The Intersection of Reinforcement Learning and Traffic Light Scheduling
Q learning with a target network and experience replay applied to traffic light controllers

Drew Kristensen
University of Puget Sound
dkristensen@pugetsound.edu

Abstract

The goal of this project is to create a policy learner to optimize the traffic flow in a non-virtual city environment. However, in order to accomplish this task, we will be using model based reinforcement learning approaches to tackle the end goal. By first training the architecture on an easily testable and repeatable environment, we can develop a model that will complete this task quicker and easier than one reliant on real world training data. One key feature in this project is keeping the inputs simple enough that they would be readily available in real life applications - that is to say the features used as inputs should be easy to observe.

Introduction

Traffic congestion is a problem that most people in urban environments have experienced at some point in their lives. Congestion not only has an impact on the humans involved in the traffic jam, but the idling vehicles emitting carbon cause environmental harm. It is estimated that in 2013, the annual time wasted in cities due to congested traffic was around 65 hours per driver, and the amount of carbon dioxide released while delayed was approximately 3.1 megatons per major city [CEBR2014]. The factors responsible for this traffic congestion can be attributed to a few sources, namely the increase in the number of vehicles on the road, inadequate infrastructure, and inefficient current intersection controls. To alleviate congestion, at least one of the above reasons must be addressed. Since construction of new infrastructure in urban environments is costly, and since cars will continue to be driven, the most realistic solution involves utilizing a smarter traffic controller.

The most basic traffic controllers use fixed time signal control, where the timings of the lights can be fitted to data in order to optimize throughput. The problem with fixed timing controls is that it cannot react to real-world changes in traffic throughput, and can lead to congestion and gridlock in urban environments. Currently, an effective solution to reducing vehicular congestion is using adaptive traffic signal control, where the timings on light signals are changed according to real world data that the traffic control agent (TCA) can collect. Current examples of these include SCOOT and TRANSYT [Robertson and Bretherton1991, Robertson1969] and prove to be much more effective than their static equivalent.

Reinforcement learning techniques have seen recent use in an attempt to develop better controllers than their fixed timing counterparts. In many of these, human crafted features are used, such as queue length and vehicle speed [Gregoire et al.2013, Arel et al.2010, Abdoos, Mozayani, and Bazzan2013], while others use machine crafted features by feeding in the entire state space consisting of vehicle positions and speeds [Gao et al.2017, Genders and Razavi2016, Wiering2000, Balaji, German, and Srinivasan2010]. In this paper, we propose using a deep Q network (DQN) with a target network [Mnih et al.2015, van Hasselt, Guez, and Silver2015] as our traffic controller over the state space in order to minimize the total squared waiting time for every car in the network. Not only do we implement a novel state space for this problem, with less knowledge required from the environment, but we also utilize an unexplored action space in order to give our traffic control agent the most freedom possible.

The paper will be composed as follows; section 2 will address previous research in the area and how it applies to this paper, section 3 will describe the proposed system, and section 4 discusses the significance of the results.

Related Work

Reinforcement learning applied to traffic control is not a novel idea, but there has been a wide variety in the approaches that have been used. Early approaches utilized small scale examples, but showed the RL can be applied to the TCA problem successfully [Wiering2000, Thorpe and Anderson1996, Brockfeld et al.2001]. Since then, increases in computational power, better reinforcement learning techniques, and the increase in popularity of neural networks have all driven novel research into this field. Previous works have explored a variety of state spaces, action spaces, reward functions, traffic simulators, network layout, and vehicle flow, which have provided a deep foundation for the field. Previous approaches have used many different state spaces, such as locations of cars in lanes, the number of queued vehicles, or the density of traffic in a lane [Genders and Razavi2016, Gao et al.2017, Wiering2000, LA and Bhatnagar2011] as well as a variety of action spaces and approaches

[Kwong et al.2011, Abdulhai, Pringle, and Karakoulas2003]. Examples in previous works include state spaces like the DTSE encoding developed by Genders, which segments streets into blocks and encodes positions of cars on the street and their speeds relative to the speed limit, or more general state spaces which encompass the full network and locations of cars in segments of roads, as used in the work done by Wiering.

On the topic of previous research, there are several key notions which we would like to examine.

First off, the state space utilized by many of the previous research projects involved total knowledge of the vehicles in the state. That is, they take their position on the edge as well as their speed. Examples include state spaces like the encoding developed by Genders, which segments streets into blocks and encodes positions of cars on the street and their speeds relative to the speed limit, or more general state spaces which encompass the full network and locations of cars in segments of roads, as used in the work done by Weiring. The most in depth exploration of constructing the state space in any real world scenario comes from work done in [Thorpe and Anderson1996]. While this encoding leads to a information dense state sapce useful for optimization, the real world examples of this would seem rather difficult to implement. It is from this, that we argue for a reduced information state space, where we keep data we feel is necesary, such as the quantities of vehicles in lanes and the number of queued vehicles, but throw out data we feel can be done without, such as exact speeds and positions. This will be discussed further in Section 2.2. This takes the middle ground of the early research, done by utilizing restricted state spaces, and recent works, which use too free a state space.

Second, a goal that some recent works have argued for is maximal freedom for the TCA in terms of action space [Wiering2000, Genders and Razavi2016]. In attempts to keep this, many approaches have been utilized that give the TCA more control than a standard fixed timing controller. Even in restricted action spaces where mandatory sequences must be followed, such a a yellow phase following a green phase, there has been good work done in allowing fluid choice for the TCA. However, all of these do serve as restrictions of the possible action space, and the optimal policy may not follow these rules. Thusly, we attempt to provide our agent with close to the maximal degree of autonomy, allowing it to change the signals as it pleases, with the only restriction being the breadth of traffic phases it can choose from.

Proposed System

Action Space

For any pair of incoming roads for a traffic light, we assume that there are 4 different signals that the traffic light can use; green straight, yellow straight, green left, yellow left. For any other pair of intersections, if any of the four mentioned are in use, then they will have red signals. From our previous assumption that there does not exist any intersection with more than 6 incoming roads, we can take our action space to be 3 sets of this pattern, so that $|A(S)| = 12$. From this, we can generalize our network to incorporate not only intersections with 4 incoming edges, but any intersection with 3 to 6 incoming roads.

In most cases, such as for any intersection with less than 6 incoming streets, there will be less than 12 actions to take. In these cases, we can still use the action space, but train it to never choose the actions it cannot take by subtracting a penalty value from the reward for a given action. For training, we can say that any output higher is clipped at whatever the maximum action value is. We can represent choosing an action by outputting an $\arg\max$ over the vector given by each light.

Another important aspect of the action space is that there is no duration included in the action decision. The implications of this is that at each time step, the network makes a decision based on the state space. This freedom in an action space has yet to be explored.

State Space

Previous state spaces rely on collecting data regarding precise locations of vehicles within their lanes and their speeds [Genders and Razavi2016]. While these provide a rich state space from which we can easily optimize, collecting these data points in the real world is a challenge. Instead, we propose a new state space which relies on the proportion of vehicles in each lane as well as the proportion of vehicles in each lane going under a speed threshold at each traffic light. Using this, our inputs stay between 0 and 1, and similarly scaled situations should have similar outcomes. Thusly, if there are ten cars on each lane in a network, it should theoretically take the same action as having one car in each lane. We categorize each lane into one of five types, dedicated left, left and straight, straight, right and straight, and dedicated right. Using this, we can capture any intersection with an arbitrary number of lanes, since for a road with two or more of any one type, we return their sum as their combined value.

How we define each of these directions within the problem is important, since obviously, for a 5 way intersection, there will intuitively be two right turns and two left turns, but no straight. Therefore, we define straight as within one eighths pi radians away from the direction the incoming street had if extended through the intersection. Using this, we hope to have a generalized layout for the state space such that it can represent any intersection. The pseudocode for this algorithm can be found in Algorithm 2 on page 6 and an example can be seen in figure 1.

Reward

In reinforcement learning, the reward function is what guides the agent during the algorithm, so it is important to choose a reward that will lead the agent in the direction that we want. We can easily replace a reward function, which ideally minimizes its value, with a cost function, which ideally minimizes its value, by negating the cost value. This allows us to maximize the negative of the cost, which is equivalent to minimizing. As in work done by Brys et. al [Brys, Pham, and Taylor2014], we propose using squared vehicle

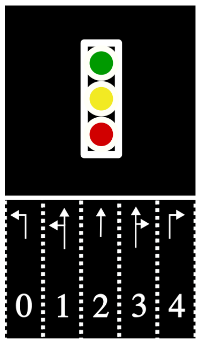

Figure 1: Example of lane classes within an intersection

delay as our cost. Our reasoning behind this decision is as follows; for a lone vehicle attempting to cross a busy street, the network with linear rewards will not be incentivised to allow them through at the cost of the vehicles on the busy street for a long time. Under the squared implementation, the network will be forced to let the vehicle through much faster, and thus preserve some individual happiness at a minimal cost to the vehicles needing to stop for a brief time.

RL Agent

For the agent, we utilize a convolutional neural network (CNN) [Lecun et al.1998, Redmon et al.2015] in order to make use of our state space in a logical and human-intuition backed manner. We argue that a CNN works better than a feed forward artificial neural network, as it is better suited for the task of pattern recognition. This can be seen in the success of CNNs in the field of image recognition and segmentation. Using this, we hope to better be able to pick out patterns in our traffic data, even with the more limited state space which we have proposed in this research. The architecture of our DQN can be seen in figure 2. We have two convolutional layers, which convolve over first the lane data for each incoming road in the junction, then secondly convolve over all of the outputs of the first layer, representing a convolution over the patterns extracted from each lane, giving us an output the operates over the entire state space in the junction. We also preform a convolution over our the inputs corresponding to our current traffic signal. Again, the intuition behind this decision is that this allows our network to pick out patterns with the light phases as well. The outputs from the convolutions are the combined in a flatten layer, which is then fed into two hidden layers with linear rectifier activation functions and one output layer.

Q Learning

We implement the Q learning algorithm with a target network and experience replay [Watkins and Dayan1992, Mnih et al.2015, Heess et al.2017]. Q learning is the process of training an approximator to mimic a Q function, where the Q function represents

$$Q(s_t, a_t, \pi) = E\left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} \middle| s_t, a_t, \pi \right\}$$

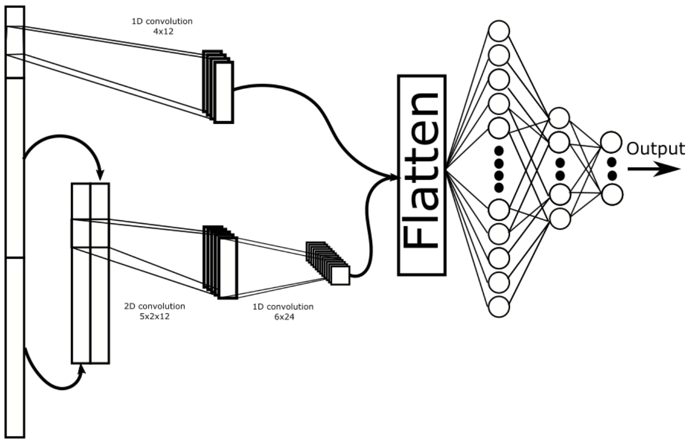

Figure 2: Architecture of the DQN

for a given state (s_t) and action (a_t), taken under some policy π. That is, the Q function outputs the expected value of all rewards from the current action and state onwards. The discount factor, γ, is a value between 0 and 1, and is used to weight the actions taken now and in the near future more heavily than the rewards from actions take in the distant future. By operating on the rewards, we can choose an action at time t with

$$a_t = \arg\max_a Q(s_t, a, \pi)$$

. The goal of Q learning is to find some policy π^* such that $\pi^* = \arg\max_\pi Q(s, a, \pi)$, for all $s \in S, a \in A$.

We train towards this policy by the iterative value update defined by setting

$$Q(s_t, a_t, \theta) \leftarrow r_t + \max_{a \in A} Q(s_{t+1}, a, \theta')$$

Where θ is the parameters for our primary DQN, and θ' holds the parameters for our target network. We use this implementation as it has been shown to better handle overestimates that single network Q learning is so susceptible to [van Hasselt, Guez, and Silver2015]. In order to update the target network, the approach that we use is Polyak averaging, as it is a simple and effective method for this task. This means that for some given λ, such that $0 < \lambda < 1$, we perform the following assignment after each training iteration:

$$\theta' = \lambda\theta' + (1 - \lambda)\theta.$$

The algorithm for training the network is given in Algorithm 1.

Algorithm 1: Q learning using a target network and experience replay

```
1  Initialize DQN with random weights $\theta$;
2  Initialize Target DQN with random weights $\theta'$;
3  Initialize $\epsilon, \gamma, \beta, N, M, \lambda$;
4  Initialize replay buffer, $\mathbf{B}$;
5  for episode $\leftarrow$ 1 to $N$ do
6      for $t \leftarrow$ 0 to 4800 do
7          $S_t \leftarrow$observed state;
8          With probability $\epsilon$
9              $A_t \leftarrow \arg\max_a Q(S_t, a, \theta)$;
10         Else
11             $A_t \leftarrow$ random$([1..12])$;
12         Simulate time step;
13         $S_{t+1} \leftarrow$ new observed state;
14         $r_t \leftarrow$ reward from $A_t$;
15         Insert $(S_t, A_t, r_t, S_{t+1})$ into $\mathbf{B}$;
16         minibatch $\leftarrow$ sample$(\mathbf{B}, 32)$;
17         $X \leftarrow$ inputs from minibatch;
18         $Y \leftarrow$ targets from minibatch;
19         Update $\theta$ through RMSProp on training data;
20         $\theta' \leftarrow \lambda\theta' + (1 - \lambda)\theta$;
21     end for
22 end for
```

In most cases, such as for any intersection with less than 6 incoming streets, there will be less than 12 possible actions for the traffic light to take. In these cases, we can still use our action space, but train the network to never choose the actions it cannot take. During training, we severely penalize any output higher maximum action value is, and retain the action currently in place. We can represent choosing an action by outputting an $\arg\max$ over the vector given by each light.

Results

Agent Settings

We implemented our DQN in TensorFlow 1.8 and we used the SUMO environment for our traffic simulator. We trained the agent for N = 4000 episodes and within each episode, we simulate one hour of traffic. Within our DQN, we used the default ADAM optimizer in TensorFlow, with the singular change being that we set $\alpha = 0.00025$. Our discount factor was 0.95 and our experience replay buffer has the capacity to store 500,000 steps. For our epsilon-greedy policy, we decrease epsilon linearly from 1 to 0.1 over 2.5 million time steps. We took our Polyak averaging constant to be $\lambda = 0.999$. The training was done a 3.5 GHz Apple Macbook Pro.

Results

Due to difficulties in implementing the Q learning algorithm, we did not have sufficient time to test our network in a capacity that we are happy with. Due to time constraints, instead of being able to train on the 4 way, 4 lane intersection that is so heavily researched, we were forced to train on a 4 way, 1 lane intersection in its stead. However, from less than one night of running, we achieved a throughput over 1000 simulated seconds of 115 vehicles, compared to the fixed timing 141 in the same time. However, during that same run, the average delay for arrived vehicles was 119 seconds for our network compared to the 19 seconds for the default fixed timing system. This represents an awful performance by our network, even relative to the most simple algorithm for traffic signal control. While disappointing, we hope to train the network more in the near future, in order to fully explore the extent to which our state space and architecture can be successful in operating as a TCA relative to other RL approaches.

Conclusion

Since the algorithm only operated successfully with such limited time, our network achieved significantly worse results than any other available algorithms for traffic controller agents. It is because of this that we feel like this research proved unsuccessful, at least with the parameters and settings that we used in our implementations. It might be worth changing these, as our state space is more actionable than previous state spaces, and thus may still have value for future works.

Future Work

In the future, we would like to explore the success of the novel state space and architecture, as mentioned in section 22. Furthermore, if this approach does seem to have merit, then exploring a variety of intersections, such as 6 way intersections, in order to better showcase the state space as well as provide a baseline for these types of intersections in the field. Lastly, testing the network on multiple intersections in the same network would be important, as this is more similar to real life scenarios and would allow us to compare our research to more algorithms that are present in previous works.

Acknowledgements

I would like to thank Dr. America Chambers, for her assistance and encouragement throughout the length of this project. I would also like to thank the authors of the various frameworks used to actualize my goals.

References

Abdoos, M.; Mozayani, N.; and Bazzan, A. L. C. 2013. Holonic multi-agent system for traffic signals control. *Eng. Appl. Artif. Intell.* 26(5-6):1575–1587.

Abdulhai, B.; Pringle, R.; and Karakoulas, G. 2003. Reinforcement learning for true adaptive traffic signal control. 129.

Arel, I.; Liu, C.; Urbanik, T.; and Kohls, A. G. 2010. Reinforcement learning-based multi-agent system for network traffic signal control. *IET Intelligent Transport Systems* 4(2):128–135.

Balaji, P. G.; German, X.; and Srinivasan, D. 2010. Urban traffic signal control using reinforcement learning agents. *IET Intelligent Transport Systems* 4(3):177–188.

Brockfeld, E.; Barlovic, R.; Schadschneider, A.; and Schreckenberg, M. 2001. Optimizing traffic lights in a cellular automaton for city traffic. 64:056132.

Brys, T.; Pham, T. T.; and Taylor, M. E. 2014. Distributed learning and multi-objectivity in traffic light control. *Connection Science* 26(1):65–83.

CEBR. 2014. The future economic and environmental costs of gridlock in 2030. Technical report, INRIX.

Gao, J.; Shen, Y.; Liu, J.; Ito, M.; and Shiratori, N. 2017. Adaptive traffic signal control: Deep reinforcement learning algorithm with experience replay and target network. *CoRR* abs/1705.02755.

Genders, W., and Razavi, S. 2016. Using a deep reinforcement learning agent for traffic signal control. *CoRR* abs/1611.01142.

Gregoire, J.; Frazzoli, E.; de La Fortelle, A.; and Wongpiromsarn, T. 2013. Capacity-aware back-pressure traffic signal control. *CoRR* abs/1309.6484.

Heess, N.; TB, D.; Sriram, S.; Lemmon, J.; Merel, J.; Wayne, G.; Tassa, Y.; Erez, T.; Wang, Z.; Eslami, S. M. A.; Riedmiller, M. A.; and Silver, D. 2017. Emergence of locomotion behaviours in rich environments. *CoRR* abs/1707.02286.

Kwong, Y.; Bolong, N.; Kiring, A.; Yang, S.; Tze, K.; and Teo, K. 2011. Q-learning based traffic optimization in management of signal timing plan. 12.

LA, P., and Bhatnagar, S. 2011. Reinforcement learning with function approximation for traffic signal control. *IEEE Transactions on Intelligent Transportation Systems* 12(2):412–421.

Lecun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. 86:2278 – 2324.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518:529 EP –.

Redmon, J.; Divvala, S. K.; Girshick, R. B.; and Farhadi, A. 2015. You only look once: Unified, real-time object detection. *CoRR* abs/1506.02640.

Robertson, D. I., and Bretherton, R. D. 1991. Optimizing networks of traffic signals in real time-the scoot method. *IEEE Transactions on Vehicular Technology* 40:11–15.

Robertson, D. I. 1969. 'transyt' method for area traffic control. *Traffic Engineering Control* 10:271–281.

Thorpe, T. L., and Anderson, C. W. 1996. Traffic light control using sarsa with three state representations. Technical report, IBM Corporation.

van Hasselt, H.; Guez, A.; and Silver, D. 2015. Deep reinforcement learning with double q-learning. *CoRR* abs/1509.06461.

Watkins, C., and Dayan, P. 1992. Technical note: Q-learning. 8:279–292.

Wiering, M. 2000. Multi-agent reinforcement learning for traffic light control. In *Proceedings of the Seventeenth International Conference on Machine Learning*, 1151–1158.

Appendix

Algorithm 2: Calculate the class of the lane

```
Function GetLaneType(Lane)
    Input  : The lane we want to compute the lane
             class for
    Output: The index of the lane with a value between
             0 and 4 inclusive
    left, straight, right $\leftarrow$ 0,0,0;
    for outbound lane connected to Lane do
        if outbound lane is straight ahead then
            straight $\leftarrow 1$;
        else if outbound lane is a right turn then
            right $\leftarrow 1$;
        else
            left $\leftarrow 1$;
        end if
    end for
    sum $\leftarrow 0\,(\text{left}) + 2\,(\text{straight}) + 4\,(\text{right})$;
    number $\leftarrow (\text{left} + \text{straight} + \text{right})$;
    return sum/number;
```