DQL For Traffic Lights Control

Riccardo Cappi (riccardoblevio@gmail.com), Sebastiano Monti (sebastiano.monti.96@gmail.com)

March 2024

Contents

1 Introduction

Modern society highly relies on transportation systems. The most important one for individuals and services' movement is urban vehicle transportation. Population's dramatic increase and subsequent vehicle ownership worsened the problem of traffic congestion, resulting in long delays for travellers, huge waste of energy and fuel, worsened air quality and increased vehicular accidents.

Two types of solution are possible for such problems. The first one is to expand road infrastructure. However, this can be a very expensive and just a short term solution. The second one is to increase the efficiency of already existing infrastructures, like, in particular, traffic light signals at intersections. Solutions of this kind can be roughly grouped into three main categories. The first regard simple fixed time signals control, in which all green and red phases are assigned to an immutable duration, without considering fluctuations in traffic demand. The second include vehicle-actuated signal control, such as underground inductive loop detectors to detect the presence of vehicles in proximity of traffic lights. In these cases inputs are processed in a rough way to determine green and red lights' duration. The last and more interesting approach involves adaptive signal control, in which the timing of traffic lights is automatically managed according to the current congestion state of the intersection.

This project aims at analyzing the last approach by taking as reference some recent research works based on Reinforcement Learning (RL). The goal is to have a RL agent which is able to choose the optimal traffic lights configuration, according to the current congestion state of the intersection, in order to minimize vehicles' jam lengths and waiting times at the intersection.

2 Background

In a reinforcement learning setting, an agent interacts with the environment to get rewards from actions. Usually, a reinforcement learning model faces an unknown Markov decision process. It consists of the set of all the states S, the action set A, the transition function δ, and the reward function R. At each discrete time t:

- the agent observes state $s_t \in S$;
- it chooses action $a_t \in A$ (among the possible actions in state s_t) and executes it;
- it receives immediate reward r_t, that can be positive, negative or neutral.
- the state changes to s_{t+1}

We assume that r_t and s_{t+1} only depend on current state and action. The agent's goal is to learn an action policy $\pi : S \to A$ that maximizes the expected sum of (discounted) rewards obtained if policy π is followed:

$$E = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $0 < \gamma \leq 1$ is called the **discount factor**. For each possible policy π the agent might adopt, we can define an evaluation function over states:

$$V^{\pi}(s) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

where $r_t, r_{t+1}, ...$ are generated executing policy π starting at state s. Then, the choice of the best actions to play becomes an optimization problem. Indeed, it comes down to finding the optimal policy π^* that maximizes the evaluation function:

$$\pi^* = argmax_{\pi} V^{\pi}(s)$$

So, how can we find the optimal policy π^*?

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*).

$$\pi^*(s) = argmax_a[r(s,a) + \gamma V^*(\delta(s,a))] \tag{1}$$

Unfortunately, learning V^* is a useful way to learn the optimal policy only when the agent has perfect knowledge of δ and r. This requires that it'd be able to perfectly predict the immediate result (i.e., the immediate reward and immediate successor) for every possible state-action transition. In many practical problems, it is impossible for the agent or its human programmer to predict in advance the exact outcome of applying an arbitrary action to an arbitrary state.

In the Q-learning framework, a numeric value $Q(s,a) \in \mathbb{R}$, called Q-value, is associated to each state-action pair. The value of Q is the reward received immediately upon executing action a from state s, plus the value (discounted by γ) of following the optimal policy thereafter.

$$Q(s,a) = r(s,a) + \gamma V^*(\delta(s,a))$$

Then, we can reformulate the optimal policy as:

$$\pi^*(s) = argmax_a Q(s,a)$$

The Q-values are estimated in the Q-learning algorithm by iterative Bellman updates: $Q_t(s,a) = Q_{t-1}(s,a) + \alpha(r + \gamma max_{a'} Q_{t-1}(s',a') - Q_{t-1}(s,a))$. In this way, if the agent learns the Q function instead of the V^* function, it will be able to select optimal actions even when it has no knowledge of the functions r and δ.

3 Operating environment

In this section we will define the operating environment of the project. First, let's define the task environment according to the **PEAS** description:

- **Performance measure:** the performance of the agent is assessed with respect to two common traffic metrics: queue length and vehicles waiting time. The goal is to find a model able to dynamically control the traffic lights of an intersection in order to reduce both the queue length and the vehicles waiting time.
- **Environment:** The architecture we aim to implement needs "eyes" in order to constantly picture the road condition in real time. This is possible with the presence of cameras installed in proximity of crossroads, but also with the increasing popularization of vehicular on-board cameras. Thank to these devices, a lot of information about roads could be extracted and transmitted to the algorithm, such as vehicles' position, speed and waiting time.

  It is difficult, however, to retrieve visual data of crossroads pertaining to urban situations, making infeasible real world experimentation. For this reason, we relied on SUMO [6], an open source traffic simulator that allows to model real world traffic behaviours.

  This software, through an API called TRACI, provides a complete control over the simulation environment elements, such as vehicles' speed and position, traffic flow's intensity in each lane, traffic light signal phases, road intersections' shape and infrastructure, etc. A visual representation of the crossroad we designed for the analysis can be seen in Figure 1. All the simulation specifications (network shape, traffic flow, etc.) are defined via .xml files that are parsed by SUMO in order to create the simulation environment. The detailed setup of the simulation is presented in Section 5.1

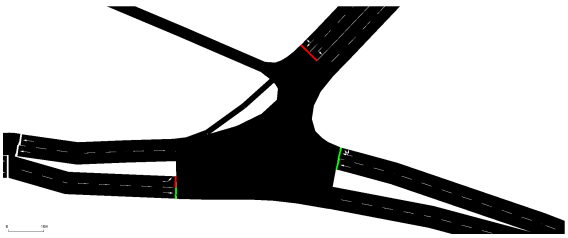


Figure 1: Representation of considered intersection network

- **Actuators & Sensors:** The agent interacts with the simulation environment using the Python APIs provided by TRACI. This library contains a set of useful methods to get the relevant simulation's information (e.g. vehicles positions, jam lengths, current signal phase, etc.) and to modify the state of the simulation (e.g. change traffic light's phase/timing).

  The agent takes as input a snapshot of the current traffic state at the intersection (more details on the state representation in Section 4.1) and returns as output a specific configuration of the traffic lights among the possible green/red lights configurations. In this context, the *sensors* can be thought as cameras which constantly detect vehicles on intersection's lanes.

Although dynamic traffic light control is an extremely complex task in the real world, SUMO allows you to operate in a more controlled environment that makes the job easier. In fact, the agent works in a **fully-observable** environment since the software gives access to the complete state of the environment at each point in time.

Furthermore, SUMO makes the environment **deterministic** and **static**. In fact, it is possible to specify the traffic flow both in a deterministic and stochastic way. For this project, we set a deterministic traffic flow in order to keep things simple. The environment is also static, because while the agent chooses an action the simulation *pauses* and resumes after the action is performed.

Lastly, the environment is **continuous**, since its state representation involves continuous values (as we will see later), and it is **unknown**, since the agent does not know the outcome of each action in advance. Clearly, the environment is also **sequential** and **single agent**.

RL is particularly suited for this kind of problem, since it does not need a complete knowledge of the environment in advance. Instead, it is able to gain knowledge by progressively interacting with the environment, learning by trial and error.

4 Methods

In order to build a reinforcement learning model for traffic lights control, we need to define the traffic state representation, the action space and the reward function.

4.1 State representation

Authors in [1][3][7] have defined the state representation on the basis of queue length of different incoming roads, while in [9] the traffic state is estimated by considering both queue length and the maximum time a vehicle has waited on each lane at the intersection. However, all these representations are abstractions of the traffic state which omit relevant information and could lead to suboptimal solutions. For example, a state representation based only on the number of queued vehicles does not take into consideration those vehicles

that are still moving.

In this project, instead, we followed the approach proposed in [8], which aims at implicitly extract all the relevant features from raw real-time traffic data by using a convolutional neural network that takes as input an image-like representation of the traffic state.

The idea is to map the region around the intersection into a grid of cells. This state-matrix is computed in the following way: each lane approaching the intersection is mapped into a Boolean-valued vector of length l, where each cell can contain a 1, which indicates the presence of a vehicle at that position, or a 0, which indicates the absence of a vehicle. Each cell of the vector corresponds to 1 meter of the lane, therefore, if we have a lane of 300 meters, the corresponding vector will have 300 cells. This process is repeated for each lane, and the final matrix is obtained by stacking all the lane vectors. The considered intersection has three incoming roads, each with two lanes (6 total lanes), where the longest lane is 309 meters and the shortest 103 meters. Therefore, this state-matrix will have a shape of (6×309). Note that a zero-padding is added to lane vectors with shorter length than the maximum length lane in order to have all equally-sized vectors.

Previous works, such as [4], combined this binary matrix with another matrix indicating vehicle's speed. However, according to the work made by authors in [8], we tried to use a stack of consecutive simulation frames so that the model is able to implicitly estimate velocity and travel direction of vehicles. In particular, the state-matrix mentioned above is computed for the last two simulation steps and the results are stacked to create a new matrix of shape $(2 \times 6 \times 309)$.

The representation built so far consists in a Boolean-valued matrix which contains the information about vehicles' positions of the last two simulation steps. However, it does not take into consideration the vehicles' waiting time. This information is embodied in the representation by computing another state-matrix where cells contain the normalized values of the vehicles' waiting times of the last simulation step. Obviously, if there is no vehicle at a specific position, its corresponding cell is 0.

Then, the final state representation consists in a $(3 \times 6 \times 309)$ matrix, where the first two layers provide information about vehicles' position and the last layer about vehicles' waiting time.

4.2 Action space

Another important aspect is how the action space is defined. Previous works proposed two different possibilities:

1. Authors in [5] proposed a system in which all the traffic light phases cyclically change in a fixed sequence, and the agent's action is to select every phase's duration in the next cycle.

2. On the other hand, most of the previous research defined the action space as the set of possible green/red lights configurations [10][8][4]. This scenario implies that each phase must have a fixed duration where green/red lights can only be a multiple of this fixed-length interval.

This project follows the second approach, as it seems to be the most popular. In particular, the action space is defined as the set of possible green/red lights configurations at the intersection. On each of the three incoming roads there is a traffic light which manages the traffic on the corresponding lanes. The combination of the individual phases of these traffic lights forms the set of the possible green/red lights configurations.

In Figure 2 all the 3 possible signal phases that can occur at the considered intersection are shown. The green and red lines represent the routes that vehicles can travel during the simulation. Vehicles on green paths are allowed to pass, while vehicles on red paths must stop. The agent selects which lanes get a green light according to these 3 configurations.

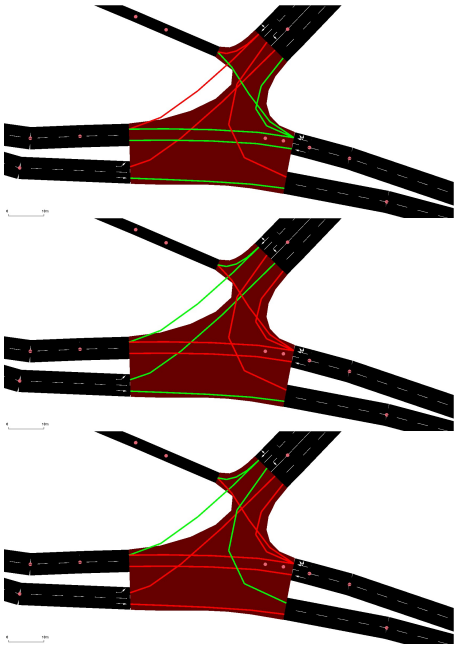


Figure 2: Possible phase configurations that can occur at the intersection

4.3 Reward

Defining a reward for the traffic control problem is not easy. The most common reward functions in literature are defined as the change in cumulative vehicle delay [2][8] and change in number of queued vehicles[7]. However, this project follows the idea proposed in [9], in which the reward is based both on the decrease in queue length and on the decrease in vehicles' waiting time. In fact, if only the decrease in queue length is used to compute the reward, a situation can occur in which only vehicles in crowded roads are allowed to pass, while those vehicles in minor roads could wait indefinitely. On the other hand, if the reward is solely based on the decrease in vehicles' waiting time, the agent will favor vehicles that have waited for a long time regardless of the queue length on other roads. For this reason, the proposed reward definition takes into consideration both components. In particular, the reward r_t is given by the following formula:

$$r_t = (J_t - J_{t+1}) - \alpha W_{t+1}$$

where J_t represents the sum of the jam lengths (in meters) observed over the lanes at time t, and W_{t+1} represents the sum of the maximum waiting times (in seconds) observed over the lanes at time $t+1$. α is a hyper-parameter that determines how much to penalize the agent for letting vehicles wait too much (in our setting $\alpha = 0.4$).

The agent receives a positive reward if the last action performed a_t leads to a state s_{t+1} in which the total queue length has been reduced and/or the maximum waiting times are maintained low.

4.4 Network architecture

The proposed architecture is a Convolutional Neural Network (CNN) that takes as input the state-matrix mentioned before and returns as output the approximations of the optimal Q-values.

The proposed CNN architecture is composed of two convolutional layers and two fully connected layers at the end. In particular, the first convolutional layer contains 16 (2×10) filters with stride (2×1) followed

by a *LeakyReLU* activation function. The second has 32 (1×4) filters with stride (1×2) followed by a *LeakyReLU* activation function and a max pooling layer of size (1×2). The first fully-connected layer has 256 nodes followed by a *LeakyReLU* activation function, while the output layer has 3 linear output neurons (one for each possible green/red lights configuration). In Figure 3 a summary of the CNN architecture is shown.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 16, 3, 300]             976
         LeakyReLU-2          [-1, 16, 3, 300]               0
            Conv2d-3          [-1, 32, 3, 149]           2,080
         LeakyReLU-4          [-1, 32, 3, 149]               0
         MaxPool2d-5           [-1, 32, 3, 74]               0
            Linear-6                [-1, 256]       1,818,880
         LeakyReLU-7                [-1, 256]               0
            Linear-8                  [-1, 3]             771
================================================================
Total params: 1,822,707
Trainable params: 1,822,707
Non-trainable params: 0
----------------------------------------------------------------
```

Figure 3: CNN architecture summary

4.5 Learning algorithm

The proposed model was trained using the Deep Q-Learning (DQL) algorithm. DQL consists in combining Q-Learning to Deep Neural Networks (DNN). Instead of estimating the Q-values for each pair (s, a), DQL resorts to deep neural networks in order to approximate the optimal Q-function.

In particular, the training algorithm proceeds as follows:

1. The agent observes state s_t and passes it as input to the CNN network, which computes the approximations of the optimal Q-values for every action. We denote these values as Q^*
2. The action a_t to be played is selected according to the softmax method. Basically, the values computed by the network are passed through a softmax function which returns a probability distribution in which higher values are associated with higher probabilities and lower values with lower probabilities. Then, a_t is selected by randomly drawing from this distribution. This is done in order to balance exploration and exploitation [1].
3. The agent executes action a_t, ends up in state s_{t+1} and gets a reward r_t.
4. An *instance* of the form $\langle s_t, a_t, r_t, s_{t+1} \rangle$ is stored into a memory D.
5. At training time, a batch of these instances is sampled uniformly from D and used to update the network. This approach is called experience replay, which is used to stabilize the learning algorithm and avoid strong correlations between consecutive samples.
6. The network is updated in the following way: for each $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in the batch:
   (a) the network computes the *prediction*: $Q^*(s_t, a_t)$
   (b) the network computes the *target*: $r_t + \gamma \, max_{a'}(Q^*(s_{t+1}, a'))$
   (c) the loss is computed:

$$\frac{1}{2}\left(r_t + \gamma \, max_{a'}(Q^*(s_{t+1}, a')) - Q^*(s_t, a_t)\right)^2$$

[1] Note that this softmax layer is not part of the network architecture. It is just a probabilistic method to select the next action

7. The mean loss computed over all the batch instances is backpropagated through the network, which is then optimized using ADAM optimizer.

8. Repeat from (1), setting $s_t \leftarrow s_{t+1}$

The model was trained using the hyper-parameters shown in Table 1, typically found in literature.

Parameter	Value
Optimizer	ADAM
Replay memory size	5000
Learning rate	0.001
Batch size	32
Discount factor γ	0.9
State matrix size	$3 \times 6 \times 309$
Epochs	45

Table 1: Agent's hyper-parameters

5 Experiments

5.1 Simulation setup

The considered intersection (Figure 1) is composed of three incoming roads, each with two lanes. In order to simulate real-life scenarios, this crossroads was drawn similarly to a real intersection located in Como at the following coordinates: (45.802155, 9.084961). The two main roads have lane lengths of $309m$ and $211m$ respectively, while the minor road has a lane length of $103m$. The max speed is $13.9m/s$, which is equal to $50km/h$, for all the roads.

On each lane, vehicles can travel following different routes through the intersection. Since we did not find a dataset of the traffic flows of Italian roads, we set the traffic flow rate to 350 vehicles per hour on each route. A scheme of the routes that vehicles can travel is shown in Figure 2. As it can be seen, the East incoming road has 4 different routes, therefore, the traffic on that road will be higher than the others. Each incoming road has a traffic light that manages the vehicles on the lanes. The set of the possible green/red lights configurations at the intersection is obtained by combining individual phases of these three traffic lights.

The minimum green/red-light phase duration is fixed at 10 simulation steps (10 seconds in the simulation environment), while the yellow-light phase duration between two neighboring phases is fixed at 5 seconds. These two fixed lengths determine how many simulation steps SUMO can run before letting the model take a new action. With this configuration, the green-light phase is guaranteed to be of at least 10 seconds.

For simplicity, we chose to generate only one vehicle's type. In particular, each vehicle length is 5 meters. After 500 simulation steps, the system stops generating vehicles and the simulation ends. The proposed model was trained for 45 epochs, where each epoch is composed by 5 complete SUMO simulations.

5.2 Results

As we said before, the proposed model was assessed with respect to two common traffic metrics: queue length and vehicles waiting time. The performance of the proposed model were compared to those of the following baseline models:

- A Multi Layer Perceptron (MLP) network with one fully-connected hidden layer of 80 nodes, followed by a *ReLU* activation function, and 3 linear output neurons. The input for the MLP consists in a vector containing the information about the current phase, the queue length (in meters) on each lane, and the maximum time (in seconds) a vehicle has waited on each lane at the intersection. The MLP was trained with the same hyper-parameters and optimization method used for the CNN.
- A simple static configuration of the traffic lights signals, in which all the phases cyclically change in a fixed sequence and each green/red-light phase has a fixed duration of 25 seconds. The yellow-light duration is still 5 seconds.
- Two models which implement the *most waiting first* (MWF) heuristic and *longest queue first* (LQF) heuristic. The first model sets a green light to the lanes that have waited the most up to the current simulation step. The second model, instead, sets a green light to the lanes in which the longest queues were observed. For both models, the green/red-light duration and the yellow-light duration are the same as the CNN model.

In particular, we compared the average queue length and the average waiting time obtained by each solution. Note that the average waiting time at the intersection is computed by averaging the maximum waiting times observed on each lane during the simulation.

In Figure 4 a comparison between the average rewards obtained by the CNN and the MLP models on each epoch is shown (red line and blue line respectively). The learning process seems to be more stable for

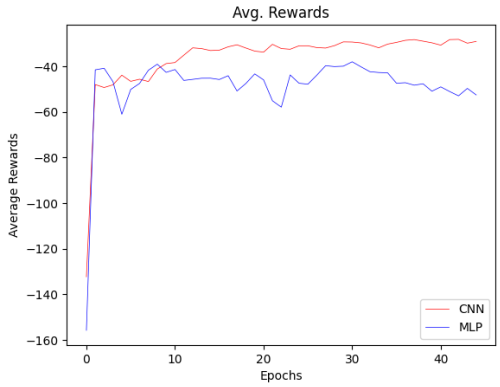


Figure 4: Average rewards obtained by the CNN and the MLP on each epoch.

the CNN-based agent, which performs significantly better than the baseline.

In Figure 5 the red line illustrates the average queue lengths obtained by the CNN-based agent during the training phase, while the blue line is related to the MLP-based model. The plot clearly shows that the CNN-based agent is able to learn an action policy that minimizes the queue length at the intersection in a more effective way than the baseline. The same is also valid for the average waiting times observed over the training epochs shown in Figure 6.

In Table 2 some statistics of the best CNN-based agent are compared to the ones of the baselines models mentioned before [2]. Particularly interesting are the average values of queue length and waiting time, which effectively resume the general behaviour of different models. It is clear that the proposed model outperforms every baseline, providing less average waiting time and queue length.

[2]Each model was tested by running a complete SUMO simulation

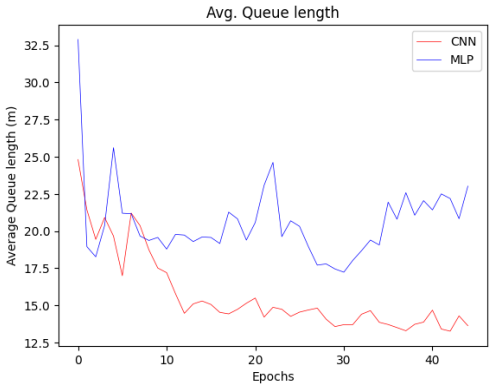


Figure 5: Average queue length in meters obtained by the CNN and the MLP on each epoch.

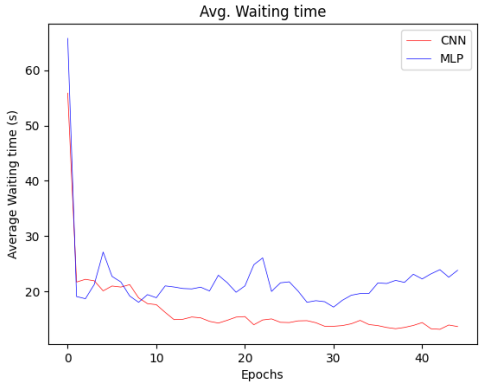


Figure 6: Average waiting time in seconds obtained by the CNN and the MLP on each epoch.

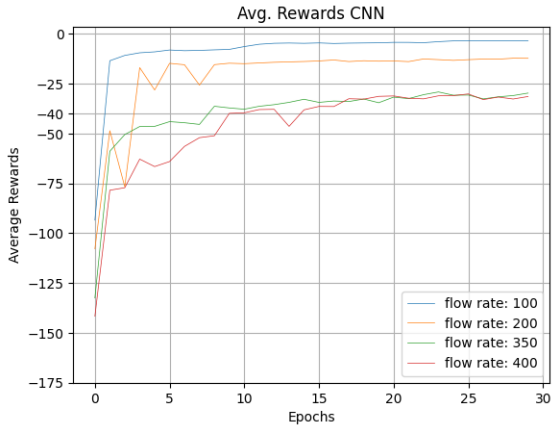
Finally, in order to assess whether the CNN-based agent concretely brings significant improvements with respect to the MLP-based one, we trained both models using different traffic flow rates. Figures 7a, 7b show the average rewards obtained by training both models in 4 different simulation setups. Each setup is equivalent to the one presented before, except for the traffic flow rate. In particular, the experienced traffic flow rates are 100, 200, 350, 400 vehicles per hour on every route. Under low traffic conditions (100, 200 vehicles per hour) the two models perform similarly. However, the CNN-based agent scales significantly better with increasing traffic intensity than the MLP model, showing that the proposed model is more robust and can deal with more complex and real scenarios.

6 Future improvements

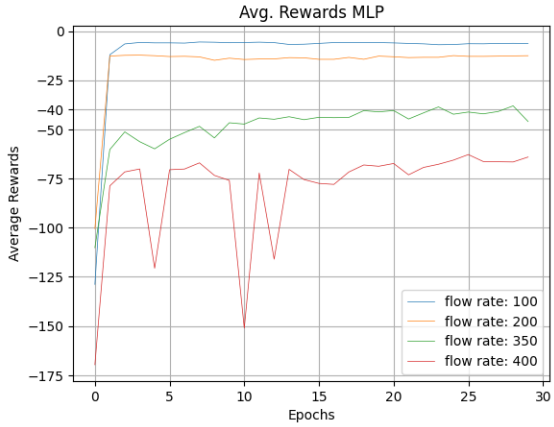
Future developments for this project may regard the implementation of an agent able to deal with traffic flows based on measured data in real urban situations. Another improvement may regard a generalization of the agent to deal with different vehicles' types. Finally, you could test deeper architectures and/or different learning algorithms (e.g., n-step Q-learning) and observe their effect on performance.

Model	Max queue length $[m]$	Max waiting time $[s]$	Avg. queue length $[m]$	Avg. waiting time $[s]$
CNN	**79.38**	**72**	**14.79**	**13.53**
MLP	124.08	131	19.08	19.48
MWF	103.15	102	21.24	21.00
LQF	101.81	156	27.20	31.61
Static	199.32	159	28.04	28.60

Table 2: Comparison of most important statistics for the analyzed models. CNN and MLP's values are obtained by testing the models that got the highest average reward during the training phase.



(a)



(b)

Figure 7: Plots of the average rewards obtained by training the CNN and MLP models considering different traffic flow rates.

References

[1] Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. Reinforcement learning for true adaptive traffic signal control. *Journal of Transportation Engineering*, 129(3):278–285, 2003.

[2] Itamar Arel, Cong Liu, Tom Urbanik, and Airton G Kohls. Reinforcement learning-based multi-agent

system for network traffic signal control. *IET Intelligent Transport Systems*, 4(2):128–135, 2010.

[3] Yit Kwong Chin, Lai Kuan Lee, Nurmin Bolong, Soo Siang Yang, and Kenneth Tze Kin Teo. Exploring q-learning optimization in traffic signal timing plan management. In *2011 third international conference on computational intelligence, communication systems and networks*, pages 269–274. IEEE, 2011.

[4] Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016.

[5] Xiaoyuan Liang, Xunsheng Du, Guiling Wang, and Zhu Han. Deep reinforcement learning for traffic light control in vehicular networks. *arXiv preprint arXiv:1803.11115*, 2018.

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

[7] Nandan Maiti and Bhargava Rama Chilukuri. Traffic signal control for an isolated intersection using reinforcement learning. In *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pages 629–633. IEEE, 2021.

[8] Seyed Sajad Mousavi, Michael Schukat, and Enda Howley. Traffic light control using deep policy-gradient and value-function-based reinforcement learning. *IET Intelligent Transport Systems*, 11(7):417–423, 2017.

[9] Mohamad Belal Natafgi, Mohamad Osman, Asser Sleiman Haidar, and Lama Hamandi. Smart traffic light system using machine learning. In *2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*, pages 1–6. IEEE, 2018.

[10] Elise Van der Pol and Frans A Oliehoek. Coordinated deep reinforcement learners for traffic light control. *Proceedings of learning, inference and control of multi-agent systems (at NIPS 2016)*, 8:21–38, 2016.