Tej Patel
Robert Cook

A Q-Learning Approach to Traffic Light Signal Control

Introduction

The rapid increase in automobiles on roadways has naturally lead to traffic congestions all over the world, forcing drivers to sit idly in their cars wasting time and needlessly consuming fuel. Ride sharing and infrastructural improvements can help mitigate this but one of the key components to handling traffic congestion is traffic light timing. Traffic light control policies are often not optimized, leading to cars waiting pointlessly for nonexistent traffic to pass on the crossing road. We feel that traffic light control policy can be greatly improved by implementing machine learning concepts. Our project focuses on implementing a learning algorithm that will allow traffic control devices to study traffic patterns/behaviors for a given intersection and optimize traffic flow by altering stoplight timing. We do this with a Q-Learning technique, similarly seen in previous works such as *Gao et. al.* and *Genders et. al*, where an intersection is knowledgeable of the presence of vehicles and their speed as they approach the intersection. From this information, the intersection is able to learn a set of state and action policies that allow traffic lights to make optimized decisions based on their current state. Our work seeks to alleviate traffic congestion on roads across the world by making intersections more aware of traffic presence and giving them the ability to take appropriate action to optimize traffic flow and minimize waiting time.

Problem Definition and Algorithm

The problem definition is a simple one: Minimize the total waiting time of all vehicles during the simulation. Total waiting time will be a measure of the total accumulated amount of time waited by vehicles at a red light. To approach this problem, we use Q-Learning, where an agent, based on the given state, selects an appropriate action for the intersection in order to maximize present and future rewards. The state-action pairs, also called Q values, are learned and saved to a table where there values are continuously updated until convergence, where an ideal policy is found. This algorithm is outlined in more detail in the following sections.

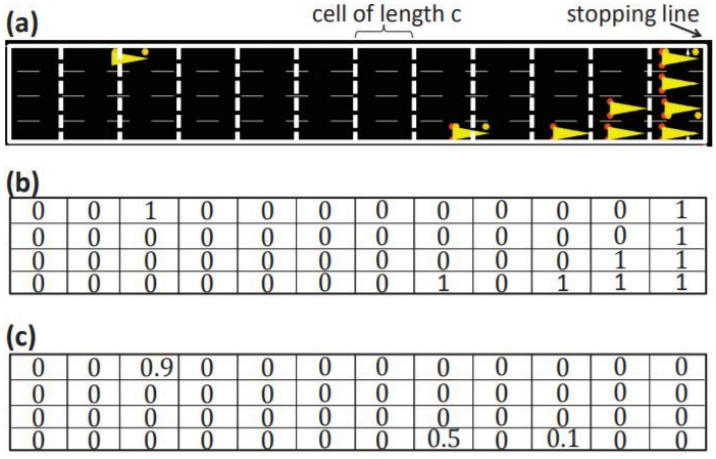
Intersection Model

Tej Patel
Robert Cook

We used the above four-way intersection, where each road has three lanes leading into the intersection: A right lane for right turns only, a middle lane for going straight only, and a left lane for turning left only. In addition, a single lane is used on each of the four roads for carrying traffic out of the intersection. This simplistic model of an intersection gives us the necessary conditions for testing our model on a four-way intersection.

We formulate traffic signal control problem as reinforcement learning problem where an agent interacts with the intersection at discrete time steps , t = 0,1,2…. , and the goal of agent is to reduce the vehicle staying time at the intersection in the long term. Specifically, such an agent first observes intersection state S_t (defined later) at the beginning of time step t, then selects and actuates traffic signals A_t. After vehicles move under actuated traffic signals, intersection state changes to a new state S_t+1. The agent also gets reward R_t (defined later) at the end of time step t as a consequence of its decision on selecting traffic signals. Such reward serves as a signal guiding the agent to achieve its goal. In time sequence, the agent interacts with the intersection as ... , S_t, A_t, R_t, S_{t+1}, A_{t+1}... . Next, we define intersection state S_t, agent action A_t and reward R_t, respectively.

Intersection State



For every edge we generate position and velocity matrices as shown above. Velocities are normalized by the speed limit. So our intersection state will be 2 matrices, one for position and one for velocity, described as follows.

$$P = \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \qquad V = \begin{bmatrix} V_0 \\ V_1 \\ V_2 \\ V_3 \end{bmatrix}$$

Tej Patel
Robert Cook

Agent Action

The agent sees the state and chooses one of two actions : 0 - turn on green lights for the horizontal roads and 1 - turn on green lights for vertical roads. Every time a state change is needed (i.e horizontal was green and agent decides to turn on green lights for vertical roads or vice versa), before executing the action the changes state, we need to undergo a transition to the next state. Transition would involve following :-

1) Change the lights for vehicles going straight to yellow.
2) Change the lights for vehicles going straight to red.
3) Change the lights for vehicles turning left to yellow.
4) Change the lights for vehicles turning left to red

Green light duration is 10 seconds and yellow light duration is 6 seconds. So at every time step, agent might decide to keep the same state or change the state.

Reward

We give rewards/punishment to agent at every time step. Agent tries to reduce the congestion by reducing the waiting time for vehicles at the intersection. We take 2 observations during each time step. First when the time step begins and second when the time step ends.

In each observation we record the waiting times of the vehicles, r1 and r2 represents the value for both the observations respectively. Finally the total reward R would be,

$$R = r_1 - r_2$$

By setting the above definition, we are punishing agent if its actions result in the increment of the waiting time and reward if waiting time is decreased.

Agent Goal

Recall that the goal of the agent is to reduce the waiting time of vehicles at the intersection in the long run. By observing the state, agent decides to make an action according to some action policy π. Since the agent aims to reduce vehicle staying time in the long run, the agent needs to find an action policy π^* that maximizes the following cumulative future reward, namely:

$$\begin{aligned} Q_\pi(s,a) &= \mathbb{E}\{R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \cdots | S_t = s, A_t = a, \pi\} \\ &= \mathbb{E}\{\sum_{k=0}^{\infty} \gamma^k R_{t+k} | S_t = s, A_t = a, \pi\} \end{aligned}$$

Tej Patel
Robert Cook

Deep Reinforcement Learning Algorithm

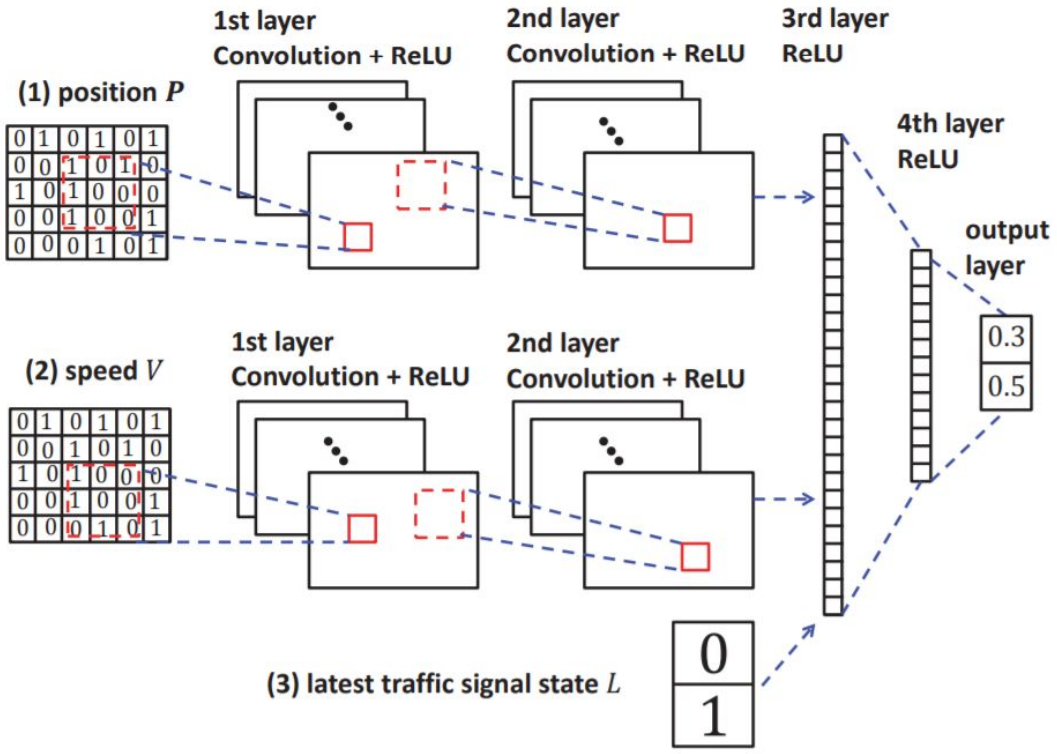
Since we have apparently infinite states to work with, storing the Q-value for each and every state is infeasible. Hence we need an approximation technique for estimating the Q-value for the given state.

For the optimal Q-values we have following recursive relationship, known as Bellman Optimality Equation,

$$Q^*(s,a) = \mathbb{E}\{R_t + \gamma \max_{a'} Q^*(S_{t+1}, a') | S_t = s, A_t = a\}$$
$$\text{for all } s \in \mathcal{S}, a \in \mathcal{A}$$

The intuition is pretty straightforward, the optimal cumulative reward agent receives is immediate reward received from executing an action and optimal future reward thereafter.

We will be approximating the Bellman Equation by training the Deep Neural Network having the following structure.



As we can see, the network input is the observed state (P, V, L). The first layer of convolution network has 16 filters of 4x4 with stride 2 and it applies relu as the activation. The second layer has 32 filters of size 2x2 with stride 1 and also applies relu. The third and fourth

layers are fully connected layers of size 128 and 64 respectively. Final layer is linear layer outputting Q values corresponding to every possible action that agent takes.

Now we will discuss about the training of the neural network. First we initialize neural network with random weights. At beginning of every time step, agent observes the current time step S_t and it forms input to the neural network and performs action A_t that has highest cumulative future reward. After performing the action, agent receives reward R_t and next state S_{t+1} by the environment. Agent stores this experience (S_t, A_t, R_t, S_{t+1}) in memory. As the memory is of limited size, oldest data is removed when memory space is full. DNN is trained by extracting training examples of type (S_t, A_t) from the memory. After collecting the training data, agent learns features θ by training the DNN network to minimize the following Mean Squared Error (MSE)

$$MSE(\theta) = \frac{1}{m}\sum_{t=1}^{m}\Big\{\big(R_t + \gamma \max_{a'} Q(S_{t+1}, a'; \theta')\big) - Q(S_t, A_t; \theta)\Big\}^2$$

Where m is the size of the input data set. Since m is large, the computational cost to calculate MSE(θ) is large, hence we use stochastic gradient descent algorithm RMSProp with minibatch of size of 32.

Experimental Evaluation

For algorithm evaluation we opted to simulate a four-way intersection using the traffic simulator SUMO. SUMO, or Simulation of Urban MObility is an open-source road simulation package that simulates traffic behavior, allows for road network construction, traffic light policy implementation, and traffic data collection. We also implemented the SUMO TraCI (Traffic Control Interface) extension which allows for dynamic control of the traffic lights at runtime. Running as a client, and calling the SUMO simulation tools as a server, TraCI not only allows for modification of traffic light timings while in simulation, but it also allows for information access of individual objects, such as vehicles. This makes it possible to extract the vehicle position and velocity matrices needed for Q-learning.

The traffic arrival process is probabilistic and generates East-West traffic with probability little bit higher than the North-South traffic. So the ideal agent should turn on green lights for E-W roads more than N-S roads. As mentioned earlier, duration for green light is 10 seconds and yellow light is 6 seconds.

During each “Episode”, or simulation run, around 1000 to 1200 vehicles were simulated over a given time interval of around 1.5 hours, approximately representing a typical traffic load during “rush hour” traffic. The learning rate of agent is 0.0002 and size of replay memory is 200.

As the traffic is skewed i.e the E-W traffic is more than N-S, static lights do not perform well, as running our simulations using static lights resulted in total waiting time of around 330k seconds. Using our DNN Agent, the best result we got was around 90k. It is about 72% improvement. The average waiting times are about 160-200k which is still significant improvement from our traditional methods.

Future Work

Unfortunately time constraints have prevented us from being able to analyze our approach when multiple intersections are present. It would be interesting to see if the same state-action pairs (Q values) would be learned or if the presence of multiple nodes would cause these to change. We generated simulations to test a four-intersection model however we have not had time to analyze this and learning Q values for multiple intersections would greatly increase simulation time which in itself already takes hours to run. Permitted more time we could expand this simulation scope and possibly consider implementing other state values apart from just the vehicle position and velocity matrices, for example, allowing an intersection to see the states of its neighbors. Such increases in state space could prove beneficial to improving traffic flow, but can also greatly increase learning time.

Conclusion

For our project we successfully implemented the Q-Learning algorithm as demonstrated in previous works by *Gao et. al.* and *Genders et. al.*, showing increased efficiency over the default static-timing paradigm, and showing improvement in the algorithm over time as it learns the state-action pairs. Over the course of 1000 "episodes" we showed an average improvement over the static-policy waiting time of around 30%, proving Q-Learning is an efficient alternative for traffic light control. As a real-world application there are of course complications that need to be addressed, such as the absence of position and velocity detection devices that make generating the state matrices possible. Barring these restrictions and assuming intersections with appropriate sensing technology to make vehicle detection possible, Q-Learning has proven itself as a viable alternative to traditional traffic control policies making it possible to reduce traffic congestion on roads around the world.

Bibliography

Bakker, Bram, et al. "Traffic Light Control by Multiagent Reinforcement Learning Systems." *Interactive Collaborative Information Systems Studies in Computational Intelligence*, 2010, pp. 475–510., doi:10.1007/978-3-642-11688-9_18.

Gao, Juntao & Shen, Yulong & Liu, Jia & Ito, Minoru & Shiratori, Norio. (2017). Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network.

Genders, Wade and Saiedeh Razavi. “Using a Deep Reinforcement Learning Agent for Traffic Signal Control.” *CoRR* abs/1611.01142 (2016): n. Pag.

Gregoire, Pierre-Luc, et al. “Urban Traffic Control Based on Learning Agents.” *2007 IEEE Intelligent Transportation Systems Conference*, 2007, doi:10.1109/itsc.2007.4357719.

Jadhao, Namrata S., and Parag A. Kulkarni. “Reinforcement Learning Based for Traffic Signal Monitoring and Management.” *International Journal of Engineering Research & Technology*, vol. 1, no. 4, June 2012.

Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. *Recent Development and Applications of SUMO - Simulation of Urban MObility*. International Journal On Advances in Systems and Measurements, 5 (3&4):128-138, December 2012