Heterogeneous Multi-Agent Deep Reinforcement Learning for Traffic Lights Control

Jeancarlo Arguello Calvo, Ivana Dusparic

School of Computer Science and Statistics, University of Dublin, Trinity College
arguellj@tcd.ie, ivana.dusparic@scss.tcd.ie

Abstract. The most promising technique used in urban traffic control (UTC) has been Reinforcement Learning (RL) due to its capacity to learn the dynamics of complex problems. The main problem of this approach is the curse of dimensionality that arises from the exponential growth of the state and action spaces because of the number of intersections.

Deep Reinforcement Learning (DRL) enhances hugely the performance of RL. DRL has proved to work very well for UTC in single agent environments. Nonetheless, when the problem scales up to multiple intersections the need for coordination becomes too complex, and as a result, the latest studies take advantage of the similarity of agents to train them as one. However, the usage of homogeneous junctions is not a real world scenario where a city has different layouts of intersections.

This study proposes the usage of Independent Deep Q-Network (IDQN) to train heterogeneous multi-agents to deal with both the curse of dimensionality and the need for collaboration. The former is handled by using Deep Q-Networks. The latter is managed with IDQN, which trains simultaneously and separately each agent allowing us to support heterogeneous agents. Unfortunately, this technique can lead to convergence problems because one agent's learning makes the environment appear non-stationary to other agents, and this problem conflicts with experience replay memory on which DQN relies. We address this issue by conditioning each agent's value function on a fingerprint that disambiguates the age of the data sampled from the replay memory.

1 Introduction

This research addresses heterogeneous multi-agents for urban traffic control (UTC) systems as the first study applying Deep Reinforcement Leaning (DRL) in a heterogeneous network layout which is a reflection of the cities in the real world. It proposes the usage of DRL to deal with the curse of dimensionality that suffers RL approaches by using Neural Networks. It also uses Indepedent Q-Learning (IQL), where each agent learns independently and simultaneously its own policy, treating other agents as part of the environment. However, the environment becomes nonstationary from the point of view of each agent, as it involves the interaction with other agents who are themselves learning at the same time, ruling out any convergence guarantees. The technique used for this problem is Deep Q-Networks which relies in a component so called experience replay memory in order to stabilize and improve the learning. However, this memory is incompatible with non-stationary environments. In order to combine the experience replay memory and IQL, we used a technique of fingerprinting which stabilizes the memory against the non-stationarity. This fingerprint disambiguates the age of the data sampled from the replay memory. We evaluate the usage of IQL with Deep Q-Networks in a multi-agent setting by using a simulation of an urban traffic control system.

2 Background and Related Work

2.1 Reinforcement Learning

Reinforcement learning (RL) is about learning optimal actions from specific situations where the agent has to discover which actions maximize a reward signal by exploring them in a trial-and-error basis. These actions may affect future situations and subsequent reward signals [12].

In RL an agent interacts with the environment. On each time step t, the agent perceives an state s_t in state space S from where it selects an action a_t in the action space A by following a policy π. The agent receives a reward r_t when it transitions to the state s_{t+1} according to the environment dynamics, the reward function *R(*s_t*,* a_t*,* s_{t+1}*)* and the transition function *T(*s_t*,* a_t*,* s_{t+1}*)*. In order to converge the algorithm faster, a discount factor $\gamma \in [0, 1]$ is applied to the total amount of reward.

RL can be either model-based or model-free (where the transition and the reward functions are unknown). The most common model-free technique is Q-Learning, where RL agents learn Q-values which are functions of state-action pair that returns a real value: *Q:* S x $A \rightarrow \mathbb{R}$. Thus, the policy is represented as:

$$\pi(s) = \arg\max_{a \in A} Q(s,a) \tag{1}$$

where the Q-value can be learned by using Q-learning updates [18]:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[R(s,a) + \max_{a \in A} Q(s',a')] \tag{2}$$

where $0 < \alpha \leq 1$ is a learning rate.

2.2 Deep Reinforcement Learning

The curse of dimensionality given by large state and action spaces make unfeasible to learn Q value estimates for each state and action pair independently as in normal tabular Q-Learning. Therefore, DRL models the components of RL with deep neural networks. The following subsections describe the methods used during the proposed research to implement the Deep Reinforcement Learning in Traffic Light problem.

Deep Q Networks [11] proposed Deep Q-Networks (DQN) as a technique to combine Q-Learning with deep neural networks. RL is known to be unstable or even to diverge when a non-linear function approximator such as a neural network is used to represent the Q value. DQN addresses these instabilities by using two insights, experience replay and target network.

DQN parameterizes an approximate value function Q(s, a; θ_i) using Convolutional Neural Networks, where θ_i are the weights of the network at iteration i. The experience replay stores the agent's experiences $\mathrm{e}_t = (\mathrm{s}_t, \mathrm{a}_t, \mathrm{r}_t, \mathrm{s}_{t+1})$ at each time step t in a dataset $\mathrm{D}_t = \mathrm{e}_1,,\mathrm{e}_t$ pooled over many episodes into a replay memory. Then, mini batches of experience drawn uniformly at random from the dataset (s, a, r, s) $\sim$ U(D) are applied as Q-updates during the training. The Q-learning update at iteration i follows the loss function:

$$L_i(\theta_i) = \ E_{(s,a,r,s)\sim U(D)}\left[\left(r + \gamma \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)^2\right] \quad (3)$$

where θ_i are the parameters of the Q-network at iteration i and θ_i^- are the target network parameters. The target network parameters are only updated with the Q-network parameters every C steps and are held fixed between individual updates.

Double DQN The max operator in standard Q-learning and DQN uses the same values both to select and to evaluate an action. It is known the this maximization sometimes produces to learn unrealistically high action values which tends to prefer overestimated values over underestimated values, resulting in overoptimistic value estimations. To prevent this, Double Q-learning decouples the selection and the evaluation [7].

For DQN architectures is not desired to fully decoupled the target as in [7] because the target network provides a intuitive option for the second value function, without having to include extra networks. For that reason, [6] propose to evaluate the greedy policy according to the online network, but using the target network to estimate its value given as result the Double DQN (DDQN) algorithm:

$$Y_t^{DDQN} = R_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_i^-) \quad (4)$$

Prioritized Experience Replay [13] presented prioritized experience replay which detaches agents from considering transitions with the same frequency that they are experienced. Prioritized replay samples more frequently transitions from which there is a high expected learning progress. as measured by the magnitude of their temporal-difference (TD) error. It samples transitions with probability p_t relative to the last encountered absolute TD error:

$$p_t \propto \left|\left(R_{t+1} + \gamma_{t+1} \max_{a'} Q\left(s', a'; \theta_i^-\right) - Q\left(s, a; \theta_i\right)\right)\right|^\omega \quad (5)$$

Where ω is a hyper-parameter that determines the pattern of the distribution. New transitions are pushed into the replay buffer memory with maximum priority, providing a bias towards recent transitions.

Dueling Network Dueling Network is a technique proposed by [17] which computes separately the value V(s) and advantage A(s, a) functions that are represented by a duelling architecture that consists of two streams where each stream represents one of these functions. These two streams are combined by an convolutional layer to produce an estimate of the state-action value Q(s, a). The dueling network automatically produces separate estimates of the state value and advantage functions without supervision. Besides that, it can learn which states are valuable, without having to explore the consequence of each action for each state. Dueling network is defined with the equation:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A\left(s,a;\theta,\alpha\right) - \frac{1}{|\mathcal{A}|}\sum_{a'} A\left(s,a;\theta,\alpha\right) \right) \tag{6}$$

2.3 Multi-Agent Reinforcement Learning

Consider a cooperative multi-agent environment where n agents identified by a ϵ A $\equiv$ {1,..., n} participate in a stochastic game G, denoted by a tuple <S, U, P, r, Z, O, n, γ >. The environment consists of states s$_t$ ϵ S, where at every time step t, each agent takes an action u_t^a ϵ U, forming a joint action $\mathbf{u}_t$ ϵ $\mathbf{U}$ $\equiv$ U^n. The transition probabilities are defined by P(s$_{t+1}$ | s$_t$, u$_t$) : S x U x S $\rightarrow$ [0, 1]. A global reward function r(s$_t$, u$_t$) : S x U $\rightarrow$ $\mathbb{R}$ is shared between all the agents.

Each agent's observations z ϵ Z follows an observation function O(s$_t$, a) : S x A $\rightarrow$ Z. Each agent a conditions its behaviour on its own action-observation history τ_a ϵ T $\equiv$ (Z x U)*, according to its policy π_a(u$_a$ | τ_a) : T x U $\rightarrow$ [0, 1]. After each transition, the action u$_t^a$ and new observation O(s$_t$, a) are added to τ_a, forming $\tau_a^{'}$. Let the joint quantities over agents be in bold, and the joint quantities over agents other than a with the subscript -a, so that, e.g., u = [u$_t^t$, $\mathbf{u}_{-a}$].

Independent DQN Independent DQN (IDQN) is an extension of Independent Q-Learning[15] with DQN for cooperative multi-agent environment, where each agent a observes the partial state s$_t^a$, selects an individual action u_t^a, and receives a team reward, r$_t$ shared among all agents. [14] combines DQN with independent Q-learning, where each agent a independently and simultaneously learns its own Q-function $Q^a(s, u^a; \theta_i^a)$ [3]. Since our setting is partially observable, IDQN can be implemented by having each agent conditioned on its action-observation history, i.e., Q$_a$(τ_a, u$_a$). In DRL, this can be implemented by given to each agent a DQN on its own observations and actions.

As mentioned before, a key component of DQN is the experience replay memory. Unfortunately, the combination of experience replay with IQL appears to be problematic because the non-stationarity introduced by IQL which provokes that the dynamics that generated the data in the agent's experience replay memory no longer indicate the current dynamics in which the agent is learning.

A fingerprint technique was introduced in [4], which states that the disadvantage of IQL is that it ignores the changing over time of the other agents' policies because it perceives the other agents as part of the environment, which causes non-stationarity on its own Q-function. Hence, the Q-function might be made stationary if it conditioned on the other agents' policies.

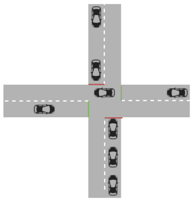
Based on the results of the study, a fingerprint must be correlated with the true value of state-action pairs given the other agents' policies. It should gradually change over training time in order to allow the model to generalise across experiences in which the other agents execute policies as they learn.

3 Design

This section outlines the approach used to design the different components of the UTC problem as a DRL problem.

3.1 State Representation

The state is a image-like representation of the current state of the simulator environment (Figure 1a), similar to the concept used in [11]. The state consists of two matrices of 64x64: (1) a binary matrix P for vehicle positions (Figure 1b), and (2) a matrix S for vehicle speeds (Figure 1c). These matrices have been used in previous works such as [5, 9, 10, 16].



(a) Simulation Environment State

$$\begin{bmatrix} 0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0&1 \\ 0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&0 \end{bmatrix}$$

(b) Position Matrix

$$\begin{bmatrix} 0.0&0.0&0.0&0.5&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.4&0.0&0.0&0.0&1.0 \\ 0.0&0.8&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \\ 0.0&0.0&0.0&0.0&0.0&0.0&0.0&0.0 \end{bmatrix}$$

(c) Speed Matrix

Fig. 1: State Representations

The locations are calculated by mapping the continuous space of the simulated environment into a discretized environment by creating a grid with cells of size C. The matrix P is a binary matrix where one indicates the presence of a vehicle and zero the absence of a vehicle (Figure 1b). The matrix S indicates the speed of the vehicle in the same cell position where a vehicle was calculated to be located in the matrix P (Figure 1c). The speed is represented as a percentage of the maximum allowed speed that it is computed by dividing the current vehicle's speed by the maximum allowed speed.

Additionally, a fingerprint is defined as: Let -a be the other agents of agent a, such that we can include the importance weights of the prioritized experience replay and the TD-Errors vectors sampled from the other agents into the observation function as θ_{--a} and TD$_{-a}$ respectively. Given that, the new observation function is O'(s) = { O(s), θ_{--a}, TD$_{-a}$ }. They are included in a matrix F.

3.2 Action Space

The action space varies depending of the structure of the intersection. For example, in a four road intersection the action space is defined as A = {NS, EW, NST; EWT} where NS stands for turning green North-South roads, EW stands for turning green East-West roads, NST stands for turning green North-South right turning, and EWT stands for turning green East-West right turning.

The duration of each phase is 1 time step. However, the agent implicitly determines how long each phase can last ranging from 1 time step up to the final time step of the simulation. The phase can be only changed by the agent when it decides to do it, therefore there is not a limit for how long a phase can take. Additional yellow phase is added during a fixed period of 3 time steps when the previous action is different than the current chosen action. This middle yellow phase reduces the risk of collisions.

3.3 Reward Function

Let $w_{i,t}$ be the ith vehicle's waiting time at time step t, and W_t the total cumulative waiting time for all the vehicles in the observation scope of the road network at time step t as shows in equation 7. The reward function is formulated in equation 8. The intention of the reward function is to reward positive the agent in a range (0.0, 1.0] where the agent's reward loses value proportional to the cumulative waiting time at time step t. Thus, the agent must keep short waiting time in order to receive higher scores, and as consequence, this reward function accomplishes the goal of reducing the driver's waiting time at a junction.

$$W_t = \sum_i w_{i,t} \tag{7}$$

$$r_t = \begin{cases} \frac{1.0}{W_t}, & \text{if cummulative waiting time W}_t \text{ is greater than 0} \\ 1.0, & \text{otherwise} \end{cases} \tag{8}$$

3.4 Deep Neural Network Architecture

The neural network is the same used in [17]. The Figure 2 illustrates the Deep Neural Network. Every agent has its own Neural Network copy in order to allow it to learn its own local policy since every agent is trained independently by using IDQN.

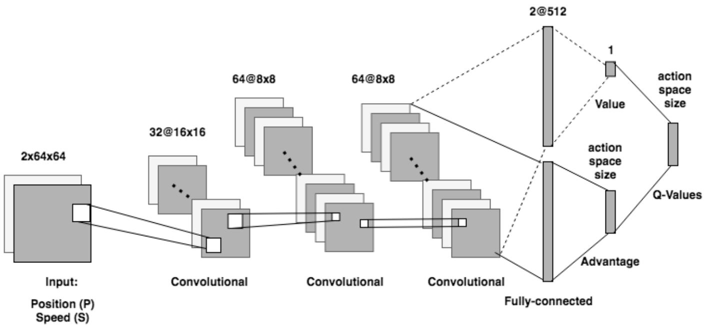


Fig. 2: The architecture of the Deep Neural Network.

3.5 Framework

We use OpenAI's baseline framework [2] to implement our Dueling Proritized DDQN. OpenAI Baselines is a set of high-quality implementations of reinforcement learning algorithms in Python which uses the library TensorFlow [1].

4 Evaluation

In this section we present an evaluation of IDQN as solution for multi-agent DRL for heterogeneous agents in traffic light control using the traffic simulator SUMO [8]. We run the test in the network layout show in Figure 3, which consists of three heterogeneous junctions.

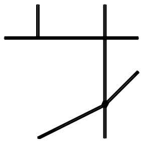


Fig. 3: Network Layout for Multi-Agent Experiments

The table 1 describes the hype-parameters used. Every episode corresponds to 1 hour of simulation time which is 3600 time steps, where every time step is 1 simulated second.

We select two metrics that have been used in several DRL studies for UTC. Every metric is taken per episode. The metrics are the following:

- **Cumulative Reward**. The reward r_t taken on every time step t is accumulated until the episode is finished.

Table 1: Multi-agent evaluation hyper-parameters

Parameter	Value
Episodes	1000
Time steps	3600000
Pre train time steps	2500
Learning rate	0.0004
Exploration ϵ	$1.0 \rightarrow 0.01$
Time steps from starting ϵ to ending ϵ	360000
Target network update	5000
Prioritization exponent α	0.6
Prioritization importance sampling β	$0.4 \rightarrow 1.0$
Discount factor γ	0.99
Replay Memory size M	30000
Minibatch size B	32

– **Average Waiting Time**. Let w_t^i be the ith vehicle's waiting time at time step t. Let W_t be the sum of all vehicles' waiting times at the time step t. The average is the sum of all W_t divided by the number of time steps in the episode (N) as shown in Equation 9. SUMO defines the waiting time for a vehicle as the time (in seconds) spent with a speed below 0.1 m/s since the last time it was faster than 0.1 m/s.

$$AWT = \frac{\sum_{N}^{t=1} W_t}{N} \tag{9}$$

We evaluate the following techniques:

– **Fixed Time / Round Robin (FT)**. This is a predefined configuration for all the traffic lights where the duration and order of the phases is fixed and pre configured.
– **IDQN without experience replay (ERM Disabled)**. It is a IDQN technique but the experience replay is disabled, therefore the agent cannot store experiences.
– **IDQN with prioritized experience replay (PERM)**. It is a normal IDQN technique which uses the experience replay as recommended for DQN.
– **IDQN with prioritized experience replay and fingerprint (PERM + FP)**. This is the proposed IDQN technique with the fingerprint to disambiguate the age of experience replays.
– **IDQN with standard experience replay (ERM)**. This is IDQN with the standard experience replay, not with prioritized experience replay.

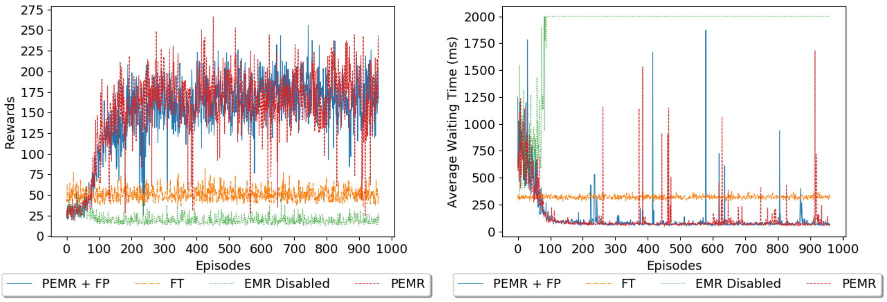
Every technique is tested in the following scenarios:

– **Low traffic load**. A scenario where the number of vehicles is reduced. It represents a normal flow where is not peek time. However, it is a good amount of cars to produce traffic jams if the traffic control is not good.

– **High traffic load**. A scenario where the number of vehicles is overwhelming. It represents the traffic load in peek times. This scenario certainly will produced long queues and traffic jams even with good control. The idea is to verify if it is possible and how much can be reduced in such a overcrowded scenario.

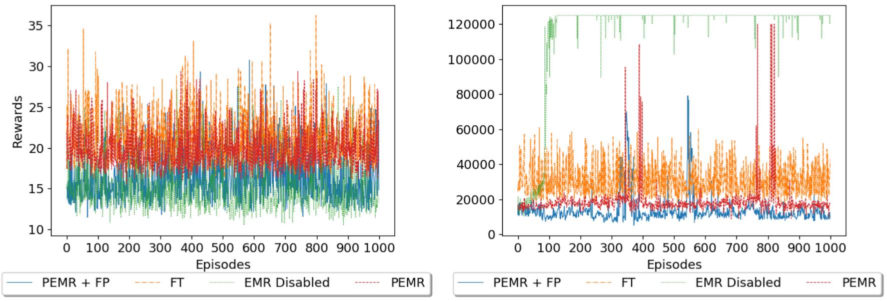
4.1 Experiments

The Figures 4 and Figure 5 present the experiments executed in low and high traffic load respectively.



(a) Cumulative Rewards per Episode (b) Average Waiting Time per Episode

Fig. 4: Low Traffic IDQN Experiments



(a) Cumulative Rewards per Episode (b) Average Waiting Time per Episode

Fig. 5: High Traffic IDQN Experiments

As can be seen from Figure 4a, *FT* gets a constant reward range that varies extremely low around 50 due to it does not adjust its performance. Also, *EMR*

Disabled gets very bad rewards, even worse than the FT, and it never learns. Therefore, disabling the experience replay is not useful for our problem. Finally, we compare *PERM* and *PERM + FP*, which show identical performance getting 400% bigger rewards than *FT*. They also learn very well in spite of the non-stationarity, increasing the rewards from around 25-100 in the exploration phase up to around 200-250 in the exploitation phase. They reach their optimal point at 300 episodes, where they get to keep a stable rewarding. Based on this similarity of results, we deduce that the fingerprint is not helping.

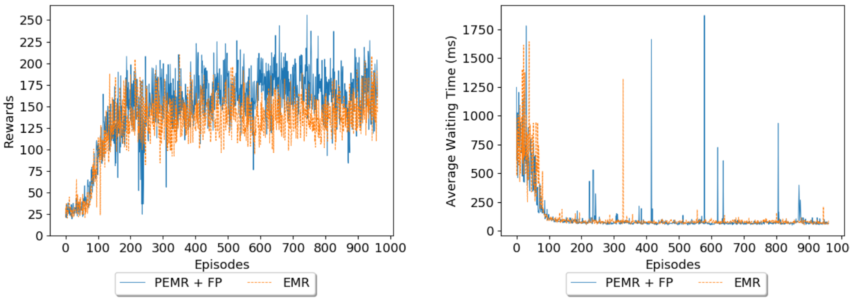
As illustrated in figure 5a, *FT* seems to make a good job in this scenario along with *PERM*, both outperforms *EMR Disabled* and *PERM + FP* in around 25%. It seems that the extra layer of the fingerprint reduces the performance of the technique under intense traffic loads. Moreover, none DRL technique gets to learn over the episodes. This can indicate that the agents needs more training for such a high load.

The Figure 4b presents the waiting time for low traffic. *FT* gets the same behaviour than the rewards. In the case of *EMR Disabled*, it produces extremely high waiting times. Alike in rewards, *PERM* and *PERM + FP* get similar performance, with very low waiting times which are approximately 300% less than *FT*. From these results we can also conclude that the fingerprinting is not helping. In high traffic load, the the results are similar. *PERM + FP* outperform slightly *PERM*. Alike the rewarding results, there is not learning across the training.

The unexpected performance of the fingerprint can be caused by:

- **Not proper selection of fingerprint.** Fingerprint do not correlate enough with the true value of state-action pairs given the other agents' policies and/or it does not vary smoothly over training, which does not allow the model to generalise across experiences in which the other agents execute policies as they learn.
- **Prioritized experience replay is good enough to deal with the non-stationarity.** Due to importance weights that allow to know what records are more valuable samples, which helps to disambiguate the age of the data.

We only verify the second hypothesis by executing a test with *EMR*. The figure 6 shows this additional experiment which compares *EMR* against *PERM + FP*. As shown in Figure 6a, both techniques learn well, but *PERM + FP* obtains higher rewards outperforming *ERM* in around 45% in the highest point of *PERM + FP*. Figure 6b illustrates how they get a similar performance in waiting time, but *EMR* is more stable than *PERM + FP* by not producing peeks. Regardless of the waiting time, we conclude the Prioritized Experience Replay helps to deal with the non-stationarity by getting bigger rewards, which indicates that *PERM* has better performance than *EMR*. The waiting time results are a direct result of the reward function, and *PERM* is getting better results. It could be a correlation problem of the reward function if the waiting time is not decreasing proportionally to the rewards. These results prove that *PERM* can be good enough to deal with the non-stationarity. However, *PERM + FP* could improve by selecting a better fingerprint.



(a) Cumulative Rewards per Episode (b) Average Waiting Time per Episode

Fig. 6: Low Traffic IDQN Experiments with Standard ERM

5 Conclusion

This paper presented IDQN as the only study which addresses heterogeneous multi-agent DRL in UTC. We implemented a IDQN by giving a Dueling Prioritized DDQN to each agent. We evaluated IDQN's performance with different configurations. We executed tests in two traffic conditions, i.e low and high traffic loads. Our experiments showed the IDQN is a suitable techniques for heterogeneous multi-agent UTC environments which can deal with the non-stationarity. The best results were obtained in the low traffic load. The high traffic load seems to need longer training time. We demonstrated that IDQN outperforms the baseline, and that the experience replay is mandatory to learn efficiently. Nonetheless, the fingerprint we chose do not enhance the normal prioritized experience replay.

Finally, PERM needs further investigation to find out better fingerprints. Also, the integration of the fingerprint as an additional matrix can be studied to verify if it is the best approach in a RL system fed with image-like states.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X., Brain, G.: TensorFlow: A System for Large-Scale Machine Learning TensorFlow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). pp. 265–284 (2016), `https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi`
2. Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.: OpenAI Baselines. \url{https://github.com/openai/baselines} (2017)

3. Foerster, J.N., Assael, Y.M., de Freitas, N., Whiteson, S.: Learning to Communicate with Deep Multi-Agent Reinforcement Learning. CoRR abs/1605.0 (2016), http://arxiv.org/abs/1605.06676
4. Foerster, J.N., Nardelli, N., Farquhar, G., Torr, P.H.S., Kohli, P., Whiteson, S.: Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. CoRR abs/1702.0 (2017), http://arxiv.org/abs/1702.08887
5. Gao, J., Shen, Y., Liu, J., Ito, M., Shiratori, N.: Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network. arXiv pp. 1–10 (2017), https://arxiv.org/pdf/1705.02755.pdf{\%}0Ahttp://arxiv.org/abs/1705.02755
6. van Hasselt, H., Guez, A., Silver, D.: Deep Reinforcement Learning with Double Q-Learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. pp. 2094–2100. AAAI'16, AAAI Press (2016), http://dl.acm.org/citation.cfm?id=3016100.3016191
7. Hasselt, H.V., Group, A.C., Wiskunde, C.: Double Q-learning. Nips pp. 1–9 (2010)
8. Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker, L.: Recent Development and Applications of {SUMO - Simulation of Urban MObility}. International Journal On Advances in Systems and Measurements 5(3&4), 128–138 (dec 2012), http://elib.dlr.de/80483/
9. Liang, X., Du, X., Wang, G., Han, Z.: Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks. Ieee Transactions on Vehicular Technology 1(Xx), 1–11 (2018), https://arxiv.org/pdf/1803.11115.pdf
10. Liu, M., Deng, J., Xu, M., Zhang, X., Wang, W.: Cooperative Deep Reinforcement Learning for Traffic Signal Control. 23rd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD), Halifax 2017 (2017)
11. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
12. Richard S. Sutton and Andrew G. Barto: Reinforcement Learning, Second Edition An Introduction. MIT Press, second edi edn. (2018)
13. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. CoRR abs/1511.05952 (2015), http://arxiv.org/abs/1511.05952
14. Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., Vicente, R.: Multiagent Cooperation and Competition with Deep Reinforcement Learning. arXiv pp. 1–12 (2015), http://arxiv.org/abs/1511.08779
15. Tan, M.: Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In: Machine Learning Proceedings 1993, pp. 330–337. Morgan Kaufmann Publishers Inc. (1993), http://linkinghub.elsevier.com/retrieve/pii/B9781558603073500496
16. Van Der Pol, E., Oliehoek, F.A.: Coordinated Deep Reinforcement Learners for Traffic Light Control. NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems (Nips) (2016), http://www.fransoliehoek.net/docs/VanDerPol16LICMAS.pdf{\%}0Ahttp://elisevanderpol.nl/papers/vanderpol{_}oliehoek{_}nipsmalic2016.pdf
17. Wang, Z., de Freitas, N., Lanctot, M.: Dueling Network Architectures for Deep Reinforcement Learning. arXiv (9), 1–16 (2016), http://arxiv.org/abs/1511.06581
18. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning 8(3-4), 279–292 (1992), http://link.springer.com/10.1007/BF00992698