Combining Deep Q-Networks and Double Q-Learning to Minimize Car Delay at Traffic Lights

Timmy A. Hussain*
Department of Aerospace and Astronautical Engineering
Stanford University
timmyh@stanford.edu

Abstract

This project investigates using a Reinforcement Learning (RL) based approach as a substitute for a periodic traffic light system functioning at an intersection. The Double Q-Learning algorithm is implemented using two neural networks as Q-value function approximators. The objective is to minimize the average car delay at the intersection by developing a robust learning policy.

1 Introduction

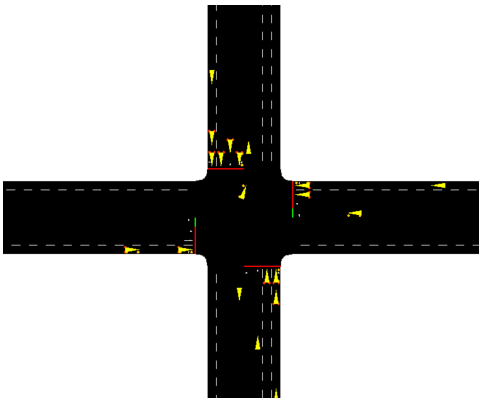
Traffic light systems which function on a purely periodic basis or have a schedule determined a priori are unable to adapt to the conditions that evolve in the environment they operate in. This project will develop a traffic light system which minimizes the average delay per vehicle by developing a policy capable of adapting to the conditions of the environment. I will be using a Double Q-learning approach to develop a policy and training two neural networks to store the associated Q-values. The input to the neural networks will be a state representation of the traffic intersection and the outputs will be the Q values of actions given the input state. The neural networks will be a series of fully-connected layers.

2 Related work

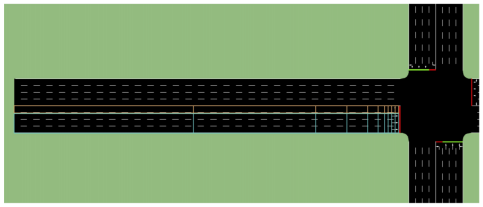
A lot of prior work exists in this field. The problem with unintelligent traffic light systems has been the topic of several research groups in the past, although the approach varies across the board. One approache tackles the problem with a multi-agent consideration which can be expanded to include several traffic light systems acting as agents in the network and able to leverage data from neighbouring nodes in decision processes[1][2]. This is essentially a reinforcement learning problem. The problem is well suited to be approached in this way as there can be a finite action space and the task of minimizing delay or some similar metric lends itself well to the reinforcement learning framework. Work in the traffic agent RL space typically takes the latter approach by incorporating the metric into a negative reward function such as proportional to the amount of delay a car experiences in two consecutive states. Some work has also been done around non-deterministic duration signals[3] which expands the action space somewhat; however, I will be drawing on the work done by Andrea Vidali[4] for much of the foundation for this project.

*Masters of Science, Aerospace/Astronautical Engineering

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)



(a) Road populated with vehicles



(b) Vidali discretization of intersection

Figure 1: Intersection model

3 Dataset and Features

3.1 Environment

As I plan to use reinforcement learning for this project, having an environment which facilitates the learning process is essential to training the model to develop the right policies. I will be using the Simulation of Urban MObility (SUMO) developed and maintained by the Institute of Transportation Systems at the German Aerospace Center[5]. I am taking advantage of a codebase provided by Vidali[4] which includes code to set up and interact with the simulation environment for the purposes of a reinforcement learning project. The environment pictured in Figure 1 consists of 4 roads which meet at an intersection.

Vehicles travelling on each of these roads follow predetermined routes with variations governed by a Weibull distribution. The action space initially appears to be a product of the three possible light colours and eight unique traffic lights. However, this can be reduced significantly. As can be seen in Figure 1, traffic light signals are paired which reduces the unique traffic lights from eight to four. Furthermore, each traffic light can only display one light colour at a time (red, yellow or green) and yellow lights are only displayed when transitioning from red to green or vice versa. Therefore, the optimal policy simply becomes what sequence of green (or red) light signals to give to optimize the chosen metric. As for duration, the duration of the opposing red light signal is necessarily determined by the duration of the green light signals and the duration of the yellow light signals is fixed. The SUMO environment facilitates all of these parameters to be configured and more including the number of vehicles spawned and the length of the simulation.

3.2 State Representation

I am currently preserving the state representation used by Vidali. As seen in Figure 1b, each road is discretized into a grid of 10 blocks of variable sizes with increased granularity closer to the traffic lights and decreased further away. The state is then represented by the number of cars in each block (essentially an occupancy grid of varying block sizes). As we have eight roads, we end up with an array $s_t \in \mathbf{R}^{80\times 1}$. This allows us to know with increased precision the number of cars closest to the traffic light which is extremely useful for developing a policy capable of dealing with situations where there are fewer cars on the road (a motivating use case for a smart traffic light system).

4 Method

The proposed algorithm here will be the Double Q-Learning algorithm as described by Hasselt[6] in his 2010 paper. It is an extension of the Q-Learning, algorithm the broad strokes of which are described below.

4.1 Q-Learning

Consider an agent which exists within an environment space and learns a policy over time by choosing actions within an action space. Actions have rewards associated with them. The agent learns an optimal policy by choosing sequences of actions which maximize the final reward over the duration of the training period. The Q-learning equation is as follows:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q^*(s_{t+1}, a_{t+1}) \tag{1}$$

where $r(s_t, a_t)$ is the reward function associated with making action a in state s at time t.

Q-Learning is governed by Bellman's Optimality principle: the optimal solution to a problem is reached by choosing the optimal solutions to sub-problems which compose the original problem. For the Q-Learning equation above, Q^* represents the optimal reward obtainable given some state s_t. It can be more formally defined as

$$Q^*(s_t, a_t) = \max_{a \in \mathcal{A}} \{Q(s_t, a_t) | s \in \mathcal{S}\} \tag{2}$$

where $\mathcal{A}$ and $\mathcal{S}$ represent the Action and State space respectively.

4.2 The Case for Double Q-Learning

Q-Learning is vulnerable to some issues which may either stop convergence from being guaranteed or ultimately lead to convergence of wrong Q-values (over- or under-estimations). As can be seen in equations 1 and 2, there is a dependence of $Q(s_t, a_t)$ on itself which leads to a high bias when trying to train a Q-network leading to overestimation of the Q values. Hasselt develops a method which avoids this by simulataneously training two networks and sampling from one while training the other:

$$Q_A(s_t, a_t) = r(s_t, a_t) + \gamma Q_B^*(s_{t+1}, a_{t+1}) \tag{3}$$

The reward function I am using takes the following form

$$r(s_t, a_t) = w_t - w_{t-1}$$

where w_t is the total wait time (wait time summed across all vehicles) accrued at time t. The total wait time accrued will increase by varying amounts depending on what action is chosen. This implicit relationship allows the agent to learn an optimal policy.

4.3 The Case for Deep Q-Networks

In lower-dimensional problems, it is tractable to reasonably store and access Q-values for each state and action pair in reasonable time. However, for most problems it is not. This is where the use of a Neural Network as an approximator to the Q-value function is useful. The neural network takes the form

$$f : \mathbf{R^n} \rightarrow \mathbf{R^m}$$

where n is the size of the state space and m is the size of the action space. It takes as inputs x which represent the state and outputs the Q-values associated with each possible action in that state. The neural network is trained to output Q-values close to the target Q-values for each action given the input state. Therefore a mean-squared loss is appropriate in this situation. I implemented the model using the Keras[7] framework to develop my neural network DQN which has 5 fully-conencted layers with ReLU activations besides the input and output linear layer.

$$\mathcal{L}_A = \frac{1}{k} \sum_{i=1}^{k} \left(\hat{y}^{(i)} - Q_A^{(i)}(s_t, a_t) \right)^2 \tag{4}$$

where $Q_A(s_t, a_t)$ is as defined in equation 3.

The original Hasselt paper did not deal with implementing the Double Q-Learning algorithm with a Deep Q-Network approach and I had to develop a few workarounds and approximations to facilitate

batch training the neural networks I used. My approach ultimately required I loop through the sampled batch twice ultimately limiting the benefits I could get from vectorization; this also factored into my decision to use a modest batch size. The Hasselt paper also made use of an adaptive learning rate

$$\alpha_t = \frac{1}{n_t^A(s_t, a_t)} \tag{5}$$

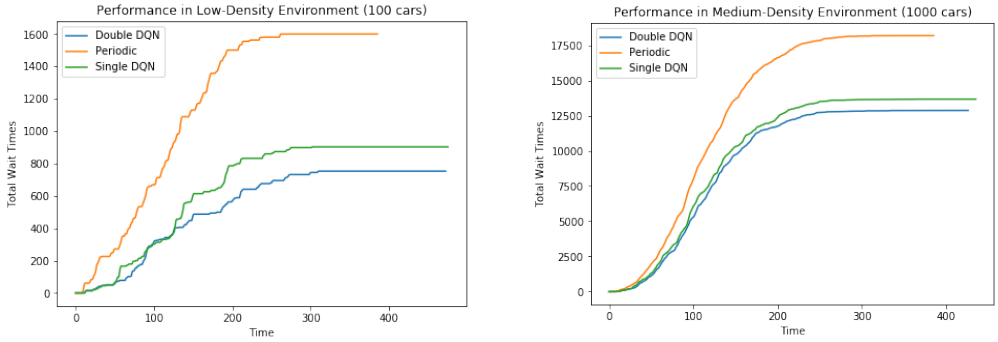
where $n^A(s_t, a_t)$ is the number of updates made on action a_t for model A. However, for a batched training process where $batch\_size$ updates are being made concurrently, I approximated α as

$$\alpha_t = 1 \div \frac{n_t^A}{num\_actions} = \frac{num\_actions}{n_t^A}$$

where n_t^A is the total number of updates made so far on model A. This makes n roughly the average update per action.

5 Results and Experiments

I ran a simulation of 5400 steps (5400 seconds) 100 times and trained the model for 800 epochs at the end of each simulation episode. Each epoch of training was done on a mini-batch of 100 randomly sampled experiences from the memory store. This was a reasonable batch size. After this training process, the model was able to reduce total wait times by 53% for the low-density case compared to the simple periodic model as shown in Figure 2. Figure 2 also shows the performance of a single Deep Q-Network trained with the same architecture. The Double DQN outperforms the single DQN in the motivating low- and medium-density cases, the high-density case is presented in the Appendices.



(a) Low density environment with 100 cars

(b) Medium density environment with 1000 cars

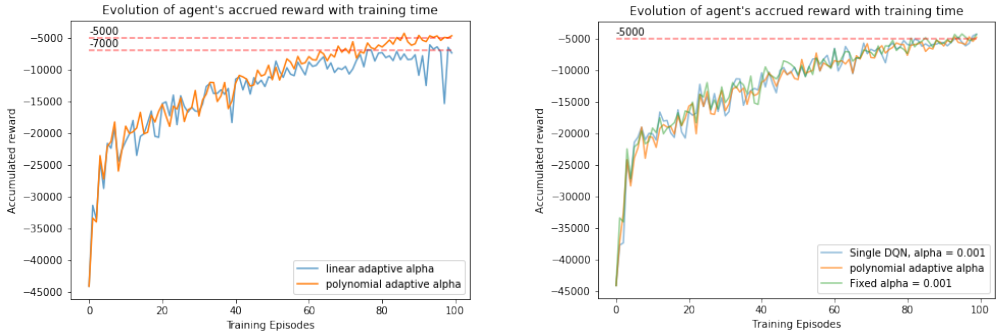
Figure 2: Agent performance in low and medium density environments

5.1 Learning Rate, α

To facilitate my training, I plotted the total negative reward accrued by the agent during a simulation episode. The expected result is that the negative reward will decrease over the course of training and converge to some maximum as the agent learns action sequences that maximize reward. This is illustrated in Figure 3 for the final model. Early in the course of my training, however, I ran into some issues with exploding gradients with the last few training episodes. These are shown in the Appendices where the reward begins to grow increasingly more negative right at the end. I determined this was likely a learning rate issue and this influenced my choice to decrease the learning rate from my initial 0.01 to 0.001. Under the 0.001 learning rate choice, the model converges and this is ultimately the learning rate the single DQN agent is trained on and the results associated with the Double DQN agent below are for an agent trained with a fixed learning rate unless otherwise specified.

However, as mentioned earlier, the Hasselt paper uses an adaptive learning rate. Initially, I didn't bother implementing this as I wanted to focus on getting a working implementation first before tuning. Once I determined that the training was highly sensitive to my choice of learning rate, however, I

decided it would be beneficial to implement the adaptive learning rate as described by Hasselt. Hasselt describes two learning rate policies, the first of which is described in equation 5 and the second of which is $\alpha_t^{0.8}$ where α_t is as described in equation 5. As suggested by Hasselt, the polynomial learning rate leads to better performance as shown below.



(a) Hasselt Comparison

(b) Fixed and polynomial comparison

Figure 3: Reward Convergence with varying alpha

Between Hasselt's two adaptive learning rate models and a fixed learning rate of 0.001, however, there is negligible difference in rate of training and reward convergence as shown in Figure 3. The single DQN also converges at roughly the same rate as the polynomial and fixed 0.001 Double DQN model. For a comparison of the performances of the Single DQN, Double DQN (fixed alpha), Double DQN (polynomial alpha) in low-, medium- and high-density environments however,

5.2 Gamma, γ

Another parameter I tuned is the parameter γ as seen in equations 1 and 3. This parameter determines how much weight we put on maximizing future rewards versus immediate reward. A γ of 0 would essentially lead to a myopic policy. Vidali in her paper proposes $\gamma = 0.75$ and I compared this to a higher-valued $\gamma = 0.95$. There is minimal issue with converging, although the model does not converge to a more positive reward at the end of training than the $\gamma = 0.75$ model as shown below. Therefore I stuck with $\gamma = 0.75$. These results are shown in the appendices.

5.3 Exploration/Exploitation

As is typical, I used an ϵ-greedy policy to deal with the exploration-exploitation question per the following formula

$$\epsilon = 1 - \frac{episode}{total\_episodes}$$

and with probability ϵ, I chose a random action from the action space. This biases exploration to the early training episodes and exploitation of your model to the later training episodes.

6 Conclusion/Future Work

In conclusion, the biggest worry over the course of this project was convergence. The reward function was simple and logical but the training was extremely sensitive to the choice of learning rate. Ultimately, the best performance arrived due to a constant learning rate of $\alpha = 0.001$. The model was trained for medium traffic density and performs well when tested in low and medium density environments. The performance between the single DQN and the Double DQN are comparable although the single DQN does better in the high-density environment.

For low traffic densities, a different reward model might be necessary: due to the sparse distribution of cars, there would not be so clear a mapping from action to reward as, for example, changing the lights from green to yellow to red while there are no cars present will have no effect on total wait times but may not be the best sequence (it might be better to simply hold at red or green). Such

a reward function would have to incorporate the cost associated with transitioning between colors which the current reward model does not.

In the development of a model robust to varying traffic densities, it might make sense to develop a model where γ is not fixed but is a learned parameter of the model. As described above, γ is a measure of how much we care about future reward. For low traffic densities, a myopic policy will likely work best as, between events, there is enough time for the agent to adjust. For high density traffic, γ should be higher because there are already several cars in the queue which will be affected by the action chosen at any time step. Therefore, instead of training the entire model with a fixed γ, it might make sense to leave that to be determined by the model in order to have a model robust to different densities. Of course, the training environment would need to be modified to spawn varying numbers of cars across training episodes.

7 Appendices

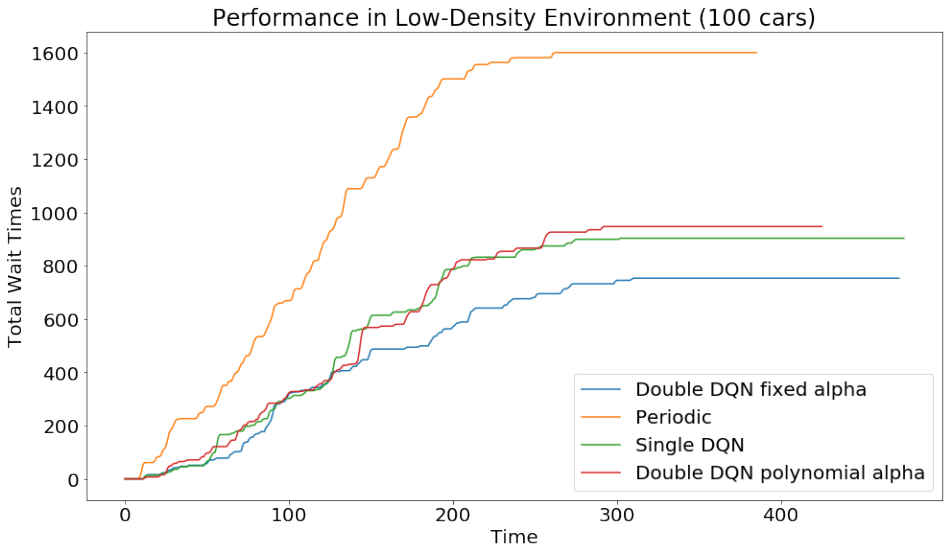


Figure 4: Low density performance

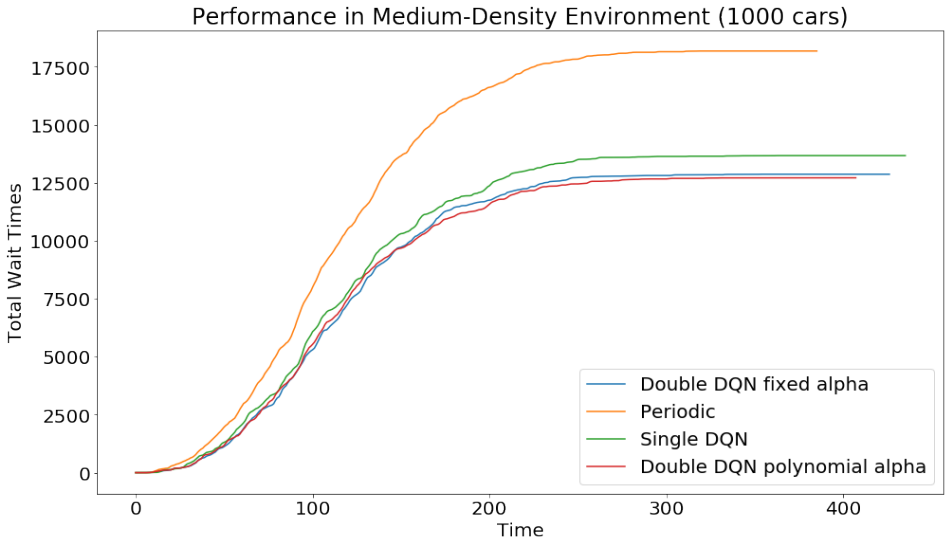


Figure 5: Medium density performance

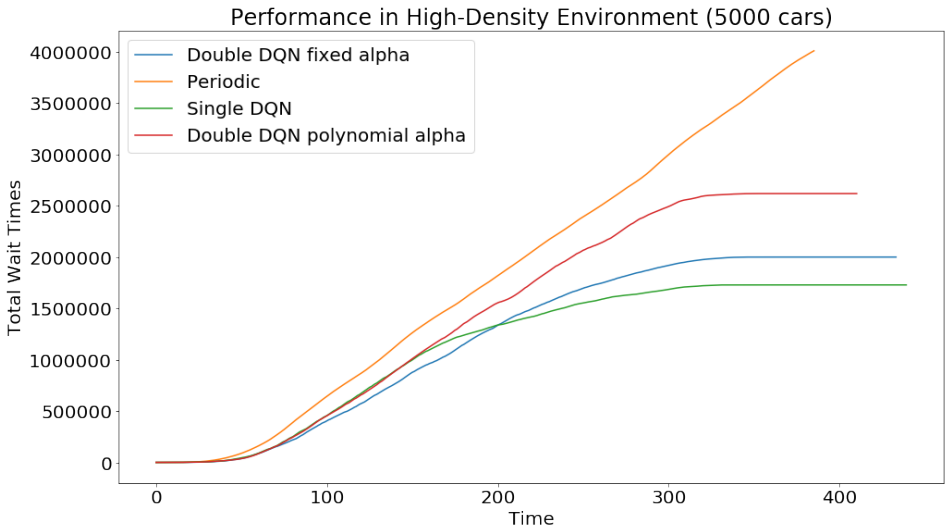


Figure 6: High density performance

Figure 7: Performance with varying gamma

References

[1] Itamar Arel et al. “Reinforcement learning-based multi-agent system for network traffic signal control”. In: *IET Intelligent Transport Systems* 4.2 (2010), pp. 128–135.

[2] Y. K. Chin et al. “Q-Learning Traffic Signal Optimization within Multiple Intersections Traffic Network”. In: *2012 Sixth UKSim/AMSS European Symposium on Computer Modeling and Simulation*. 2012, pp. 343–348. DOI: 10.1109/EMS.2012.75.

[3] X. Liang et al. “A Deep Reinforcement Learning Network for Traffic Light Cycle Control”. In: *IEEE Transactions on Vehicular Technology* 68.2 (2019), pp. 1243–1253. DOI: 10.1109/TVT.2018.2890726.

[4] Andrea Vidali. “Simulation of a traffic light scenario controlled by a Deep Reinforcement Learning agent”. MA thesis. University of Milano-Bicocca, May 2018.

[5] Daniel Krajzewicz et al. “Simulation of modern traffic lights control systems using the open source traffic simulation SUMO”. In: *Proceedings of the 3rd Industrial Simulation Conference 2005*. EUROSIS-ETI. 2005, pp. 299–302.

[6] Hado Hasselt. “Double Q-learning”. In: *Advances in neural information processing systems* 23 (2010), pp. 2613–2621.

[7] Francois Chollet et al. *Keras*. 2015. URL: `https://github.com/fchollet/keras`.