TÉCNICO LISBOA

Learning Control Policies in Smart Cities from Physical Data

Mykhaylo Marfeychuk

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Dr. Tiago Santos Veiga
Prof. Pedro Manuel Urbano Almeida Lima

Examination Committee

Chairperson: Prof. Maria Luísa Torres Ribeiro Marques da Silva Coheur
Supervisor: Dr. Tiago Santos Veiga
Member of the Committee: Prof. João Paulo Salgado Arriscado Costeira

November 2020

Acknowledgments

I would like to thank my partner for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible.

I would also like to thank my parents, for the opportunity to attend University and for their support throughout all these years.

I would also like to acknowledge my dissertation supervisors Prof. Pedro Lima and Dr. Tiago Veiga for their insight, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.

To each and every one of you – Thank you.

Abstract

Motor vehicle emissions are the primary contributors to the increase in ambient pollution levels. Rapid urbanization and lack of a good solution to manage the traffic are forcing cities to take drastic measures against the automotive industry. In this thesis, we build a case study of the Norwegian city of Trondheim's traffic, and create a realist simulation based on real world data, that simulates the traffic and the emissions. We then propose a Reinforcement Learning based solution that controls the access to the different regions of the city to optimize the traffic given a desired metric. We also take a look at different improvements, like using a multi-agent system and using pre-generated data for the training phase. We compare the obtained results with the baseline and against a reactive agent. At the end we assess the solution's strengths and weaknesses, and propose possible future improvements.

Keywords

Deep Learning; Reinforcement Learning; Multi-Agent System;

Resumo

As emissões dos veículos a motor são a principal contribuição para o aumento dos níveis da poluição ambiente. A rápida urbanização e a falta de uma boa solução para gerir o trânsito está a forçar as cidades a tirar medidas drásticas contra a indústria automóvel. Nesta dissertação, é feito um estudo de caso do trânsito em Trondheim e é criada uma simulação realista, a partir de dados do mundo real, que simula o trânsito e as emissões. Seguidamente, é proposta uma solução baseada em Aprendizagem por Reforço que controla o acesso a diferentes regiões da cidade por forma a optimizar o trânsito, dada uma métrica desejada. Também são vistos diferentes aperfeiçoamentos, como a utilização de um sistema multi-agente e utilização de dados pré-gerados para a fase de treino. Os resultados obtidos são comparados com o cenário base e contra um agente reactivo. No final, avaliam-se os pontos fortes e fracos da solução, e propõem-se possíveis futuras melhorias.

Palavras Chave

Aprendizagem Profunda; Aprendizagem por Reforço; Sistema Multi-agente;

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor Critic
AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application Programming Interface
CHs	Hydrocarbons
CNN	Convolutional Neural Network
CO	Carbon Monoxide
CO2	Carbon Dioxide
CSV	Comma Separated Values
DQN	Deep Q-Networks
EEA	European Economic Area
EFTA	European Free Trade Association
GA	Genetic Algorithm
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HBEFA	Handbook Emission Factors for Road Transport
ISR	Institute for Systems and Robotics
LSTM	Long Short-Term Memory
MDP	Markov Decision Process
NEAT	NeuroEvolution of Augmenting Topologies
NILU	Norwegian Institute for Air Research

NOx	Nitrogen Oxide
OSM	OpenStreetMap
PM	Particulate Matter
PPO	Proximal Policy Optimization
ReLU	Rectified Linear Unit
REST	Representational State Transfer
RL	Reinforcement Learning
SARSA	State-Action-Reward-State-Action
SUMO	Simulation of Urban Mobility
SWIG	Simplified Wrapper and Interface Generator
TCP	Transmission Control Protocol
TD	Temporal Difference
TraCI	Traffic Control Interface
XML	Extensible Markup Language

1

Introduction

Contents

1.1 Motivation

Motor vehicle emissions contribute to ambient pollution levels with carcinogen toxins. Exposure to these toxins can also cause non-cancer health effects, such as neurological, cardiovascular, respiratory, reproductive and/or immune system damage. Many of today's traffic lights and road accesses, are controlled by a timer or a predefined pattern. This solution may have a lot of overhead in situations where the traffic doesn't follow the pattern. This in turn has a direct correlation to the pollution increase within the city.

The rapid urbanization and traffic-related pollution are forcing cities to take drastic actions against the motor industry. An example of this is Norway, where between 2009 and 2012 the air quality in a number of Norwegian cities and densely populated areas was not in line with the European Economic Area (EEA) rules. Due to this, the case was brought to the European Free Trade Association (EFTA) court, which forced Norway to find solutions to limit the number of pollutant particles below a certain threshold. Norway is not alone in having problems with the limit values. Air pollution is still a widespread problem across the EEA, particularly in big cities, where emissions from vehicles are a major contributor to poor air quality.

Recently there have been successful traffic control solutions that use statistical analysis to find a more refined pattern of the traffic flow. Using these analyses it is possible to find more efficient algorithms that control the traffic signals and city access. The main drawback of the statistical solutions is that they have trouble acting on edge cases, and generally have trouble generalizing when the complexity of the system increases.

Reinforcement Learning (RL) is a method of learning an optimal set of actions that maximize an accumulated discounted reward, by simply giving positive and negative feedback. In reinforcement learning, an agent applies an optimal action given an environment state. In some cases, the handcrafted state features may not be enough to fully model the environment, thus a more sophisticated approach like Deep Reinforcement Learning is used, which, when given a feature set, learns the necessary correlations and reduces necessary feature space to a minimum. Deep Reinforcement Learning is based on the same principle as RL but uses Deep Neural Networks to estimate the state-action values. With this method it is possible to model the traffic pattern with all its intricacies without explicitly specifying any predefined behaviour or the relation between features. The necessary criteria are the environment state and the behavioral reward. By using RL, it is possible to develop better traffic control solutions that perform better at reducing the pollution than existing solutions, while still maintaining the same amount of traffic.

The AI4EU consortium was established to build the first European Artificial Intelligence On-Demand Platform and Ecosystem. The platform will leverage industry-led pilots to demonstrate its capabilities, and the resources and innovation produced by these pilot projects will enable real applications and foster more innovation that will support the European AI community in the long run. The Institute for Systems

and Robotics (ISR) collaborates with one of the pilot projects, with the goal of implementing solutions to better citizens lives by using Artificial Intelligence (AI) to improve pollution monitoring, prediction and support decision-making in the city of Trondheim.

Trondheim was one of the cities with highest pollution level in Norway, before the implementation of stricter regulation, but still to this day the city struggles with high pollution levels. Due to this and the abundance of traffic and pollution data, the city was chosen as the main subject of the thesis case study.

1.2 Project Goals

As described by the pilot project, the goal is to develop a solution using Reinforcement Learning to improve the citizens lives by reducing the pollution caused by traffic. It is also relevant to study the traffic flow to better understand the primary culprits of the increased pollution levels. The understanding of the traffic is necessary for providing a better traffic control, with the objective of optimizing for lower levels of pollution.

The goal is to implement a solution that reduces the pollution in the area by optimizing the traffic flow, using Deep Learning and Reinforcement Learning based algorithms. The solution, given the readings from the air pollution sensors and the traffic density, needs to reduce the dimensionality of the feature set, i.e. use only relevant features, and determine the current environment state. In many cases the sensors data is plagued with noise and uncertainty, so the solution needs to compensate for that, and it is also required to be able to work as desired with as little information as possible, as it is hard to collect perfect knowledge from the real world. With the states, a Deep Neural Network needs to learn an optimal policy, on how to control the traffic. The policy dictates which actions to be executed, in this case, by opening and closing access to certain roads. The objective is to change the traffic flow in such a way that it lowers the pollution levels caused by the traffic, without drastically impacting the traffic flow.

The implementation is going to be trained and tested on a simulation based on real traffic behaviour in the city of Trondheim in Norway. One of the reason for selecting Trondheim for the simulation, is that this is the home of the AI4EU partner in charge of this pilot, namely the company Telenor.

This project will support the existing research, possibly challenging the obtained results, bring new insights to the topic and possibly new approaches on how to improve the city traffic and improve the pollution problem.

2

Background

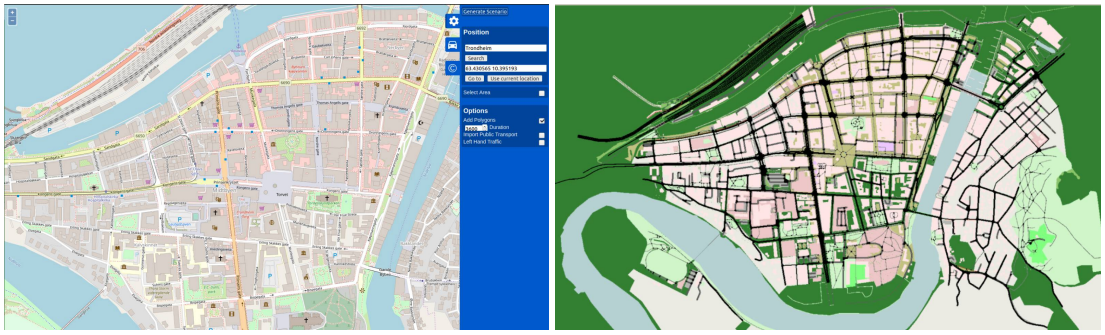
Contents

2.1 SUMO: Multimodal Traffic Simulator

Simulation of Urban Mobility (SUMO) [1] is an open source continuous road traffic simulation package designed to handle large road networks. It is mainly developed by employees of the Institute of Transportation Systems at the German Aerospace Center. The SUMO simulator was primarily developed to support the traffic research community. By providing a high-level API for the simulation, SUMO lets the researchers focus on the implementation of the research topic. SUMO simulator was primarily designed with two major goals, to be fast and portable. As such, the simulator was developed into multiple components, where not all are required to run the simulation. As an example, the visualization component is not required. This approach allows for the simulator to be run without having to instantiate the simulators interface. This allows to run automatically multiple parallel simulations.

SUMO's key features are the Microscopic simulation, where the vehicles, pedestrians and public transport are explicitly modeled, the Cross-Platform support and the supported import formats. Additionally SUMO provides multiple configurable emission models, which can be assigned to the simulated vehicles. This tool is extremely relevant for this project.

A tool called OpenStreetMap (OSM) Web Wizard is provided, that based on the selection of an OSM map excerpt automatically generates a SUMO simulation scenario. This tool drastically speeds up the scenario generation.



(a) OSM Web Wizard **(b)** SUMO Simulation Scenario

Figure 2.1: SUMO simulation scenario generated by OSM Web Wizard tool

SUMO provides a Traffic Control Interface (TraCI), which allows to retrieve values of simulated objects and to manipulate their behaviour. TraCI uses a Transmission Control Protocol (TCP) based client/server architecture to provide access, and supports multiple simultaneous clients. To have a more efficient coupling without the need for socket communication, the TraCI Application Programming Interface (API) is provided as a C++ library, Libsumo. Libsumo provides pre-built language bindings for Java and Python, and provides support for other programming languages via the Simplified Wrapper and Interface Generator (SWIG) software tool.

2.2 Deep Learning

Neural Networks [2] are a machine learning tool used to model an $f(x)$ function to map x to y. This approach is based on Connectionism Theory [3] that tries to explain mental phenomena using Artificial Neural Networks (ANNs). A Neural Network consists of a large number of neuron units joined together in a connectivity pattern. Units in an ANN are usually segregated into three classes: input, hidden, and output.

Neural Networks are a collection of connected nodes called artificial neurons. Each neuron receives an arbitrary number of weighted inputs, performs a mathematical operation on the aggregated sum of the inputs, and passes the result to the next set of neurons. Some commonly performed operations are the sigmoid function or the Rectified Linear Unit (ReLU) function, these act as a threshold, similar to biological neurons. Typically these neurons are structured into layers, each layer connecting to the next, and so on, as represented in Figure 2.2. The input layer, receives its inputs from an external source, and is connected to a hidden layer which, in turn, is connected to the output layer. When there are multiple hidden layers, these networks are called Deep Neural Networks. As mentioned, all the connections have a respective weight, and these weights are the key feature that make these neural networks perform so well. The success of the Neural Networks is due to the network being able to model non-linear functions.

Passing data from the input layer through the network, up to the output layer, is called forward propagation. For the network to output the desired values, it needs to be trained. Training the network, means changing the weights, for this the gradient descent optimization algorithm is used. The error between the given output and the expected output is computed, and the weights for the output neurons are updated using the gradient descent. This process is made sequentially for all the network's neurons. This process is called Backpropagation.

Another approach of training a Neural Network, is to use the NeuroEvolution of Augmenting Topologies (NEAT) [4] method. NEAT is a Genetic Algorithm (GA) for the generation of evolving artificial neural networks developed by Ken Stanley in 2002. The algorithm generates a batch of Neural Networks, and based on their performance, selects the best networks. Small variations to these networks are made, which is called Mutation. These mutated networks represent the next batch. Over time, the networks overfit on a semi-optimal solution. Usually this approach is slower and more resource intensive than the backpropagation approach. The advantage of using NEAT, is that it allows to find a close to optimal topology of the Neural Network.

2.2.1 Convolutional Neural Network

Another relevant variation of Neural Networks, are the CNNs [5], these are optimized to process input structures like images or maps. These networks have two special types of layers, Convolutional Layers

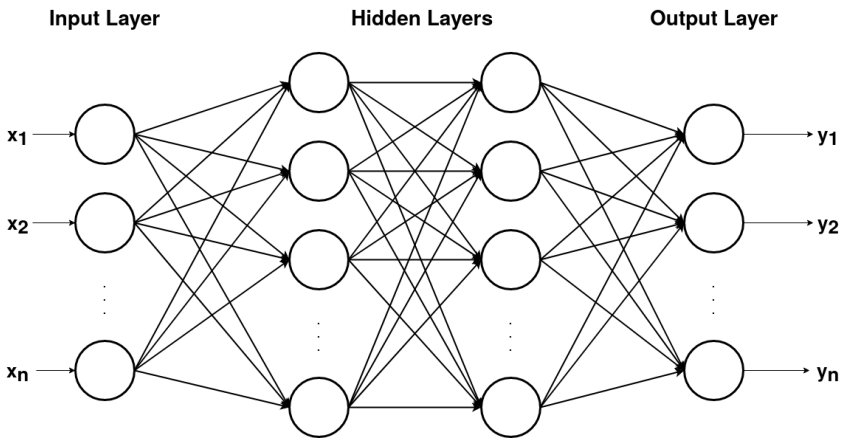


Figure 2.2: Neural Network Representation

and Pooling Layers. Convolutional Layers perform a sliding window algorithm with what is called a filter. These filters are a matrix of numbers, and represent functions like average, Sobel or Gaussian. The Pooling Layers are used to down sample the number of inputs. A Pooling Layer is positioned after each Convolutional Layer. The output of the last layer is then flattened into a one dimensional vector and connected to a fully connected Neural Network. A general flow of a Convolutional Neural Network (CNN) is represented in Figure 2.3.

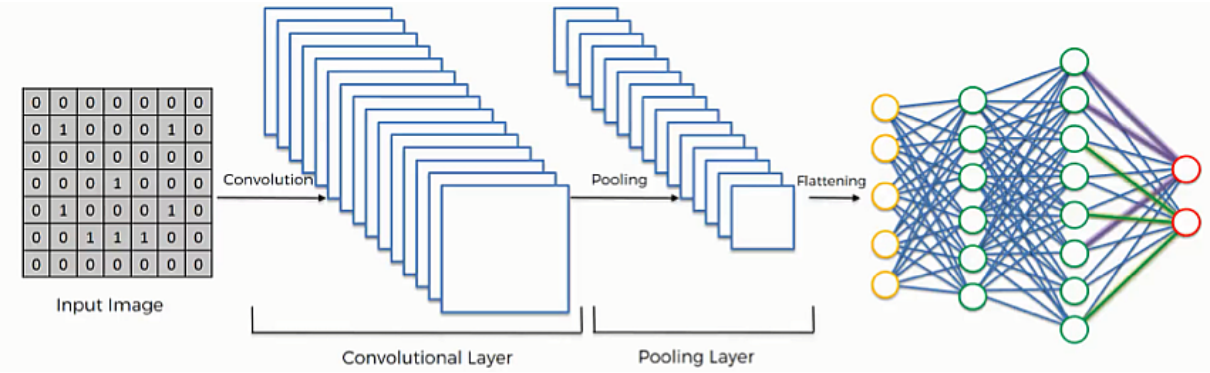


Figure 2.3: Convolutional Neural Network model [6]

In some cases different data types may be added alongside the image to provide more relevant information. For this a mixed-data neural network is used. This model is built by creating a separate neural network for each data type and then, taking the output from the input branches, combining them into a combined neural network, as shown in Figure 2.4. It is also important to make sure that the input information is normalized the same way, and a similar activation function is used throughout the network.

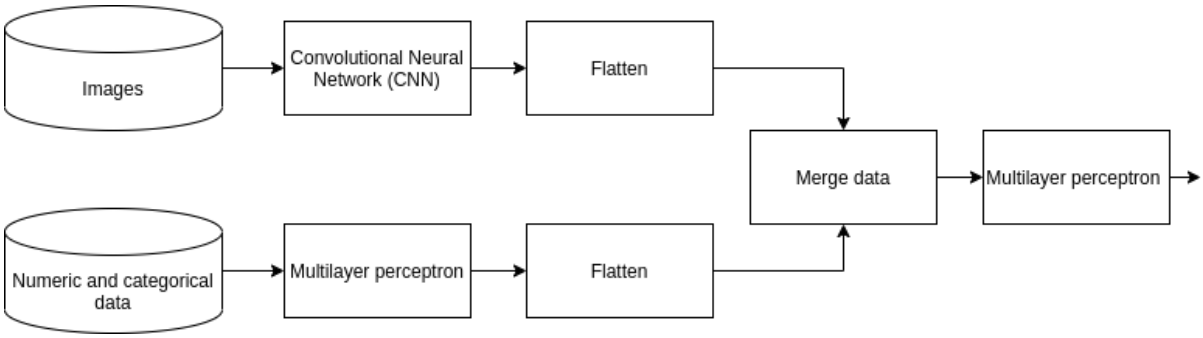


Figure 2.4: Mixed Neural Network model

2.3 Markov Decision Process

A Markov Decision Process (MDP) [7] is a framework for modeling decision making in situations where the outcomes of actions are uncertain. MDPs model the interaction between an agent and the environment, as represented in Figure 2.5.

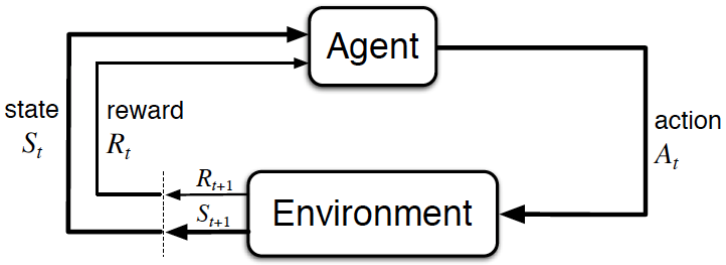


Figure 2.5: MDP Framework [8]

A **Markov Decision Process (MDP)** model can be characterized by:

- **States** S**:** The set of all the possible states the environment could be in.
- **Actions** A**:** The set of actions that the agent can execute.
- **Transition probability** $P(s_{t+1} = s' | s_t = s, a_t = a)$**:** A success probability matrix of moving from one state to another, by executing a certain action.
- **Reward** R_t**:** A positive reward received for executing the action that takes the agent from one state to the next.

The Markov Decision Process' solution is the optimal policy (or an approximation for complex models) that maximizes the expected performance criteria, for the specific model.
In a state, taking into account the transition probability and a reward function the agent computes the

best action to take. The action can maximize the immediate reward or a long-term reward, based on the Performance Criteria to Optimize that is used.

In the case where the agent knows all the states, the transition probability matrix and the reward function, the problem is considered to be model-based. With those elements the policy, $\pi(s)$, can be computed even before ever taking any action. There are two model-based algorithms for solving an MDP, policy iteration and value iteration.

The Policy iteration [9] algorithm starts with a random policy, then it calculates the value function of that policy using (2.1). Then the algorithm calculates an improved policy based on the previous value function using (2.2). The algorithm repeats this process until the optimal policy is reached. It is guaranteed that each policy is a strict improvement over the previous one, unless it is already optimal.

$$V^{\pi}(s) = R_t + \gamma \sum_{s'} V^{\pi}(s') P(s'|s,a) \tag{2.1}$$

$$\pi(s) = \arg\max_{a} \left[R_t + \gamma \sum_{s'} V^{\pi}(s') P(s'|s,a) \right] \tag{2.2}$$

where:

$V^{\pi}(s)$	=	value function
$\pi(s)$	=	policy
R_t	=	reward obtained from moving from state s to s'
γ	=	discount factor

An algorithm for Policy Iteration is:

Algorithm 2.1 Policy Iteration

function POLICYITERATION(π_0)
  $t \leftarrow 0$
  repeat
    Compute V^{π_t}
    $\pi_{t+1}(s) \leftarrow \arg\max_{a} \left[R_t + \gamma \sum_{s'} V^{\pi_t}(s') P(s'|s,a) \right]$ for all states s
    $t \leftarrow t + 1$
  until $\pi_t = \pi_{t-1}$
return π_t

The value iteration [7] algorithm, starts with a random value function. This function is then improved by an iterative process, using (2.3) until the optimal value function is achieved. After the optimal value function is achieved, the policy is easily calculated using (2.4)

$$V^{*}(s) = \max_{a} \left[R_t + \gamma \sum_{s'} V^{*}(s') P(s'|s,a) \right] \tag{2.3}$$

$$\pi(s) = \arg\max_a [R_t + \gamma \sum_{s'} V^*(s') P(s'|s,a)] \tag{2.4}$$

where:

$V^*(s)$ = value function of optimal policy
$\pi(s)$ = policy
R_t = reward obtained from moving from state s to s'
γ = discount factor

An algorithm for Value Iteration is:

Algorithm 2.2 Value Iteration

function VALUEITERATION
  $t \leftarrow 0$
  $V_0(s) \leftarrow 0$ for all states s
  repeat
    $V_{t+1}(s) \leftarrow \max_a \left[R_t + \gamma \sum_{s'} V_t(s') P(s'|s,a) \right]$ for all states s
    $t \leftarrow t + 1$
  until convergence
return V_t

2.4 Reinforcement Learning

There are situations where the model of the world isn't known, the methods used in these situations are known as model-free methods. For this, a Reinforcement Learning algorithm is used to solve the MDP, where the agent tries to approximate a policy by interacting with the environment. This is useful when the agent needs to be used in previously unseen environment, but it can also lead to the Cliff Walking problem, where the agent is near a cliff and decides to take the action of walking off it which, in most cases, is an undesirable action.

The goal of the agent is to maximize its expected total, future, reward. It does this by adding the maximum reward attainable from future states to the reward for achieving its current state, effectively influencing the current action by the potential future reward. This potential reward is a weighted sum of the expected values of the rewards of all future steps starting from the current state. Off-policy learner learns the value of the optimal policy independently of the agent's actions. Q-Learning [10] is a greedy algorithm, which means that it is going to try to maximize it's cumulative reward, even if there is a high negative risk to the agent.

$$\underbrace{Q(s,a)}_{\text{new value}} \leftarrow (1-\alpha)\cdot \underbrace{Q(s,a)}_{\text{old value}} + \alpha \cdot \overbrace{\left[R_t + \gamma \cdot \underbrace{\max_a Q(s',a')}_{\substack{\text{estimate of optimal} \\ \text{future value}}}\right]}^{\text{learned value}} \tag{2.5}$$

where:

$Q(s,a)$	$=$	quality of a state-action combination
α	$=$	learning rate
R_t	$=$	reward obtained from moving from state s to s'
γ	$=$	discount factor

An algorithm for Q-learning is:

Algorithm 2.3 Q-learning

function QLEARNING
  $t \leftarrow 0$
  $s_0 \leftarrow$ initial state
  Initialize Q
  while Q is not converged **do**
    Choose action a_t from s_t based on Q and the chosen exploration strategy
    Execute action, observe new state, s_{t+1}, and reward, R_t
    $Q(s,a) \leftarrow (1-\alpha)\cdot Q(s,a) + \alpha \cdot \left[R_t + \gamma \cdot \max_a Q(s',a')\right]$
    $t \leftarrow t+1$
  return Q

State-Action-Reward-State-Action (SARSA) [11] works on the same principle as the Q-Learning algorithm, but SARSA is more conservative. It uses the actual action taken to update Q instead of maximizing over all possible actions as done in Q-learning. If there is a risk of getting a large negative reward close to the optimal path, Q-learning will tend to trigger that reward whilst exploring, whilst SARSA will tend to avoid a dangerous optimal path and only slowly learn to use it, as shown in Figure 2.6.

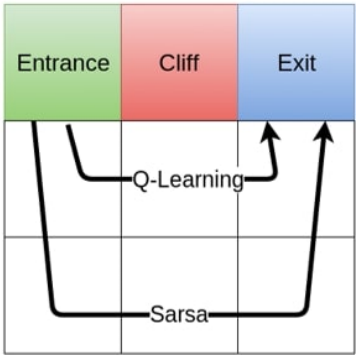


Figure 2.6: Cliff Walking problem [12]

SARSA is also more exploratory than Q-Learning which means, in the long run, it can find more

efficient and safe routes for the agent to take.

$$Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R_t + \gamma \cdot Q(s',a')] \tag{2.6}$$

where:
$Q(s,a)$ = quality of a state-action combination
α = learning rate
R_t = reward obtained from moving from state s to s'
γ = discount factor

An algorithm for SARSA is:

Algorithm 2.4 SARSA

function SARSA
  $t \leftarrow 0$
  $s_0 \leftarrow$ initial state
  Initialize Q
  Choose action a_t from s_t based on Q and the chosen exploration strategy
  while Q is not converged **do**
    Execute action, observe new state, s_{t+1}, and reward, R_t
    Choose action a_{t+1} from s_{t+1} based on Q and the chosen exploration strategy
    $Q(s,a) \leftarrow (1-\alpha) \cdot Q(s,a) + \alpha \cdot [R_t + \gamma \cdot Q(s',a')]$
    $t \leftarrow t+1$
  return Q

2.4.1 Deep Q-Learning

Although Q-learning is a very powerful algorithm, its main weakness is lack of generality. The problem is, that when there is a state that is not in the Q-learning table, the algorithm will have no clue which action to take, and in many applications it is impractical to represent every possible state. To solve this, DeepMind proposed a solution [13] to this problem by replacing the state-action matrix by a Neural Network, to estimate the Q-value function. This Neural Network, given an input state, produces the Q-value, meaning, the network acts as the policy. To train a Neural Network, desired outputs are required. These are generated by a simplified version of the Q-learning update method, represented in (2.7). The difference between the given and the desired outputs is calculated by using (2.8), which is then used by the Backpropagation algorithm to update the network's weights.

$$y_j \leftarrow R_j + \gamma \max_{a'} \hat{Q}(s'_j,\ a') \tag{2.7}$$

$$\Delta y_j \leftarrow (Q(s_j,\ a)\ -\ y_j))^2 \tag{2.8}$$

where:

y_j = vector of the expected state-action value for transition j
$\hat{Q}(s'_j, a')$ = target state-action value for transition j
$Q(s_j, a)$ = main state-action value for transition j
R_j = reward obtained from moving from state s_j to s'_j
γ = discount factor

When training, the network tends to prioritize the experience from the latest examples. This lowers the ability of the network to generalize, as the experience from older examples is overwritten with the latest experiences. To solve this, an Experience Replay technique is used, where a stack of examples is saved. During the training, a random batch of these examples is selected and used to train along with the latest examples. This has proven to improve the generalization while lowering the number of training cycles.

An algorithm for Deep Q-Learning with Experience Replay is:

Algorithm 2.5 Deep Q-learning with Experience Replay

procedure DEEP_Q-LEARNING(S, A, R, γ, ϵ)
  Initialize replay memory $\mathcal{D}$ to capacity N
  Initialize main network Q
  Initialize target network $\hat{Q}$
  while not converged **do**
    Select a random action a with probability ϵ
    Select $a = \max_a Q(s,\ a)$ with probability $1 - \epsilon$
    perform a, observe reward R and next state s'
    store experience $(s,\ a,\ R,\ s')$ in experience replay memory D
    if enough experiences are stored in $\mathcal{D}$ **then**
      sample random minibatch of N transitions from D
      for every transition $(s_j,\ a_j,\ R_j,\ s'_j)$ in minibatch **do**
        if transition j is terminal **then**
          $y_j = R_j$
        else
          $y_j = R_j + \gamma \text{max}_{a'} \hat{Q}(s'_j,\ a')$
        Compute the mean squared error loss $\mathcal{L} = \frac{1}{N} \sum_{j=0}^{N-1} (Q(s_j,\ a_j) - y_j)^2$
        Update Q using the stochastic gradient descent algorithm by minimizing the loss $\mathcal{L}$
        Every C steps, copy weights from Q to $\hat{Q}$

2.4.2 Advantage Actor-Critic

Although Deep Q-Learning achieved huge success in higher dimensional problems, the action space is still discrete. However, in many applications, especially physical control tasks, the action space is continuous. If the action space is discretized too finely, it becomes massive. Thus, a new approach was introduced, Actor-Critic, which aims to merge both value-based and policy-based methods.

Advantage Actor-Critic (A2C) [14] is an algorithm that uses the actor-critic architecture, represented in Figure 2.7, where the main idea is to split the model in two: one for computing an action based on

a state and another one to produce the Q values of the action. The actor takes as input the state and outputs the best action. It essentially learns the optimal policy (policy-based). The critic, on the other hand, evaluates the action by computing the value function (value-based).

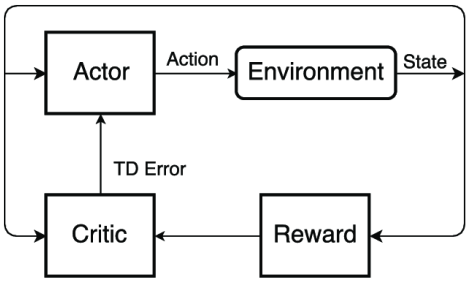
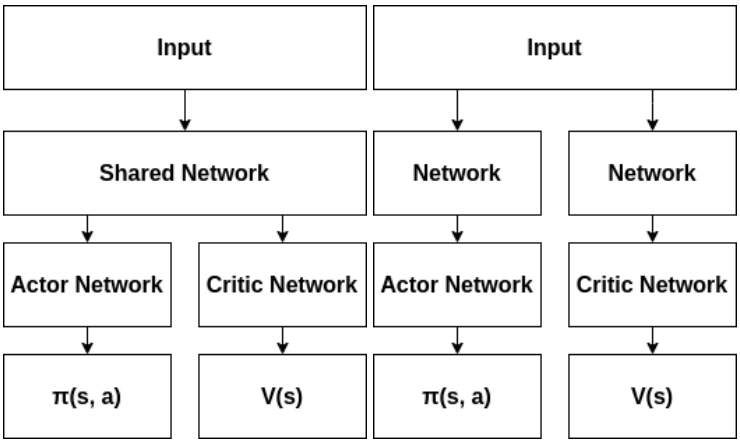


Figure 2.7: Actor-Critic Architecture

There are two variations for the actor-critic architecture, to be seen in Figure 2.8, which affect the way the models get trained. The first variation is where the networks share a common network layer. This helps the models to learn similar initial patterns. Another approach is to separate the models and train them independently. It is important to notice that the update of the weights happens at each step (Temporal Difference (TD) Learning) and not at the end of the episode, as opposed to policy gradients, and it is independent of the variation of the architecture.



(a) Shared networks **(b)** Separate networks

Figure 2.8: Variations of actor-critic approach

A2C expands upon the architecture, by introducing the advantage value, which the critic learns instead of the Q values. The advantage value is derived from the Q value, as shown in (2.9), and it helps to reduce the high variance of policy networks and stabilize the model.

$$Q(s,a) = V(s) + A(s,a) => A(s,a) = Q(s,a) - V(s) => A(s,a) = r + \gamma V(s') - V(s) \tag{2.9}$$

3

Related Work

Contents

Traffic control optimization has been an active field of research since the time when cars started to be introduced massively into our lives, and implementations using AI, especially Reinforcement Learning, have been an active field of research for the last 20 years. Due to the limitations of computational power, however, effective solutions couldn't be implemented in the early days. However, in the recent years, with the big technological advancements and improvements in learning algorithms created a new spark for this field research, and the exponential increase of vehicles on the road increases the need of better solutions.

In this chapter we'll take a look at existing solutions to the problem. We'll also take a look at their implementations, results and limitations.

All the insights gathered from the published works are relevant for this project. This project will improve upon the published work, and challenge the solutions and the results.

3.1 Reinforcement Learning Based Traffic Control

Lin et al proposed [15] Deep RL based approach to optimize the traffic flow by better controlling the traffic lights. The proposed approach is to act on the intersections which are a superset of four traffic lights. Each intersection has four possible phases which have a specific order of transition, which represent the traffic direction and subsequently the traffic light states, as shown in the Figure 3.1.

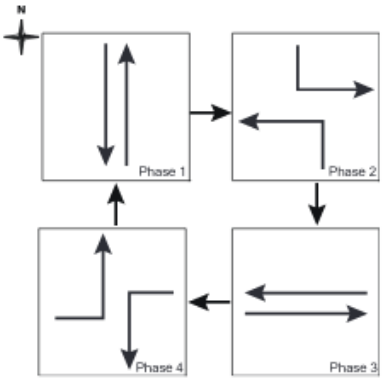


Figure 3.1: 4 phases of traffic light, **Straight(NS,SN)**, **TurnLeft(NE,SW)**, **Straight(WE,EW)**, **TurnLeft(WN,ES)** in each intersection [15]

The approach uses an A2C based deep learning model. The model receives as input a 3D tensor, as shown in 3.2, that represents the entire traffic grid. The model's output is a simplified discrete action space, in a shape of $< N_{tls}, 2 >$, where 2 indicates the number of the discrete probabilities of choices, either maintaining or switching to the next phase for each intersection, and N_{tls} is the number of traffic lights.

In the paper, the agent's reward was divided into two parts. One part is a global reward that leads

the agent to optimize a global behaviour of the whole network. For this, the net outflow of the whole network was used. The net outflow is calculated by subtracting the input vehicles $||Veh_t^{(in)}||$ from the output vehicles $||Veh_t^{(out)}||$ within the selected area at each time step t (3.1). When a congestion or collision occurs, the simulator often teleports the vehicles to the position where the vehicle should have been. Because of this only the vehicles that weren't teleported are counted for the net outflow.

$$r_t^{Global} = ||Veh_t^{(out)}|| - ||Veh_t^{(in)}|| \tag{3.1}$$

The other part is a local reward, that helps the agent learn a local behaviour relative to the intersecting. It is defined as the absolute negative difference between queue length (3.2).

$$r_t^{TLS_i} = -|\max q_t^{WE} - \max q_t^{NS}| \tag{3.2}$$

For each intersection TLS_i, q_t^{WE} is the number of halting vehicle in lanes from west to east or vice versa.

The complete hybrid reward function is formed by summing both the global and local rewards (3.3).

$$r_t = \beta r_t^{Global} + (1-\beta)\frac{1}{N_{TLS}}\sum_{i}^{N_{TLS}} r_t^{TLS_i} \tag{3.3}$$

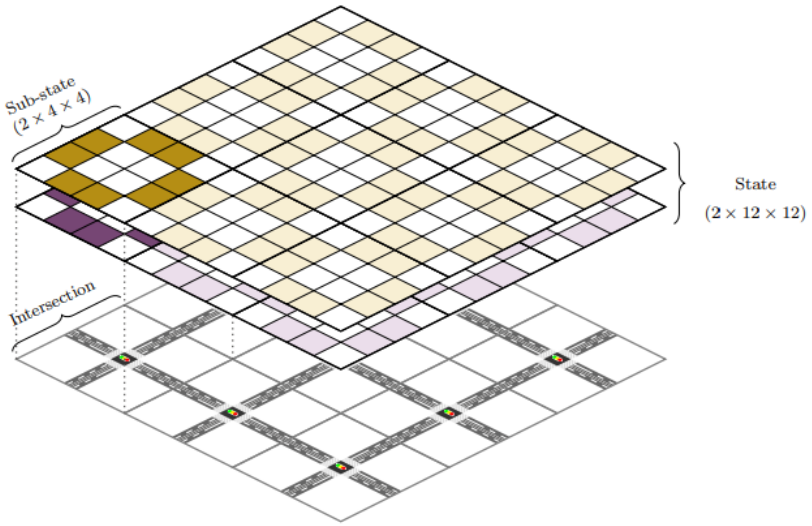


Figure 3.2: The traffic grid and corresponding formatted tensor [15]

The solution was trained on randomly generated traffic. Multiple simultaneous simulations were ran during each episode, with each simulation having its own actor to maximize sampling.

The agent was able to increase the throughput by 25.19% and 37.81% compared with fixed-time

and vehicle-actuated controllers, while the average waiting time reduces by 18.68% and 28.54% at the same time. The problem that was encountered that when the traffic system becomes over-saturated, the performance gets close to the fixed-time controller.

The team faced exactly the same challenge as this thesis, scalability. The proposed model has $< N_{tls}, 2 >$ outputs. With each added new traffic light the number of possible combinations grows exponentially. Running simulations on a city wide simulations would have billions of combinations. The model should increase in size, as the proposed model wouldn't be able to generalize, which in turn would require more training data. The team had great success while testing on randomly generated simulations scenarios. An interesting study would to test the solution on real world based simulations.

3.2 Multi-agent Based Road Traffic Control

Arel et al [16] introduced a control system where the model-based reinforcement learning approach is utilized to optimize traffic signal in a network aiming at minimizing travel time.

The team proposes a multi-agent approach, where in a five-intersection system, Figure 3.3, each intersection has an agent deciding the traffic light state. There are two types of agents where the only difference is the type of information they receive. The outbound intersection agents only have access to the local traffic statistics, while the central intersection agent has access to all states of its neighbouring intersections. Each intersection follow a two-ring structure. Each ring has four phases giving a total of eight combinational actions. In a multi-intersection network, each intersection agent individually selects an action among the available eight-phase combinations.

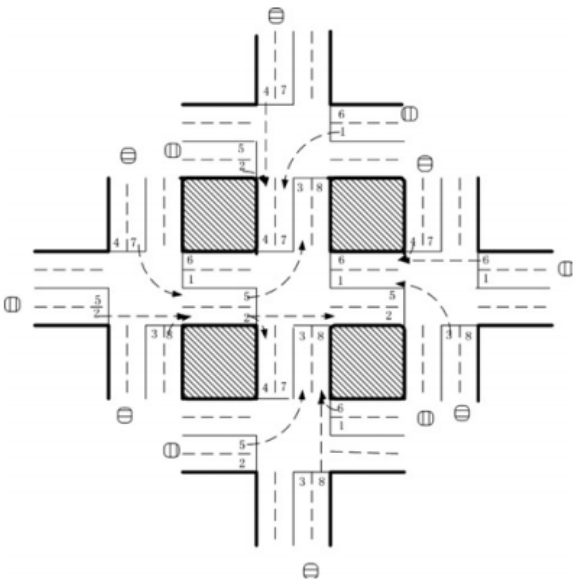


Figure 3.3: Five-intersection, centrally connected vehicular traffic network

The outer intersection agents employ the longest-queue-first algorithm, while the inner intersection agent uses the RL based approach. When arrival rates are low, the longest-queue-first scheduling algorithm performs slightly better than the multi-agent Q Learning system.

For the reward, the traffic delay difference between the current and previous timesteps was used, this may be positive or negative (3.4). D_{new} and $D_{current}$ are the previous and current intersection total delays.

$$r = \frac{D_{last} - D_{current}}{\max[D_{last}, D_{current}]} \tag{3.4}$$

The authors state that the current solution is scalable, although the paper only shows a single example with only one RL agent. It would be interesting to see how the solution behaves on a city wide simulation. Also the method is less flexible as the disturbances (e.g., lane changing) are ignored.

Alegre [17] implemented a multi-agent approach using RLlib and SUMO to control Traffic Lights. In the approach, each traffic light has it's own independent agent. Each agent receives the states of the neighboring lanes, which includes the amount of cars, density, waiting time, and the traffic lights current state. At every few steps the agents have to decide to which phase they should change the Traffic Light.

Given the flexibility of the RLlib library, he also created a variation where only a single agent is used to control all the traffic lights. RLlib having many RL algorithms already implemented, allows easily to easily change the used algorithm with minimal changes.

Being this project implemented in SUMO and RLlib, the project will acts as an example on how to create a RL SUMO wrapper. It can't act as a starting point, as the goal of our implementation is to be modular.

Chu et al [18] propose a Multi-agent RL scalable approach for the adaptive traffic control problem.

The proposed approach uses a synchronous multi-agent A2C algorithm to train the agents, where each intersection has an agent which decides the red-green combinations of the traffic lights. The agent receives the *wait* which measures the cumulative delay of the first vehicle, *wave* which measures the total number of approaching vehicles, and the neighbor policies [19]. The deep learning model also contains an Long Short-Term Memory (LSTM) layer, as shown in Figure 3.4, which stores information that is relevant for learning long-term dependencies.

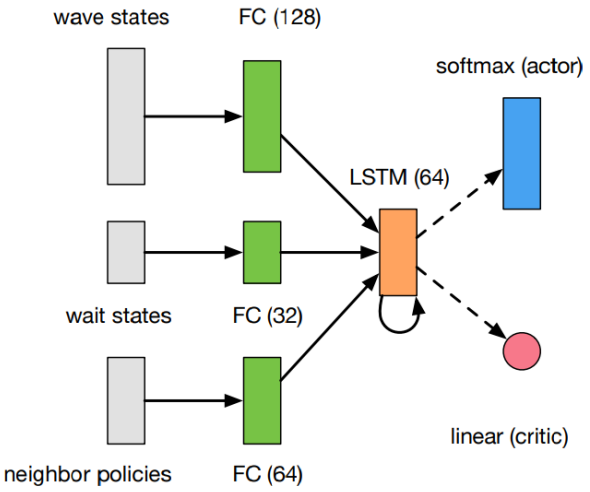


Figure 3.4: Proposed Neural Network model [18]

For the reward the authors propose to optimize for the *queue* which measures the queue length along each incoming lane, the *wait* and *a* which is the trade-off coefficient (3.5). This reward only contains a local reward, and not a global reward like other proposed approaches.

$$r_{t,i} = - \sum_{ji \in E, l \in L_{ji}} (queue_{t+\Delta t}[l] + a \cdot wait_{t+\Delta t}[l]) \tag{3.5}$$

The approach was compared to a Deep Q-Networks (DQN) [20] based and single A2C agent based approaches. The DQN wasn't able to converge, and the multi-agent approach converged faster than a single agent. To test the scalability of the solution, the mentioned approaches were tested on a Monaco based simulation, where the multi-agent outperformed the other two approaches.

4

Use Case: Traffic and Pollution in the City of Trondheim

Contents

SUMO provides many necessary features, such as traffic flow simulation, traffic lights control and vehicle emissions, which are necessary for this case study.

A major drawback of the SUMO simulator is the city wide pollution simulation. The simulation only provides instantaneous emissions of the vehicle, thus, extra work is required to simulate the pollution propagation throughout the simulated environment.

To guarantee the credibility of the simulation, real world sampled data is used to tune the simulation for it to be as close as possible to the real world. The general overview of the simulation creation is shown on Figure 4.1.

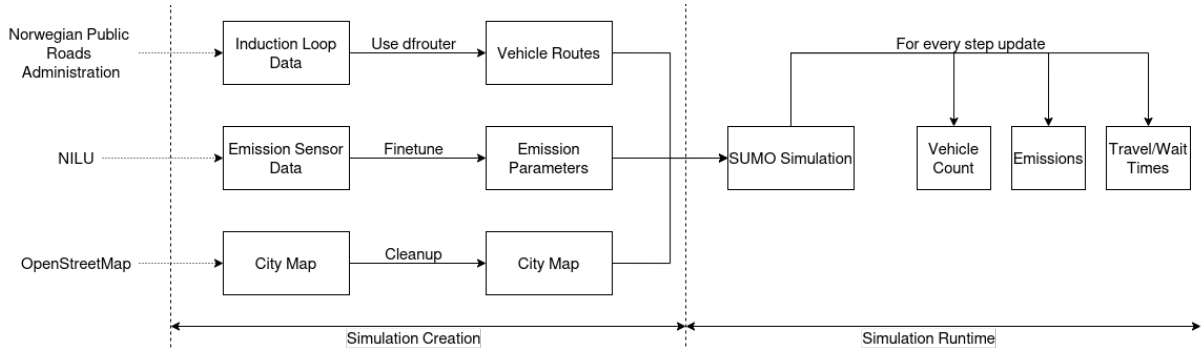


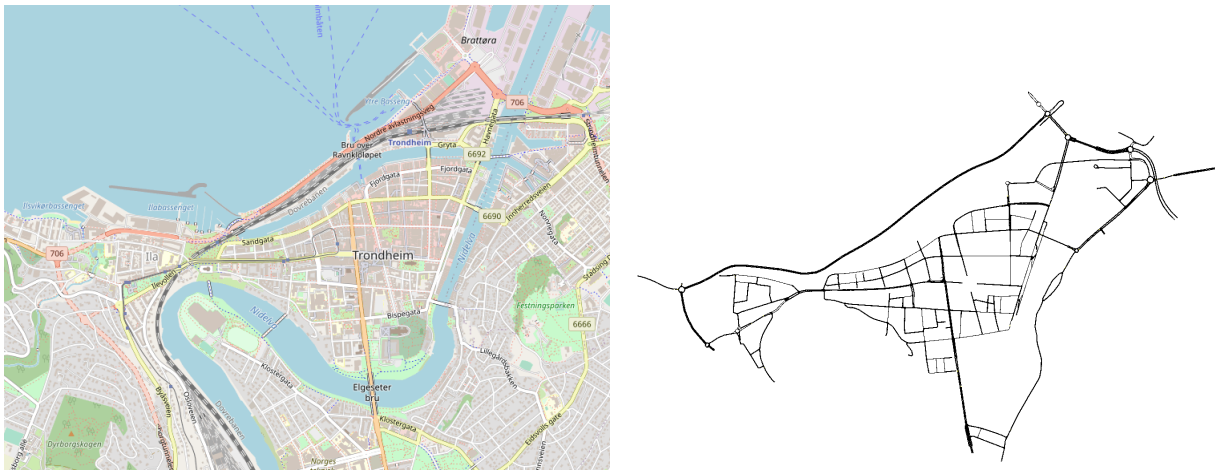
Figure 4.1: Environment Flow Overview

4.1 Trondheim Map

OSM is an open source collaborative project to create a free editable map of the world. The project tracks the topology of the environment, geocoding of addresses, place names, and route planning. OSM also provides a Representational State Transfer (REST) API that can be used to programmatically download the desired information. SUMO provides a variety of tools to process OSM maps, although the functionality is limited only to the topology. OSM Web Wizard is the tool that was used to create the maps, as it provides an intuitive Graphical User Interface (GUI) to select the desired area and its conditions, like imported information, and pre-generated traffic.

Multiple, differently sized maps were created to test the capabilities of the simulator and for rapid prototyping. The larger the map the more computationally expensive it is to run the simulation. The final map size, shown in Figure 4.2, was created based on the primary routes, and the available sensor information. Increasing the size beyond this point will only add unnecessary complexity and introduce more error to the simulation.

SUMO also provides various components, like footpaths, waterway, roads, house and garden decorations, etc. By default the tools used to generate the map, also include many unnecessary components.



(a) OSM Web Wizard view

(b) Generated SUMO Simulation Map

Figure 4.2: Comparison between the real map and the generated SUMO map

For this work they were removed from the map for simulation purposes, as these increase the simulation's complexity without adding any benefit.

The cleaned up map consists of lanes, edges, and intersections. An edge corresponds to a road, which is composed of multiple lanes. Furthermore, multiple edges are used to create roundabouts. By default the generated map has connection issues between the edges, as such the vehicles can perform illegal moves. To solve this, many of the edges needed to be curated manually.

A grid system was used on the map to track the environment state and simplify the interactability with the environment components. The map is split into 17×16 cells, where each cell is $200\ m^2$ and is composed of multiple edges. Furthermore the cell tracks the number of vehicles present in it at any current simulation step, as well as the pollution values, and the travel and waiting times. Another state of the cell is related to the action that may be performed on the cell. A cell may be open or closed, which propagates the state to the edges. If a cell is closed, the vehicles are prohibited to travel on the edges that lay inside of the cell.

4.2 Traffic Modelling

Poor traffic management and traffic jams in a city are two of the primary contributors to bad air quality. The set goal for this project is to study the behaviour of the traffic and implement a solution that improves the traffic flow. Ideally the experiments would be conducted in the real world, to guarantee the fidelity of the results, but unfortunately this is not possible, so a simulation will be used. The simulated traffic should match the real world traffic as much as possible to guarantee the trustworthiness of the experiments. To do this, the generated traffic flow is done based on real world samples.

4.2.1 Sensors

An inductive-loop traffic detector is a detection system which uses the principle of electromagnetic induction to detect or measure objects. It is widely used in cities to detect passing vehicles or to control traffic signals at an intersection of roads. An induction loop is usually embedded in the road and the circuit connected to this loop can detect changes in its inductance when a vehicle passes over or stops on the loop. This type of system can be used to detect vehicles as well as to tune the timing of traffic signals.

The Norwegian Public Roads Administration[1] hosts a variety of induction-loop detectors located in relevant entry points of the city of Trondheim, as can be seen in Figure 4.3. The sample rate is per hour basis and counts how many vehicle passed through the sensors, and logs the estimated size of the vehicle. The size distinction is relevant, as differently sized vehicles represent different emission categories. The relevant information included in the dataset is:

- Timestamp of the sampled hour
- Location of the sensor
- Road name
- Direction of the sensor road
- Number of vehicles measured to be shorter than 5.6 m.
- Number of vehicles measured to be greater than or equal to 5.6 m

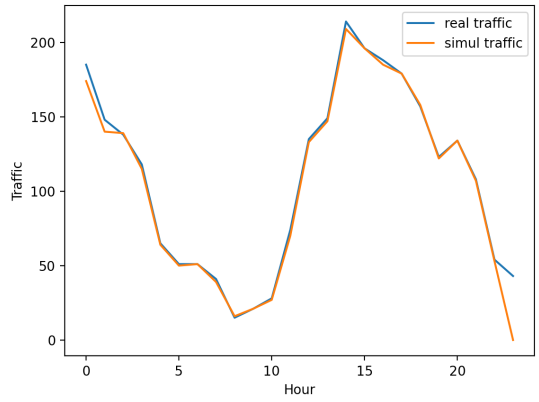
Figure 4.3: Trondheim induction loop locations (blue spots on the image)

[1] Norwegian Public Roads Administration website, https://www.vegvesen.no/

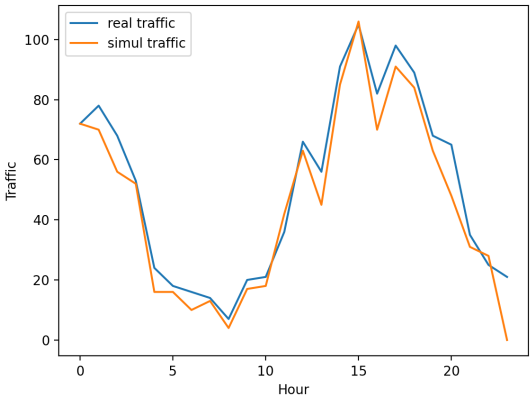
An alternate option would be to use the data from the OSM. OSM is a community driven reporting project, thus the data is questionable and may be unreliable.

4.2.2 Real World based Traffic

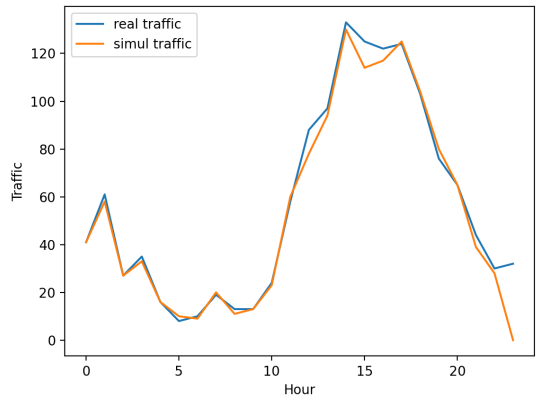
By having the number of vehicles passing through each induction loop, and the relative location of the induction loop in the simulated environment, we can use tools such as DFrouter to generate the traffic. DFrouter is a tool that tries to create traffic where the simulated induction-loop vehicle count matches the desired vehicle count. Shown in Figure 4.4 is a comparison between the real vehicle count and the simulation. The variation in some cases is due to small anomalies and modifications made in the generated map.



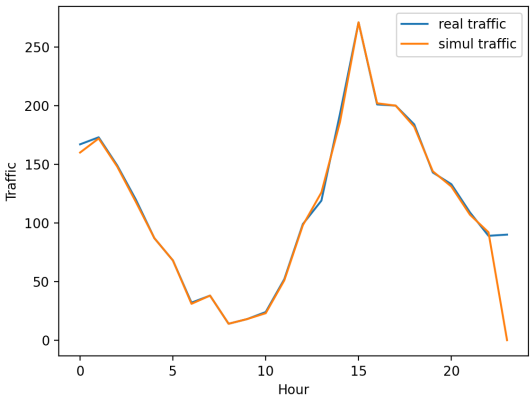
(a) Søndre Ilevollen Sensor, Sentrum Direction



(b) Brattørbrua Sensor, Kjøpmannsgata Direction



(c) Nye Ilsvikøra Sensor, Flakk Direction



(d) Søndre Ilevollen Sensor, Byåsen Direction

Figure 4.4: Example comparisons between real traffic and simulated

4.2.3 Dynamic Traffic

The vehicle routes are originally set up based on the real world data. When the environment state changes, for example when a road is closed, the vehicles need to react to the change and generate a different route to guarantee the arrival at the destination.

By default the dynamic routing is not enabled, and the simulator manually adds and removes vehicles to maintain the specified induction-loop count. When enabled, at each simulation step all the vehicles check if all the edges in their route are reachable, if not then a new route is calculated. By default SUMO uses A* to deduce a new route instead of Dijkstra, as it is often faster. A* uses the metric in (4.1) for bounding travel time to direct the search.

$$h(x) = \frac{euclidean\ distance}{maximum\ vehicle\ speed} \tag{4.1}$$

In some instances there may be a synchronization issue where two vehicles may try to enter the same junction at the same time. This causes a grid lock, which is solved only if one of the vehicles gives way to the other. By default the SUMO simulator removes these vehicles to ensure that the original induction loop counts are met. This functionality was disabled as it doesn't represent real world behaviour, and the behaviour may be exploited.

During a grid-lock, the traffic jam may extend onto the roads where the vehicles are initially created. When this happens, new vehicles are not able to be introduced into the simulation.

4.3 Pollution Modelling

In many areas, vehicle emissions have become the dominant source of air pollutants, including Carbon Monoxide (CO), Carbon Dioxide (CO2), Hydrocarbons (CHs), Nitrogen Oxide (NOx), and Particulate Matter (PM). Many of the particles are affected heavily by the external environment sources, and some don't have any data available, so the primary focus of the simulation will be to study and simulate the NOx particles.

The simulator provides an accurate vehicle emissions values for the particles previously mentioned, but these are instantaneous emissions, so an extra step is added to the simulation pipeline which tries to simulate the emission propagation throughout the environment based on real world samples.

4.3.1 Real World data

The Norwegian Institute for Air Research (NILU)[2] is an independent, nonprofit institution that focuses on researching and making society aware of the causes and consequences of climate change and pol-

[2]Norwegian Institute for Air Research website, https://www.nilu.com/

lution. NILU hosts a variety of air quality sensors located throughout Norway, including three located in Trondheim, as can be seen in Figure 4.5. The amount of sensors is not enough to create an accurate simulated model of the pollution, as the pollution is not only affected by the traffic, but also by many environmental factors, but they can be used to roughly approximate the propagation of emissions throughout the environment. NILU provides the sensor coordinates, Air Quality index and direct sensor samples through a REST API.

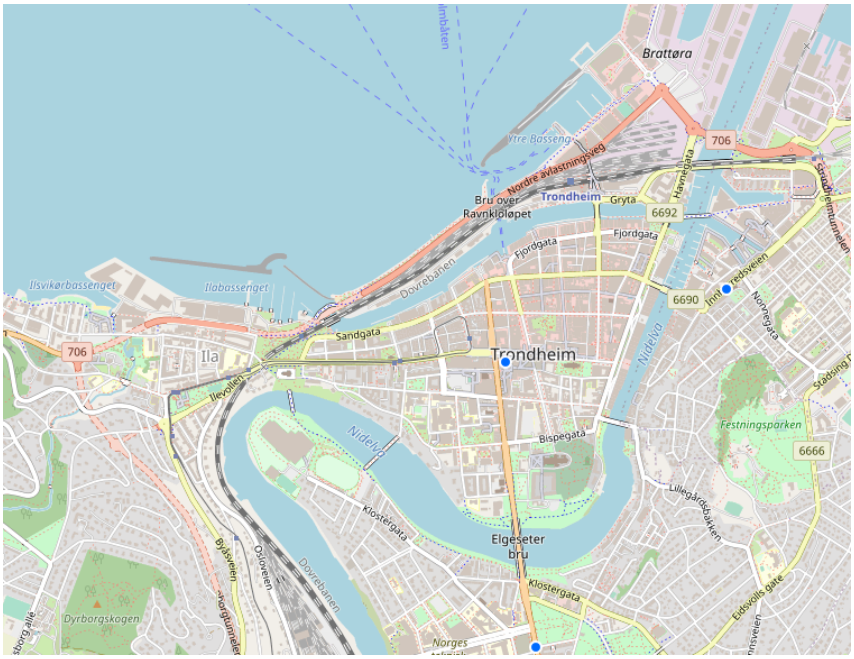


Figure 4.5: Trondheim Air Quality Sensor locations (blue spots on the image)

4.3.2 Vehicle emissions model

Each vehicle emits different pollutants which vary in type and quantity. These emissions depend on the model of the vehicle, type, fuel, wear down, and travel speed. As such, a vehicle emissions model which is able to fit different vehicle variations is required.

SUMO simulator provides various open source and commercial models, as shown in Table 4.1. The Handbook Emission Factors for Road Transport (HBEFA) [21] v3.1 model is used, as it is the latest more accurate open source model available.

Table 4.1: Measurements covered by different emission models

model	CO	CO_2	CHs	NO_x	PM	fuel consumption	electricity consumption
HBEFA v2.1-based	x	x	x	x	x	x	-
HBEFA v3.1-based	x	x	x	x	x	x	-
PHEMlight	x	x	x	x	x	x	-
Electric Vehicle Model	-	-	-	-	-	-	x

4.3.3 Modelling

The SUMO simulator only provides instantaneous emissions for the vehicle and the road, meaning that the values are only available for the last step of the simulation. Because of this, we need to implement our own emission tracker and simulation. To model the environment, we'll use the same methodology of splitting the map into cells, where each cell represents a section of the map, but in this case it tracks the multiple pollutants present in each section. At each time step the vehicle contributes to the emissions of the cell it's located in, by concatenating the vehicle pollution values to the current cell value.

To make the emission behaviour more realistic, two extra features are used, pollution propagation and decay. At each step the pollution values are reduced by a certain amount, to simulate the particle settling down and decaying. Because the real world is a dynamic environment where the particles aren't bound to a specific area, a Gaussian Blur is also applied to simulate the propagation through the neighboring cells. So at each step for each cell, the pollution values are multiplied by the decay value and the Equation (4.2) is applied where the σ corresponds to the dissipation. Ideally, the weather and wind dynamics would be needed to make a more realistic emission simulation but this goes beyond the scope of this thesis.

$$G(x, y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.2}$$

The per pollutant decay and dissipation values can be seen in Table 4.2. These values were selected by performing an iterative search with the goal of minimizing the error between the real world emissions and the simulation emissions.

Table 4.2: Per-emissions type decay and dissipation values

	CO	**CO_2**	**CHs**	**NO_x**	**PM**
Decay	0.9999	0.8	0.999914	0.996	0.995
Dissipation	0.3	0.4	0.3	0.3	0.3

4.4 Environment State Tracking

The SUMO simulator only handles the traffic simulation and provides TraCI API to interact with the simulation. However this is only the most basic functionality, so it doesn't have the functionality that is needed. To solve this, an environment was created that wraps the SUMO simulation and adds extra functionality. The built environment follows a modular methodology, where a core system exists, and optional modules can be provided to augment and provide more functionality. This was primarily done because different experiments require different requirements, and adding more complexity may increase the simulation time and provide functionality that may not be needed for the specific use case. The core structure and modules implemented can be seen in Figure 4.6.

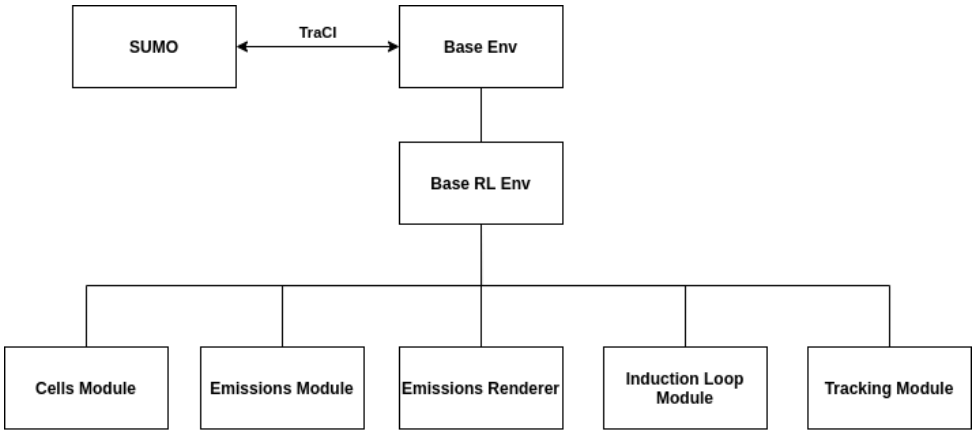


Figure 4.6: Modular Framework Structure

4.4.1 Core

The core of the environment is called *BaseEnv*, and it is responsible for handling the SUMO simulation and optional modules. *BaseEnv* receives the simulation map, start/stop time steps, update frequency, logging directory, and additional modules. When a *BaseEnv* is created, it initializes and runs a sumo simulation. Depending on the parameters, the SUMO GUI version may be used, which provides a graphical interface of the simulation. At every simulation step, all of the module's update functions are called. It is the responsibility of each module to implement the desired functionality. *BaseEnv* also exposes a reset function, which when called, resets the simulation and the modules.

The goal of the project is to create an RL based agent that acts upon the environment. The used RL library requires a gym environment which needs to provide certain functions and parameters, so that the library can interact with the environment. Because of this, another *Class* was created *BaseRLEnv* which wraps the *BaseEnv* and implements the *Gym.Env* interface. The *BaseRLEnv* is also a core Class, meaning that for each experiment, a custom environment needs to be created that implements the *BaseRLEnv* abstract methods.

4.4.2 Modules

As previously mentioned the project work behaves in a modular manner, where each module implements a desired functionality and should be only used when needed. During the simulation environment creation, a list of modules are passed as one of the parameters. These modules must implement specific methods. The environment takes care of calling the module base functions, but it is the module's responsibility to have the desired functionality implemented. The modules implemented for the project are the following:

4.4.2.A Cells Module

Cells module creates the cell matrix of the map, where each cell corresponds to each region of the map. The module keeps track of certain information like, all the roads that belong to the cell, the number of vehicles, the travel and waiting time, and the current state. The module provides functions which are used by the agents to change the state of the cells to open or closed. Other modules keep the reference of the Cells module, as they are dependant on the cell matrix.

4.4.2.B Emissions Module

Emission module keeps track of the per cell emission values. For each emission type, the module creates a cell grid matrix which contain the emission type values. At every step it gets the current step emission values of all the vehicles and adds them to the corresponding matrix position. To simulate the propagation emission a decay and Gaussian Blur is applied, as explained in Section 4.3.3.

4.4.2.C Emissions Renderer Module

Emissions renderer module plots the emission grid for each emission type. For every step the renderer gets the emission values from the Emissions module and plots them highlighting the areas that have the most pollutants. This module is an alternative to the in-built SUMO renderer, which requires to run the simulation in GUI mode.

4.4.2.D Induction Loops Module

Induction loops module keeps track of how many vehicles pass through every induction loop for every hour. At the end of the simulation the module writes the vehicle count to a CSV file. There are two versions of the module. One that reads the values directly from TraCI, but this one is inaccurate because of the timestep rounding. And another that gets the vehicle count from a temporary XML file that the SUMO simulator generates.

4.4.2.E Tracking Module

Tracking module tracks general statistical information about the simulator, like the number of closed cells, aggregated emission values, max emission value, wait and travel times, number of created and arrived vehicles. This module samples the information every 15 minutes and writes them to a CSV file. Optionally the module can create plots of the data. The data writing is done on a separate thread to not add extra delay to the simulation. Additionally no other module interacts with this module.

5

Solution Proposal

Contents

The proposed approach uses an RL agent which learns an optimal solution by interacting with the environment. This chapter will cover the details of the proposed solution, which includes how the problem is modelled, the proposed approach and the environment, including different training tactics.

5.1 Proposed Approach Overview

The primary goal is to create an RL agent that acts upon the environment. Following the traditional RL practices, the number of action outputs would be the combination of possible cell states, which for this problem where with 82 cells would be 2^{82} actions, which is extremely computationally expensive. The proposed approach treats each cell as an independent state. The agent then chooses an action for each cell. The number of possible combinations still remains the same, but the problem becomes more manageable.

Every 15 minutes of the simulation, the agent has to perform actions on the environment. The agent goes over every cell of the map and decides if the cell should stay open or closed. After this, a binary grid map is generated which represents which cells should be closed or open.

Every cell has a state, if the agent acts only by using the cell state, it will be blind to the surrounding cells and will act poorly. This will happen because the state of one road affects the future state of another road. To solve this, the agent will always have access to the state of all the cells. This way the agent can find correlation between different cells. Another important property passed in the state is on which cell the agent is acting, if this is not passed, the agent will learn to always perform the same action on all the cells.

5.2 Environment Representation

Reinforcement Learning algorithms require for the environment to be represented in a specific manner. At each step the environment should provide the observation, the available actions, and the reward.

As previously stated, the agent will interact with each cell independently meaning that each cell has its own observation, reward, and available actions.

5.2.1 Observation State

The observation represents the current state of the environment, and is provided to the agent at every action step. At each step, the agent needs to decide the action to perform for each cell. The way the problem is formulated, each cell will have its own observation, action and reward.

There are two ways to represent the observation. One way is for each cell interaction to provide only information about the cell that is being interacted with. The problem is that the agent won't be able to

take the neighboring cells into account. Another way is to provide information about all the cells. The issue is that for each cell action the observation will be the same, meaning that the agent will overfit on a specific action, and always choose the same action for all the cells.

The proposed observation has a mix of both, the state of all the cells as well as the current action cell's specific information. This way the agent will choose the best action for the current cell while taking into account the state of all the surrounding cells.

Due to the fact that the city is represented as a $m \times n$ grid, and that the state of one road may directly affect the state of another, a CNN is proposed for the initial layers of the model. The primary reason for this is that the model receives a matrix input, and the Convolutional Layers are ideal to find correlations in multi-dimensional represented data. The defined observation is a $m \times n \times p$ matrix which contains information about the map, and an extra vector input which represents general information like the current time of day. The time of day is important, as the traffic differs throughout the day.

The matrix input is comprised of the following data:

- **Emissions:** Represents the current per cell emission values.
- **Number of Vehicles** Contains the number of the vehicles that are present in each cell.
- **Cell Closing State:** Is a binary matrix that represents the currently closed cells.
- **Action Cell:** This matrix represents on which cell the agent is acting upon. One value is set to 1 and the rest is set to 0. This matrix changes for each acting cell.

Extra input is provided in the form of a one-hot encoded vector. The vector represents the action number of the day and is calculated by (5.1). In the future more information could be provided, like weather information, current date, etc.

$$action\_number = \lceil \frac{current\_day\_minutes}{action\_every\_n\_minutes} \rceil \tag{5.1}$$

5.2.2 Actions

For each cell, the agent can perform one of two actions, open or close the cells. Changing the cell state also changes the state of all the roads inside the cell. There are some cells on which the agents can't act, and in some cells the road is not changed. For example a cell that also includes a road where the cars are created. If the cell is closed, all the roads except the car spawning road are closed. This is because the agents tend to exploit certain features of the simulation.

Another situation to take into account, is that some roads will belong to two different cells simultaneously. This is because the SUMO simulator doesn't allow to define which section of the road should stay

open or closed, only the state of the entire road. This may lead to some undesired behaviours, where one cell may close the road, and another one will open it. It is the job of the agent to learn this behaviour.

5.3 Implementation

An agent is taught a policy by interacting with the environment. This policy is then used to choose which action to perform at a specific state. There are a variety of RL algorithms that can be used to train the agent, like DQN, Proximal Policy Optimization (PPO) [22], A2C/A3C. The algorithm used for this project is A2C. The primary reason for using A2C model is that it is parallelizable, meaning that it can run multiple simulations simultaneously. Furthermore, A2C was chosen over Asynchronous Advantage Actor Critic (A3C) as in practice A2C converged faster.

Each agent has a neural network that when given a state, produces a action. The primary reason for using a neural network is that the current problem can't be represented as a Q-table. For each cell, the environment state is passed through the network, which outputs the action that should be performed.

5.3.1 Deep Learning Model

There is no clear way of finding the optimal performing network structure without trial and error. The proposed model is comprised of two consecutive convolutional layers. The task of these layers is to find spatial correlation between the input data. After each of the convolutional layers, there is a Max Pooling layer which performs down-sampling, which helps to reduce the dimensionality of the data. After this, the max pooling layer connects to a fully connected network, with an additional input. This extra input contains categorical data about the general state of the environment. The fully connected network is comprised of two dense layers and a Softmax layer. For each dense layer a Dropout regularization technique is used, which randomly disables certain neurons from training.

The model uses ReLU as the primary activation function, this is because the input data can have abstract data ranges, meaning that activation functions like Sigmoid or tanh would perform poorly. The generated neurons are initialized with a normal distribution.

The model overview is shown in Figure 5.1, and the detailed layer parameters are shown in Table 5.1.

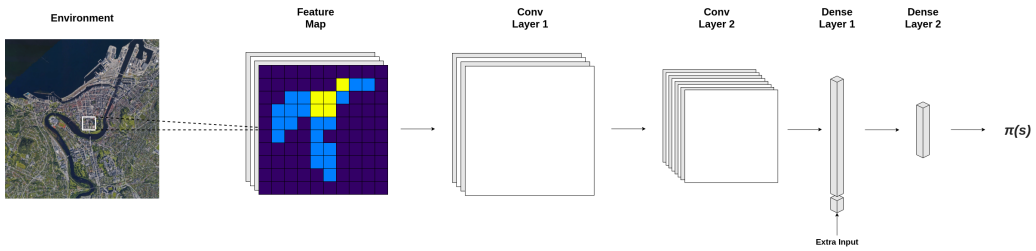


Figure 5.1: Used Network Pipeline

Table 5.1: Neural Network Parameters (f=filters, k=kernel, s=stride)

Layer Name	Activation Shape	Activation Size	# Parameters
Input Layer 1	(16, 17, 4)	0	0
Convolutional Layer 1(f=16, k=[3,3], s=1)	(16, 17, 16)	4352	592
Max Pooling 1	(15, 16, 16)	3840	0
Convolutional Layer 2(f=32, k=[4,4], s=1)	(15, 16, 32)	7680	8224
Max Pooling 2	(14, 15, 32)	6720	0
Input Layer 2	(96, 1)	96	0
Fully Connected Layer 1	(144, 1)	144	981648
Fully Connected Layer 2	(64, 1)	64	9280
Softmax	(2, 1)	2	130

5.3.2 Multi-Agent Framework

Even though the action space is greatly reduced, the amount of information present is still too vast for a single agent to learn. Ideally each cell would have it's own agent which would learn a policy only applicable to that cell, and not a global one. Taking this approach would require a lot of computational power as, for example, where a single model required 400 Megabytes of Graphics Processing Unit (GPU) memory, 82 agents would require on average 32 Gigabytes of GPU memory.

Taking into account the hardware limitation, a compromised approach is proposed where the map is split into 4 regions, represented in Figure 5.2. Each region has its own agent that acts upon the cells in that region, as represented in Figure 5.3. Each agent has its own network which needs to be trained independently. For each step, each agent iterates over the cells in its region and decides if the cell should be open or closed. All the agents receive the same input where the only variation is the location of the cell. This is so the agent has information about the global state of the map, and the behaviour of the other agents.

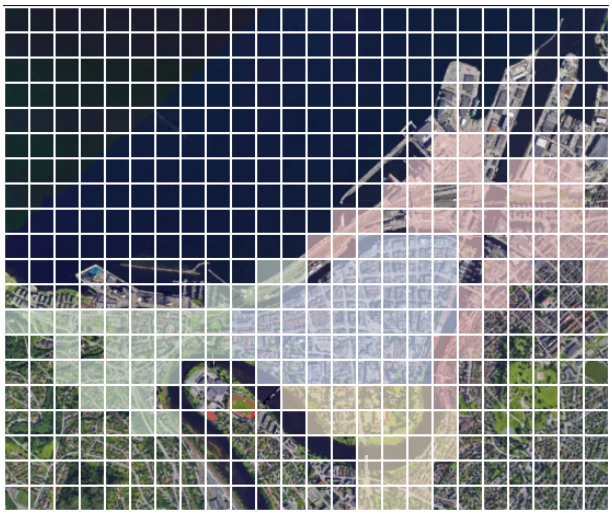


Figure 5.2: Map split into multi-agent regions. Each color corresponds to an agents actionable cells.

This approach introduces a new challenge, cooperation. The agents will have to learn to increase a global reward by cooperating. This approach should also decrease the number of required simulations and the size of the models, as there is less information to be learned per agent.

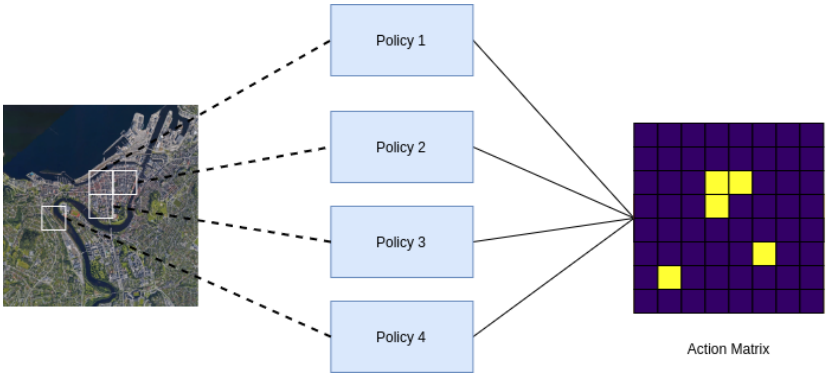


Figure 5.3: Multi-Agent Approach

5.4 Training Environment

The training simulation is performed on a single day for a period of 24 hours, where every 15 minutes the agents has to perform an action. During the simulation the agent will have to decide an action for each cell for each action step, which during a period of 24 hours, 4 times an hour and 82 cells gives a total of 7872 actions. It is important, during a simulation, to provide many interactions for the agent, because of how long a simulation takes. On average to simulate a day takes around 80 seconds, and the simulation needs to be performed hundreds of times for an agent to be able to learn a valid policy.

Another proposal is to use pre-generated data. Here the simulations are run with random actions, and their state is stored. The agent is then trained with the saved data. The issue is that the data is static, but the agent will keep generating actions. To solve this, an action mask is added to the model which forces the agent to take the desired action, which in this case is the action that is specified in the generated data. By using this method, the training is faster, as the environment doesn't need to be simulated again.

6

Results

Contents

In this chapter we'll take a look and discuss the results from the testing of the solution proposed in Chapter 5. We'll also take a look at the findings discovered during the development and the reason why certain development decisions were made. Baseline is a simulation where there are no agents acting upon. All the results will be primarily compared to the baseline.

6.1 Experiment Description

Most of the days showed little variation between one another. The training was performed on a single day, and tested on a different one. This is to test the agents ability to generalize. The training is limited to a single day because of the time a training sessions takes.

The training wasn't limited to a specific number of episodes, this is because different rewards functions require different amount of data. In general, the more complex a reward function is, the longer it takes to learn. The training was stopped only when the agents behaviour was the same for the last few episodes.

6.1.1 Vehicle Throughput

To test the ability of the vehicles to dynamically change their route and test if the agent is able to learn a simple reward function, a throughput experiment was devised. In this experiment a single agent is used and the goal is to learn a policy that maximizes the traffic flow in the city. The agent is able to close and open any given cell. The reward function is the percentage of the vehicles that successfully arrive to their destination. The wait and travel time were not used, as they introduce a more complex behaviour to learn and easily can be exploited, like for example, closing all the cells reduces the travel and wait time.

6.1.2 Pre-generated Data

A day simulation is generally slow, and to train an RL, requires hundreds or in some cases thousands of simulations. A solution proposed is to run hundreds of simulations with random actions and save the information like the closed cells, emission values, number of vehicles for each step. The agents are then to be trained on this data. During the training the agent will try to make actions, but because the data is static, it's not possible. To solve this, an action layer is added to the end of the network. This forces the agent to always select the predefined action.

To test this approach, a single agent is trained with the goal of maximizing the throughput as in the previously described experiment. The reason for this is that we then can compare both results.

6.1.3 Reactive Agent

To test the efficacy of the RL agent, it needs to be compared with other solutions. As there are no available implementations that run on the same environment, an ad hoc agent is proposed. The proposal in question is a reactive agent. The only functionality that the agent has, is to check the pollution levels in each cell and close the ones that exceed a specific threshold, as represented in the Algorithm 6.1. The reason for such an agent is because every city in Europe has a limitation of pollution that an area in any given time cannot surpass. Two tests are performed, one where the threshold is 100 NOx ($\mu g/m^3$) per cell, and the other where the threshold is 50.

Algorithm 6.1 Reactive Agent Decision Flow

```
procedure REACTIVE_AGENT(threshold, cells)
    for cell in cells do
        get cell emissions
        if cell_emissions >= threshold then
            Close cell
        else
            Open cell
```

6.1.4 RL Agent

The goal is to develop an RL agent that reduces the pollution in the city. For this an adequate reward function needs to be found. For this an iterative search was performed, until the correct reward parameters were found.

If the reward is just the max cell emissions, meaning the value of the cell that has the highest emissions value, the agent learns to exploit an obvious behaviour, which is closing all the cells. The logic behind this, is that if there are no cars circulating, there are no emissions. The reason why total emissions weren't used as a reward is because it is a global reward, and the agent has trouble optimizing for it. Ideally a mix of global and local rewards is needed.

When the reward is the travel time or waiting time, a similar behaviour as to max cell emissions is found. If there are no vehicles, the value is always at 0. Because this reward has a similar behaviour as the max cell emissions, they can't be used as a compound reward.

The only reward that can be mixed with the emissions, is the number of vehicles that arrived, as they have an opposite behaviour. This way the agent tries to minimize the pollution while guaranteeing that the vehicles reach the destination. Minimizing the emissions is also not ideal, as it will try to minimize the value to 0 and in some cases may penalize heavily the traffic. Solution to this is to try to maintain the emissions between a desired threshold. The final reward function is as shown in (6.1).

$$reward = \frac{(t - mce)}{200} + \frac{av}{eav} \tag{6.1}$$

where:

t = emissions threshold
mce = current highest cell emission
eav = expected step arrived vehicles
av = step arrived vehicles

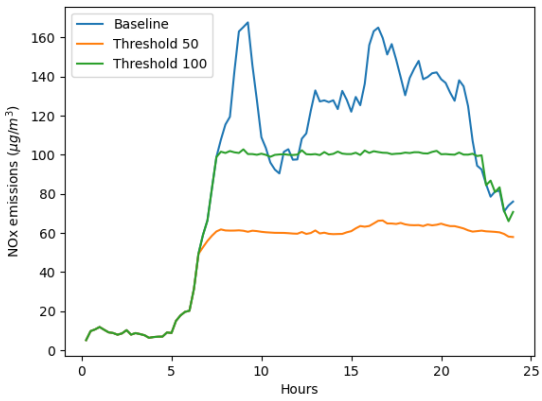
Having a reward function, the final experiments can be conducted. First, a experiment is done to test the performance of a single agent against a multi-agent system. The best approach will then be used for the further experiments. Two additional experiments are conducted where the only variation is the threshold value, one with 100 and 50 NOx.

6.2 Results

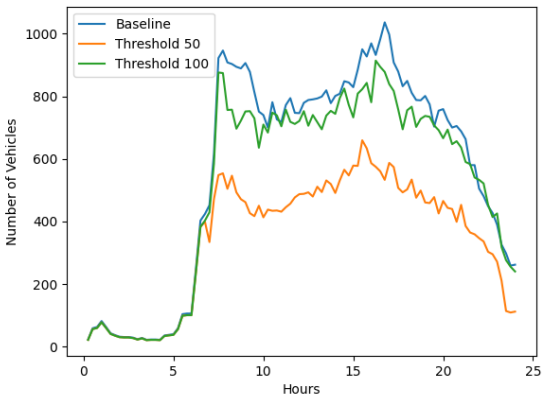
In this chapter the results of each experiment will be presented with a respective explanation for the agents behaviour. The results show the agents direct effect on the environment, and not the training performance of the model.

6.2.1 Reactive Agent

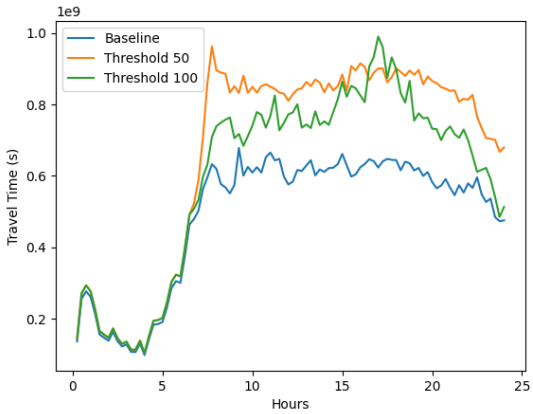
As shown in Figure 6.1, the 100 limit agent performs well in maintaining the max emissions under the threshold, with minor reduction in arrived vehicles. On the other hand, the 50 limit agent has trouble in achieving the max emissions threshold, and it reduces drastically the number of arrived vehicles. This is because the agent takes drastic actions, closing too many cells and reducing the effectiveness of the traffic. Another interesting side effect can be seen, is when a cell is closed all the vehicles in it get stuck. The vehicles are prohibited from circulating, but they still produce emissions which increase the cell emissions, which make the agent keep the cell closed. This is also the reason why the waiting time is constantly increasing, because more and more vehicles get stuck in closed cells. The problem of the vehicles being stuck in the cell could be easily solved by modifying the simulation and removing the vehicles from the closed cells. This wasn't done because in the tests the RL agent exploited this behaviour and closed all the cells, which in turn inhibits any vehicle from entering the city.



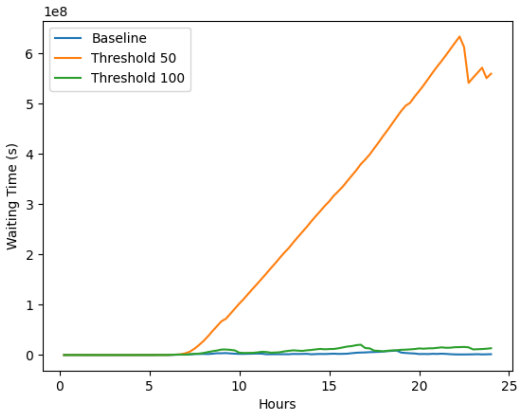
(a) Max Cell Emissions Value



(b) Arrived Vehicles to Destination



(c) Travel Time



(d) Waiting Time

Figure 6.1: Reactive agent results

6.2.2 Vehicle throughput

While testing the agent for the arrived vehicles optimization, in some cases the agent learned a different pattern of behaviours. In general the agent learned that opening all the cells maximizes the traffic throughput, but in some cases, the agent learned that some cells are irrelevant for the traffic, and keeping them closed doesn't affect the traffic flow. This is because, when those cells are closed, there is always a different route that the vehicles could take. The cells in question can be seen in Figure 6.2.

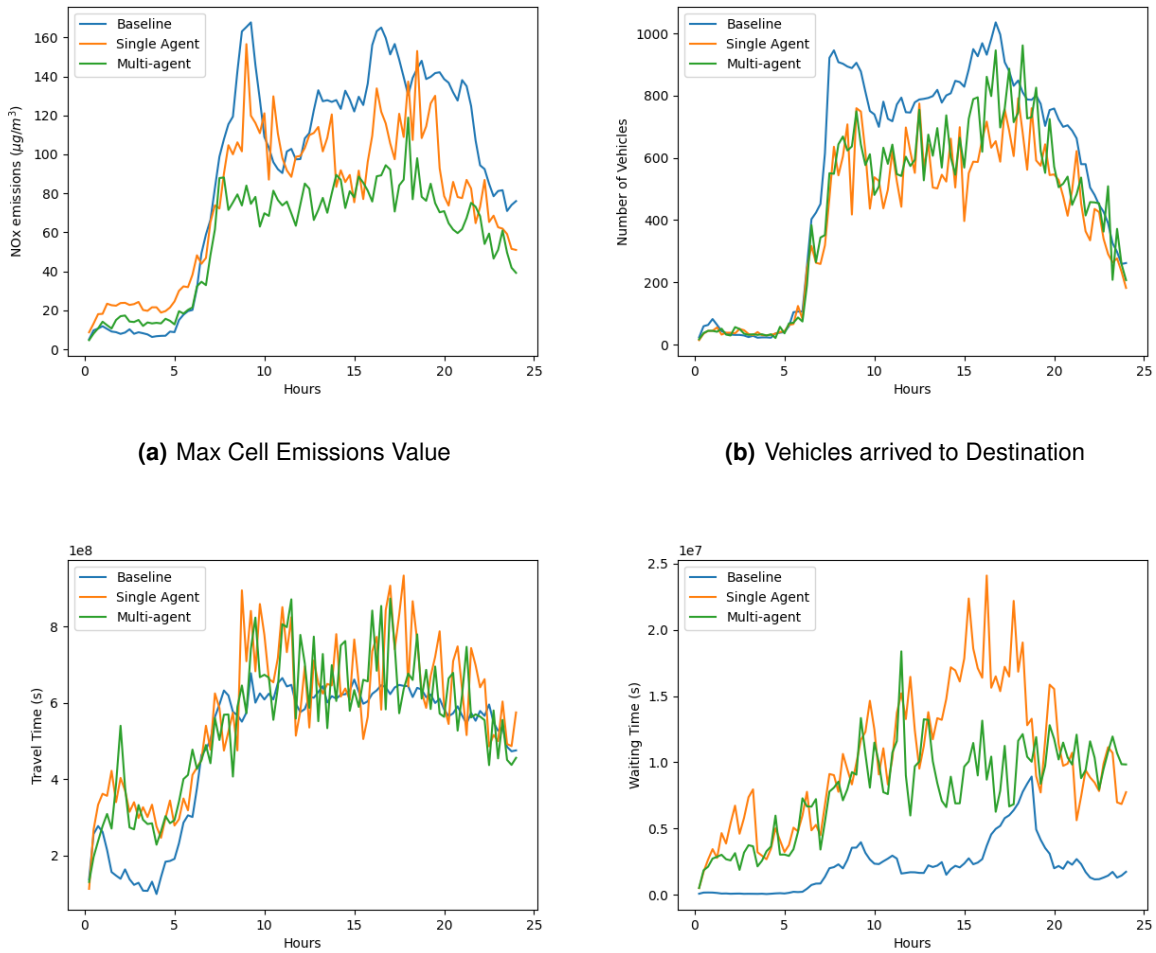
Figure 6.2: Cells that are not relevant for the traffic

6.2.3 Pre-generated Data

Training the agent on pre-generated data sped up the training drastically, which in turn allowed more testing and rapid prototyping. This unfortunately didn't work, as the agents always overfitted on a wrong solution or didn't learn a solution at all, compared to a normally trained agent. This may be due to the lack of training data, as the agent would require more examples because of the lack of exploration.

6.2.4 RL Agent

A test was made comparing a single agent, against a multi-agent system. As shown in the Figure 6.3, the multi-agent approach was able to learn a better policy compared to a single agent. With different reward functions, the multi-agent approach was able to converge faster on a solution compared to a single agent. Because of this, all of future trainings were performed on the multi-agent system.



(a) Max Cell Emissions Value

(b) Vehicles arrived to Destination

(c) Travel Time

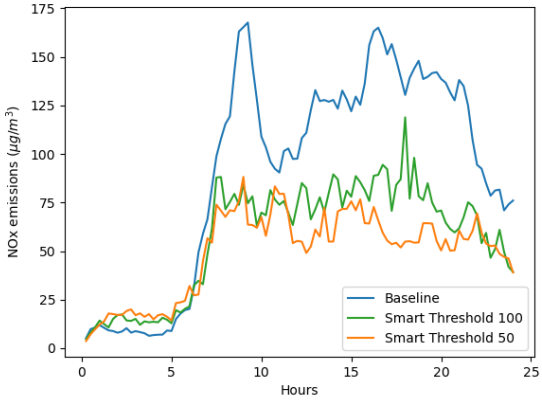
(d) Waiting Time

Figure 6.3: Comparison between Single Agent and Multi-agent approaches

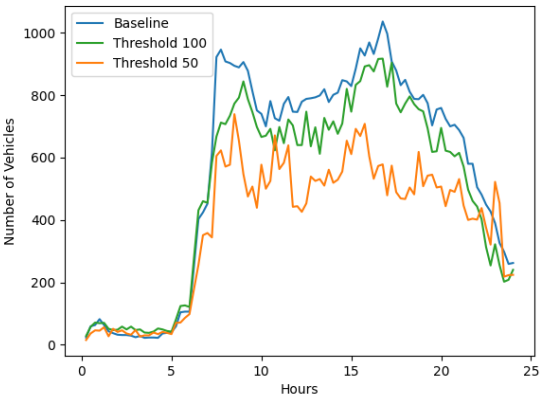
The agents were trained with two different threshold values, 100 and 50 NOx ($\mu g/m^3$), the same as the reactive agent. The test results can be seen in Figure 6.4. The agents with the 100 limit threshold are able to learn correctly the policy and act accordingly. The agents are able to reduce the pollution below the threshold with minimal traffic impact. There is a spike where the value goes over the threshold, this is due to accumulation of cars in a closed cell. Training more the agents and with more varied data will solve the issue. The travel time was impacted as the vehicles have to find alternative longer routes.

Contrary, the 50 limit threshold agents are not able to minimize the value below the threshold. This is because, to achieve lower emissions, the number of vehicles needs to be reduced. Currently there is no way to directly control the number of vehicles that are created. What happens is the agent closes cells which cause a traffic jam that extends to the roads where the cars are created. When this happens

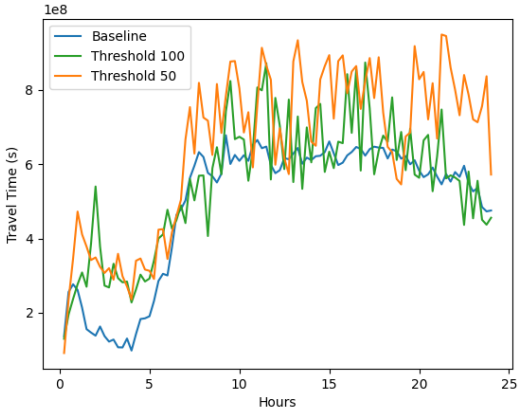
new cars can't be introduced into the simulation, and this explains the lower number of vehicles. The agents are able to find an equilibrium between both rewards, emissions and the number of arrived cars. The waiting time also increases because the vehicles get stuck in closed cells, or in roads without any alternative routes due to the closed cells.



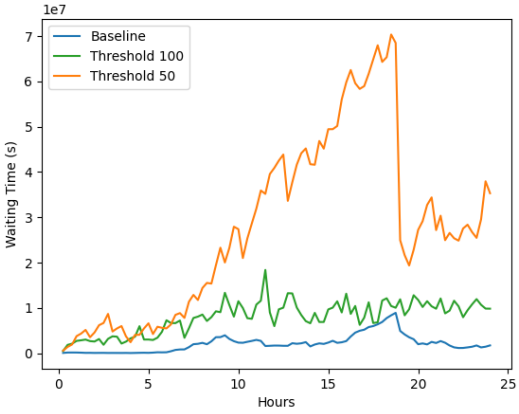
(a) Max Cell Emissions Value



(b) Vehicles arrived to Destination



(c) Travel Time

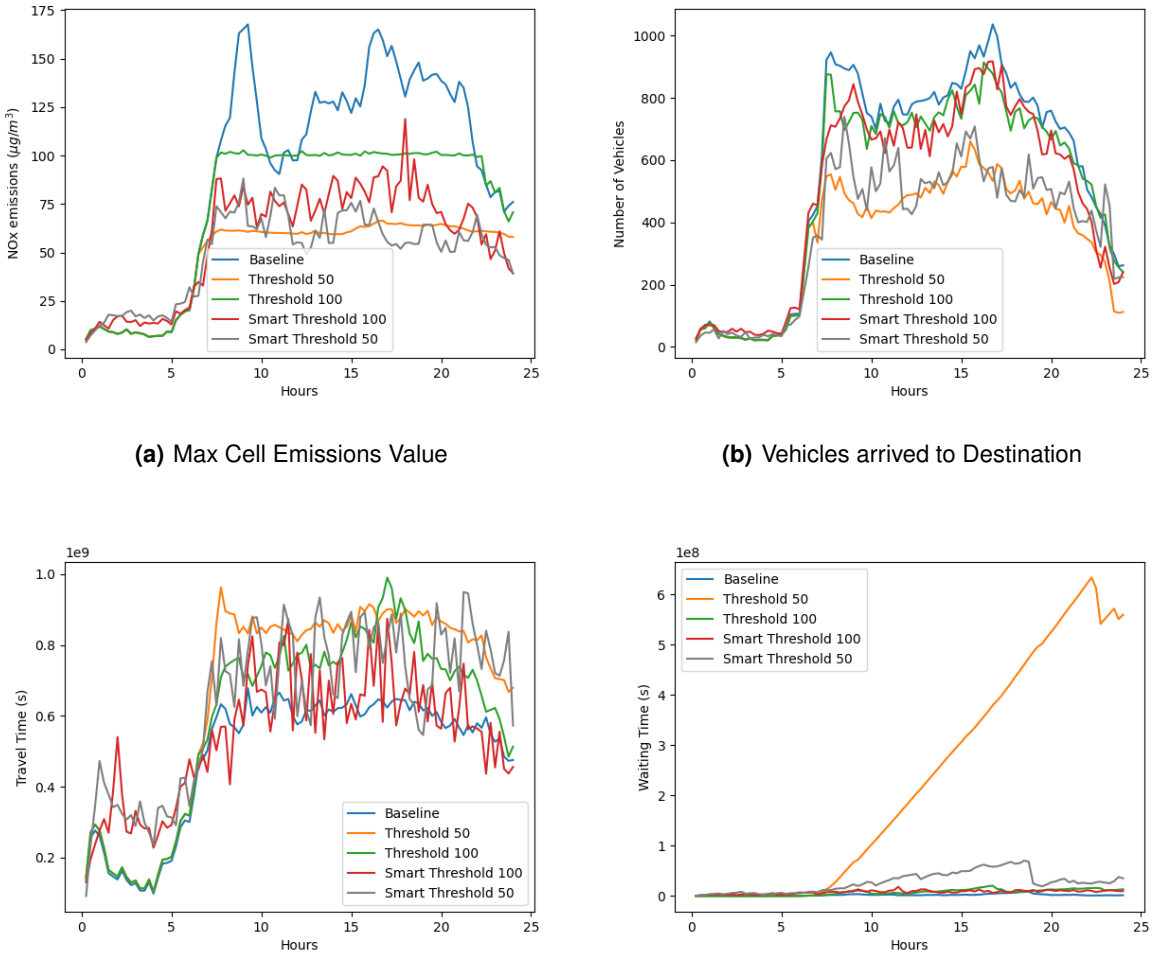


(d) Waiting Time

Figure 6.4: RL agent results

6.2.5 Comparison

Taking a look at the Figure 6.5, we can see some interesting patterns. Looking at the agents with 100 NOx ($\mu g/m^3$) threshold, we can see that the RL agent is able to reduce the emissions more than the reactive agent. This is because the RL agent leverages the emissions more than the arrived vehicles, thus a small decline in arrived vehicles.



(a) Max Cell Emissions Value

(b) Vehicles arrived to Destination

(c) Travel Time

(d) Waiting Time

Figure 6.5: Results comparison between Reactive Agent, RL Agent and Baseline

For the 50 NOx ($\mu g/m^3$) threshold, both agents struggle to maintain the below the threshold. The reactive agent can't lower the emissions more due to the poor management of the cells, while the RL agent is due to the emissions-arrived equilibrium. It is also relevant to notice, that with this threshold, the agents prioritize lowering the emissions more than maintaining vehicle flow. In general both agents have similar results, except for the waiting time. The Reactive agent closes the cells with the vehicles, blocking them from moving, this in turn increases the waiting time, while the RL agent tries to avoid this behaviour. The RL agent is able to learn this behaviour because of the arrived vehicles reward part. When the vehicles are stuck in a cell, they are not able to arrive to the destination, in turn lowering the reward.

It may seem counter-intuitive that vehicles taking longer routes don't increase the pollution. This is because when the vehicles take longer routes, they in fact produce more emissions, but they produce the emissions in less concentrated areas which allows for the emissions to dissipate faster.

The Table 6.1 shows a more statistical results comparison.

Table 6.1: Agent differences relative to the Baseline

	Avg. Max Emissions	Avg. Arrived Vehicles	Avg. Travel Time	Avg. Waiting Time
Reactive Agent Limit 100	-19.13%	-8.21%	15.69%	192.40%
Reactive Agent Limit 50	-47.39%	-37.67%	28.14%	9073.79%
RL Agent Limit 100	-36.97%	-10.53%	11.51%	275.15%
RL Agent Limit 50	-45.99%	-31.07%	31.70%	1195.48%

6.2.6 Week Simulation

To test the generalization of the approach, the system was trained on a single day and tested on an entirely different week. As seen in the Figure 6.6, the agent is able to generalize and act accordingly even though that the agent wasn't trained on those days. This shows that the agent can act on previously unforeseen data and with more training, the performance can improve.

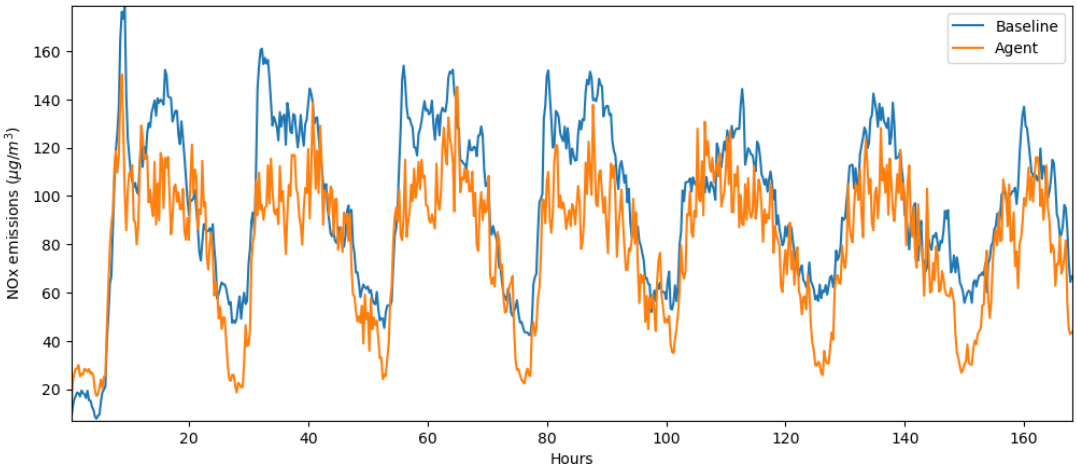


Figure 6.6: Agent acting on a 7 day simulation

7

Conclusion

Contents

In this chapter, we'll take a look at the solution, reflecting on the results, and presenting suggestions for future work.

7.1 Conclusions

Vehicle are some of the primary environment pollutant. Rapid urbanization and poor traffic are starting to create health hazards in big cities. Because of this tougher regulations are being imposed on cities and car industry. One of the primary issues, is the poor management of the city traffic, as for example sub-optimal traffic light times which cause huge accumulation of vehicles in one area. Currently these are controlled by manually crafted models, which in many cases are not adapting to the traffic behaviour change.

Since it's not possible to test a solution on the real world, a simulator is required. A simulation based on SUMO was created that generates traffic based on real world sampled data. Additionally vehicle emission propagation was modelled based on real data. The simulator was also extended to implement the OpenAI Gym format which is a common RL practice. Using a Gym environment allows to use existing RL frameworks.

We provide an RL solution to reduce the city level emissions by managing the access to specific areas of the city. The proposed approach splits the map into a cell grid and uses multiple agents that assess the state of each of the cells. Each of the agents is responsible for managing a specific region of the city, which are comprised of multiple cells. Each of the agents contains a Convolutional Neural Network which is used to find the patterns on the input data, and learn a desired policy.

An approach using pre-generated data was proposed. Here multiple simulations with random actions were run and the intermediate states were stored. The agents were then trained on this data. This approach would allow a more controlled environment and faster training. Unfortunately the agents were not able to learn a valid policy and overfitted on a specific behaviour. Although the poor agent's behaviour, it doesn't eliminate the possibility of using such approach in the future.

Ideally each map cell would have its own policy as this would allow to have a more optimal behaviour, but this would require a lot of computational resources. A compromised solution was introduced, where the map was split into four regions and each region had it's own policy. The agents then had to cooperate together to maximize a global reward. This approach was compared with a single policy approach. The multi-agent approach performed better than a single agent. This is because a single agent can't generalize well on so many actionable spaces. A solution would be to increase the complexity of the model, but this would require more resources and more training compared to just using a multi-agent approach.

The agents were trained on different reward functions and compared with a reactive agent and the

baseline. The RL agents perfomed better than the reactive agent at minimizing the emissions below the threshold. When more weight was given to minimizing the emissions, a bigger reduction in number of vehicles was seen. The RL agents also showed that some regions of the city are not necessary to the traffic throughput, as there are alternative routes that the vehicles could take.

With more training and improvements the agents could perform even better and be reused for other cities.

7.2 Future Work

For future work, the simulation could be improved by providing additional information, like weather, day type, etc. This would allow the agents to better understand the behaviour of the traffic and the effect that the weather has on the emissions. Additionally the emissions simulation could be improved to better model the real world behaviour where for this the environment details like the environment topology and weather needs to be taken into account. With more computational power and time, the agents could be trained on more days, which would make the behaviour to be more robust and prone to outliers.

The simulation speed could be improved, as there are multiple bottlenecks like emissions simulation and garbage collection. Some parts could also be rewritten in C++ as in some cases it is in order of magnitude faster than Python. Another reason is that the SUMO simulator originally is written in C++, thus a communication layer between C++ and Python is required which adds extra latency. Additionally, PyTorch could also be used instead of TensorFlow, as in limited tests, it showed to perform better on consumer grade hardware. Switching to PyTorch would also require to replace RLlib by a different framework as there is some functionallity that is not yet implemented for PyTorch.

The current implementation will also be dockerized and hosted as a component based experiment on the AI4EU Experiments platform. The simulator and the agent will have its own Docker [23] containers which will be able to be used to build a pipeline and be deployed as an online service.

Bibliography

[1] P. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using sumo," in *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018. [Online]. Available: https://elib.dlr.de/124092/

[2] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, ser. Cornell Aeronautical Laboratory. Report no. VG-1196-G-8. Cornell University, New York, 1962. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/256582.pdf

[3] D. Hebb, *The Organization of Behavior: A Neuropsychological Theory*, ser. A Wiley book in clinical psychology. John Wiley & Sons Inc., New York, 1949. [Online]. Available: http://s-f-walker.org.uk/pubsebooks/pdfs/The_Organization_of_Behavior-Donald_O._Hebb.pdf

[4] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: http://nn.cs.utexas.edu/?stanley:ec02

[5] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, May 2015.

[6] S. Team, "Convolutional neural networks (cnn): Summary," 2018, online; accessed December 6, 2020. [Online]. Available: https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-summary/

[7] R. Bellman, *Dynamic Programming*. Princeton University Press, Princeton, New Jersey, 1957, ch. XI. [Online]. Available: https://www.gwern.net/docs/statistics/decision/1957-bellman-dynamicprogramming.pdf

[8] D. Kolmas, "Markov decision processes," 2019, online; accessed December 6, 2020. [Online]. Available: http://www.damiankolmas.com/rl/Marcov-Decission-Process/

[9] R. Howard, *Dynamic Programming and Markov Processes*. The Technology Press of Massachusetts Institute of Technology and John Wiley & Sons, Inc., New York .

London, 1960, ch. 4. [Online]. Available: https://www.gwern.net/docs/statistics/decision/1960-howard-dynamicprogrammingmarkovprocesses.pdf

[10] C. J. C. H. Watkins, "Learning from delayed rewards," Ph.D. dissertation, King's College, Cambridge, UK, May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[11] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," Cambridge University Engineering Department, Tech. Rep., 1994.

[12] S. Exchange, "Convergence of q-learning and sarsa," 2017, online; accessed December 6, 2020. [Online]. Available: https://stats.stackexchange.com/questions/317636/convergence-of-q-learning-and-sarsa

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[14] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *SIAM Journal on Control And Optimization*. MIT Press, 2001, pp. 1008–1014. [Online]. Available: https://papers.nips.cc/paper/1786-actor-critic-algorithms.pdf

[15] Y. Lin, X. D. andLi Li, and F.-Y. Wang, "An efficient deep reinforcement learning model for urban traffic control," *CoRR*, vol. abs/1808.01876, 2018. [Online]. Available: http://arxiv.org/abs/1808.01876

[16] I. Arel, C. Liu, T. Urbanik, and A. G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control," *IET Intelligent Transport Systems*, vol. 4, no. 2, pp. 128–135, 2010.

[17] L. N. Alegre, "Sumo-rl," https://github.com/LucasAlegre/sumo-rl, 2019.

[18] T. Chu, J. Wang, L. Codecà, and Z. Li, "Multi-agent deep reinforcement learning for large-scale traffic signal control," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[19] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): Methodology and large-scale application on downtown toronto," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1140–1150, 2013.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *DeepMind Research*, December 2013. [Online]. Available: https://arxiv.org/pdf/1312.5602v1.pdf

[21] "The handbook of emission factors for road transport," Jan 2010. [Online]. Available: https://www.hbefa.net/e/index.html

[22] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *OpenAI Research*, 2017. [Online]. Available: https://arxiv.org/pdf/1707.06347.pdf

[23] D. Merkel, “Docker: lightweight linux containers for consistent development and deployment,” *Linux journal*, vol. 2014, no. 239, p. 2, 2014.