IET Intelligent Transport Systems

Research Article

Traffic light control using deep policy-gradient and value-function-based reinforcement learning

ISSN 1751-956X
Received on 9th May 2017
Revised 8th July 2017
Accepted on 10th August 2017
E-First on 20th September 2017
doi: 10.1049/iet-its.2017.0153
www.ietdl.org

Seyed Sajad Mousavi[1] ✉, *Michael Schukat*[1], *Enda Howley*[1]
[1]*Discipline of Information Technology, National University of Ireland, Galway, Galway, Ireland*
✉ *E-mail: s.mousavi1@nuiglaway.ie*

Abstract: Recent advances in combining deep neural network architectures with reinforcement learning (RL) techniques have shown promising potential results in solving complex control problems with high-dimensional state and action spaces. Inspired by these successes, in this study, the authors built two kinds of RL algorithms: deep policy-gradient (PG) and value-function-based agents which can predict the best possible traffic signal for a traffic intersection. At each time step, these adaptive traffic light control agents receive a snapshot of the current state of a graphical traffic simulator and produce control signals. The PG-based agent maps its observation directly to the control signal; however, the value-function-based agent first estimates values for all legal control signals. The agent then selects the optimal control action with the highest value. Their methods show promising results in a traffic network simulated in the simulation of urban mobility traffic simulator, without suffering from instability issues during the training process.

1 Introduction

With regard to fast growing population around the world, the urban population in the 21st century is expected to increase dramatically. Hence, it is imperative that urban infrastructure is managed effectively to contend with this growth. One of the most critical consideration when designing modern cities is developing smart traffic management systems. The main goal of a traffic management system is reducing traffic congestion which nowadays is one of the major issues of megacities. Efficient urban traffic management results in time and financial savings as well as reducing carbon dioxide emission into atmosphere. To address this issue, a lot of solutions have been proposed [1–4]. They can be roughly classified into three groups. The first is pre-timed signal control, where a fixed time is determined for all green phases according to historical traffic demand, without considering possible fluctuations in traffic demand. The second is vehicle-actuated signal control, where traffic demand information is used, provided by inductive loop detectors on an equipped intersection to decide to control the signals, e.g. extending or terminating a green phase. The third is adaptive signal control, where the signal timing control is managed and updated automatically according to the current state of the intersection (i.e. traffic demand, queue length of vehicles in each lane of the intersection and traffic flow fluctuation) [5]. In this paper, we are interested in the third approach and aim to propose two novel methods for traffic signal control by leveraging recent advances in machine learning and artificial intelligence fields [6, 7].

Reinforcement learning (RL) [8] as a machine learning technique for traffic signal control problem has led to impressive results [2, 9] and has shown a promising potential solver. It does not need to have a perfect knowledge of the environment in advance, for example, traffic flow. Instead they are able to gain knowledge and model the dynamics of the environment just by interacting with it. An RL agent learns based on trial and error. It receives a scalar reward after taking each action in the environment. The obtained reward is based on how well the taken action is and the agent's goal is to learn an optimal control policy, so the discounted cumulative reward is maximised via repeated interaction with its environment. Aside from traffic control, RL has been applied to a number of real-world problems such as cloud computing [10, 11].

Typically, the complexity of using RL in real-world applications such as traffic signal management, grows exponentially as state and action spaces increase. To deal with this problem, function approximation techniques and hierarchical RL (HRL) approaches can be used. Recently, deep learning has gained huge attraction and has been successfully combined with RL techniques to deal with complex optimisation problems such as playing Atari 2600 games [7], Computer Go program [12] etc., where the classical RL methods could not provide optimal solutions. In this way, the current state of the environment is fed into a deep neural net [e.g. a convolutional neural network (CNN) [13]] trained by RL techniques to predict the next possible optimal action(s).

Inspired by the successes of combining RL with deep learning paradigm and with regard to the complex nature of environment of traffic signal control problem, in this paper we aim to use the effectiveness and power of deep RL to build adaptive signal control methods in order to optimise the traffic flow. Although a few previous studies have tried to apply deep RL in the traffic signal control problem [14, 15], in this research the state representation is different. Also, one of our methods uses policy-gradient (PG) method which does not suffer from oscillations and instabilities during the training process and can take full advantage of the available data of the environment to develop the optimal control policy.

We propose adaptive signal controllers by combination of two RL approaches (i.e. PG and action-value function) and a deep convolution neural network, which perceive embedded camera observations in order to produce control signals in an isolated intersection. We conduct simulated experiments with our proposed methods in simulation of urban mobility (SUMO) traffic simulator.

The rest of this paper is organised as follows. Section 2 provides related work in the area of traffic light control (TLC). Section 3 gives a brief review of RL techniques which we have used in this research. Section 4 presents how to formulate the TLC problem as an RL task and the proposed methods to solve the task. Then, Section 5 provides simulation results and the performance of the proposed approaches. Finally, Section 6 concludes this paper and gives some directions for future research.

2 Related work

A lot of research has been done in academic and industry communities to build adaptive traffic signal control systems. In

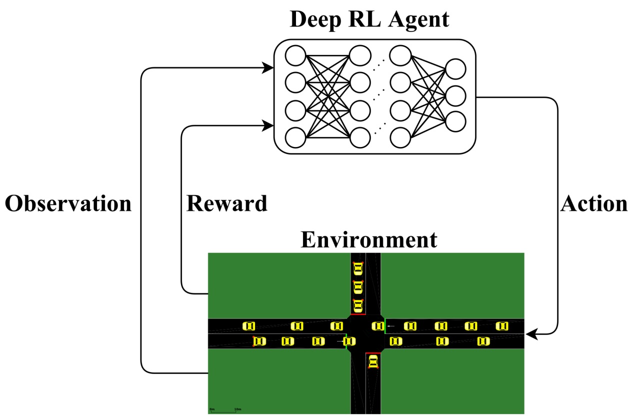

Fig. 1 *Deep RL agent of traffic signal control*

particular, significant research has been conducted employing RL methods in the area of traffic light signal control [16–20]. These works have achieved promising results. However, their simulation testbeds have not been mature enough to be comparable with more realistic situations. Developing advanced traffic simulation tools have made researchers develop novel state representation and reward functions for RL algorithms, which could consider more aspects of complexity and reality of real-world traffic problems [3, 5, 21–24]. All these attempts viewed the TLC problem as a fully observable Markov decision process (MDP) and investigated whether Q-learning algorithm can be applied to it. However, Richter's study formulated the traffic problem as a partially observable MDP (POMDP) and applied PG methods to guarantee local convergence under a partial observable environment [25].

By utilising advances in deep learning and its application to different domains [11, 26, 27], deep learning has gained attention in the area of traffic management systems. The previous research has used deep stacked auto-encoders (SAEs) neural networks to estimate Q values, where each Q-value is corresponding to each available signal phase [28]. It considered measures of speed and queueing length as its state in each time step of learning process of its proposed method. Two recent studies by van der Pol and Oliehoek [14] and Genders and Razavi [15] provided deep RL agents that used deep Q-network [7] to map from given states to Q values. Their state representations were a binary matrix of the positions of vehicles on the lanes of an intersection, and a combination of the presence matrix of vehicles, speed and the current traffic signal phase, respectively. However, we use raw visual input data of the traffic simulator snapshots as system states. Moreover, in addition to estimating Q-function, one of the proposed methods directly maps from the input state to a probability distribution over actions (i.e. signal phases) via deep PG method.

3 Background

In this section, we will review RL approaches and briefly describe how RL is applied to real-world problems where the number of states and actions are extremely high, so that the regular RL techniques cannot deal with them.

3.1 Reinforcement learning

A common RL [8] setting is shown in Fig. 1, where an RL agent interacts with an environment. The interaction is continued until reaching a terminal state or the agent meets a termination condition. Usually, the problems that RL techniques are applied to are treated as MDPs. An MDP is defined as a five tuple $\langle S, A, T, R, \gamma\rangle$, where S is the set of states in the state space of the environment, A is the set of actions in the action space that the agent can use in order to interact with the environment, T is the transition function, which is the probability of moving between the environment states, R is the reward function and $\gamma \in [0, 1]$ is known as the discount factor, which models the importance of the future and immediate rewards. At each time step t, the agent perceives the state $s_t \in S$ and, based on its observation, selects an action a_t. Taking the action, leads to the state of the environment transitions to the next states $s_{t+1} \in S$ regarding the transition function T. Then, the agent receives reward r_t which is determined by the reward function R.

The goal of the learning agent in RL framework is to learn an optimal policy $\pi: S \times A \to [0, 1]$ which defines the probability of selecting action a_t in state s_t, so that with following the underlying policy the expected cumulative discounted reward over time is maximised. The discounted future reward, R_t, at time t is defined as follows:

$$R_t = E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}\right], \tag{1}$$

where the role of the discount factor γ is to trade-off the worth of immediate and future rewards. In most real-world problems, there are many states and actions which make it impossible to apply classic RL techniques, which consider tabular representations for their state and action spaces. For example, in the problem of traffic light optimisation, that we are interested in this paper, the state spaces are continuous.

Hence, it is common to use function approximators [29] or decomposition and aggregation techniques such as HRL approaches [30–32] and advance HRL [33].

Different forms of function approximators can be used with RL techniques. For example, linear function approximation, a linear combination of feature of state and action spaces f and learned weights w (e.g. $\sum_i f_i w$) or a non-linear function approximation (e.g. a neural network). Until recently, the majority of work in RL has been applying linear function approximators. More recently, deep neural networks (DNNs) such as CNNs, recurrent neural networks, SAE etc. have also been commonly used as function approximators for large RL tasks [6, 34]. The interested readers are referred to [35] for a review of using DNNs with RL framework.

3.2 Deep learning and deep Q-learning

Deep learning techniques are one of the best solutions to address high-dimensional data and extract discriminative information from the data. Deep learning algorithms have the capability of automating feature extraction (the extraction of representations) from the data. The representation is learned through the data which are fed directly into deep nets without using human knowledge (i.e. automated feature extraction). Deep learning models contain multiple layers of representations. Indeed, it is a stack of building blocks such as auto-encoders, Restricted Boltzmann machines and convolutional layers. During training, the raw data is fed into a network consisting of multiple layers. The output of each layer which is non-linear feature transformations is used as inputs to the next layers of the DNN. The output representation of the final layer can be used for constructing classifiers or those applications which can have the better efficiency and performance with abstract representation of the data in a hierarchical manner as inputs. A non-linear transformation is applied at each layer on its input to try to learn and extract underlying explanatory factors. Consequently, this process learns a hierarchy of abstract representations.

One of the main advantages of DNNs is the capability of automating feature extraction from raw input data. A deep Q-learning network (DQN) [6] uses this benefit of deep learning in order to represent the agent's observation as an abstract representation in learning an optimal control policy. The DQN method aggregates a DNN function approximator with Q-learning to learn action-value function and as a result a policy π, the behaviour of the agent which tells the agent what action should be selected for each input state. Applying non-linear function approximators such as neural networks with model-free RL algorithms in high-dimensional continuous state and action spaces has some convergence problems [36]. The reasons for these issues are: (i) consecutive states in RL tasks have correlation. (ii) The underlying policy of the agent is changing frequently, because of slight changes in Q values. To cope with these problems, the DQN provides some solutions which improve the performance of the algorithm significantly. For the problem of correlated states, DQN uses the previously proposed experience replay approach [37]. In this way, at each time step, the DQN stores the agent's experience (s_t, a_t, r_t, r_{t+1}) into a data set D, where s_t, a_t and r_t are the state, chosen action and received reward, respectively, and s_{t+1} is the state at the next time step. To update the network, the DQN utilises stochastic minibatch updates with uniformly random sampling from the experience replay memory (previously observed transitions) at training time. This negates strong correlations between consecutive samples. Another approach to deal with aforementioned convergence issues, which we also examine in this research, is the PG methods. This approach has demonstrated better convergence properties in some RL problems [38].

3.3 PG methods

A PG method tries to optimise a parameterised policy function by gradient-descent method. Indeed, PG methods are interested in searching policy space to learn policies directly, instead of estimating state-value or action-value functions. Unlike the traditional RL algorithms, PG methods do not suffer from the convergence problems of estimating value functions under non-linear function approximation or in the environments which might be POMDPs. They can also deal with the complexity of continuous state and action spaces better than purely value-based methods [38]. PG methods estimate policy gradients using Monte Carlo estimates of the policy gradients [39]. These methods are guaranteed to converge to a local optimum of their parameterised policy function. However, typically PG methods result in high variance in their gradient estimates. Hence, in order to reduce the variance of the gradient estimators, some methods subtract a baseline function from the policy gradients. The baseline function can be calculated in different manners [40, 41]. By inspiring these features of PG methods and successes of neural networks in automatic feature abstractions, we use DNNs to represent an optimal traffic control policy directly in the traffic signal control problem.

4 System description

In this section, we will formulate TLC problem as an RL task by describing the states, actions and reward function. We then present the policy as a DNN and how to train the network.

4.1 State representation

We represent the state of the system as an image $s_t \in R^d$ or a snapshot of the current state of a graphical simulator {e.g. SUMO-graphical user interface (GUI) [42]} which is a vector of raw pixel values of current view of the intersection at each step of simulation (as shown in Fig. 1). This kind of representation is such as putting a camera on an intersection which enables it to view the whole intersection. The state representation in the TLC literature usually uses a vector representing the presence of a vehicle at the intersection, a Boolean-valued vector where a value 1 indicates the presence of a vehicle and a value 0 indicates the absence of a vehicle [14, 43] or a combination of the presence vector with another vector indicating the vehicle's speed at the given intersection [15]. Regardless of these states representations that are using a prior knowledge provided, they make assumptions which are not generalisable for the real world. For instance, they discretise a lane segment of an intersection into cells with a constant length c which is supposed to be the vehicle length to build the vehicle's speed and presence vectors. However, by feeding the state as an image to a CNN, the system can detect the location and the presence of all vehicles with different lengths and as a result the vehicles' queue on each lane. Furthermore, by stacking a history of consecutive observations as input, the convolutional layers of a deep network are able to estimate velocity and travel direction of vehicles. Hence, the system can implicitly benefit from this information as well.

4.2 Action set

To control traffic signal phases, we define a set of possible actions A = {North/South green (NSG), East/West green (EWG)}. NSG allows vehicles to pass from North-to-South and vice versa and also indicates the vehicles on East/West route should stop and not proceed through the intersection. EWG allows vehicles to pass from East to West and vice versa and implies the vehicles on North/South route should stop and not proceed through the intersection. At each time step t, an agent regarding its strategy chooses an action $a_t \in A$. Depending on the selected action, the vehicles on each lane are allowed to cross the intersection.

4.3 Reward function

Typically, an immediate reward $r_t \in \mathbb{R}$ is a scalar value which the agent receives after taking the chosen action in the environment at each time step. We set the reward as the difference between the total cumulative delays of two consecutive actions, i.e.

$$r_t = D_{t-1} - D_t, \tag{2}$$

where D_t and D_{t-1} are the total cumulative delays in the current and previous time steps. The total cumulative delay at time t is the summation of the cumulative delay of all the vehicles appeared from $t = 0$ to current time step t in the system. The positive reward values imply the taken actions led to decrease the total cumulative delay and the negative rewards imply an increase in the delay. With regard to the reward values, the agent may decide to change its policy in certain states of the system in the future.

4.4 Agent's policy

The agent chooses the actions based on a policy π. In the policy-based algorithm, the policy is defined as a mapping from the input state to a probability distribution over actions A. We use the DNN as the function approximator and refer its parameters $\boldsymbol{\theta}$ as policy parameters. The policy distribution $\pi(a_t | s_t; \boldsymbol{\theta})$ is learned by performing gradient descent on the policy parameters. In the value-function-based algorithm, the DNN is utilised to estimate the action-value function. The action-value-function maps the input state to action values, which each represents the future reward that can be achieved for the given state and action. The optimal policy can then be extracted by performing a greedy approach to select the best possible action.

4.5 Objective function and system training

There are many measures such as maximising throughput, minimising and balancing queue length, minimising the delay etc. in the traffic signal management literature to consider as the learning agent's objective function. In this research, the agent aims to maximise the reduction in the total cumulative delay, which empirically has been shown to maximise throughput and to reduce queue length (more details discussed in Section 5.3).

The objective of the agent is to maximise the expected cumulative discounted reward. We aim to maximise the reward under the probability distribution $\pi(a_t | s_t; \boldsymbol{\theta})$.

$$J(\boldsymbol{\theta}) = E_{\pi_\theta}\left[\sum_{t=0}^{T} \gamma^t r_t\right] = E_{\pi_\theta}[R]\,. \tag{3}$$

We divide the system training based on two RL approaches: value-function-based and policy-based. In *value-function-based approach*, the value function $Q_\pi(s, a)$ is defined as follows:

$$Q_\pi(s, a) = E_\pi[r_t + \gamma \max_{a'} Q(s', a') | s, a], \tag{4}$$

where it is implicit that $s, s' \in S$ and $a \in A$. The value function can be parameterised $Q(s, s; \boldsymbol{\theta})$ with parameter vector $\boldsymbol{\theta}$. Typically, the gradient-descent methods are used to learn parameters, $\boldsymbol{\theta}$ by trying

```
1:  Initialize parameters, $\theta$ with random values
2:  Initialize replay memory $M$ with capacity $L$
3:  for each simulation do
4:      initialize $s$ with current view of the intersection
5:      repeat                                  # each step in the simulation
6:          choose action $a$ according to $\epsilon$-greedy policy
7:          take action $a$, observe reward $r$ and next state $s'$
8:          store transition $(s, a, r, s')$ in $M$
9:          $s \leftarrow s'$
10:         $b \leftarrow$ sample random minibatch of transitions from the
            replay memory, $M$
11:         for each transition $(s_j, a_j, r_j, s'_j)$ in $b$ do
12:             if $s'_j$ is terminal then
13:                 $y_i \leftarrow r_j$
14:             else
15:                 $y_j = r_j + \gamma max_{a'} Q(s'_j, a'; \theta^-)$
16:             end if
17:             update parameters $\theta$ according to equation (7)
18:         end for
19:     until $s$ is terminal
20: end for
```

Fig. 2 *Algorithm 1: Deep value-function-based RL agent of traffic signal control with experience replay*

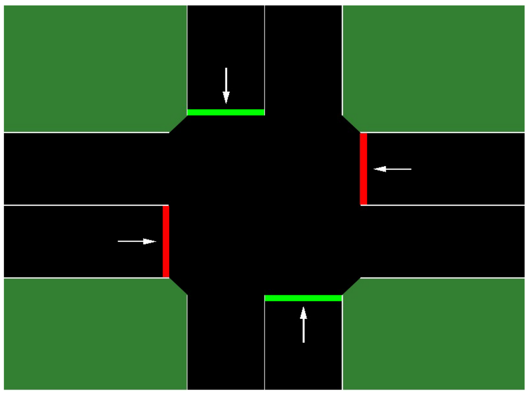

Fig. 3 *Algorithm 2: Deep PG-based RL agent of traffic signal control*

to minimise the following loss function of mean-squared error in Q values,

$$J(\theta) = E_\pi[(r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}) - Q(s, a; \boldsymbol{\theta}))^2], \tag{5}$$

where $r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta})$ is the target value. In the DQN algorithm, a target Q-network is used to address the instability problem of the policy. The network is trained with the target Q-network to obtain consistent Q-learning targets by keeping the weight parameters ($\boldsymbol{\theta}^-$) used in the Q-learning target fixed and updating them periodically every N steps through the parameters of the main network $\boldsymbol{\theta}$. The target value of the DQN is represented as follows:

$$y_i = r + \gamma \max_{a'} Q(s', a'; \boldsymbol{\theta}^-), \tag{6}$$

where $\boldsymbol{\theta}^-$ are parameters of the target network. The stochastic gradient-descent method is used in order to optimise (5). The parameters of the deep Q-learning algorithm are updated as follows:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} + \alpha(y_i - Q(s, a; \boldsymbol{\theta}_i))\nabla_{\theta_i} Q(s, a; \boldsymbol{\theta}_i), \tag{7}$$

where y_i is the target value for iteration i and α is a scalar learning rate. Algorithm 1 (see Fig. 2) presents the pseudo-code for the training algorithm.

In *policy-based approach*, the gradient of the objective function represented in (3) is given by:

$$\nabla_\theta J = \sum_{t=0}^{T} E_{\pi_\theta}[\nabla_\theta \log(a_t | s_t; \boldsymbol{\theta}) R_t]. \tag{8}$$

This (8) is standard learning rule of the REINFORCE algorithm [44]. It updates the policy parameters $\boldsymbol{\theta}$ in the direction $\nabla_\theta \log(a_t | s_t; \boldsymbol{\theta})$, so that the probability of action a_t at state s_t is increased if it has led to high cumulative reward; however, it is decreased if the action has result in a low reward. The gradient estimate in (2) results to have high variance. It is common to reduce the variance by subtracting a baseline function $b_t(s_t)$ from the return R_t, without changing expectation. Commonly, an estimate of the state-value function is used as the baseline, $b_t(s_t) = V^{\pi_{\theta_v}}(s_t)$. Thus, the adjusted gradient is $\nabla_\theta \log(a_t | s_t; \boldsymbol{\theta})(R_t - b_t(s_t))$. The value $R_t - b_t$ is known as the *advantage function*.

With regard to the advantage actor–critic method [45], computing a single update is done by selecting actions using the underlying policy for up to M steps or till a terminal state is met. In this way, the agent obtains up to M rewards from the environment at each update point and updates the policy parameters after every $n \le M$ steps regarding n-step returns. The vector parameters $\boldsymbol{\theta}$ are updated through the stochastic gradient-descent method:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \sum_t \nabla_\theta \log(a_t | s_t; \boldsymbol{\theta}) A(s_t, a_t; \boldsymbol{\theta}, \boldsymbol{\theta}_v), \tag{9}$$

where $A(s_t, a_t; \boldsymbol{\theta}, \boldsymbol{\theta}_v)$ is an estimate of the *advantage function* corresponding $\sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n V(s_{t+n}; \boldsymbol{\theta}) - V(s_t; \boldsymbol{\theta}_v)$, where n might have different values with respect to the state, up to M. This process is an actor–critic algorithm, the policy $\pi(a_t | s_t; \boldsymbol{\theta})$ refers to the actor and the estimate of the state-value function $V^{\pi_{\theta_v}}(s_t)$ implies to the critic [45, 46]. Algorithm 2(see Fig. 3) shows the pseudo-code for the training algorithm.

5 Experiment and results

In this section, we present the simulation environment, where our experiments have been done. We then describe the details of the DNN utilised including hyper-parameters to represent the agent's policy.

5.1 Experiment setup

We have used the SUMO [42] tool to simulate traffic in all experiments. SUMO is a well known open source traffic simulator which provides useful application programming interfaces and a GUI view to model large road networks as well as some possibilities to handle them. In particular, we utilised SUMO–GUI v0.28.0 as it allows to have snapshots of each step of the simulation. The intersection geometry used in this paper is shown in Fig. 4. There are four incoming lanes to the intersection and four outgoing lanes from the intersection. To generate traffic demands from different directions (i.e. North-to-South and West-to-East and vice versa) to the road network, we randomly sample from a uniform probability distribution with the probability of 0.1 to model vehicle generation at each 3600 time steps.

5.2 System architecture and hyper-parameters

We took the snapshots from the SUMO–GUI and did some basic pre-processing. The snapshots are converted from red–green–blue representation to grey-scale and resized them to 128×128 frames. To enable our system to memorise a history of the past observations, we stacked the last four frames of the history and provided them to the system as input. So, the input to the network was a $128 \times 128 \times 4$ image. We applied approximately the same architecture of the deep Q-network (DQN) algorithm introduced by Mnih *et al.* [6, 7]. The network consists of a stack of two convolutional layers with 16 8×8 and 32 4×4 filters with strides 4 and 2, respectively. The final hidden layer is fully connected with 256 hidden nodes. All three hidden layers are followed by a

```
1:  Initialize parameters, $\theta$, $\theta_v$ with random values
2:  Initialize step counter $t \leftarrow 0$
3:  for each simulation do
4:      initialize $s$ with current view of the intersection
5:      $t_{start} = t$
6:      repeat
7:          perform action $a$ according to policy $\pi(a|s;\theta)$
8:          observe reward $r$ and next state $s'$
9:          $t \leftarrow t + 1$
10:     until $s$ is terminal or $t - t_{start} == M$ (Max step)
11:     if $s$ is terminal then
12:         $R = 0$
13:     else
14:         $R = V(s;\theta_v)$
15:     end if
16:     for $i \in \{t-1, ..., t_{start}\}$ do    # $n <= M$ times
17:         $R \leftarrow r_i + \gamma R$
18:         $\theta \leftarrow \theta + \alpha \nabla_\theta log(a_i|s_i;\theta)(R - V(s_i;\theta_v))$
19:         $\theta_v \leftarrow \theta_v + \frac{\partial (R - V(s_i;\theta_v))^2}{\partial \theta_v}$
20:     end for
21: end for
```

Fig. 4 *Intersection geometry for the traffic simulation*

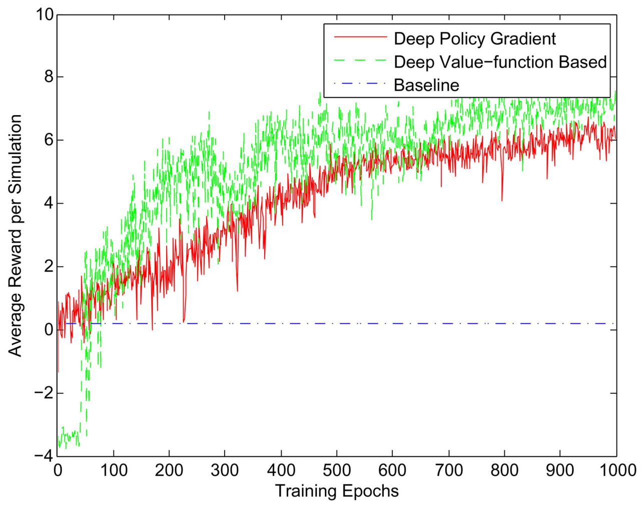

Fig. 5 *Comparison of performance of the average reward received during the evaluation time for the proposed method and the baseline*

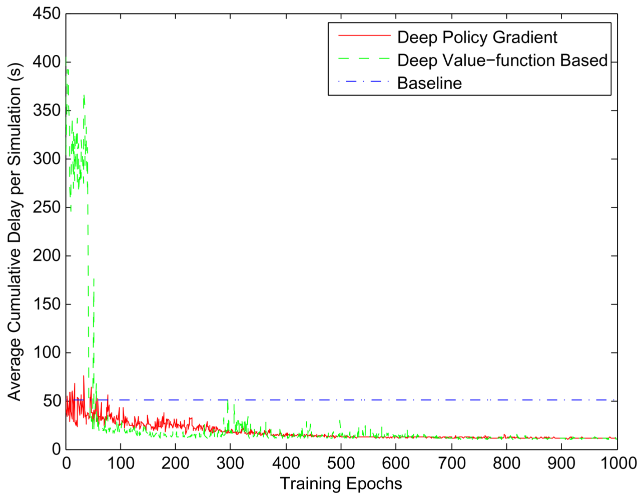

Fig. 6 *Average cumulative delay per vehicle using the suggested model and the baseline during the evaluation time*

rectifier non-linearity. The main difference with the network architecture of the DQN method is the last layer, where the last layer of DQN is a fully connected linear layer with a number of output neurons [i.e. Q values $Q(a, s)$] corresponding to each action in a given Atari 2600 game, while in policy-based model the last layer represents two sets of outputs, a softmax output resulting in a probability distribution over the actions A [i.e. the policy $\pi(a, s)$], and a single linear output node resulting in the estimate of the state-value function $V(s)$. For value-function model we used the architecture, the same as the DQN. The output layer is corresponding to action values. In all of our experiments, the discount factor was set to $\gamma = 0.99$ and all weights of the network were updated by the Adam optimiser [47] with a learning rate $\alpha = 0.00001$ and with mini batches of size M (up to 32), the maximum number of steps that the agent can take to follow its policy and afterwards need to update it. The network was trained for about 1050 epoch, ~2 million time steps. Each epoch is corresponded ten episodes and each episode was a complete SUMO–GUI simulation. The learned policies by the agent was evaluated every ten episodes by running SUMO–GUI for five episodes and averaging the resulting rewards, total cumulative delay and queue length.

To evaluate our proposed method, we also built a shallow neural network (SNN) with one hidden layer. The hidden layer has 64 hidden nodes followed by a rectifier non-linearity. The output layer is a fully connected linear layer with a number of output neurons corresponding to each traffic signal phase in the intersection. Two vectors are used as input state of the network. The first representing the number of queued vehicles at the lanes of the intersection (i.e. North, South, East and West) and the second corresponding to the current traffic signal phase of the intersection. SNN is trained with the same hyper-parameters and optimisation method (i.e. the gradient decent algorithm) as the proposed methods.

5.3 Results and discussion

To evaluate the performance of the proposed methods, we compared them against a baseline traffic controller, a controller that gives an equal fixed time to each phase of the intersection. We ran SUMO–GUI simulator for the proposed model using the configuration setting explained in Section 5.2 and compared the average reward, average total cumulative delay and average queue length achieved to the baseline. Fig. 5 shows the received average reward while the agent follows a certain policy. As shown in Fig. 5, the proposed method performs significantly better than the baseline and results more reward magnitudes by doing more epochs. This gradually increasing reward reflects the agent's ability to learn an optimal control policy in a stable manner. Unlike using deep RL for estimating the Q values in traffic light optimisation problem [14], the proposed agent does not suffer stability issues. To assess the learned policy by the agent, two of the most common performance metrics in the traffic signal control literature is implemented: the cumulative delay and queue length. Figs. 6 and 7 illustrate the performance comparison of the leaning agent regarding average cumulative delay time and average queue length metrics, respectively, to the baseline, while the agent is following the learning policy over time. The plots clearly show the agent is able to find a policy resulting minimising queue length and total cumulative delay. Moreover, these graphs reveal that by using the reward function for reducing cumulative delay, the intersection queue length is reduced as well as the total cumulative delay of all vehicles.

We also compared the proposed methods with the SNN, which is a shallow neural network with one hidden layer. Table 1 reports a comparison of the proposed models and the SNN model in terms of the average and standard deviation (μ, σ) of average queue length, average cumulative delay time and the received average reward metrics. The results on Table 1 are calculated from the last 100 training epochs of each method. Comparing the metrics shown in Table 1, demonstrates that the proposed models significantly outperform the SNN method. On the basis of the data in Table 1 we can induce 67 and 72% reductions in the average cumulative delay and queue length for the PG method and 68 and 73% reductions for value-function-based method compared with the SNN. Furthermore, we can see that the proposed methods have received average rewards superior to the SNN. Considering these results, it

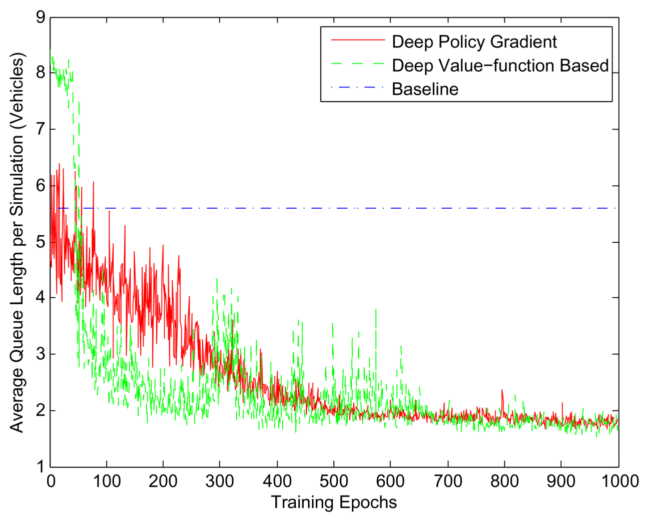

Fig. 7 *Average queue length of the intersection using the proposed model and the baseline during the evaluation time*

Table 1 Comparison of performance of the proposed methods against the SNN

Evaluation metric (μ, σ; $n = 100$)	Policy based	Value-function based	SNN
queue (vehicles)	(1.79, 0.073)	(1.74, 0.10)	(5.55, 0.73)
cumulative delay, s	(11.25, 0.39)	(11.01, 0.69)	(41.40, 7.31)
reward (r)	(6.14, 0.29)	(7.14, 0.44)	(1.73, 0.62)

is obvious that the policy gradient and value-function agents could learn the control policies better than the SNN approach.

6 Conclusion

In this paper, we applied deep RL algorithms with focusing on both policy and value-function-based methods to traffic signal control problem in order to find optimal control policies of signalling, just by using raw visual input data of the traffic simulator snapshots. Our approaches have led to promising results and showed they could find more stable control policies compared with the previous work of using deep RL in traffic light optimisation. In our work, we developed and tested the proposed methods in a small application, extending the work for more complex traffic simulations, for instance considering many intersections and multiple agents to control each intersection, using multi-agent learning techniques to handle coordination problem between agents would be a direction for future research.

7 Acknowledgments

We thank Peter Corcoran, IP manager and Consultant at FotoNation company, for letting us to work with their GPU cluster.

8 References

[1] Li, L., Wen, D., Yao, D.: 'A survey of traffic control with vehicular communications', *IEEE Trans. Intell. Transp. Syst.*, 2014, **15**, (1), pp. 425–432

[2] Balaji, P., German, X., Srinivasan, D.: 'Urban traffic signal control using reinforcement learning agents', *IET Intell. Transp. Syst.*, 2010, **4**, (3), pp. 177–188

[3] Abdoos, M., Mozayani, N., Bazzan, A.L.: 'Holonic multi-agent system for traffic signals control', *Eng. Appl. Artif. Intell.*, 2013, **26**, (5), pp. 1575–1587

[4] Li, L., Wen, D.: 'Parallel systems for traffic control: a rethinking', *IEEE Trans. Intell. Transp. Syst.*, 2016, **17**, (4), pp. 1179–1182

[5] El. Tantawy, S., Abdulhai, B., Abdelgawad, H.: 'Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (marlin-atsc): methodology and large-scale application on downtown Toronto', *IEEE Trans. Intell. Transp. Syst.*, 2013, **14**, (3), pp. 1140–1150

[6] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.*: 'Playing atari with deep reinforcement learning', arXiv preprint arXiv:13125602, 2013

[7] Mnih, V., Kavukcuoglu, K., Silver, D., *et al.*: 'Human-level control through deep reinforcement learning', *Nature*, 2015, **518**, (7540), pp. 529–533

[8] Sutton, R.S., Barto, A.G.: '*Introduction to reinforcement learning*' (MIT Press, 1998)

[9] Prashanth, L., Bhatnagar, S.: 'Reinforcement learning with function approximation for traffic signal control', *IEEE Trans. Intell. Transp. Syst.*, 2011, **12**, (2), pp. 412–421

[10] Duggan, M., Flesk, K., Duggan, J., *et al.*: 'A reinforcement learning approach for dynamic selection of virtual machines in cloud data centres'. Sixth Int. Conf. Innovating Computing Technology, 2016

[11] Duggan, M., Duggan, J., Howley, E., *et al.*: 'An autonomous network aware VM migration strategy in cloud data centres'. 2016 Int. Conf. Cloud and Autonomic Computing (ICCAC), 2016, pp. 24–32

[12] Silver, D., Huang, A., Maddison, C.J., *et al.*: 'Mastering the game of go with deep neural networks and tree search', *Nature*, 2016, **529**, (7587), pp. 484–489

[13] LeCun, Y., Bottou, L., Bengio, Y., *et al.*: 'Gradient-based learning applied to document recognition', *Proc. IEEE*, 1998, **86**, (11), pp. 2278–2324

[14] van der Pol, E., Oliehoek, F.A.: 'Coordinated deep reinforcement learners for traffic light control', 2016

[15] Genders, W., Razavi, S.: 'Using a deep reinforcement learning agent for traffic signal control', arXiv preprint arXiv:161101142, 2016

[16] Wiering, M., *et al.*: 'Multi-agent reinforcement learning for traffic light control'. ICML, 2000, pp. 1151–1158

[17] Abdulhai, B., Pringle, R., Karakoulas, G.J.: 'Reinforcement learning for true adaptive traffic signal control', *J. Transp. Eng.*, 2003, **129**, (3), pp. 278–285

[18] Brockfeld, E., Barlovic, R., Schadschneider, A., *et al.*: 'Optimizing traffic lights in a cellular automaton model for city traffic', *Phys. Rev. E*, 2001, **64**, (5), p. 056132

[19] Khamis, M.A., Gomaa, W., El.Mahdy, A., *et al.*: 'Adaptive traffic control system based on Bayesian probability interpretation'. Conf. Electronics, Communications and Computers (JEC-ECC), 2012, Japan-Egypt, 2012, pp. 151–156

[20] Khamis, M.A., Gomaa, W., El.Shishiny, H.: 'Multi-objective traffic light control system based on Bayesian probability interpretation'. 2012 15th Int. IEEE Conf. Intelligent Transportation Systems (ITSC), 2012, pp. 995–1000

[21] Chin, Y.K., Bolong, N., Kiring, A., *et al.*: 'Q-learning based traffic optimization in management of signal timing plan', *Int. J. Simul. Syst. Sci. Technol.*, 2011, **12**, (3), pp. 29–35

[22] Arel, I., Liu, C., Urbanik, T., *et al.*: 'Reinforcement learning-based multi-agent system for network traffic signal control', *IET Intell. Transp. Syst.*, 2010, **4**, (2), pp. 128–135

[23] Khamis, M.A., Gomaa, W.: 'Adaptive multi-objective reinforcement learning with hybrid exploration for traffic signal control based on cooperative multiagent framework', *Eng. Appl. Artif. Intell.*, 2014, **29**, pp. 134–151

[24] Khamis, M.A., Gomaa, W.: 'Enhanced multiagent multi-objective reinforcement learning for urban traffic light control'. 2012 11th Int. Conf. Machine Learning and Applications (ICMLA), 2012, vol. **1**, pp. 586–591

[25] Ritcher, S.: 'Traffic light scheduling using policy-gradient reinforcement learning'. Int. Conf. Automated Planning and Scheduling., 2007

[26] Deng, L.: 'A tutorial survey of architectures, algorithms, and applications for deep learning', *APSIPA Trans. Signal Inf. Process.*, 2014, **3, p**.e2

[27] Khamis, M., Gomaa, W., Galal, B.: 'Deep learning is competing random forest in computational docking', arXiv preprint arXiv:160806665, 2016

[28] Li, L., Lv, Y., Wang, F.Y.: 'Traffic signal timing via deep reinforcement learning', *IEEE/CAA J. Autom. Sin.*, 2016, **3**, (3), pp. 247–254

[29] Xu, X., Zuo, L., Huang, Z.: 'Reinforcement learning algorithms with function approximation: recent advances and applications', *Inf. Sci.*, 2014, **261**, pp. 1–31

[30] Barto, A.G., Mahadevan, S.: 'Recent advances in hierarchical reinforcement learning', *Discrete Event Dyn. Syst.*, 2003, **13**, (4), pp. 341–379

[31] Ghazanfari, B., Mozayani, N.: 'Enhancing nash q-learning and team q-learning mechanisms by using bottlenecks', *J. Intell. Fuzzy Syst.*, 2014, **26**, (6), pp. 2771–2783

[32] Mousavi, S.S., Ghazanfari, B., Mozayani, N., *et al.*: 'Automatic abstraction controller in reinforcement learning agent via automata', *Appl. Soft Comput.*, 2014, **25**, pp. 118–128

[33] Ghazanfari, B., Mozayani, N.: 'Extracting bottlenecks for reinforcement learning agent by holonic concept clustering and attentional functions', *Expert Syst. Appl.*, 2016, **54**, pp. 61–77

[34] Lange, S., Riedmiller, M.: 'Deep auto-encoder neural networks in reinforcement learning'. 2010 Int. Joint Conf. Neural Networks (IJCNN), 2010, pp. 1–8

[35] Mousavi, S.S., Schukat, M., Howley, E.: 'Deep reinforcement learning: an overview'. Intelligent Systems Conf., 2016

[36] Tsitsiklis, J.N., Van Roy, B.: 'An analysis of temporal-difference learning with function approximation', *IEEE Trans. Autom. Control*, 1997, **42**, (5), pp. 674–690

[37] Lin, L.J.: '*Reinforcement learning for robots using neural networks*' (Fujitsu Laboratories Ltd., 1993)

[38] Sutton, R.S., McAllester, D.A., Singh, S.P., *et al.*: 'Policy gradient methods for reinforcement learning with function approximation'. NIPS, 1999, vol. **99**, pp. 1057–1063

[39] Baxter, J., Bartlett, P.L., Weaver, L.: 'Experiments with infinite-horizon, policy gradient estimation', *J. Artif. Intell. Res.*, 2001, **15**, pp. 351–381

[40] Schulman, J., Heess, N., Weber, T., *et al.*: 'Gradient estimation using stochastic computation graphs'. Advances in Neural Information Processing Systems, 2015, pp. 3528–3536

[41] Wierstra, D., Förster, A., Peters, J., *et al.*: 'Recurrent policy gradients', *Log. J. IGPL*, 2010, **18**, (5), pp. 620–634

[42] Krajzewicz, D., Erdmann, J., Behrisch, M., *et al.*: 'Recent development and applications of SUMO-simulation of urban mobility', *Int. J. Adv. Syst. Meas.*, 2012, **5**, (3&4), pp. 128–138

[43] Thorpe, T.L., Anderson, C.W.: '*Traffic light control using SARSA with three state representations*' (IBM Corporation, 1996)

[44] Williams, R.J.: 'Simple statistical gradient-following algorithms for connectionist reinforcement learning', *Mach. Learn.*, 1992, **8**, (3-4), pp. 229–256

[45] Mnih, V., Badia, A.P., Mirza, M., *et al.*: 'Asynchronous methods for deep reinforcement learning'. Int. Conf. Machine Learning, 2016

[46] Degris, T., Pilarski, P.M., Sutton, R.S.: 'Model-free reinforcement learning with continuous action in practice'. 2012 American Control Conf. (ACC), 2012, pp. 2177–2182

[47] Kingma, D., Ba, J.: 'Adam: a method for stochastic optimization', arXiv preprint arXiv:14126980, 2014