STM JOURNALS

Journal of
Artificial Intelligence Research & Advances

ISSN: 2395-6720
Volume 8, Issue 3, 2021
DOI Journal: 10.37591/JoAIRA

http://computers.stmjournals.com/index.php?journal=JoAIRA&page=index

Research JoAIRA

Comparative Study of Reinforcement Learning Algorithms on Traffic Light Control System

Partha Ghosh[1,*], Anirban Dan[2], Abhi Goswami[2], Nilagnik Chakraborty[2], Amit Chakraborty[2]

Abstract

With the changing times, the need of upgraded and efficient state-of-the-art traffic light control system is truly required. How well do various state-of-the-art algorithms handle complex real-life situations, more technically speaking, how well the reward function operates and minimizes the waiting time, is the prime question of the hour. Here we have successfully employed three reinforcement learning agents-REINFORCE, D3QN and DDPG each of which can independently handle large volumes of traffic movement and minimize signal waiting time, decision-making capability is strictly stochastic and beyond blind guess. The D3QN agent is characterized as a fast learner with only value-based approach. REINFORCE utilizes policy-based optimization and performs close to D3QN when trained on a higher number of episodes. DDPG utilizes both value and policy based approach to reduce waiting time; however the training time is significantly higher due to its complex network architecture. Our proposed work lays a foundation on multi-agent as well as mixed modelling decision making algorithms for traffic light control system which shall be a significant step in not only bringing down emissions and preserving ecology but also open various new research domains, especially in real time multimedia processing by intelligent agents.

Keywords: Traffic Light Control System (TLCS), Reinforcement Learning (RL), Deep Reinforcement learning (DRL), Deep Q Network (DQN), Deep Deterministic Policy Gradient (DDPG), Dueling Double Deep Q-Network (D3QN)

INTRODUCTION

A problem in today's society is the quantity of traffic because of the excessive use of cars and similar vehicles for transportation increasing day by day. Intersection contributes to the present downside joined or many traffic flows should look forward to another. Traffic management systems, within the kind of traffic lights are accustomed to handle these kinds of conflicting traffic flows. However, there are inefficiencies within these systems. It is highly complicated to implement a smart traffic light system. Many external factors such as the existence of several traffic lights, pedestrian lights, multiple vehicle movements, and pedestrian movements play a decisive role when controlling such a system. Any blunders may bring about congestion and the lack of time, cash and effort. In addition, in a city, traffic flow through one intersection can affect other intersections increasing the level of complexity.

A crucial part of making the streets smarter is utilizing traffic signals that use sensor data, communication, and intelligent adaptive algorithms to decrease traffic congestion [1]. In other words, signals should have the capability to adjust in real-time to traffic conditions resulting in

***Author for Correspondence**
Partha Ghosh
E-mail: parth_ghos@rediffmail.com

[1]Assistant Professor, Department of Computer Science and Engineering, Government College of Engineering & Ceramic Technology, Kolkata, West Bengal, India
[2]Student, Department of Computer Science and Engineering, Government College of Engineering & Ceramic Technology, Kolkata, West Bengal, India

Received Date: December 06, 2021
Accepted Date: December 12, 2021
Published Date: December 20, 2021

Citation: Partha Ghosh, Anirban Dan, Abhi Goswami, Nilagnik Chakraborty, Amit Chakraborty. Comparative Study of Reinforcement Learning Algorithms on Traffic Light Control System. Journal of Artificial Intelligence Research & Advances. 2021; 8(3): 33–45p.

minimizing both the hassle and cost of commuting as well as reducing the carbon footprint of the vehicles by making a system which reduces fuel wastage during signal wait as well as minimizing traffic interference. A smart transportation system is expected to also minimize the number of accidents which will lead to fewer fatalities, fewer patients in hospitals and less pressure on emergency response systems. This will in turn release funds which might be used to make the life of citizens better rather than buying more supplies for hospitals and emergency vehicles and increasing their number.

The system is also expected to benefit individual commuters by lowering insurance rates (since there are fewer accidents happening), fuel cost and cost of lost time and arriving late to work/school. We came up with staggering results where we saw that up to 22% total travel time was wasted in stoppages. The average speed of vehicles during office hours was approximately 20–25 km/h. The pedestrians and local auto services added more complexity.

Road congestions are increasing the pollution level every day, a statistic by the Centre for Science and Environment (CSE) revealed that congestion on Delhi roads is increasing 7% annually as about 537 cars and 1,158 two-wheelers are added every day on these roads, the organization estimates [2]. The awareness to use public transport is somewhat lacking. The articles also stated almost no difference between peak and non-peak hours where traffic flows at an averages speed of 26 km/h. Several procedures have been used in the past to lower the vehicle waiting time, for example by adding extra road capacity or by deploying round about. But those techniques are high priced and are not in any respect realistically feasible. Also, various traffic scheduling methodologies have been brought up all across the globe. But they all work well in certain state space or situation.

With the arrival of device learning, diverse complicated troubles were solved efficiently. This has been possible due to the sudden increase in the amount of data coming from various gadgets. Traffic light control comes under the section of unsupervised algorithms called reinforcement learning where agent has no data about the environment beforehand. There are various reinforcement learning to choose to solve the given task. Our focus will be to compare the algorithms with the help of the simulator and derive meaningful conclusions which will help us to use the right algorithm to tackle traffic control system in right way. Future work will depict how our conclusions go with real life scenario. For the given task in hand, all the experiments have been done first in isolated simulating environment. For this purpose, we used Simulation of Urban Mobility (SUMO) which depicted a real-life traffic intersection scenario.

RELATED WORK

A lot of work has already been done in this field. MARL i.e. Multi-agent reinforcement learning was introduced by Chu *et al.* to handle large state-action space in practical traffic control scenario [3]. They first introduced MARL in context of A2C and applied it on Monaco traffic network. Their method outperformed all existing MARL based algorithms.

After various DRL algorithms came into picture, the need of an efficient hybrid DRL in TLCS was brought by Lin *et al*. to handle the complex intersection control problem [4]. They used a hybrid reward function comprising of local and global reward. They trained the agent as an Actor-Critic model and adopted clipped PPO as policy update procedure. The waiting time went down by 18.65 and 28.45%.

Abdul Aziz *et al*. utilized RMART aka R-Markov Average Reward Technique in TLCS by drawing comparison with Q-learning and SARSA [5]. The model outperformed them when data from neighborhood increased. The paper stated that RMART not only lowered release of polluting gases but also conserved a lot of energy.

Schulman *et al*. [6] at OpenAI handled the limitations of REINFORCE [7] algorithm by proximal policy based algorithms. The essence of these algorithms lied in calculating the clipped-objective surrogate function to perform multiple updates on it via gradient ascent. Actor-critic algorithms introduced by Minh *et al*. are at the junction of both value-based and policy methods by eliminating the negatives and combining the positive aspects of both [7]. They come out to be with less variability and increased stability as the subtraction of expected cumulative reward from baseline is done. It also requires lesser samples than policy-based methodology. Bakker *et al*. were one of the first to research and improve MARL algorithms in traffic control systems [8]. They improved their network via three extensions-traffic congestion, observation of traffic states by applying belief states and managing the behavior of multiple agents via max-plus algorithm. Lowrie developed and utilized SCATS in Sydney, Australia [9]. It showed intelligent management of the signal phases at the traffic signal, which means that it tries to search out the most effective phasing (i.e. cycle times, part splits and offsets) for a traffic (for individual intersections also as for the entire network). It makes automated decisions from a server in response to the information derived from different sensors planted on road.

Yang *et al*. structures a vehicle detector and classifier by using a cost effective magnetic resistive sensor [10]. They had taken into consideration the functions of magnetic detection alerts, with unique emphasis on especially congested huge towns and used system gaining knowledge of set of rules primarily based totally to discover automobiles inside an unmarried lane and section the car alerts efficiently in step with the time facts of automobiles getting into and leaving the sensor tracking area. Smiths *et al*. created Surtrac's smart traffic-management system [11]. The Surtrac system utilizes radar-based sensors and the cameras located at each traffic light intersection, inputs them in various AI algorithms to build a timing plan and finally communicates with traffic intersections downstream so they can plan.

PROPOSED WORK

Sequence followed in paper:

1. Introduction to all the reinforcement algorithms considered in this paper.
2. Brief overview of SUMO simulator and necessary information regarding it.
3. Fit state vectors to various RL algorithms to generate reward and waiting time graphs with respect to no. of epochs.
4. The results are then analyzed based on our empirical experiments and at last we mention the future scope of the proposed work and references used throughout the paper.

The flow diagram of proposed work is shown below in Figure 1 where we discuss all the stages of our work in brief.

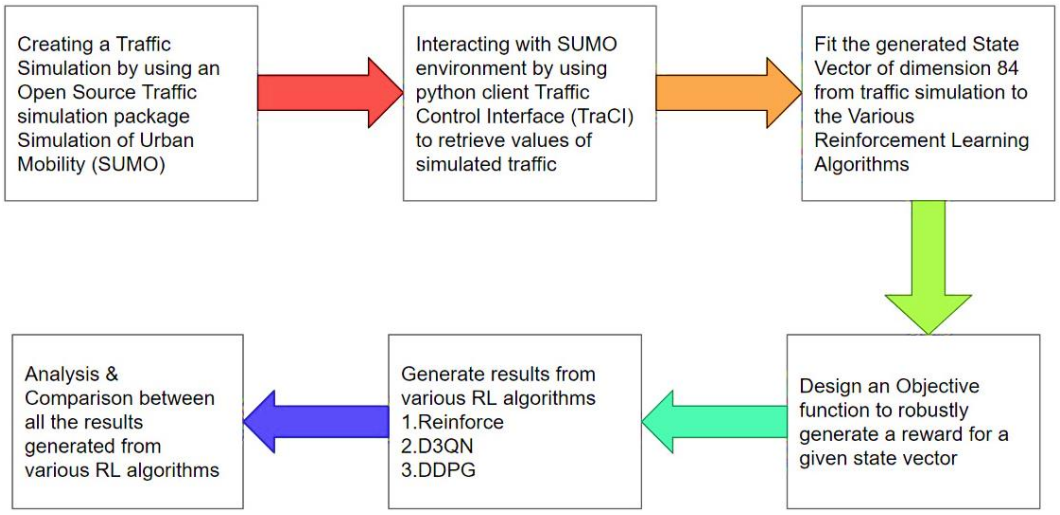

Figure 1. Sequence of tasks followed to accomplish the proposed work.

ALGORITHMS USED

The following are the algorithms used in this paper with their pseudocode:

Dueling Double Deep Q-Network

Q-Learning learns the motion-price feature Q(s, a) to apprehend how true to take a motion at a specific state. In Q studying we created a reminiscence desk to save Q values for all combos of states and actions. However, DQN, proper to its call makes use of a deep neural community that acts as a feature approximator [12]. It produces a vector of motion values because the closing layer of neural community. The most fee suggests the motion to take. Whereas in traditional reinforcement mastering algorithms in which best one Q fee is produced at a time, the DQN in a single ahead by skip produces a Q fee for each viable motion. The DQN consists of three convolution layers with RELU activation, one fully connected layer with one fully connected linear output layer. But still convergence of optimal value function is not guaranteed. There are chances of oscillation of network weights. This happens due to correlation between states and actions which promotes unstability. As we realize, each motion impacts the subsequent country in a few ways, therefore a series of skilled tuples may be surprisingly correlated. To combat this kind of correlation, we use experience replay where during interaction with environment we save each experienced tuple in a buffer and then sample out a small set of tuples from it in order to learn. Thus, we get threat to research from person tuples a couple of times, memorize uncommon occurrences, pattern skilled tuples from this buffer at random, and in fashionable use our enjoy well. Experienced replay helps us to combat one kind of correlation, i.e. between successive experience tuples. There is every other form of correlation among the goal and the parameters we are changing. We use fixed target setup by fixing the function parameters used to generate our target (Figure 2).

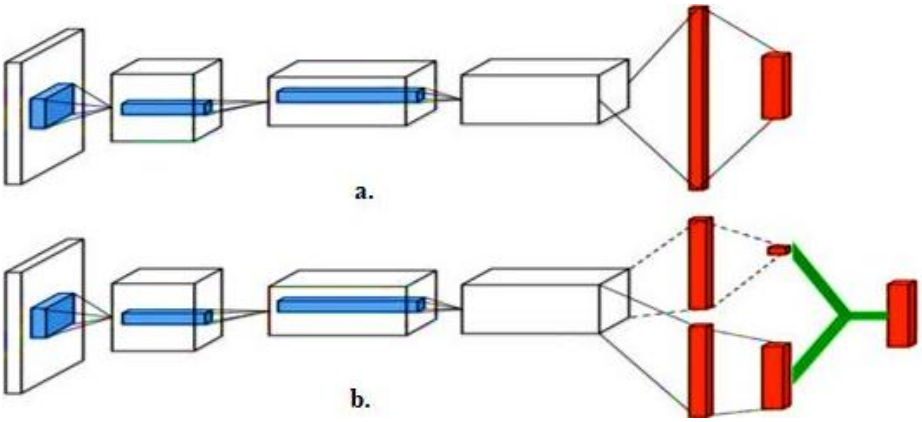

Figure 2. (a) Single stream Q-network; (b) Dueling Architecture [12].

Here, we discuss the pseudo code of DQN. First we initialize replay memory. Use of circular queue is preferred. Then initialize the weights. Then we feed four frames as input and did preprocessing like image resize and gray scaling. As the algorithm was created to tackle games, it runs algorithm for every episode. There are foremost strategies on this algorithm. First, sampling the environment by performing certain actions and storing the observed experienced tuples in the replay memory. Second, where we select a random small batch of tuples from this memory and use gradient descent update step to learn from this batch. Following is the pseudo code of Double DQN from its paper [12]:

Algorithm 3 DDQN

a'

input: D: empty replay buffer; θ: initial network parameters, θ−copy of θ

input: N_r: replay buffer maximum size; N_b: training batch size; N⁻: target network replacement freq.

for episode e∈{1, 2, . . . , M} **do**

Initialize frame sequence x ← ()

for t ∈{0, 1, . . .} **do**

Set state s ← x, sample action a~π_b
Sample next frame x^t from environment ϵ given (s, a) and receive reward r, and append x^t to x
if |x|>N_f **then** delete oldest frame $x_{t_{min}}$ from x **end**
Set s' ←x, and add transition tuple (s, a, r, s') to *D*,
replacing the oldest tuple if |D|≥N_r
Sample a minibatch of N_b tuples (s, a, r, s')~Unif(*D*)
Construct target values, one for each of the N_b tuples:
Define a^{max} (s' ; θ)=arg $max_{a'}$ Q(s', a'; θ)

$$y_j = \begin{cases} r \ if \ s' \text{ is terminal} \\ r + \gamma Q(s', a^{max}(s';\ \theta);\ \theta^-), \text{otherwise} \end{cases}$$

Do a gradient descent step with loss $||y_j - Q(s, a;\ \theta)||^2$
Replace target parameters θ^-←θ every N^- steps
end
end

Various enhancements in the Deep Q-network were carried out in the paper [12]. Two of the most important ones are Double DQN and dueling architecture.

Double Deep Q-networks

One of the issues arising in Deep Q-network is overestimation of action values. This arises in the arg max term in below equation due to lack of gathered information during the early stages which makes it difficult to select a best action. To clear up this, we pick out the fine the use of one set of parameters θ_i, however examine it the use of a unique set of parameters θ^-. This prevents the algorithm from receiving incidental high rewards in the beginning stages. Following is the DDQN target:

y_i^{DDQN}=r+γQ (s', arg max Q (s',a'; θ_i); θ^-).
a'

The Dueling Network Architecture

The center concept of dueling networks is to apply streams, one which estimates the country fee characteristic and one which estimates the benefit for every movement A (s, a). Finally, the favored Q values are received with the aid of using combining the country and benefit values. The notation is given below:

Q(s, a; θ, α, β)=V(s; θ, β)+(A(s, a; θ, α)−$\frac{1}{|A|}$ $\sum_{a'}$ A(s, a' ; θ, α))

Reinforce Algorithm

REINFORCE is a Monte-Carlo variation of coverage gradients [13]. In value based methods, to derive an optimal policy we first have to evaluate the optimal action value function. We used table for small state spaces or neural network for large state spaces. But cannot we derive the policy directly without caring about the value function? Yes, we accomplish this via a class of algorithms known as policy-based methods. REINFORCE falls in the sub class of it known as policy gradient methods where optimal policy is estimated via gradient ascent. The goal is to find the values of the weights in the neural network that maximize the expected return. The algorithm is as follows:

First we initialize a random policy and using it we collect a list of state, action and reward. Then we compute total reward of trajectory and compute an estimate of the gradient of expected reward. Lastly we update the policy using gradient ascent with learning rate alpha. The process then repeats.

Here is the pseudo code for REINFORCE [14]:
Algorithm 2 REINFORCE
function REINFORCE
Initialize θ arbitrarily

for each episode $\{s_1, a_1, r_2, \ldots\ldots\ldots s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
for t=1 to T-1 **do**
$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta (s_t, a_t)\, v_t$
end for
end for
return θ
end function

So, the flow of the algorithm is:
Step 1: Perform a trajectory roll-out using the current policy.
Step 2: Store log probabilities (of policy) and reward values at each step.
Step 3: Calculate discounted cumulative future reward at each step.
Step 4: Compute policy gradient and update policy parameter.
Step 5: Repeat step 1 to step 4.

REINFORCE being the starting point of policy gradient methods, suffers from certain short comings. First of all, the update process is very inefficient. Secondly, the gradient estimate is very noisy as by chance the collected trajectory may not represent the policy. And lastly, there is no clear credit assignement. All these will be resolved in our next policy based algorithm DDPG.

Deep Deterministic Policy Gradient (DDPG)

There are a class of algorithms which are at an intersection of policy based methods like REINFORCE and value based methods such as DQN. Here critic evaluates the state value function using the TD estimate and calculates the advantage function which is further used to train the actor.

DDPG is a different kind of actor-critic method [15]. It could be considered as approximate DQN as the critic in DDPG is used to approximate the maximum from all the Q values of next state instead of learning it as a baseline. DQN agent does fairly well in discrete action space, but problems falling in continuous action space cannot be solved. This is one of the issues DDPG considers.

In DDPG, we use two deep neural networks. Here the actor deterministically searches for optimal policy. By deterministically, we mean to say the network outputs a single action. It is unlike stochastic policies where we learn the probabilistic distribution over the actions. The critic learns to evaluate the optimal action value function by using actor's best believed action. Three other interesting aspects of DDPG are introduction of noise, use of replay buffer and doing soft updates to the target network.

In DQN we had two copies of network weights. One regular and other target. Regular network is the most up to date network because it is the one we are training while we use the target network for prediction. In the original paper, target network was updated every 10000-time steps.

In DDPG, we have copies of community weights for every community. Actor has a regular and target and similarly critic also has a regular and target. But in DDPG, the goal networks are up to date the usage of a gentle updates strategy.

A soft update procedure slowly mixes the weights of regular network with the weights of target network. So every time step, we make the target network 99.99% similar to target network and 0.01% of regular network.

In theory, soft update strategy helps us in reaching fast convergence. Finally, noise is added into network so that agent can add more experience via exploration.

Pseudo-code of the DDPG Algorithm [16]:

Algorithm 1 DDPG Algorithm

Randomly initialize critic network Q(s, $a|\theta^Q$) and actor μ(s|θ^μ) with weights θ^Q and θ^μ.

Initialize target network Q' and μ′ with weights $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer R
for episode =1, M **do**
Initialize a random process *N for* action exploration
Receive initial observation state s_1
for t=1, T **do**
Select action a_t=μ($s_t|\theta^{\mu}$)+N_t according to the current policy and exploration noise
Execute action a_t and observe reward r_t and observe new state s_{t+1}
Store transition (s_t, a_t, r_t, s_{t+1}) from R
Sample a random mini batch of *N* transitions (s_i, a_i, r_i, s_{i+1}) from R
Set y_i=r_i+$\gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|(\theta^{Q'})$
Update critic by minimizing the loss: L=$\frac{1}{N}\sum_i \left(y_i - Q(s_i, a_i|\theta^{Q})\right)^2$
Update the actor policy using the sampled policy gradient:
$\nabla_{\theta^{\mu}} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^{Q})|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})_{s_i}$
Update the target networks:
$\theta^{Q'} \leftarrow \tau\theta^{Q}+(1-\tau)\,\theta^{Q'}$
$\theta^{\mu'} \leftarrow \tau\theta^{\mu}+(1-\tau)\,\theta^{\mu'}$
end for
end for

Simulation of Urban Mobility (SUMO) [17]

To run our reinforcement algorithms in simulator we chose SUMO. SUMO is an open supply site visitor's simulation package. It turned into evolved withinside the year 2001. It is a pretty advanced system in which we can build traffic network and assign routes within a network, develop traffic light algorithm and simulate and analyze traffic within this network, given origin-destination (OD) matrix. In SUMO, there is a cfg file and rou file coded in XML.

In this routing file we decide vehicle type, its properties as well as the route they are going to take throughout the network. Moreover within vehicle object we can decide acceleration speed, deceleration speed, max speed and sigma value which adds randomization to make situation realistic. SUMO uses, by default, Deijkstra's algorithm to route pass. So we can just give start and end edge.

For the underlined work, we have also used TraCI i.e. traffic control interface. It is an API that gives us access to run SUMO simulation.

The various TraCI operations we can do over SUMO are:

- Receive traffic light values.
- Receiving information regarding all lanes.
- Receive information about vehicle.
- Receive information about persons.
- Receive information about vehicle types.
- Receive information about routes.

After thorough experimentation on SUMO simulator, we were able to derive all the use cases which can mimic a real life intersection. The basic three cases of intersection are shown below:

When all the lanes have minimum traffic i.e. a scenario which replicates a night time scenario is shown in Figure 3a. We also have a situation where all the lanes are heavily crowded replicating intense traffic moment is shown in Figure 3b. At last when only one of the lanes is heavily crowded is shown in Figure 3c.

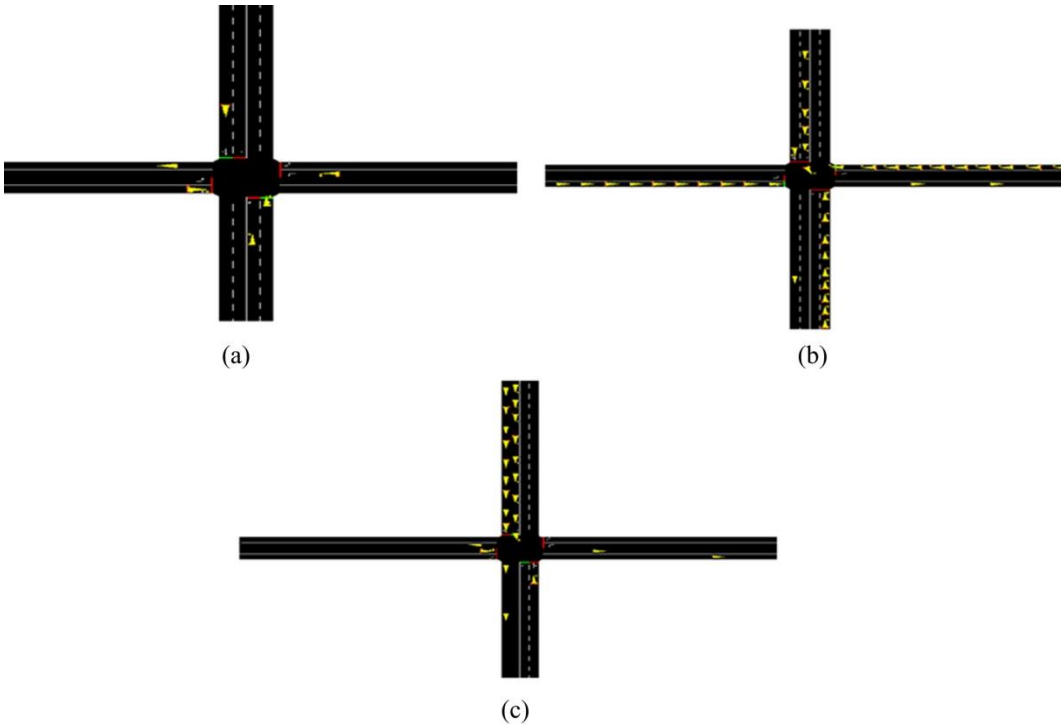

Figure 3. (a) Minimum traffic in all intersections; b) Maximum traffic in all intersection; c) Partially congested scenario.

EXPERIMENTAL RESULTS AND ANALYSIS

The present work also is focused on comparing the three state-of-the-art RL algorithms i.e. REINFORCE; double dueling DQN and single DDPG. We conducted extensive experiments in the SUMO environment. We have considered the four lane intersection and total number of steps used to train the agent was 2500×100 where 100 is total number of episodes. Total range of motors have been inside variety of 2550–2750. Pytorch (Python 3.6 API) was used to train all the models. Detailed comparison is made by using these three algorithms based on waiting time and expected cumulative reward. Then graphs were plotted based on the results we acquire.

REINFORCE Algorithm

Starting with the REINFORCE algorithm, we plotted the reward on the y-axis and episode on the x-axis. The basic criteria we considered for rewarding was waiting time. More the waiting time of car, the environment puts higher negative reward. If waiting time is less, the environment puts less negative reward. The following hyper parameter values gave us the best results-discount factor or gamma as 0.9, learning rate of agent as 0.001 and trained the agent over whole batch size which is number of steps per episode (in our case 2500). The overall network architecture where we input a state vector of 84 dimension into a fully connected 4-layered networks with 256, 32, 8, 4 nodes in the respective layer. The total training time was 40 min 23 sec. If we put close attention on reward-episode curve, agent initially explores the environment and thus takes a lot of random guesses which results in higher negative rewards. But as episodes go, by 21st episode it starts to understand the environment, in other words, starts gaining higher positive rewards than previous episode. Thus, curve becomes smoother and equable (Figure 4).

Then we plot waiting time on y-axis with episode number on x-axis. It is logical to say rewards and waiting time are inversely proportional i.e. when waiting time is less; agent gets less negative reward and vice-versa. Thus in the waiting time graph, initially, waiting time fluctuates a bit as agent explores the environment after which the waiting time starts to slowly minimize (Figure 5). By 97th episode, the total signal waiting time substantially decreased to 7.49 sec. After training through 100 episodes,

the agent ended up with waiting time of 8.8 sec. While in the testing environment, the waiting time for inference was 7.32 sec.

Figure 4. Score vs. episode plot of REINFORCE.

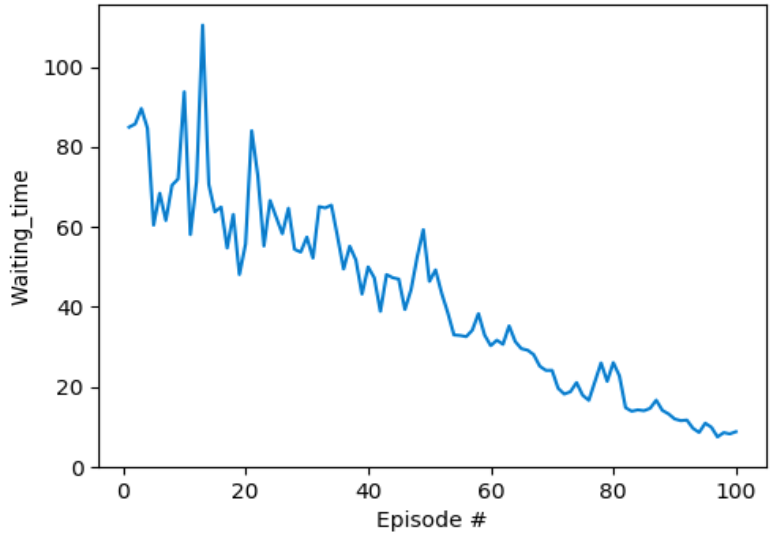

Figure 5. Waiting time vs. episode plot for REINFORCE.

Dueling Double Deep Q-Network (D3QN)

The hyper parameter values that gave the best results are as follows: discount factor or gamma as 0.99, interpolation parameter as 0.001, learning rate of agent as 0.0005, batch size as 64 and updated the network after every two steps. The network architecture consists of base net with input state vector of 84 and other two layers with 256 and 32 nodes. The value net also consists of two layers with 8 and 1 nodes. At last, there is an advantage net with 8 and 4 nodes, where 4 being the action dimension. It took 56 min 37 sec to train the network with 100 episodes.

After few initial episodes, agent started exploitation of the environment and took more accurate actions. After a certain episode, the reward reached a plateau (Figure 6).

From Figure 7 we can draw inference that total signal waiting time reduced to 7.76 sec after reaching 90th episode of training and after completion of 100 episodes it ended up with 8.66 sec. While in the testing environment the waiting time for inference was 6.75 sec.

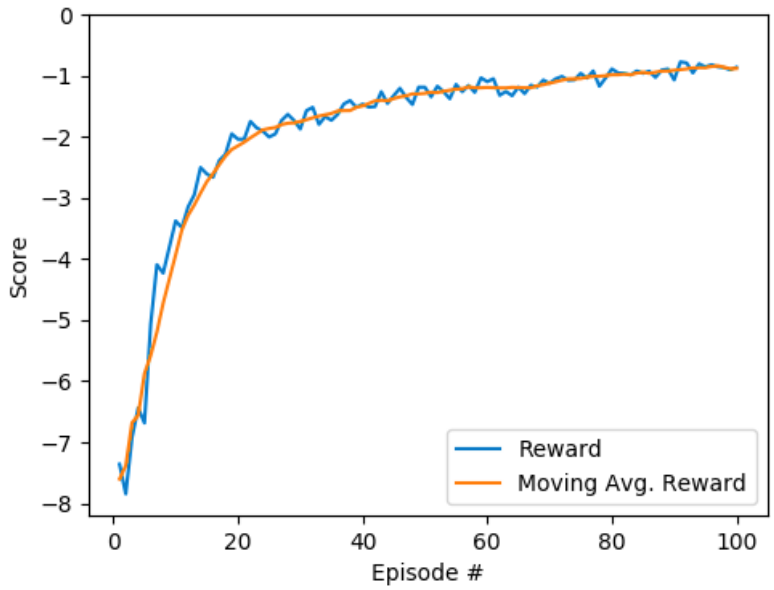

Figure 6. Cumulative reward vs. episodic steps in Dueling Double DQN.

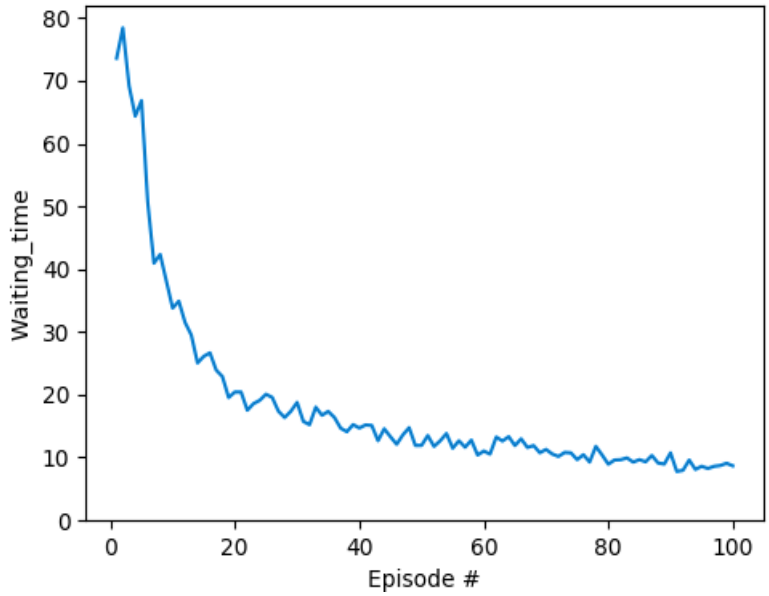

Figure 7. Waiting time vs. episodes in Dueling Double DQN.

Single Deep Deterministic Policy Gradient

As we know from initial discussion, DDPG is a combination of value based and policy based where it tries to use the value-based techniques to further reduce the variance of policy based methods which gives the extra stability. Feature vector of size 84 is fed into the DDPG agent. To achieve maximum reduction of waiting time, the following hyper parameter values were used: discount factor gamma was set as 0.99. The soft update of target parameters was set as 0.001. Learning rate of both actor and critic was set as 0.000005. Mini batch size was set as 256. The buffer size was set as 1,000,000. The network architecture of DDPG for both actor and critic consists of four fully connected layers. The first layer had 84 nodes which is equal to the state vector dimension. The center layers are 64, 16 nodes and variety of nodes withinside the output layer is 4. The total training time to run 100 episodes was 5 h 53 min. Batch Normalization was employed to reduce substantial noise. In the initial part of curve (1st–20th episode), the agent does not receive significant rewards due to high exploration (Figure 8).

Moreover, DDPG being a highly complex algorithm, the initial waiting times as large as the fuzziness in the network is too high. Exploration and exploitation is done in alternate episodes (1+1) which causes high noise initially. After 10 explorations, agent starts getting steadily incrementing random score and later it is able to quickly approach the optimal minima. The below graph shows that the total signal waiting time gets reduced to 22.78 sec after 97 episodes of training (Figure 9).

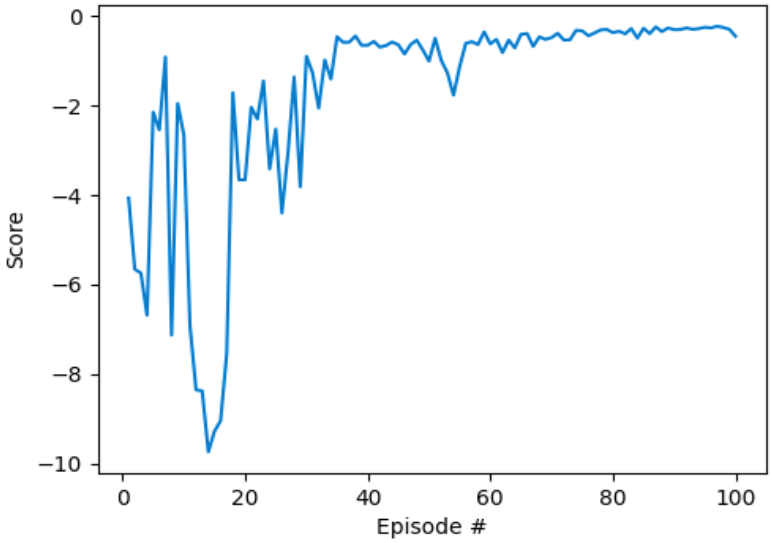

Figure 8. Score vs. Episode for DDPG.

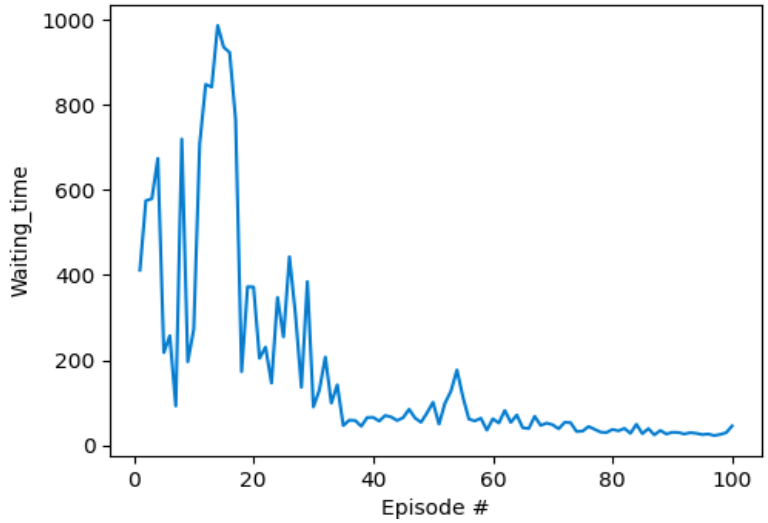

Figure 9. Waiting time vs. episode for DDPG.

The Rate of Decrease

The graphs of REINFORCE and dueling double DQN are compared in Figure 10. Here we can certainly say that D3QN agent receives high number of less-negative rewards than REINFORCE and waiting time of D3QN dropped quite smoothly as compared to REINFORCE though still both met at nearly same saturation point.

The waiting time comparison bar chart for the following three RL algorithms is shown below in Figure 11. As we can see clearly, D3QN is having the least waiting time of 6.75 sec. Moreover the smooth curve also further clarifies this statement. The REINFORCE having the smooth curve decreased the waiting time by 7.32 sec over the interval of 100 episodes. The DDPG is the algorithm with the highest waiting time i.e. 22.78 sec due to the sudden peaks of negative rewards.

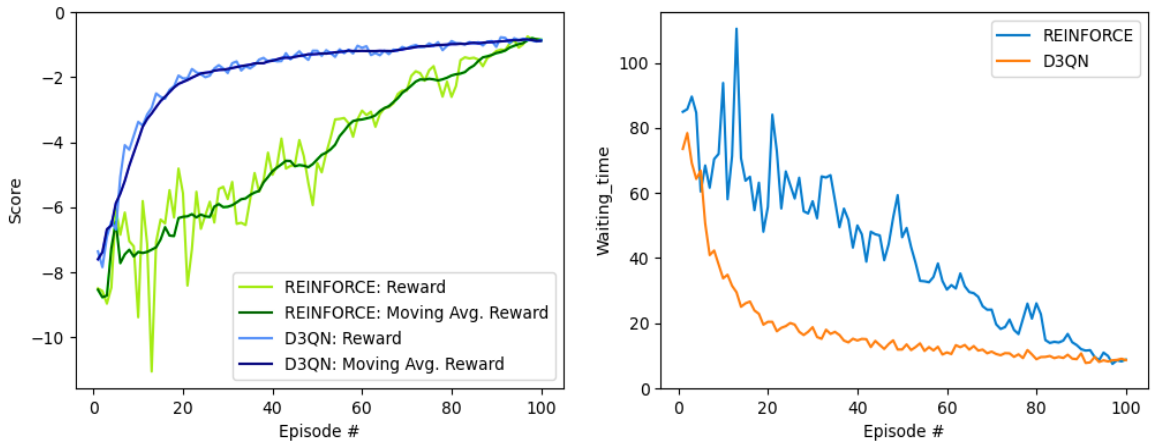

Figure 10. Simultaneous comparison of REINFORCE and dueling double DQN.

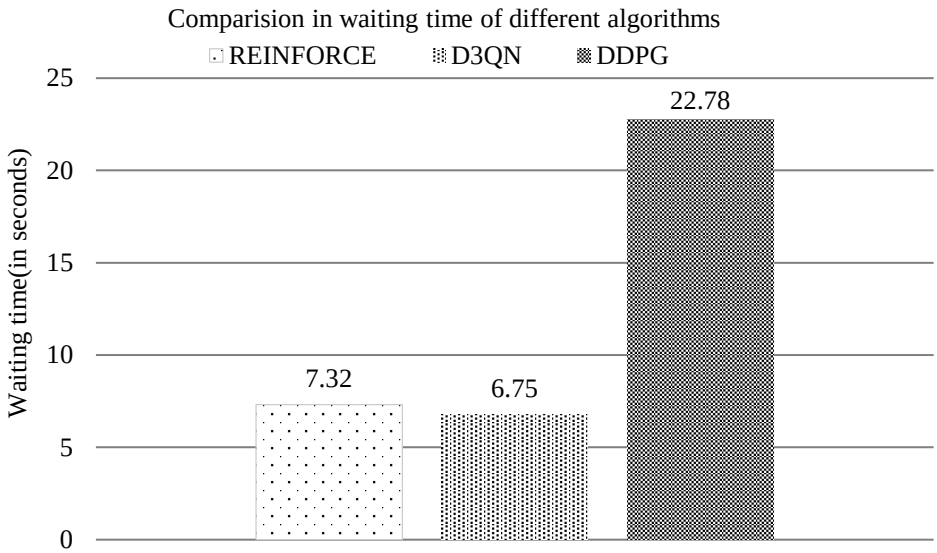

Figure 11. Comparison in waiting time of different algorithms.

CONCLUSIONS

The graphs clearly demonstrate that we have successfully constructed constantly learning agents. Also the trend in the graphs indicates decision making process is strictly stochastic and not merely random guesses. Convergence with D3QN was smooth and it is indeed a fast learner with only value based approach as compared to DDPG. Moreover, D3QN and REINFORCE has given almost same output after 100 episodes i.e. both converged to more or less a same saturation point. DDPG being a complex network architecture took 5 h 53 min for training over 100 episodes.

Scope for Future Work

A unique ensemble of IOT based devices and powerful AI algorithms will be created [18]. We also will try to predict congestion at a specific place using previous traffic data [19]. Moreover, signal wait duration to be reduced by 10–12% in a real time scenario which will also help in cutting down travel cost by lowering fuel wastage. Easy vehicle identification and tracking interface for public authorities with a huge database of categorized inputs [20]. One of the future plans to build a strongly connected ecosystem of smart devices to enhance the scalability of the entire system and use all benefits of social networking sites [21].

REFERENCES

1. Duale Ali Y, *et al.* Building Smart Traffic Control. United States Patent. LLC: Quartz Auto

Technologies; 2017 Apr.
2. Souparno. Congestion, low traffic speeds worsen pollution in Delhi. Centre for Science & Environment.
3. Tianshu Chu, *et al*. (2019 May). Multi-Agent Deep Reinforcement Learning for Large-scale Traffic Signal Control. [Online]. arXiv:1903.04527 [cs.LG].
4. Yilun Lin, *et al*. (2018 Apr). An Efficient Deep Reinforcement Learning Model for Urban Traffic Control. [Online]. arXiv:1808.01876 [cs.AI].
5. Abdul Aziz HM, *et al*. Learning based Traffic Signal Control Algorithms with Neighbourhood Information Sharing: An Application for Sustainable Mobility. J Intell Transp Syst. 2018; 22(1): 40–52.
6. John Schulman, *et al*. (2017 Aug). Proximal Policy Optimization Algorithms. [Online]. arXiv:1707.06347 [cs.LG]**.**
7. Minh, *et al*. (2016). Asynchronous Methods for Deep Reinforcement Learning. [Online]. Deep Mind. 2016.
8. Bram Bakker, *et al*. Traffic Light Control by Multiagent Reinforcement Learning Systems. Berlin, Heidelberg: Springer; 2010; 475–510.
9. Lowrie PR. SCATS: A traffic Responsive method of Controlling Urban Traffic. Darling Hurst, NSW Australia: Roads and Traffic Authority NSW; 1990.
10. Bo Yang, *et al*. Vehicle Detection and Classification for Low-Speed Congested Traffic with Anisotropic Magneto resistive Sensor. IEEE *Sens J*. Feb 2015; 15(2): 1132–1138.
11. Stephen Smith, *et al*. Surtrac-Intelligent Traffic Signal Control. TRB 2013 Annual meeting.
12. Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, Nando de Freitas. (2015). Dueling Network Architectures for Deep Reinforcement Learning. [Online]. arXiv:1511.06581 [cs.LG]
13. Tingwu Wang. Learning Reinforcement Learning by Learning REINFORCE. Machine Learning Group, University of Toronto. (http://www.cs.toronto.edu/~tingwuwang/REINFORCE.pdf)
14. Fei-Fei Li, Justin Johnson, Serena Yeung. Reinforcement Learning. Lecture 14. 2017. http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf
15. https://spinningup.openai.com/en/latest/algorithms/ddpg.html
16. Lillicrap, *et al*. (2015). Continuous Control with Deep Reinforcement Learning. [Online]. https://arxiv.org/abs/1509.02971
17. SUMO. https://sumo.dlr.de/docs/Tutorials.html
18. Merenda M, Porcaro C, Iero D. Edge Machine Learning for AI-Enabled IoT Devices: A Review. Sensors. 2020; 20(9): 2533.
19. Houli D, Zhiheng L, Yi Z. Multiobjective Reinforcement Learning for Traffic Signal Control Using Vehicular Ad Hoc Network. EURASIP J Adv Signal Process. 2010; Article ID 724035: 1–7.
20. Djalalov M, Nisar H, Salih Y, Malik AS. An algorithm for vehicle detection and tracking. 2010 International Conference on Intelligent and Advanced Systems, Manila. 2010; 1–5.
21. Essien A, Petrounias I, Sampaio P, *et al.* A deep-learning model for urban traffic flow prediction with traffic events mined from twitter. World Wide Web. 2020; 24(8): 1345–1368.

