Deep convolutional Q-learning for traffic lights optimization in Smart Cities

Riccardo Cappi[1], Sebastiano Monti[1] and Davide Tosi[2]

[1]University of Padova, Padova - Italy
[2]Università degli studi dell'Insubria, Varese - Italy

Abstract

Autonomous traffic control is an important and active field of research that could potentially lead to remarkable improvements in congestion management and consequent delay and air pollution reductions. In this paper, we propose a deep reinforcement learning model to achieve autonomous traffic lights control at an intersection in a simulated environment. The model consists of a Convolutional Neural Network (CNN) that takes as input an image-like representation of the traffic state and is trained, using the Deep Q-Learning algorithm (DQL), to maximize a reward function based both on the decrease in queue length and maximum waiting times. We show that this approach reduces average waiting time and average queue length when compared to several baselines, such as a multi-layer perceptron architecture with a simpler state space representation and four non-parametric models, which implement the most waiting first heuristic, the longest queue first heuristic, an actuated traffic control scheme, and a simple static configuration of the traffic lights, respectively. The designed approach suggests its applicability in future smart cities for real traffic light control systems.

Keywords

Reinforcement Learning, Deep Q-learning, Traffic lights, Convolutional Neural Networks, Smart Cities

1. Introduction

The advancement of smart technologies during the past 10 years, such as IoT devices, big data analytics, and artificial intelligence methods, has led to the emergence of Smart Cities. One of the key components of the smart urban environment is the optimization of urban vehicle transportation, directly impacting traffic congestion, costs, and emissions [1]. Two types of solutions are possible to address this challenge. The least efficient one, in terms of costs and durability, consists in the expansion of road infrastructures, while the most functional one involves increasing the efficiency of already existing infrastructures, such as traffic light signals at intersections [2]. The latter can be implemented through several algorithms, such as static traffic light phases or vehicle-actuated signal control. However, the most promising techniques for adaptive signal control seem to be based on Reinforcement Learning (RL) [3]. This paper aims at implementing a RL-based agent able to dynamically control the traffic light phases of an intersection in order to minimize jam lengths and vehicles' waiting times. In particular, we implemented a Convolutional Neural Network (CNN), trained using the Deep Q-Learning (DQL) algorithm, which takes as input an image-like representation of the traffic state. We employed a state space definition that combines discrete traffic state encoding (DTSE) [2] with vehicles' waiting times in order to consider both space and time information. We also defined a reward function according to the best-performing approaches proposed in literature, which involves both the variation in queue length and waiting times. We evaluated the performance of our model by comparing it with that of different baselines, such as a multi-layer perceptron architecture with a simpler state space representation and four heuristic-based models. We show that our approach performs better than the baselines in reducing the average queue length and the average waiting time at the considered intersection.

ATT'24: Workshop Agents in Traffic and Transportation, October 19, 2024, Santiago de Compostela, Spain
riccardo.cappi@studenti.unipd.it (R. Cappi); sebastiano.monti@studenti.unipd.it (S. Monti); davide.tosi@uninsubria.it (D. Tosi)
https://github.com/riccardocappi/Deep_Reinforcement_Learning_For_Traffic_Lights (R. Cappi)
0000-0002-3718-5892 (R. Cappi)

The next sections are organized as follows: Section 2 summarizes the most common algorithms and methodologies present in literature in the field of adaptive traffic lights control. Section 3 briefly describes the reinforcement learning paradigm. Section 4 defines the components of the operating environment in which the agent works, such as the performance measures and the employed simulation software. Section 5 provides details regarding the state space, action space and reward function, as well as describing the learning algorithm and network architecture. Section 6 details the experimental setup and the obtained results, while Section 7 summarizes the conducted research.

2. Related works

A lot of research has been done using reinforcement learning to build adaptive traffic signal control systems. These works mainly differ in the state representation of the environment, the action space of the agent and the reward function. Authors in [4][5] defined the state representation on the basis of queue length of different incoming roads, while in [6] 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, authors in [2] pointed out that these abstract representations of the traffic state may omit relevant information and lead to suboptimal solutions. For this reason, other works employed an image-like representation by defining a Boolean-valued matrix whose cells can contain a value of one, indicating the presence of a vehicle, or zero, indicating its absence [7]. In [2][8], this matrix is further combined with another that indicates vehicles' speed at the intersection. In this paper, instead, we aim at developing a model able to automatically learn high-level state representations without providing as input too many handcrafted features. To this purpose, we implemented a convolutional neural network that takes as input an image-like representation of the traffic state, exploiting the idea mentioned above. However, we propose a state definition that takes into consideration both the position and the waiting times of vehicles, and additionally uses a stack of consecutive simulation frames to make the model able to implicitly estimate vehicles' velocity and travel direction, following the idea proposed in [9].

An important aspect of reinforcement learning for traffic lights control is how the action space is defined. Previous works proposed two different possibilities: (1) authors in [8] proposed a system in which all the phases cyclically change in a fixed sequence to guide vehicles through the intersection. In that system, the agent's action is to select the phase duration in the next cycle. (2) On the other hand, most of the previous research defined the action space as the set of possible signal phase configurations (i.e., all the allowed green/red light configurations at the intersection) [7][9][2]. In this scenario, the agent's action consists in selecting which lanes get a green light by choosing one of the allowed green/red light settings. Since the agent does not optimize the duration of each phase, green/red light timings can only be a multiple of a fixed-length interval. We chose to use the second action space definition, as it seems to be the most popular.

Another key component is the reward function. A lot of reward definitions have been proposed in literature, such as change in cumulative vehicle delay [10][9] and change in number of queued vehicles [5]. However, authors in [6] suggest to define a reward function that is based both on the decrease in queue length and on the decrease in vehicles' waiting times. This approach is also proposed in [11], where the results show that if the reward is exclusively based on queue length metrics, the model could leave some cars wait for an indefinite period of time. Therefore, in order to avoid situations of this kind, we decided to design our reward function following the latter approach.

3. Background

In a reinforcement learning setting, an agent interacts with the environment to get rewards from its 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 an immediate reward $r_t = R(s_t, a_t)$, that can be positive, negative or neutral;
- the state changes to $s_{t+1} = \delta(s_t, a_t)$.

Assuming 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. For each possible policy π the agent might adopt, we can define an evaluation function over states:

$$V^{\pi}(s) = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \tag{1}$$

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 (1) for all states s:

$$\pi^* = argmax_{\pi} V^{\pi}(s), \ (\forall s). \tag{2}$$

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:

$$Q(s, a) = R(s, a) + \gamma V^{\pi^*}(\delta(s, a))$$

where $\delta(s, a)$ denotes the state resulting from applying action a to state s. Then, we can reformulate (2) 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 V^{π^*}, it will be able to select optimal actions even if it has no knowledge of R and δ.

4. Operating environment

In this section, we define the operating environment in which the agent works.

Simulation environment: since it is difficult to retrieve real traffic data and perform real-world experimentation, we relied on SUMO [12], an open source traffic simulator that makes it possible to model real-world traffic behavior. This software, through an API called TraCI, provides complete control over the simulation environment elements, such as vehicles' speed and position, traffic flow's intensity on each lane, traffic light phases, the shape of the intersection, etc.

Performance measures: the performance of the agent is assessed with respect to two common traffic metrics: queue length and vehicles' waiting times. The goal is to find a model able to dynamically control the traffic lights of an intersection in order to minimize these two metrics.

Although dynamic traffic light control is an extremely complex task in the real world, SUMO allows you to operate in a more controlled environment. Specifically, the agent works in a fully-observable environment, since the software gives access to its complete state at each point in time. For this paper, we also defined a deterministic environment by setting a non-stochastic traffic flow generation. This makes the analysis simpler, but it is also one of the biggest limitations of this work. Clearly, the environment is also sequential and single agent.

5. 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.

5.1. State space

We propose a state representation that takes into consideration both vehicles' positions and waiting times. The idea is to map each lane approaching the intersection into a Boolean-valued vector, where each cell can contain a 1, indicating the presence of a vehicle at that position, or a 0, indicating its absence. Each cell of the vector corresponds to 1 meter of the lane. The matrix of vehicles' positions is then obtained by stacking all the lane vectors. Given an intersection with l lanes, where the longest lane is m meters, this intermediate state representation s' consists of a $(l \times m)$ matrix. Note that a zero-padding is added to lane vectors with a length less than the maximum length lane in order to have all equally-sized vectors.

Then, the s' representation is enriched by using a stack of consecutive simulation frames to make the model able to implicitly estimate vehicles' velocity and travel direction. In particular, s' is computed for the last p ($p = 2$ in our setting) simulation steps, yielding a new $(p \times l \times m)$ matrix, denoted as s''.

The s'' representation built so far consists of a Boolean-valued matrix that contains the information about vehicles' positions of the last p simulation steps. However, it does not take into consideration the waiting times. This information is embodied in the representation by computing another state matrix, whose cells contain the normalized values of the vehicles' waiting times of the last simulation step. Then, the final state representation s is a $(p + 1 \times l \times m)$ matrix.

5.2. Action space

To handle traffic at the intersection, the agent selects which lanes get a green light according to a set of three possible green/red light configurations. On each of the three incoming roads there is a traffic light that 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 light configurations. In Figure 1, all the three

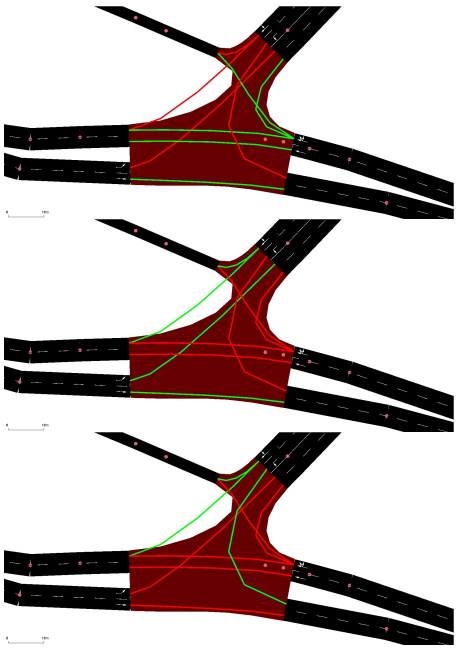

Figure 1: Possible phase configurations that can occur at the intersection

possible signal phases that can occur at the considered intersection are shown. 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.

5.3. Reward function

The proposed definition of the reward function takes into account both the variation in queue length and waiting times. 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} with shorter total queue length and/or low maximum waiting times.

5.4. Network architecture

The proposed architecture is a convolutional neural network that takes as input the state matrix mentioned in Section 5.1 and returns as output an approximation of the optimal Q-values. The model is composed of two convolutional layers and two fully connected layers at the end. In particular, the first convolutional layer consists of 16 (2×10)-filters with stride (2×1) followed by a *LeakyReLU* activation function. The second layer 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 light configuration). In Figure 2, a summary of the CNN architecture is shown. We designed the convolutional kernels so that, ideally, they compute high-level representations of each road separately. Then, the joint information among the different roads is merged by the network in the last two fully connected layers.

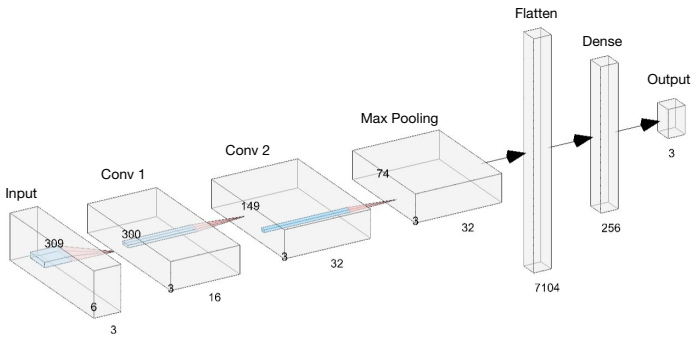

Figure 2: CNN architecture summary

5.5. Learning algorithm

The proposed model was trained using the Deep Q-Learning (DQL) algorithm, shown in Algorithm 1, which consists in combining Q-Learning with Deep Neural Networks (DNN). The employed hyper-parameters are shown in Table 1, which are typically found in literature.

6. Experiments

6.1. Simulation setup

The considered intersection (Figure 1) is composed of three incoming roads, each with two lanes. In order to simulate real-life scenarios, the intersection was designed similarly to a real one located in Como (IT) at the following coordinates: $(45.802155,\ 9.084961)$. The two main roads' lengths are

Algorithm 1 Deep Q-Learning with Experience Replay

procedure DQL FOR TRAFFIC LIGHTS CONTROL
  Initialize replay memory D to capacity L
  Initialize policy network Q with random weights θ
  Initialize target network $\hat{Q}$ with random weights $\theta^- = \theta$
  Create simulation environment env
  $epoch \leftarrow 0$
  $episode \leftarrow 0$
  while $epoch < N$ **do**
    $s_1 \leftarrow env$.reset()
    $episode \leftarrow episode + 1$
    for $t = 1, T$ **do**
      Select action a_t sampling from $softmax(Q(s_t; \theta))$
      $s_{t+1}, r_t \leftarrow env$.step($a_t$)
      Store transition $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in D
      Sample a mini-batch of transitions $\langle s_j, a_j, r_j, s_{j+1} \rangle$ uniformly from D
      if s_{j+1} is terminal **then**
        $y_j \leftarrow r_j$
      else
        $y_j \leftarrow r_j + \gamma \max_{a'} \hat{Q}(s_{j+1}, a'; \theta^-)$
      $loss = smooth\_L1\_loss(y_j, Q(s_j, a_j; \theta))$
      Optimize θ, using ADAM, according to $loss$
      $\theta^- \leftarrow \tau\, \theta + (1 - \tau)\, \theta^-$
    if $episode$ mod 5 = 0 **then**
      $epoch \leftarrow epoch + 1$
      Evaluate Q

Table 1
Agent's hyper-parameters

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

$309m$ and $211m$ respectively, while the minor road's length is $103m$. The max speed is $13.9m/s$, which is equal to $50km/h$, on each road. On each lane, vehicles can travel following different routes through the intersection. Due to the difficulty of finding a dataset of the traffic flows of Italian roads, we set the traffic flow rate to 450 vehicles per hour on each route. A scheme of the routes that vehicles can travel is shown in Figure 1. We can observe that the east incoming road has 4 different routes; therefore, the traffic on that road will be higher than on the others. 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's 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 of 5 complete SUMO simulations.

6.2. Results

As we said before, the proposed model was assessed with respect to two common traffic metrics: queue length and vehicles' waiting times. We compared the performance of the proposed model with that of the following baselines:

- 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 of the MLP consists of 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, following the approach proposed in [6]. The MLP was trained with the same hyper-parameters and optimization method used for the CNN.
- Two traffic control systems provided by default by SUMO: (1) the first one is a simple Static configuration of traffic light 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, while the yellow-light duration is still 5 seconds. (2) The second system is the default implementation of the gap-based Actuated traffic control scheme, which dynamically adjusts traffic light phases' durations whenever a continuous stream of traffic is detected.
- Two models that implement the *most waiting first* (MWF) heuristic and *longest queue first* (LQF) heuristic. The first model sets a green light to lanes in which vehicles waited the most, up to the current simulation step. The second model, instead, sets a green light to 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.

Table 2
Performance comparison of the analyzed models. CNN and MLP's values are obtained by testing the models that got the highest average reward during the training phase.

Model	Max queue length $[m]$	Max waiting time $[s]$	Avg. queue length $[m]$	Avg. waiting time $[s]$
CNN	**87.54**	**70**	**15.78**	**13.49**
MLP	124.33	159	20.71	19.41
MWF	181.21	157	38.76	36.67
LQF	140.01	241	36.65	41.17
Static	199.43	214	31.15	30.94
Actuated	178.21	117	27.67	26.14

Table 2 shows the performance of the tested models. It is clear that the proposed agent performs better than every baseline, providing less average waiting time[1] and less average queue length. We can also see that the non-parametric methods such as MWF, LQF, Actuated and Static heuristics perform dramatically worse than the RL-based agents. Therefore, we continue the analysis by exploring the differences between the two neural network models.

In Figure 3, a comparison between the average rewards obtained by the CNN and the MLP on each epoch is shown (red line and blue line, respectively). The learning process seems to be more stable for the CNN-based agent, which performs better than the baseline. However, we can observe a rapid increase in the rewards obtained by the MLP agent at the end of the training. This suggests that, even

[1] the average waiting time at the intersection is computed by averaging the maximum waiting times observed on each lane during the simulation

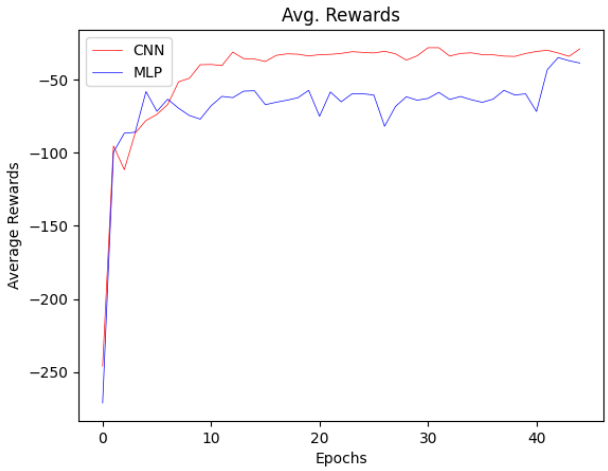

Figure 3: Average rewards obtained by the CNN and the MLP over the epochs.

if the CNN model provides better results in this experiment, the MLP does not perform dramatically worse. The same result can be deduced by looking at the average queue lengths and average waiting times obtained by the two architectures over the epochs, shown in Figure 4. For this reason, in order to

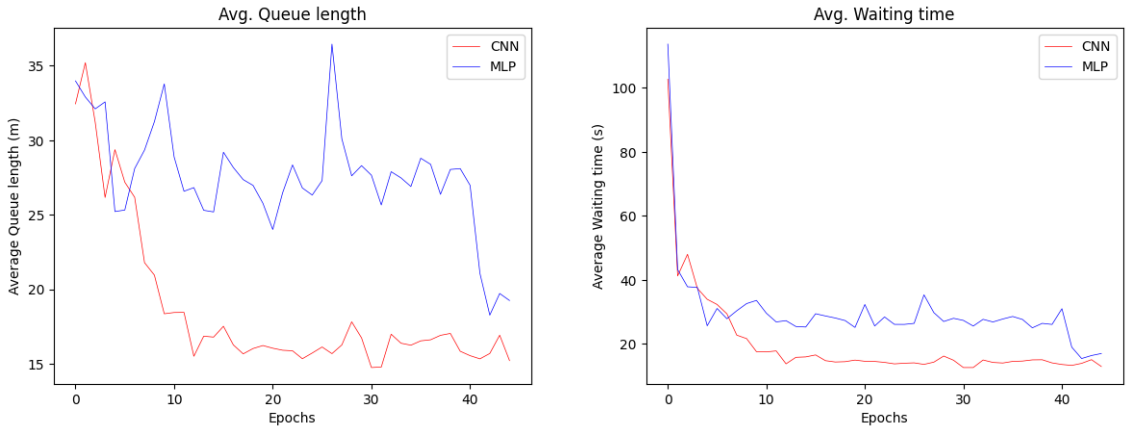

Figure 4: Average queue length (meters) and average waiting time (seconds) obtained by CNN and MLP on each epoch

assess whether the CNN-based agent concretely brings significant improvements with respect to the MLP-based one, we compared both models by training them under different traffic conditions. Figure 5 shows the box plots of the average rewards obtained by training both models in 4 different simulation setups, featuring increasing traffic intensities. Each setup is equivalent to the one presented in Section 6.1, with 350, 450, 550 and 700 vehicles per hour, respectively. The results show that, under low traffic conditions, the two models perform very similarly. However, the CNN-based agent scales better than the baseline with increasing traffic intensity, showing that the proposed model is more robust and can deal with more complex scenarios.

7. Conclusions

Smart cities and the planet urgently ask for environmental emissions to be reduced while improving the quality of life for citizens. To this end, Artificial Intelligence can provide researchers with instruments and tools to help this virtuous process. In this paper, a new CNN-based approach has been designed and tested to improve queue length and vehicle waiting times for traffic light control systems. The

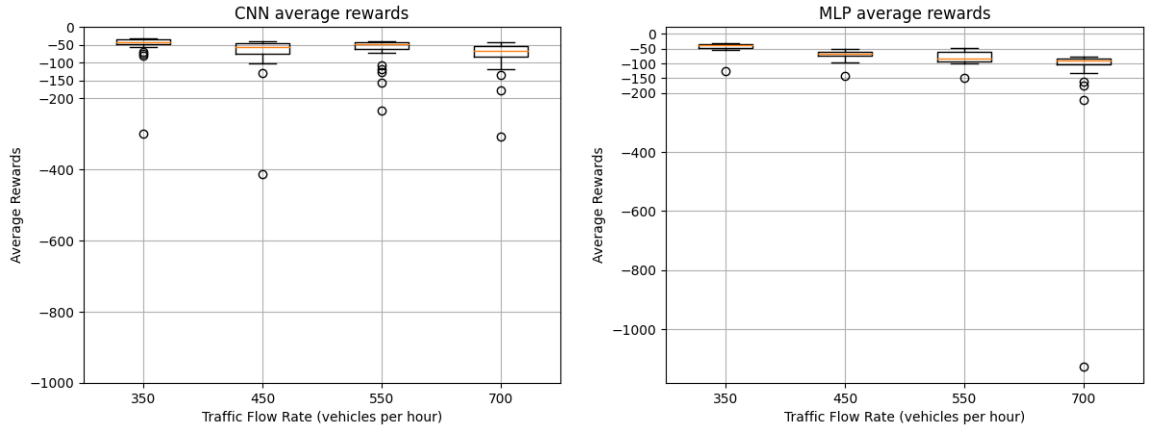

Figure 5: Average rewards obtained by training CNN and MLP agents considering different traffic flow rates

proposed approach has been extensively experimented against five baseline models. The results show that CNN models perform better than baselines. This opens the possibility of testing our approach in real-life conditions and in future Smart Cities that will exploit intelligent traffic light control systems.

References

[1] D. Tosi, Cell phone big data to compute mobility scenarios for future smart cities, International Journal of Data Science and Analytics 4 (2017) 265–284. URL: https://doi.org/10.1007/s41060-017-0061-2. doi:10.1007/s41060-017-0061-2.

[2] W. Genders, S. Razavi, Using a deep reinforcement learning agent for traffic signal control, arXiv preprint arXiv:1611.01142 (2016).

[3] R. Chen, F. Fang, N. Sadeh, The real deal: A review of challenges and opportunities in moving reinforcement learning-based traffic signal control systems towards reality, arXiv preprint arXiv:2206.11996 (2022).

[4] Y. K. Chin, L. K. Lee, N. Bolong, S. S. Yang, K. T. K. Teo, Exploring q-learning optimization in traffic signal timing plan management, in: 2011 third international conference on computational intelligence, communication systems and networks, IEEE, 2011, pp. 269–274.

[5] N. Maiti, B. R. Chilukuri, Traffic signal control for an isolated intersection using reinforcement learning, in: 2021 International Conference on COMmunication Systems & NETworkS (COMSNETS), IEEE, 2021, pp. 629–633.

[6] M. B. Natafgi, M. Osman, A. S. Haidar, L. Hamandi, Smart traffic light system using machine learning, in: 2018 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET), IEEE, 2018, pp. 1–6.

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

[8] X. Liang, X. Du, G. Wang, Z. Han, Deep reinforcement learning for traffic light control in vehicular networks, arXiv preprint arXiv:1803.11115 (2018).

[9] S. S. Mousavi, M. Schukat, E. Howley, Traffic light control using deep policy-gradient and value-function-based reinforcement learning, IET Intelligent Transport Systems 11 (2017) 417–423.

[10] I. Arel, C. Liu, T. Urbanik, A. G. Kohls, Reinforcement learning-based multi-agent system for network traffic signal control, IET Intelligent Transport Systems 4 (2010) 128–135.

[11] B. Koohy, S. Stein, E. Gerding, G. Manla, Reward function design in multi-agent reinforcement learning for traffic signal control (2022).

[12] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken,

J. Rummel, P. Wagner, E. Wießner, Microscopic traffic simulation using sumo, in: The 21st IEEE International Conference on Intelligent Transportation Systems, IEEE, 2018. URL: https://elib.dlr.de/124092/.