%
β_4	lights	15.3%
β_5	max speed	25.9%
β_6	surface type	18.3%
β_7	number of traces	8.9%

Table 6.11: hyper-parameters G=147

episodes	1000000
lr	0.01
gamma	0.99
target update freq	0
loss	MSE
epsilone	expodinal decay
epsilone start	1
epsilone end	0.1
Buffer size	250 KB

The algorithm took about 4 hours to end the training, and we can see from the metrics graph that it even reached a max reward of 8500 and mean reward of −5877.18444, the loss coverage was very fast, and it gave outstanding results for this large scale of nodes. The algorithm gave a path similar to the shortest path algorithm, even though we weight of 14 at the slope of the road. However, after analyzing the area, we found out that the selected area is flat with a slope of 0.0005. And as we can see from the graph in which the red dots represent the path that the agent needs to take to reach the goal destination.

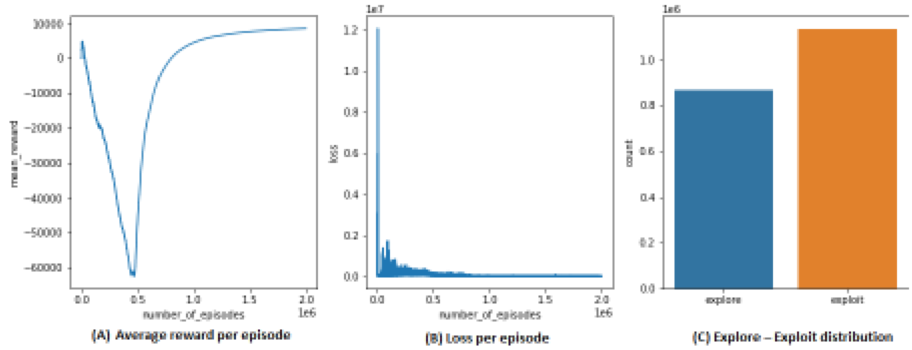


Figure 6.17: Training metrics

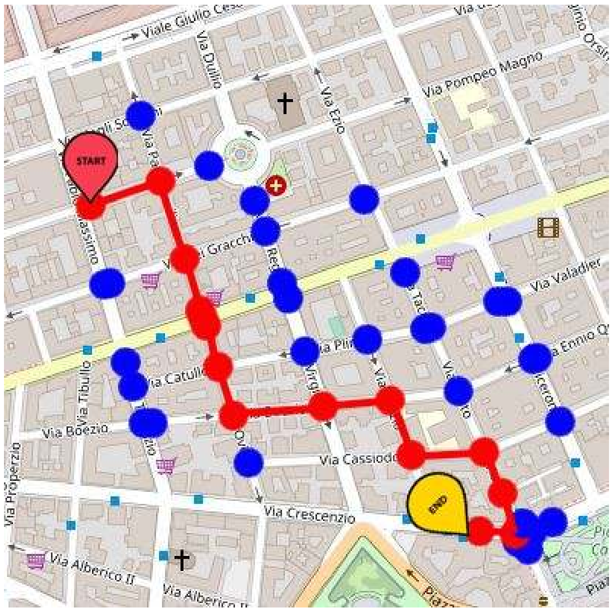


Figure 6.18: Agent path

For the sake of exploration, we wanted to see if we increased the slope of one of the roads that the agent used in the previous training, if the agent would change the road to a different one. The road we changed its slope appears in figure 6.1.5 and we used the same weights and hyperparameters in table 6.11 6.10

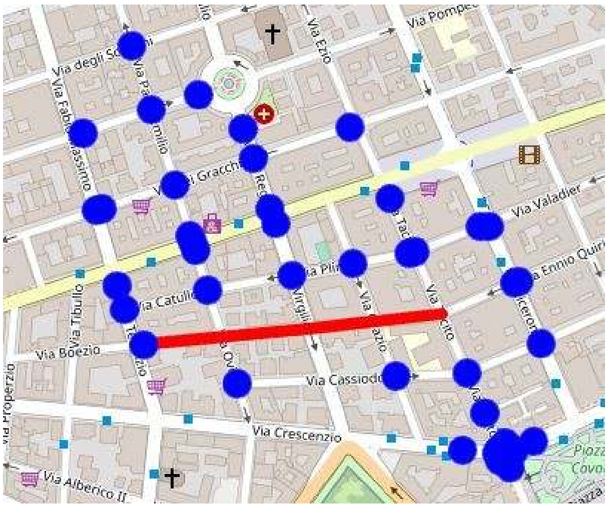


Figure 6.19: Map View

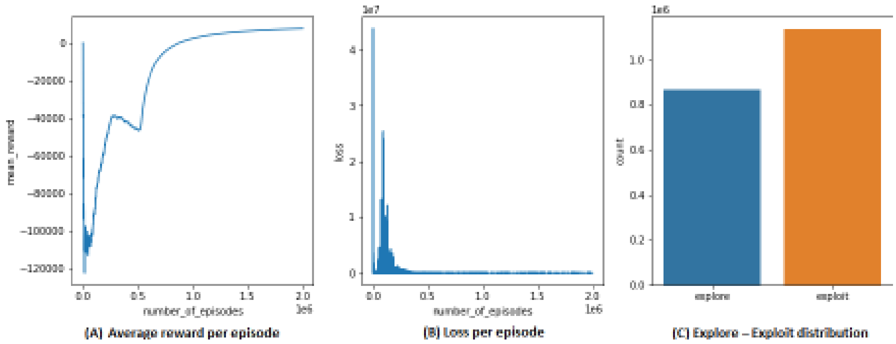


Figure 6.20: Training Metrics

The agent took 4 hours to finish the training, and we can see from figure one that the reward curve kept increasing until it reached the maximum reward of 7500, whereas, the loss was decreasing. The most important thing that the agent avoided the road was that we manually increased its slope and used another path for it 6.1.5.

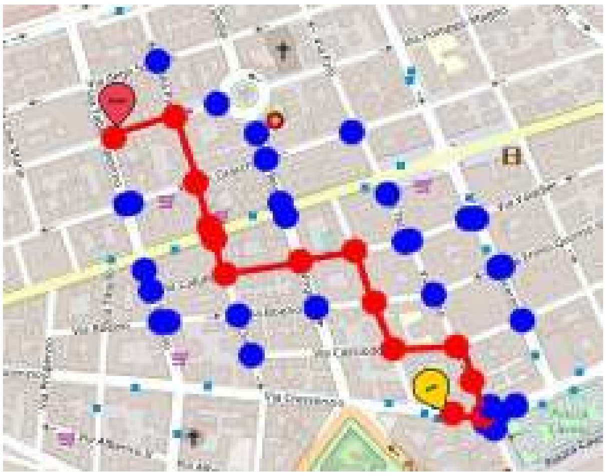


Figure 6.21: Agent Path

6.2 Conclusion

In this chapter, we experimented multiple DRL models. The first model we used is DQN which we have tested in 3 phases. In the first phase, we selected a small number of nodes G=22, and the agent gave us a Maximum reward of 4700 in 7:03 minutes, which is an excellent performance in a short amount of time. In the second phase, we increased the number of nodes to G=47, and the agent gave us great results of a maximum reward of 7450 in 20 minutes. However, in the third phase, when we increased the number of nodes to G=147, the algorithm did not perform well due to the enormous number of paths the agent needed to explore at the early stages, which will take much time and more computational power. In our experiment, we used the second model which is A3C, that uses the Actor-Critic method and multithreading techniques to find the optimal policy. Unfortunately, the algorithm did not perform sufficiently, and the agent did not give us the expected results; we achieved a maximum reward of -110043 and an average reward of -64387189. Finally, we experimented the Duel DQN that uses the same method of DQN but utilizes different neural network architecture. The algorithm performed well, as we reached a maximum reward of 7500, and the agent avoided the roads that did not meet the end-user requirements.

Chapter Seven

Conclusion and Future Work

Chapter Seven

Conclusion and Future Work

7.1 Conclusion

Deep Reinforcement Learning (DRL) is the future of Reinforcement Learning. We can use DRL to achieve a human-like performance, from playing chess to training a robot that walks in an unbounded area. However, the application of reinforcement learning in Geographical Inform in Systems (GIS) is still limited and needs further investigation.

There has been a growing interest in many countries to work on reducing the pollution caused by gas emissions from cars and many people turned into using bicycles as their main mobility. As a result, this has forced the city municipalities to plan new cycle paths for the entire city, taking into consideration all age ranges who ride bicycles; from youngsters to the elderly. For instance, a path chosen by an 18-years old person can be different from the one chosen by people in their 60s, who most likely would prefer following a path with different characteristics than the younger ones (e.g., a path that avoids any possible uphill's). All of this constituted a problem to the municipalities which drives them to not design bike lanes considering only the shortest path, but should also cover other factors like; the slope of the road, shadows, facilities on the road, the type of bicycle (e.g., E-bike), traffic.

On the other hand, the end-users who use mobile apps to find the shortest routes or traffic-free routes, do not always care about reaching their destinations faster, but need to find paths that suit their preferences. For example, the user might prefer to take a cycle lane with shadows, on a road that has cafes and public bathrooms or charging stations. The user might also want to avoid high traffic roads like roads near of schools during the drop off and pick up times, or uphill's, which might slow them down. None of the aforementioned preferences are acknowledged in any of the current routing apps.

To solve these issues, we provided a proof of concept by using a virtual agent to help the municipalities plan the most convenient cycling lanes in the city, considering multiple preferences (e.g., road slope, public bathrooms, road lights). This method allows the mobility officer to generate multiple paths between two locations based on different scenarios and even help them plan the entire city based on the municipalities' interests. On the other side, the end-users will have the option to choose a path based on their interests. Each user can choose a different path every time they use this method, based on their mood and interest; they might take the shortest path, or one with multiple restaurants or supermarkets, or a path with e-bike chargers, or even all of these preferences together.

We achieved this result by posing the following two sets of research questions:

1) **RQ1** Data related questions:
   a) Are there publicly available sources of information that contain rich geographical data?
   b) Are the publicly available data sufficient for training and testing a virtual agent that's able to compute personalized smart paths?
   c) Which are the best mechanisms to be used to integrate all the aforementioned geographical information?
2) **RQ2** Algorithmic related questions:

a) What is the possible reward modulization which encompasses all the collected rich geographical information?

b) Which is the best states representation that should be adopted when a personalized travel path problem is formulated as a reinforcement learning-based agent?

c) Which is the best Neural Network architecture that must be employed by our Agent?

d) Which is the best deep reinforcement learning algorithm that can be used in this specific application scenario?

Outcomes of the Thesis:

Hereafter we present the detailed answers that we have found for each one of the presented research questions:

- **RQ1.a**: After extensive research on different geographical data sources, we find two sources of public APIs that we can be used to extract GIS data: Overpass API from OpenStreetMap and earth engine API from google. These APIs provide sufficient GIS features, starting from roads, shops, amenities, road lights, elevation data, and even the trees in some countries. In our thesis, we mainly used the overpass API to extract the information of the roads and amenities, and we used the Earth engine API to extract the elevation data.
- **RQ1.b**: The extracted data from both APIs proved to be sufficient enough to be used in our training and testing of our DRL agent, but with some minor drawbacks. Since overpass API depends on the contributions of OpenStreetMap users to add and update the data, a lot of the key features, especially the amenity, need an update. For example, new shops have opened, and others have recently closed, or some roads

have changed some of their features, like the max speed or the number of lanes in them, so this might not give us new precise results.

- **RQ1.c**: We also proposed a mechanism to integrate all the data sources extracted into one data set in a JSON file. We constructed it by using the longitude and latitude for each road intersection from the overpass data. Then we set the keys of the JSON file based on the road intersection ID. The values are set according to the information extracted from the overpass API. The new calculated information, like the slope, is extracted from the Earth Engine's elevation data based on the longitude and latitude intersection from the Overpass API. All of this information represents the road attribute that we used later in the reward system. We believe that this method can handle not only two data sources, but multiple ones. We can use this data set in reinforcement learning projects and different applications.

- **RQ2.a**: We also constructed a reward system mechanism that covers all extracted features and gives each feature a weight. The weights can be changed based on the user's interests, and these weights will reflect the user's interest in the agent's actions. For each feature, we set an equation that calculates the reward for this specific feature, and at the end, we created a mathematical formula that combined all the rewards into one final reward value that the agent will receive. This rewards system can handle a massive number of features, but one of the drawbacks of it is that if we had to add other new features, we need to create a specific mathematical equation to get its rewards and edit the final equation to include it into the final reward.

- **RQ1.b**: Moreover, we constructed our agent environment as a graph representation based on our data set, formulated as a dictionary. Each node represents an intersection, and the information of the roads is saved on the edges between two nodes. Also, both state-space and action-space for our Agent depend on the number of nodes in the graph. The action spaces

A for our DRL equals the total number of nodes in the graph, $A = V$, Also state-space is $SP = N 2$ since the Agent can start from any V and end to any V. Finally, in our training, we decided to encode each state as vectors of zeros. Where the current state and goal state are represented in the vector as one, this method helped the neural network understand the Agent's current position and the destination of the training, which allowed the converging of the loss faster. This state representation was solid for extracting the information in the edges quickly and efficiently by the Agent. It helped us automate how we can represent the action-space and state-space in case of increasing or decreasing the nodes number.

- **RQ1.c**: Concerning our neural network architecture, we decided to use the Dueling network structure to build a network that computes the advantage A and value functions V separately. Then, we combine the two values into a single q-function at the final layer. Both the A and V need only two hidden layers of 64,32. This architecture proves its efficiency compared to the standard architecture, which computes only the q-function based on the input of the state.

- **RQ1.d**: Finally, we find out that using the Dueling DQN algorithm is the best algorithm for our personalized travel path. Due to the Neural Network architecture that combines both computing the advantage and value function to get the value of **Q-function**. This method helps us find the optimal policy for our Agent in a shorter time, unlike other algorithms and other NN architectures that have been tested before. Also, this algorithm is a value-based function that uses epsilon-greedy that combines both exploration and exploitation, which uses random action or a neural network to choose the action. In comparing DQN and A3C, the Dueling DQN performed faster, and the Agent reached an average reward earlier than both algorithms.

7.2 Current Limitations

Nevertheless, our project needs further exploration and more research, and it has some draw- backs.

- Since overpass API depends on the contributions of OpenStreetMap users in adding and up- dating the data, lots of the key features (e.g. amenity) need to be updated; for example, new shops have opened, others recently closed, or some roads have changed some of their features, like; the maximum speed or the number of lances in the road, which leads to us to no new precise results.
- While constructing the data set, some features have been generalized for the whole road and saved the same results in the edges between intersections. This problem didn't affect our training sine the areas we selected have some similarities between blocks, but when we implement it all over the city, this method will affect the training.
- Another thing we faced in our training of the model is the time it needs to finish its training. Some testing of our projects took three days to complete, and that is because of the complexity of state representation, so we have much future work to implement.

7.3 Future Works

The study provided in this thesis presents several of potential possibilities for future investigation.

- Use the amenities and shop data that we extracted to set up rewards to help the user use the road that has certain facilities that might interest him/her.
- Extract the shadow data from the API earth engine. This will add a capability of using the roads that have shadows in them and avoiding the ones that do not.
- Search for a free and reliable traffic API to support our tracer rewards feature. Explore more ways to shorten the training time in the multi-threading training by using multiple CPUs, especially in the Value-based method for DQN and Dulling DQN.
- Adding other features related to road safety like barriers, accidents and Bandits.
- Explore more algorithms that use policy-based methods, like PPO.
- Train the Agent to use every node as start and goal state. This will show the true capability of the Neural network in handling multiple paths and different states.

References

Co-Reyes, J. D., Miao, Y., Peng, D., Real, E., Levine, S., Le, Q. V., . . . Faust, A. (2021). Evolving reinforcement learning algorithms. *arXiv preprint arXiv:2101.03958*.

Divyam, R. (2017, August). Deep reinforcement learning for bipedal robots. Re- trieved from https://repository.tudelft.nl/islandora/object/uuid% 3A0fac495f-f87a-4a61-a80f-5f901323379a

Earthengine. (2021). *earthengine.* Retrieved 17-3-2021, from https://earthengine .google.com/

Geography. (2021). *Geographic database.* Retrieved 17-3-2021, from https://geography .name/geographic-database/

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. Retrieved from https://books.google.co.in/books?id=Np9SDQAAQBAJ

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, san diego, ca, usa, may 7-9, 2015, conference track proceedings.* Retrieved from http:// arxiv.org/abs/1412.6980

Lamouik, Y., & Sabri. (2018, April). Deep neural network dynamic traffic routing system for vehicles. *IEEE*. Retrieved from https://ieeexplore.ieee.org/document/ 8354012 doi: 10.1109/ISACV.2018.8354012

Liu, Y. C., Zhang, & Chen. (2019, September). A new algorithm of the best path selection based on machine learning. *IEEE*, *7*. Retrieved from https://ieeexplore.ieee.org/ document/8823853 doi: 10.1109/ACCESS.2019.2939423 medium. (2020). *graphs.* Retrieved 20-2-2020, from https://medium.com/ syncedreview/introduction-to-deep-learning-for-graphs-and-where-it-may-be-heading-75d48f42a322

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., . . . Kavukcuoglu, K. (2016, 20–22 Jun). Asynchronous methods for deep reinforcement learning. , *48*, 1928–1937. Re- trieved from https://proceedings.mlr.press/v48/mniha16.html

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M.

(2013). Playing atari with deep reinforcement learning. Retrieved from http://arxiv .org/abs/1312.5602(cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013)

Openstreetmap. (2021a). *amenity.* Retrieved 17-3-2021, from https://wiki.openstreetmap.org/wiki/Key:amenity

Openstreetmap. (2021b). *API.* Retrieved 17-3-2021, from https://wiki.openstreetmap .org/wiki/API

Openstreetmap. (2021c). *Elements.* Retrieved 17-3-2021, from https://wiki.openstreetmap.org/wiki/Elements

Openstreetmap. (2021d). *taginfo.* Retrieved 17-3-2021, from https://taginfo.openstreetmap.org/keys/highway#map

Openstreetmap. (2021e). *taginfo.* Retrieved 17-3-2021, from https://taginfo.openstreetmap.org/keys/amenity#map

OpenStreetMap contributors. (2017). *Planet dump retrieved from https://planet.osm.org.* https://www.openstreetmap.org.

Rashid, T. (2016). *Make your own neural network* (1st ed.). North Charleston, SC, USA: CreateS- pace Independent Publishing Platform.

Schulman, J. (2020, Sep). *Proximal policy optimization.* OpenAI. Retrieved from https:// openai.com/blog/openai-baselines-ppo/

Sutton, R. S., & Barto, A. G. (2018a). *Reinforcement learning: An introduction*. Cambridge, MA, USA: A Bradford Book.

Sutton, R. S., & Barto, A. G. (2018b). *Reinforcement learning: An introduction*. MIT press.

Wang, Y., Novack, & Zipf. (2020, July). Quiet route planning for pedestrians in traffic noise polluted environments. *IEEE*. Retrieved from https://ieeexplore.ieee.org/document/ 9139350 doi: 10.1109/TITS.2020.3004660

Zhiguang Cao, S. K. G. C. L. Z. X. Z., Hongliang Guo. (2020, March). Using reinforcement learning to minimize the probability of delay occurrence in transportation. *IEEE*, *69*. Retrieved from https://ieeexplore.ieee.org/document/8952783 doi: 10.1109/TVT.2020.2964784

Appendix A

Thesis Appendix

A.1 Code Organization

In this section, we will present how we organized the code for data collection, a3c algorithm, dual DQN, and DQN. Each file contains all the codes shared in the GitHub repository. Also, GitHub will be the original colab notebooks that we used as the primary reference in this thesis.

A.1.1 Data Extraction

In data extraction folder, you will find two notebooks:

- *gps tracer.ipynb*
- *data.ipynb*

gps tracer.ipynb

You only need to define the bounded box to extract all of the GPS tracers that we mentioned in 4, and the output of this notebook will be a JSON file that will be used later.

Data.ipynb

Similar to the previous notebook, you only need to define what bounded box you need to extract the data from and it will extract a JSON file that will be used in all of the algorithms.

A.1.2 DQN

In the DQN repository you will find you 6 python files:

- *Qnetwork.py* Contains the neural network.
- *create graph.py* which creates a sup graph from the data based on the number of nodes.
- *ddqn.py* Includes the training function and testing function of our agent.
- *device.py* Includes the code that is used to make the agents work on GPU.
- *environment.py* The environment of our DRL, which has the step, reset and other functions that help the agent move around the environment. It also includes the reward system.
- *main.py* Which is the main file that will run the whole code.

A.1.3 Dual DQN

In the dual DQN repository you will find 6 python files:

- *Qnetwork.py* Contains the neural network
- *create graph.py* Creates a sub graph from the data, based on thes number of nodes.
- *ddqn.py* Includes the training and testing functions for our agent.
- *device.py* Includes the code that is used to make the agents work on GPU.
- *environment.py* The environment of our DRL, which has the step, reset and other functions that help the agent move around the environment. It also includes the reward system.
- *main.py* The main file that will run the whole code.

A.1.4 A3C

In the A3c repository you will find 6 python files:

- $ActorCritics_f.py$ Includes the network,loss and return calculations.
- *Agent.py* Includes the training and testing functions for our agent.
- $SharedAdam_f.py$ Includes the shared Adam that will be used by multiple agents.
- $a3c_plot_f.py$ The plot of the final result.
- $environment_f.py$ The environment of our DRL, which has the step, reset and other functions that help the agent move around the environment. It also includes the reward system.
- *gather eps.py* The class that collects the data from the training and saves it into one dictio- nary.
- *main.py* The main file that will run the whole code.

A.2 OpenStreetMap Tags tables

In this section we will present the hole values and its description as mentioned in OpenStreetMap website for each feature that we extracted from Overpass API and used in our implementation

A.2.1 Highway values

Table A.1: Commonly used values with highway tag (OpenStreetMap,2021)

Key	Value	Comment
highway	motorway	A restricted access major divided highway, normally with 2 or more running lanes plus emergency hard shoulder. Equivalent to the Freeway, Autobahn, etc..
highway	trunk	The most important roads in a country's system aren't motorways. (Need not necessarily be a divided highway.)
highway	primary	The next most important roads in a country's system.(Often link larger towns.)
highway	secondary	The next most important roads in a country's system.(Often link towns.)
highway	tertiary	The next most important roads in a country's system.(Often link smaller towns and villages)
highway	unclassified	The least important through roads in a country's system – i.e. minor roads of a lower classification than tertiary,but which serve a purpose other than access to properties. (Often link villages and hamlets.) The word 'unclassified' is a historical artifact of the UK road system and does not mean that the classification is unknown; you can use highway=road for that.
highway	residential	Roads that serve as access to housing, without the function of connecting settlements. Often lined with housing.
highway	motorway link	The link roads (sliproads/ramps) leading to/from a motorway from/to a motorway or lower class highway.Normally with the same motorway restrictions.
highway	trunk link	The link roads (sliproads/ramps) leading to/from a trunk road from/to a trunk road or lower class highway.
highway	primary link	The link roads (sliproads/ramps) leading to/from a primary road from/to a primary road or lower class highway.
highway	Secondary link	The link roads (sliproads/ramps) leading to/from a secondary road from/to a secondary road or lower class highway.
highway	tertiary link	The link roads (sliproads/ramps) leading to/from a tertiary road from/to a tertiary road or lower class highway.
highway	living street	For living streets, which are residential streets where pedestrians have legal priority over cars, speeds are kept very low, and where children are allowed to play on the street.

highway	pedestrian	For roads used mainly/exclusively for pedestrians in shopping and some residential areas which may allow access by motorized vehicles only for very limited periods of the day.To create a ’square’ or ’plaza’ create a closed way and tag as pedestrian and also with area=yes.’
highway	track	Roads for mostly agricultural or forestry uses. To describe the quality of a track, see tracktype=*. Note: Although tracks are often rough with unpaved surfaces, this tag is not describing the quality of a road but its use. Consequently, if you want to tag a general use road, use one of the general highway values instead of track.
highway	escape	For runaway truck ramps, runaway truck lanes, emergency escape ramps, or truck arrester beds. It enables vehicles with braking failure to safely stop.

A.2.2 Amenity values

Table A.2: Commonly used values that can be used with amenity tag (OpenStreetMap,2021)

Key	Value	Comment
amenity	bar	Bar is a purpose-built commercial establishment that sells alcoholic drinks to be consumed on the premises.They are characterised by a noisy and vibrant atmosphere, similar to a party and usually don’t sell food.
amenity	bbq	BBQ or Barbecue is a permanently built grill for cooking food, which is most typically used outdoors by the public. For example these may be found in city parks or at beaches. Use the tag fuel=* to specify the source of heating, such as fuel=wood;electric;charcoal. For mapping nearby table and chairs, see also the tag tourism=picnic site. For mapping campfires and firepits, instead use the tag leisure=firepit.
amenity	cafe	Cafe is generally an informal place that offers casual meals and beverages;typically, the focus is on coffee or tea. Also known as a coffeehouse/shop, bistro or sidewalk cafe. The kind of food served may be mapped with the tags cuisine=* and diet=*. See also the tags amenity=restaurant;bar;fast food.

amenity	drinking water	Drinking water is a place where humans can obtain potable water for consumption. Typically, the water is used for only drinking. Also known as a drinking fountain or bubbler.
amenity	fast food	Fast food restaurant (see also amenity=restaurant). The kind of food served can be tagged with cuisine=* and diet=*.
amenity	food court	An area with several different restaurant food counters and a shared eating area. Commonly found in malls, airports, etc.
amenity	university	An university campus: an institute of higher education
amenity	bicycle parking	Parking for bicycles
amenity	bicycle repair station	General tools for self-service bicycle repairs, usually on the roadside; no service
amenity	bicycle rental	Rent a bicycle
amenity	bus station	May also be tagged as public transport=station.
amenity	car rental	Rent a car
amenity	car sharing	Share a car
amenity	car wash	Wash a car
amenity	vehicle inspection	Government vehicle inspection
amenity	charging station	Charging facility for electric vehicles
amenity	ferry terminal	Ferry terminal/stop. A place where people/cars/etc. can board and leave a ferry.
amenity	fuel	Petrol station; gas station; marine fuel; Streets to petrol stations are often tagged highway=service.
amenity	grit bin	A container that holds grit or a mixture of salt and grit.
amenity	motorcycle parking	Parking for motorcycles
amenity	parking	Car park. Nodes and areas (without access tag) will get a parking symbol. Areas will be coloured.Streets on car parking are often tagged highway=service and service=parking aisle.
amenity	parking entrance	An entrance or exit to an underground or multi-storey parking facility.Group multiple parking entrances together with a relation using the tags type=site and site=parking. Do not mix with amenity=parking.
amenity	parking space	A single parking space. Group multiple parking spaces together with a relation using the tags type=site and site=parking.Do not mix with amenity=parking.
amenity	taxi	A place where taxis wait for passengers.
amenity	atm	ATM or cash point: a device that provides the clients of a financial institution with access to financial transactions.

amenity	bank	Bank or credit union: a financial establishment where customers can deposit and withdraw money,take loans, make investments and transfer funds.
amenity	bureau de change	Bureau de change, money changer, currency exchange, Wechsel, cambio – a place to change foreign bank notes and travellers cheques.
amenity	baby hatch	A place where a baby can be, out of necessity, anonymously left to be safely cared for and perhaps adopted.
amenity	clinic	A medium-sized medical facility or health centre.
amenity	dentist	A dentist practice / surgery.
amenity	doctors	A doctor's practice / surgery.
amenity	hospital	A hospital providing in-patient medical treatment. Often used in conjunction with emergency=* to note whether the medical centre has emergency facilities (A&E (brit.) or ER (am.))
amenity	nursing home	Discouraged tag for a home for disabled or elderly persons who need permanent care. Use amenity=social facility + social facility=nursing home now.
amenity	pharmacy	Pharmacy: a shop where a pharmacist sells medications dispensing=yes/no - availability of prescription-only medications
amenity	social facility	A facility that provides social services: group & nursing homes, workshops for the disabled, homeless shelters, etc.
amenity	veterinary	A place where a veterinary surgeon, also known as a veterinarian or vet, practices.
amenity	arts centre	A venue where a variety of arts are performed or conducted
amenity	brothel	An establishment specifically dedicated to prostitution
amenity	lounger	An object for people to lie down.
amenity	marketplace	A marketplace where goods and services are traded daily or weekly.
amenity	monastery	Monastery is the location of a monastery or a building in which monks and nuns live.
amenity	photo booth	Photo Booth – A stand to create instant photo.
amenity	place of mourning	A room or building where families and friends can come, before the funeral, and view the body of the person who has died.
amenity	place of worship	A church, mosque, or temple, etc. Note that you also need religion=*, usually denomination=* and preferably name=* as

		well as amenity=place of worship. See the article for details.
amenity	police	A police station where police officers patrol from and that is a first point of contact for civilians
amenity	post box	A box for the reception of mail. Alternative mail-carriers can be tagged via operator=*
amenity	post depot	Post depot or delivery office, where letters and parcels are collected and sorted prior to delivery.
amenity	post office	Post office building with postal services
amenity	prison	A prison or jail where people are incarcerated before trial or after conviction
amenity	public bath	A location where the public may bathe in common, etc. Japanese on-sen, Turkish bath, hot spring
amenity	public building	A generic public building. Don't use! See office=government.
amenity	ranger station	National Park visitor headquarters: official park visitor facility with police, visitor information, permit services, etc
amenity	recycling	Recycling facilities (bottle banks, etc.). Combine with recycling type=container for containers or recycling type=centre for recycling centres.
amenity	refugee site	A human settlement sheltering refugees or internally displaced persons
amenity	sanitary dump station	A place for depositing human waste from a toilet holding tank.
amenity	shelter	heightA small shelter against bad weather conditions. To additionally describe the kind of shelter use shelter type=*.
amenity	shower	Public shower or bath.
amenity	telephone	Public telephone
amenity	toilets	Public toilets (might require a fee)
amenity	townhall	heightBuilding where the administration of a village, town or city may be located, or just a community meeting place
amenity	vending machine	A machine selling goods – food, tickets, newspapers, etc. Add type of goods using vending=*
amenity	waste basket	A single small container for depositing garbage that is easily accessible for pedestrians.
amenity	waste disposal	A medium or large disposal bin, typically for bagged up household or industrial waste.
amenity	waste transfer station	A waste transfer station is a location that accepts, consolidates and transfers waste in bulk.

amenity	watering place	Place where water is contained and animals can drink
amenity	water point	Place where you can get large amounts of drinking water
amenity	user defined	All commonly used values according to Taginfo

الملخص

استخدام نموذج التعليم المعزز العميق لتصميم بنية تحتية للتنقل المستدام خاص بالدراجات الهوائية

قد ازداد اهتمام العالم بأكمله في الآونة الأخيرة بالقضايا البيئية المتزايدة، وتعمل العديد من الدول على الحد من تأثير البشر على البيئة من خلال تبني استراتيجيات التنمية المستدامة المختلفة. إنَّ أحد الإجراءات التي يتم تعزيزها لمواجهة هذه المشكلة هو تشجيع استخدام الدراجات كوسيلة أساسية للنقل، فإذا أصبح ركوب الدراجات هو الوسيلة الأساسية للنقل، فستنشأ الحاجة إلى فتح طرق ومسارات جديدة تناسب احتياجات راكبي الدراجات. في هذه الأطروحة، سوف نتناول المشكلات ونقدم اقتراحات لحل المشاكل التي قد يواجهها راكبو الدراجات فيما يتعلق بتضاريس المدينة (على سبيل المثال، أنواع الطرق وسطح الطريق والمنحدرات).

تقدم هذه الأطروحة حلاً يحث على استخدام عامل الذكاء الاصطناعي الذي يوظف التعليم المعزز والشبكة العصبية للعثور على أفضل مسار بطريقة يتم تخصيصها حسب تفضيلات المستخدم. قدمنا أولاً عملية جمع البيانات وكيف سيتم استخدام هذه البيانات بطريقة متاحة وسهلة من قبل الوكيل. وبعد ذلك، اختبرنا العديد من خوارزميات التعليم المعزز للعثور على الطريقة الأكثر ملاءمة لاستخدامها في السيناريو الخاص بنا الذي يتسم بالتحديات. لقد قمنا أيضًا بتحويل الخريطة إلى رسم بياني يمثل بيئة التعليم المعزز العميق، وقمنا بتحويل كل ميزة إلى مكافأة فرعية في نظام المكافآت المعقد لدينا. وأخيرًا، قمنا بتدريب العديد من نماذج التعليم المعزز.

تظهر النتائج أن خوارزمية Dual Deep Q Network حققت أفضل نتيجة؛ فلقد حققنا 7500 مكافأة تراكمية في أقل من 5 ساعات من وقت التدريب، وتمكن وكيلنا من تصميم المسار بناءً على مواصفات المستخدم النهائي وتجاوز جميع الطرق التي لا تستوفي المعايير.