2018 IEEE Intelligent Vehicles Symposium (IV)
Changshu, Suzhou, China, June 26-30, 2018

Adaptive Traffic Signal Control with Deep Recurrent Q-learning

Jinghong Zeng[1], Jianming Hu[2], *IEEE Member*, and Yi Zhang[2], *IEEE Member*
Department of Automation, Tsinghua University, Beijing, 100084, China
[1]zengjh16@mails.tsinghua.edu.cn
[2]{hujm, zhyi}@mail.tsinghua.edu.cn

***Abstract*— The application of modern technologies makes it possible for a transportation system to collect real-time data of some specific traffic scenes, helping traffic control center to improve the traffic efficiency. Based on such consideration, we introduce a variant deep reinforcement learning agent that might take advantage of the real-time GPS data and learn how to control the traffic lights in an isolated intersection. We combine the recurrent neural network (RNN) with Deep Q-Network, namely DRQN and compare its performance with standard Deep Q-Network (DQN) in partially observed traffic situations. The agent is trained by using Q-learning with experience replay in traffic simulator SUMO, so as to generate traffic signal control policy. Based on the experiments, both DQN and DRQN method are able to adjust its traffic signal timing policy to specific traffic environment and achieve lower average vehicle delay than fixed time control. In addition, the recurrent Q-learning method gets better simulation result than standard Q-learning method in the environment of different probe vehicle proportion.**

I. INTRODUCTION

A set of well-configured traffic lights plays a critical role in relieving urban traffic jams and improving its efficiency in certain area. However, the optimization of traffic signal timing in a large urban traffic network is a quite difficult and challenging problem[1]. The dynamic traffic environment is time-variant, affected by the randomness of human behavior, daily weather and even some economic activity, making the predefined traffic signal timing become out of date. Hence, it is a reasonable idea that a well-designed traffic signal control (TSC) agent must be able to adjust its timing schemes according to the real-time traffic demand[2][3].

Intelligent transportation system (ITS) integrates the traditional transportation system with sensor network and wireless communication technologies makes it possible to collect large volumes of real-time vehicle data from a road network. Previous researchers had shown us the capabilities of fuzzy logic[4], neural network[5][6] and reinforcement learning[7][8] methods in taking advantage of such real-time traffic data and building an adaptive traffic lights control agent. Especially, the recent achievements of deep reinforcement learning[10][11][12][13] also proved its potential to endow vehicles or traffic infrastructures with some kind of intelligence.

*This work is partially supported by National Key R&D Program in China (2016YFB0100906), Beijing Municipal Science and Technology Program (D171100006417003), Alberta-Tsinghua Collaborative Research Fund (20173080020) and National Natural Science Foundation of China under Grant No.61673232.

Reinforcement learning[9] learns the state-action mapping from its experience by maximizing the expectation of accumulated future reward, usually with a function approximation to generalize the learning result to the exponentially expanding state and action spaces. With the remarkable deep learning[14][15] technology as function approximation to realize powerful function mappings, deep reinforcement learning could accomplish more difficult and complex tasks. Li *et al.*[12] proposed a deep Q-network with stacked auto-encoders (SAE) to approximate the optimal Q value function, where the algorithm takes the number of queue vehicles as states and trained with Q-learning algorithm to learn an optimal control policy at a single intersection. Genders *et al.*[13] proposed to use a different state representation, the discrete traffic state encoding (DTSE), to maintain the complete information around an isolated intersection and use convolutional neural networks (CNN)[14] to extract spatial features, which also produced fairly well simulation results. The previous studies on this subject mainly focus on how to apply reinforcement learning in building an adaptive traffic signal control agent. However, most of reinforcement learning theory bases upon the Markov Decision Process (MDP), a simplified model that is not suitable when the environment is non-stationary[16]. Using Reinforcement Learning in a non-stationary environment requires continuous learning with adaptive Q-learning rate and the appropriate action exploration[17].

It is a more suitable way to model the non-stationary environment as Partially Observable MDP (POMDP)[18]. There are at least two elements making the traffic environment at a single intersection become partially observed: (1) the pattern of incoming traffic flow at an intersection is time-variant and unknown exactly, (2) the vehicles around the intersection are just part of whole traffic flow in the traffic network. Especially, the accessible vehicle data are generally collected from the on-board GPS device of partial vehicles, i.e. probe vehicle, which upload their real-time GPS data to the remote data center. However, solving POMDP problem is intractable. Traditional approaches maintain the *belief states* to estimate the probability distribution over the states and some of memory-based approaches suggest solving POMDP problem with recurrent Q-learning method[19].

In this paper, we treat the traffic environment at a single intersection as non-stationary environment and attempt to solve the TSC problem with memory-based reinforcement learning method. We use recurrent neural network (RNN)

[20] to handle the sequence of observations and approximate the optimal action value function. The neural network is trained by using experience replay and recurrent Q-learning algorithm. Finally, we compare its performance with standard Q-network method and fixed-time control in partially observed traffic environment.

This paper is organized as follows. In Section II, we first present the base idea of reinforcement learning algorithm, include the deep recurrent Q-learning idea; then in Section III, we show how to model the single intersection control problem as reinforcement learning problem; Section IV provides the detail of simulation configuration and the comparison results; finally, we conclude this paper in Section V.

II. REINFORCEMENT LEARNING FOUNDATION

A. Reinforcement Learning Algorithm

Reinforcement learning agent could learn different strategies via the interaction with specific environment[9]. The basic idea of this process is the MDP, which consists of five necessary components, including the state space S, action space A, reward R, transition probability P and a discount factor γ. The agent interacts with the environment and gains an observation S_t from it, then takes an action A_t according to the state S_t and its current policy π at every time step t. After that, the environment might return a reward R_t as feedback signal and the agent learns the optimal policy π_* by maximizing the expectation of accumulated future reward, which also defines the value when agent is under a state s. The state value function is denoted as

$$v_\pi(s) = E_\pi \left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots | S_t = s \right]. \quad (1)$$

The learning target of reinforcement learning is to find an optimal policy π_* to maximize the state value function $v_\pi(s)$.

$$v_{\pi_*}(s) = \max_\pi v_\pi(s) \quad (2)$$

We denotes the cumulative future reward when the agent performs action a in state s as $Q(s,a)$, namely the Q-value or action value function. The optimization of (2) is equivalent to maximize the following target.

$$v_{\pi_*}(s) = \max_a Q_{\pi_*}(s,a) \quad (3)$$

Hence, the optimal deterministic policy is to take the action of maximal Q-value when given s if the Q-value function is learned appropriately. A representative model-free reinforcement learning method for learning action-value function is Q-learning algorithm[9]. It iteratively updates Q-value function by using

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)), \quad (4)$$

where α is the Q-learning rate, r is the instant reward, s' and a' represent the state and action at next decision step respectively. It gradually updates the Q-value function toward the ideal optimal value which defined by the Bellman optimality equation.

B. Deep Reinforcement Learning

Reinforcement learning has the capability to generalize the Q-value estimation to some unseen samples when combined with generalization methods. With deep neural network as a powerful nonlinear function approximation tool[10][11], deep reinforcement learning could learn complex Q-value function even when there are too many state-action pairs to be explored one by one.

Deep neural network is a deep stacked perceptron model, trained by the classic backpropagation algorithm. It uses gradient descent algorithm to adjust its inner parameters Θ gradually. Generally, the loss function of deep reinforcement learning is defined as

$$L(\Theta) = \frac{1}{2} E_{s,a,r,s'}[(r + \gamma \max_{a'} Q'(s',a') - Q(s,a))^2], \quad (5)$$

where $r + \gamma \max_{a'} Q'(s',a')$ plays like the learning target in supervised learning. The Q-value function with superscript $Q'(s',a')$ denotes the Q-value function estimation from another neural network, namely target network, which only updates its inner parameters from the primary neural network after a certain number of steps[11].

It is a practical way to stabilize the learning process with experience replay and target network[11]. Experience replay maintains a memory to store the experience that the agent has recently visited, and randomly samples a batch of experience from memory as training data at each training step. With experience replay, it is able to break the correlation between successive samples. Target network uses another neural network to compute the learning target and updates its weights after a certain number of steps, which is helpful to stabilize the learning target during reinforcement learning process.

C. Deep Recurrent Q-Learning

The previous Q-value function definition means that the Q-value is merely related to the current state s and corresponding action a. While in a non-stationary environment, the Q-value of state-action pairs might change as environment dynamics changes. The environment dynamics plays like the hidden variable in POMDP, which is invisible for the agent. Treating environment as POMDP means that the Q-value is not only related to the state (observation) and action, but also the observation history. There are several ways of solving this problme, e.g., the traditional *belief states* method and memory-based reinforcement learning method. Memory-based approaches use recurrent neural network to encode the observation history[19]. Hausknecht and Stone[21] proposed the Deep Recurrent Q-network (DRQN) to play Atari games after introducing partial observability to the games. In this paper, we build the deep recurrent Q-network with the long short-term memory (LSTM), one kind of RNN structures that has been applied in sequence modeling successfully[20], to control a single intersection.

III. PROBLEM MODEL

The single intersection model is built with SUMO (**S**imulation of **U**rban **MO**bility)[22], a open-source road

traffic simulation environment. We build the following four-way intersection, with two straight-going lanes, one left-turn lane and a lane for straight-going and right-turn in each direction of the intersection.

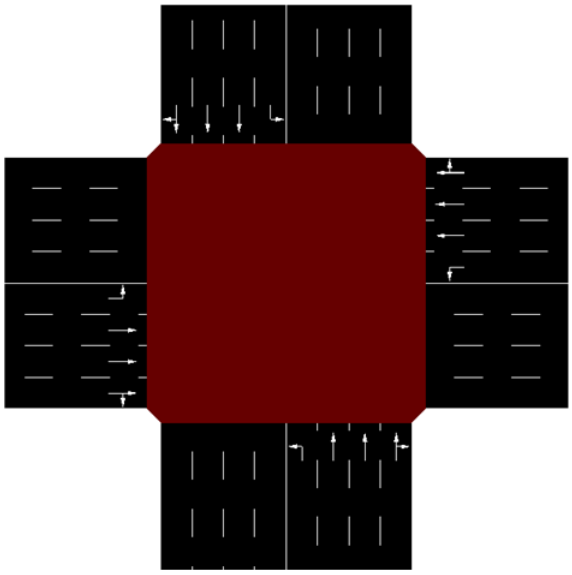

Fig. 1. The intersection model

The prerequisites of applying reinforcement learning in traffic signal control problem is the appropriate definition of the corresponding state space S, action space A and reward function R.

A. State Space

Previous researchers have shown that the current phase information (e.g. duration of phase) is useful for learning a proper control policy and the necessary state component is the combination of vehicle information (e.g. vehicle number and queue length) around the intersection[8]. It is a more attractive idea to learning the state information from the raw traffic data by agent itself. That is why DTSE[13] aims to maintain the original information as complete as possible. However, the location and speed of vehicle generally have measurement uncertainty, e.g., the GPS device could only be accurate to within about 10 meters for civilian uses, which makes the precise location and speed are not accessible. Here we follow the idea of map matching which is used to correct the error of GPS, and propose a similar way to transform the traffic situation of one road into several vectors. We divides the road of length l into discrete segments of length c to discretize the specific traffic scene, as shown in Fig. 2 for example.

Fig. 2. Road segments in one direction

The data of each vehicle are mapped into a small road segment, so that we could use a density vector d and a speed vector v to represent the traffic density and average speed in one direction. The density vector $d = [1.0, 0.625, 0.25, 0.0, 0.125, 0.25]^T$ and the speed vector $v = [0.0, 0.06, 0.2, 0.0, 0.6, 0.7]^T$ in Fig. 2. Each entity of density vector indicates how many vehicles are currently inside the corresponding segment, divided by the containable vehicles number of segment. Each entity of speed vector indicates the average normalized speed of vehicles inside the segment, divided by the maximum allowed speed. We also use one-hot vector p to encode current green phase. Because we only make decision at four green phase, we used four-dimensional one-hot vector to represent the current green phase. Finally, we combine the density and speed vector of four direction to construct a density matrix $D = [d_1, d_2, d_3, d_4]$ and speed matrix $V = [v_1, v_2, v_3, v_4]$, and the corresponding state representation is $S = \{D, V, p\}$.

B. Action Space

We use the simulator to construct an intersection with four green phases, i.e. a straight phase, a left-turning phase, a straight phase and left-turning phase for the direction orthogonal to the first one. The phase sequence is fixed as

$$\textbf{NSG} \Rightarrow NSY \Rightarrow \textbf{NSLG} \Rightarrow NSLY \Rightarrow \textbf{EWG} \Rightarrow EWY \Rightarrow \textbf{EWLG} \Rightarrow EWLY \Rightarrow \textbf{NSG} \Rightarrow \cdots,$$

where NSG means a straight green phase for north-south direction, EWG means a straight green phase for east-west direction, Y means a yellow light interval τ_y and L means a left-turning phase. There is not all-red light interval setting and all the green phases have minimum green time t_{min} and maximum green time t_{max}.

The required optimized variables are phase extension and phase skip at four green phases, making the traffic signal control become acyclic. Hence, the action space of agent could be described as $A = \{0, 1\}$. If the traffic signal control agent choses an action '0', the agent will extend the current green phase a little time span τ_g. If the current green time is greater than or equal to t_{max} after phase extension, the traffic lights will switch to the next green phase according to the phase sequence and the agent makes decision again after the green phase has been running for minimum green time t_{min}. That is what the agent will do when it selects an action '1', meaning phase skip.

C. Reward Definition

There are multiple evaluation criteria to distinguish whether the traffic situation is improved by the control policy, e.g. queue length and cumulative delay of vehicle. Here we measure the total number of halting vehicle before the stop line at time step t and denote it as NV_t. We use

$$R_t = NV_t - NV_{t+1}, \tag{6}$$

as reward function, so that when the number of halting vehicle decreases between time step t and $t+1$, the agent will receive a positive reward to encourage its decision.

D. Agent Settings

During training, the agent selects the action with ϵ-greedy strategy, with probability ϵ the agent will selects a random action and with probability $1-\epsilon$ the agent will select the highest-value action. While training, the probability ϵ_n decreases as

$$\epsilon_n = \max\{\epsilon_{min}, \epsilon_{max} - \frac{n}{N}(\epsilon_{max} - \epsilon_{min})\}, \qquad (7)$$

where n is the current training step, N is a large enough integer to trade off the exploration and exploitation. The loss function we use is Huber loss function instead of square loss function, as Mnih *et al.*[11] suggested.

IV. SIMULATION RESULTS

In this section, we build the similar deep Q-network (DQN) and deep recurrent Q-network (DRQN), and compare their performances in the control of an isolated intersection. The traffic environment model is built with SUMO. We use the framework of reinforcement learning in keras, i.e. keras-rl[23], to build the standard DQN control agent. By integrating with the framework, we realize our own DRQN control agent with some extra python scripts.

A. Network Settings

Fig. 3 shows the basic recurrent Q-network. We combine the density and speed matrix as a small two-channel image, and use two convolutional layers to extract useful features. The first layer of this part has 4 convolutional filters of size 2×2 applied with strides $(1, 2)$ and the second layer has 8 convolutional filters of size 2×2 applied with strides $(1, 1)$. Without zero-paddding, the outputs of last convolutional layer will be flattened into a 112-dimensional vector. The next part of this network is a simple embedding layer to encode the phase vector into a 10-dimensional vector. At last, the concatenation of 112-dimensional and 10-dimensional vector is treat as the input of LSTM with 32 hidden units and the successive layers are a full-connected layer with 8 outputs and a full-connected layer with 2 outputs. The activation functions are rectifier nonlinear activation functions (ReLU), expect that LSTM uses tanh and sigmoid functions, and output layer uses linear function. The comparative standard DQN is built by replacing the LSTM layer with a full-connected layer with 32 hidden units.

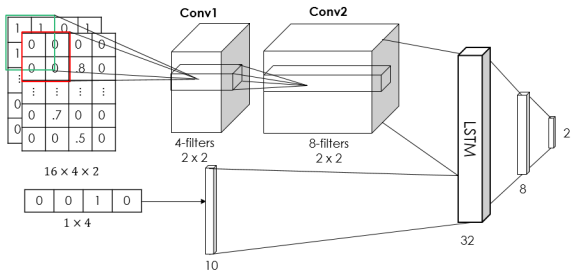

Fig. 3. The recurrent Q-network model

There are two ways to update the weights of RNN, namely *bootstrapped sequential updates* and *bootstrapped random updates*[21]. Both of these update methods will converge to similar performance, here we perform a *bootstrapped random updates* to adjust the parameters of RNN. At each training time step, we will randomly sample a fixed-length sequence of experiences from replay memory and updates begin at the start of sequence and proceed forward to the end of sequence. The sequence of experiences is first used to generate learning Q-targets by using target network recursively. Because the LSTMs hidden state is zeroed at the beginning, the predicted Q-targets will not be accuracy at the beginning. Hence, we only use the latter half of predicted Q-targets as training targets, and the error signals are back-propagated through time to train the DRQN. The standard DQN is trained based on the randomly sampling experiences and backpropagation algorithm.

Both of the networks are trained by using Adam[24] algorithm with learning rate of 0.00025; the replay memory size is 80000 and the target network update interval is 250; the sampling experience length is set as 15 and the sampling batch size is 32.

B. Simulation Settings

The four-way intersection has 500 meters approach in each direction and we only consider a road section of length l to be 256 meters. The segment length c is 16 meters, which divides the road section into 16 discrete segments, making a 16×4 density matrix and a 16×4 speed matrix. The vehicle length is 5 meters and the minimum gap between vehicles is 2.5 meters. The maximum vehicle speed is 22.22 m/s (80km/h) and the road speed limit is 16.67 m/s (60km/h).

For traffic signal timing parameters, we set the green phase interval τ_g to be 2 seconds and the yellow light interval τ_y to be 4 seconds. The minimum green time t_{min} is 6 seconds and maximum green time t_{max} is 60 seconds.

The probability ϵ that the agent selects a random action decreases from $\epsilon_{max}(1.0)$ to $\epsilon_{min}(0.1)$ as (7) during training. The total training step is 1000000 and N is equal to 800000.

C. Convergence of Algorithm

We first test the convergence of both DQN and DRQN method. The vehicles arrive at the intersection by following a Poisson process approximated here by a binomial distribution. The arrival probability of straight-going vehicles in North-South (N-S) or South-North (S-N) direction is $1.0/4.0$, meaning a straight-going vehicle in this direction is generated every 4 seconds in average. The detailed configuration is shown in the following Table I.

Fig. 4 shows the convergence of DQN and DRQN. Each episode represents one hour of traffic simulation and the performance is evaluated by the traffic metric, i.e., the average waiting time of vehicles that travel through the intersection. The fixed-time control is set up according to the proportion of arrival probabilities, i.e., $(NSG, NSLG, EWG, EWLG) = (60s, 16s, 40s, 10s)$ and all of the yellow light intervals τ_y are 4 seconds as mentioned above. The reinforcement

TABLE I
THE ARRIVAL PROBABILITIES SETTINGS

Direction	Arrival Probabilities		
	Left-turning	*Straight-going*	*Right-turning*
N-S	1/16	1/4	1/16
S-N	1/16	1/4	1/16
E-W	1/24	1/6	1/24
W-E	1/24	1/6	1/24

learning agent explores the control policy at the beginning, with high probability ϵ to select a random action. With the training time going by, the agent gets positive or negative rewards depended on whether or not it has taken a correct action to reduce the halting vehicle number. The agent gradually exploits the explored policy and reduces the queue length and average waiting time. Finally, both of DQN and DRQN method achieve the better performance than fixed-time control, as shown in Fig. 4. There is no reason to hope that DRQN could outperform the standard DQN methods in such stationary and fully observed environment. But DRQN explores a little more efficiently than DQN as it benefits from the *bootstrapped random updates*, which uses a sequence of experiences to update its inner parameters at each training step.

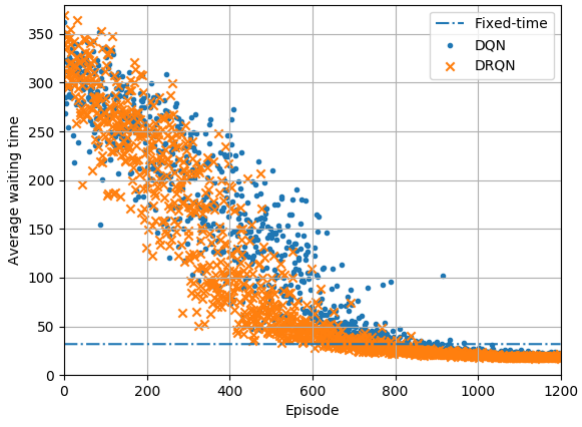

Fig. 4. Average waiting time in training progress

The final measurement result is shown in Table II, the metric is the average result of 20 testing episodes with the some simulation configuration shown in Table I.

TABLE II
THE ARRIVAL WAITING TIME

Metric	Methods		
	Fixed-time	*DQN*	*DRQN*
Average waiting time(s)	31.82	17.58	17.71

D. Partially Observable Effect

The real-world environment is full of uncertainty and it is generally impossible to access the true full state of environment, making the environment become POMDP. For example, in real-time vehicle data collection, only a portion of vehicles will upload their location and speed to the data center, acting as the probe vehicle to detect the whole traffic environment. The penetration rate is usually different in specific traffic scene and even changes according to different traffic hours. Here we consider the ratio of probe vehicle and the delay of data transmission. We change the penetration rate of probe vehicle by altering the ratio of "observable" vehicles, of which the location and speed information is visible during simulation. This will reduce the values of density vector and correspondingly increase the sparsity of both density and speed vector.

Fig. 5 shows the performances under different penetration rates. The reinforcement leaning agent is trained and tested in the environment with the same probe vehicle proportion. Each data point in graph represents the average value of 20 testing episodes with the corresponding penetration rate setting. The result shows that, for standard Q-network, the average waiting time increases as decreasing penetration rate. However, it could benefit from the discrete presentation of states and keep stable performance at the beginning. Adding recurrence to Q-network makes it more reliable and able to learn similar performance at different penetration rates. It could better distinguish the actual environment state by combining current state input with the state of memory cells in LSTM.

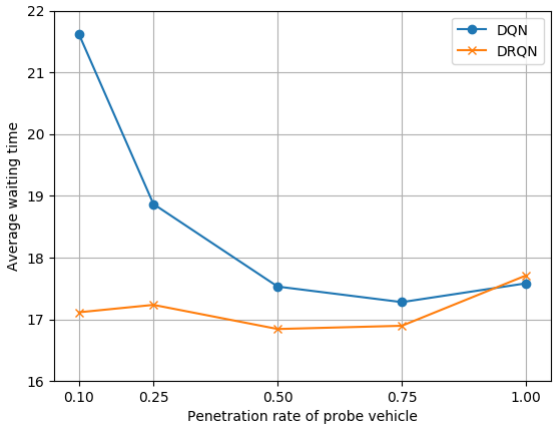

Fig. 5. Average waiting time in different penetration rates. Agent is trained and tested in the same penetration rate.

The generalization capacities of agents trained under the penetration rate of 50% are shown in Fig. 6. It is surprising to see DRQN agent has better performance than DQN agent indeed, as it has longer stable interval than DQN around 50%. The result indicates that DRQN agent not only benefit from the discrete representation, but also the condensed historical information in memory cells, which might be similar in different penetration rates.

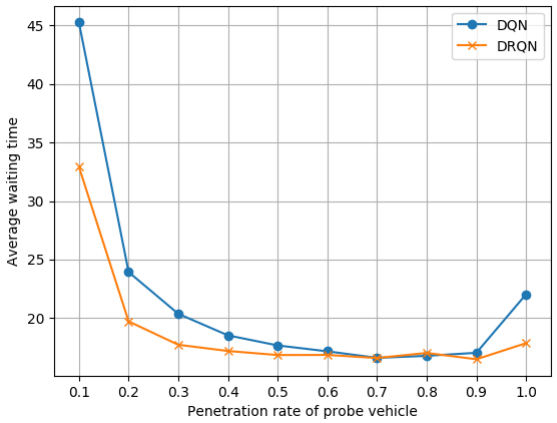

Fig. 6. Average waiting time in different penetration rates. Both of the agents are trained under the penetration rate of 50% but tested in different penetration rates.

The next benefit of using recurrent Q-learning is its ability to carry historical information and make correct decision even without seeing the current environment state. We train and test both agents in the environment of 50% penetration rate, with probability of 50% we omit the current vehicles information and input zero density and speed vectors at each time step. The average waiting time of standard Q-network is 77.47 seconds, even worse than fixed-time control. However, DRQN still keep a performance of 16.74 seconds with better robustness.

V. CONCLUSIONS

We compare the performance of recurrent Q-network and standard Q-network, showing that adding recurrence to Deep Q-Learning allows better approximating the policies in partially observed environment. Combining the historical state and current input help to better distinguish the actual environment state. Recurrent Q-learning is more stable in lower penetration rate of probe vehicle, showing the possibility of using real-time GPS positioning data to control traffic lights.

However, applying reinforcement learning in real world application is still a difficult challenge. The simulation in this paper is set up for convenience, further comparison need to be done with larger network structures and more complex traffic situation. Furthermore, reinforcement learning is actually an environment-dependent algorithm. The real-world application would need continual exploration and exploitation to adjust the learned policy in training process, especially when in a non-stationary environment.

It is a more attracting idea to first learn the real-time traffic pattern and apply different reinforcement learning agents according to the detected traffic pattern. Our future work will focus on the memory network approach for learning traffic pattern and solving traffic signal control problem.

REFERENCES

[1] Webster F V. Traffic signal settings [J]. Road Research Technical Paper, 1958, 39.

[2] Mirchandani P, Head L. A real-time traffic signal control system: architecture, algorithms, and analysis [J]. Transportation Research Part C Emerging Technologies, 2001, 9(6): 415–432.

[3] Gradinescu V, Gorgorin C, Diaconescu R, et al. Adaptive Traffic Lights Using Car-to-Car Communication [C]. IEEE, Vehicular Technology Conference - Vtc2007-Spring. IEEE, 2007: 21–25.

[4] G. Nakamiti and F. Gomide, "Fuzzy sets in distributed traffic control," in Proc. 5th IEEE Int. Conf. Fuzzy Syst., 1996, pp. 1617–1623.

[5] D. Chin, J. Spall, and R. Smith, "Evaluation of system-wide traffic signal control using stochastic optimization and neural networks," in Proc. Amer. Control Conf., 1999, vol. 3, pp. 2188–2194.

[6] D. Srinivasan, M. Choy, and R. Cheu, "Neural networks for real-time traffic signal control," IEEE Trans. Intell. Transp. Syst., vol. 7, no. 3, pp. 261–272, Sep. 2006.

[7] Abdulhai B, Karakoulas G J, Pringle R. Reinforcement learning for true adaptive traffic signal control [J]. Journal of Transportation Engineering, 2003, 129(3): 278–285.

[8] Bingham E. Reinforcement learning in neuro-fuzzy traffic signal control [J]. European Journal of Operational Research, 2001, 131(2): 232–241.

[9] Sutton R S, Barto A G. Reinforcement Learning: An Introduction, Bradford Book [J]. IEEE Transactions on Neural Networks, 2005, 16(1): 285–286.

[10] Mnih V, Kavukcuoglu K, Silver D, et al. Playing Atari with Deep Reinforcement Learning [J]. Computer Science, 2013.

[11] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning [J]. Nature, 2015, 518(7540): 529.

[12] Li L, Lv Y, Wang F Y. Traffic signal timing via deep reinforcement learning [J]. IEEE CAA Journal of Automatica Sinica, 2016, 3(3): 247–254.

[13] Genders W, Razavi S. Using a Deep Reinforcement Learning Agent for Traffic Signal Control [J]. 2016.

[14] Lécun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition [J]. Proceedings of the IEEE, 1998, 86(11): 2278–2324.

[15] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks [C]. International Conference on Neural Information Processing Systems. Curran Associates Inc. 2012: 1097–1105.

[16] Oliveira D D, Bazzan A L C, Silva B C D, et al. Reinforcement Learning based Control of Traffic Lights in Non-stationary Environments: A Case Study in a Microscopic Simulator [C]. European Workshop on Multi-Agent Systems Eumas'06, Lisbon, Portugal, December. DBLP, 2006.

[17] Noda I. Recursive Adaptation of Stepsize Parameter for Non-stationary Environments [M]. Adaptive and Learning Agents. Springer Berlin Heidelberg, 2009: 525–533.

[18] Choi S P M, Yeung D Y, Zhang N L. Hidden-Mode Markov Decision Processes for Nonstationary Sequential Decision Making[C]. Sequence Learning — Paradigms, Algorithms, and Applications. DBLP, 2001: 264–287.

[19] Lin L, Mitchell T. Memory Approaches to Reinforcement Learning in Non-Markovian Domains [M]. Carnegie Mellon University, 1992.

[20] Hochreiter S, Schmidhuber J. Long Short-Term Memory [J]. Neural Computation, 1997, 9(8): 1735–1780.

[21] Hausknecht M, Stone P. Deep Recurrent Q-Learning for Partially Observable MDPs [J]. Computer Science, 2015.

[22] Bieker L. Recent Development and Applications of SUMO — Simulation of Urban MObility [J]. International Journal on Advances in Systems & Measurements, 2012, 3&4(3 and 4): 128–138.

[23] Matthias P. Keras-RL, https://github.com/matthiasplappert/keras-rl. GitHub Repository, GitHub, 2016.

[24] Kingma D P, Ba J. Adam: A Method for Stochastic Optimization [J]. Computer Science, 2014.