A Deep Reinforcement Learning Approach to Traffic Management

Osvaldo Castellanos
College of Engineering and Computer Science
University of Texas Rio Grande Valley
Edinburg, Texas, USA
osvaldo.castellanos01@utrgv.edu

***Abstract*—In this paper, we aim to investigate the applicability of deep reinforcement learning techniques to solve traffic congestion in road intersections. This ranges from reviewing foundational reinforcement learning concepts, to the formulation of the intersection problem, and finally to extended investigations in the application of multi-agent deep reinforcement learning techniques.**

Current research in traffic signal control that also applies multi-agent reinforcement learning train agents in specific, or even single, environments. Our goal is to train a system that can be practical in multiple settings, under multiple conditions. Overall, in this project, our goal is to apply the system to simulations of large-scale areas.

***Index Terms*—Deep Reinforcement Learning, Traffic Control, deep learning, machine learning**

I. Introduction

Inner-city traffic congestion causes more than just personal frustration while driving. It's a serious contributor to higher levels of air pollution, fuel consumption, and noise pollution [5]. In the United States, traffic infrastructure designed to alleviate congestion range in different ways. One solution is to employ fixed-time traffic signal controls. This type of solution is commonly used in cities and easy to set up and maintain. However, because the timing-signal is predetermined based on historic traffic data, it is not well an efficient solution when traffic congestion deviates from the data that was used to calibrate it. Fixed-time traffic signal controls are not dynamic to the different peaks and lows of congestion throughout a day, seasons, or even special events such as concerts.

Another solution is adaptive traffic signal control. This solution may be more expensive than fixed-time due to the extra hardware installation and maintenance that is required to make traffic signal controls responsive to real-time traffic conditions. However, research has shown that it is an effective way to reduce congestion. There are different approaches to making traffic signals adaptive, and in this paper we will focus on reinforcement learning techniques.

Research in applying reinforcement learning techniques to traffic congestion has expanded in the last few years [3]–[5]. And yet, certain empirical constraints have limited the founded results to actually help real-life congestion. Studies limit their focus to simplified parameters in their environments, such as limited lanes, driving behavior, and traffic congestion. Aside from addressing the listed limitations in current studies, research in applying multi-agent reinforcement learning is the next step.

Thanks to the Dwight D. Eisenhower Fellowship for support.

II. Background

The field of deep reinforcement learning is making large strides of progress in being applied to various types of applications with varying levels of success. As of today, it is hoped that it will achieve viable methods for creating autonomous systems [1]. Although reinforcement learning techniques have been researched since the late 1980's [7], they have not been as prominent in machine learning research as supervised, and to an extent unsupervised, learning has. The rapid progress of the last decade in neural networks and deep learning helped introduce a spotlight on reinforcement learning after the group at DeepMind used deep reinforcement learning to teach an agent to play Atari games [2]. Following the research laid out by the DeepMind group, different applications of deep reinforcement learning have been investigated: from robotics and control theory, to web crawling, to solving traffic congestion problems.

The following subsections will outline the necessary knowledge on reinforcement learning and how it applies to the traffic congestion problem.

A. Formal Reinforcement Learning

Reinforcement learning has a rich, mathematical history particularly with work of the 1950s from the American mathematician Richard Bellman. Bellman is famous in computer science for dynamic programming and his interest in mathematically finding optimal solutions to control problems [7]. However, before delving further into Bellman's work, it would be meaningful to provide a broad overview of the process in how learning takes place.

In the reinforcement learning model, a decision-making entity known as an agent can gather feedback on its decisions from an external environment. Ideally, an agent's decisions, or actions, will have a consequence that alters the state of the environment in some way. Throughout this process, the environment has the ability to provide an agent with a reward signal. The reward signal's purpose is to provide information to the agent as to whether the action just taken aligns with

an agents' overall strategy. For example, if an agent were a puppy and its environment was a puppy training center, then a puppy's policy would be to repeat the actions that give it the most treats.

A way of formulating problems into mathematical equations that can help us apply Bellman's work in optimality, is to transform the problem into a Markov Decision Processes (MDPs). An MDP is a 5-tuple that consists of:

- S set of states,
- A set of actions,
- P transition probability functions,
- R reward function,
- G discounting factor for future rewards

With this information, we can use algorithms based on the Bellman optimality equations to learn from an agents interaction with an environment which action will yield the highest reward. Figure 1 is a visual representation of the interaction between the agent and the environment, and how the parameters of the MDP model the learning process.

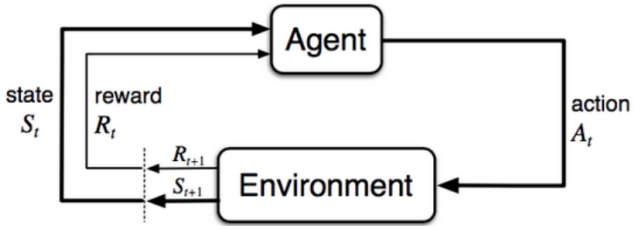

Fig. 1. Abstract model of the learning process.

Some important terminology to keep in mind going forward are the agent's policy, reward signal, value function, and model of the environment. The relationship between these terms and its application to traffic signal control will be developed further in the following section.

III. Traffic Intersections as RL Environments

In this section, we will see how traffic congestion will be modeled as an environment in our reinforcement learning approach.

The environment in our model is a road intersection between four one-lane roads. For now, the cars populating the road are simulated with two different settings. One setting is to have cars populate either the horizontal or vertical lanes at a lower rate than the other. The second setting is to populate both lanes with cars at similar rates. This way, we can compare the performance of the learning algorithm between both kinds of traffic.

We define the agent to be a traffic signal controller in an intersection. The agent's action-set will be limited to be 0, 1, where 0 encodes the light sequence where the horizontal lanes are green, and the vertical lanes are red. Following the work of [3], we define the sequence of traffic signal lights as statuses. The initial status encoded as "0" and represents a green light on the horizontal lane and a red light on the vertical lane. The "1" status is the inverse: a green traffic light on the vertical lane, and a red traffic light on the horizontal lane. In this model, we do not include a unique status for the yellow light in the sequence. For now, we will have a strict sequence for the yellow traffic light that will depend on the current status and the action decided by the agent's policy.

As an example, let us refer to Figure 1. If our current state in the environment is status 0, and the agent's policy chooses action 0, that means that the horizontal lane continues to be green. In this case, there is no need for a yellow light to occur. Only in the case where we switch from one status to the other will a yellow traffic light be required. The reasoning is that this will allow cars ahead of a safe breaking distance, such as the middle of the intersection, to drive through safely.

A. DQN

A popular reinforcement learning algorithm that is we are currently testing is the Deep Q-Network algorithm. To do so, we use a traffic simulator that interfaces with an api called gym.

The essence of the DQN algorithm is that the agent picks an action with the highest Q-value. The Q-value can be thought of us as the Quality of taking action A, in state S. The agent then learns by being able to store the consequences of that action, and updates a neural network to approximate the Q-value for the next set of actions, in the next set of states.

The main equation of Q-learning is the Q-star function:

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

This function will be approximated by the neural network using a loss formulated below:

$$L(\theta) = [Q(s,a) - r - \gamma \max_{a'} Q(s',a')]^2$$

IV. Simulator Design

This section will describe the best features and advantages of the main technologies used to develop the project. Additionally, the structure of the code and instructions on how to access and run the code will be outlined.

A. OpenAI Gym

OpenAI is a non-profit organization dedicated to research in general artificial intelligence. As part of their mission, they developed a set of programs to help the reinforcement learning research community share and expand on each others' research. Gym is one of those programs and has had a major impact on the community. Gym makes it accessible, through an application program interface (API), to train and compare reinforcement learning agents in multiple kinds of environments that come within gym.

Additionally, gym was designed to make it easy to develop custom environments, all while being able to leverage the features that make gym a powerful tool for the reinforcement learning researcher.

B. Pygame

An important open-source package used in this project is the python library Pygame. It is a popular library for making multimedia applications, like games, built on top of the excellent SDL library. Like SDL, pygame is highly portable and runs on nearly every platform and operating system.

While OpenAI's gym includes many free and basic environments, mujoco requires an additional verification and set up step that is not ideal. Similarly, other research in traffic congestion uses an external simulator that has less than ideal set up requirements. The openness and portability of Pygame made it an attractive engine to create the traffic simulation as an environment to interface with gym. Direct advantages of using Pygame include that its portable to all major operating systems, does not require OpenGL, has multi core CPU support, and uses optimized C, and Assembly code for core functions.

C. Structure of the Programs on the GitHub Repository

All of the code used for this project is available in two separate repositories hosted online on GitHub. The first repository is solely for the traffic simulator python file that manages the simulation in Pygame, as well as the file structure to be installed and compatible with the methods of gym.

Before cloning the GitHub repository of the code, it's required to install the dependencies of the project: gym, pygame, keras, tensorflow.

The GitHub url for the repository is " https://github.com/oscastellanos/gym-traffic". Anyone (using a terminal on their operating system)can use the git commands "git clone https://github.com/oscastellanos/gym-traffic" in order to download a copy of the code to the directory of their choosing. The next steps is to go to the directory and run the commands "pip install -e ."

Now, whenever you import gym to your new script, you may also import gym-traffic to make a traffic gym environment.

The second repository is for the DQN-traffic.py script. The GitHub url for this repository is "https://github.com/oscastellanos/DQN-traffic" and it can be cloned using the same command described above. The main purpose of these files is to provide a clear program that trains our agent to perform in the gym-traffic environment of above. The code provides the DQN solver class, and a score logger file that saves rewards per run to a csv file and plots it as a png image.

1) TrafficSimulator: The traffic simulator file has four main types of classes, and two additional types that inherit from these for object extensions. The container and controller class of the file is the TrafficSim class that supports the methods that will instantiate the other objects in the simulation, as well as interface with external libraries, such as gym. The other classes provide the abstractions for horizontal and vertical shapes of cars, and the different flows of traffic of each lane, such as north-bound traffic flows, east-bound traffic flows, etc. As an example, a screenshot of the running program is shown below in Figure 2.

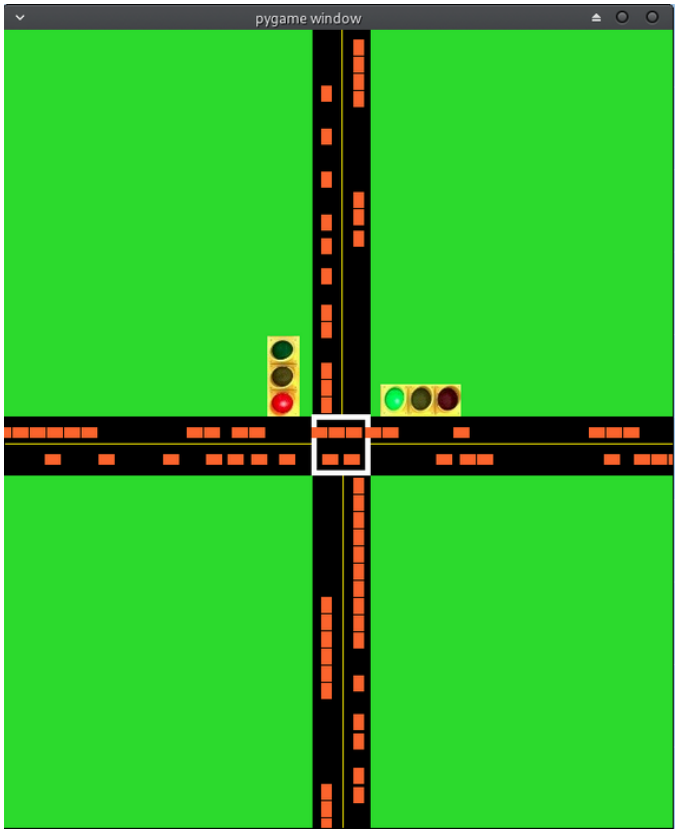

Fig. 2. Snapshot of an intersection in the traffic simulation

2) TrEnv (Traffic Environment): The TrEnv file provides the methods required by gym in order to work properly with the framework. These methods are:

- step,
- render,
- reset

3) DQN-traffic: The DQN-traffic file initates a traffic signal agent to use a DQN learning algorithm to solve the traffic congestion problem.

An overview of the process is shown on Figure 3.

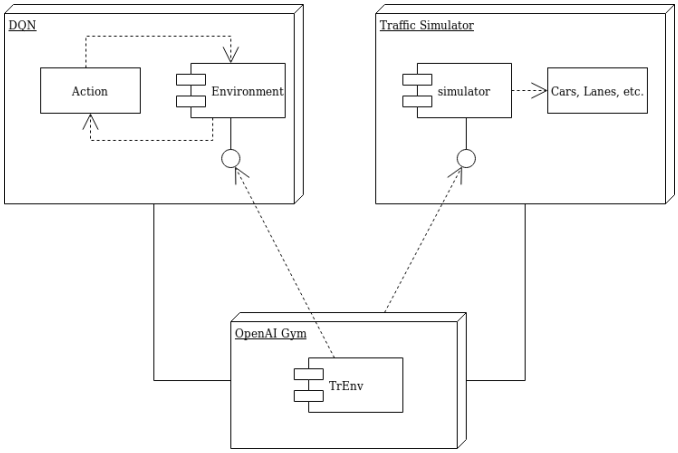

Fig. 3. How the API's communicate during learning.

V. Experiments

In order to debug and make sure the implementation of the code was behaving as expected and especially to verify that the learning algorithm was functioning, we ran experiments using the DQN-traffic.py file.

Examples of the output are shown below:

```
2494
Status is 1. Action is 1.
Scores: (min: 46, avg: 1302.375, max: 2510)

2510
Status is 1. Action is 1.
Scores: (min: 46, avg: 1317.4814814814815, max: 2526)

2526
Status is 1. Action is 1.
Scores: (min: 46, avg: 1332.4146341463415, max: 2542)

2542
Status is 1. Action is 1.
Scores: (min: 46, avg: 1347.1807228915663, max: 2558)

2558
Status is 1. Action is 1.
Scores: (min: 46, avg: 1361.7857142857142, max: 2574)

2574
Run: 1, exploration: 0.7255664080186093, score: 2574
Episode done in 2574 steps, total reward -63230.00
```

Fig. 4. At the end of training one epoch.

Initially, there were a few bugs in the simulator that prevented us to train more than 10,000 time steps per trial.

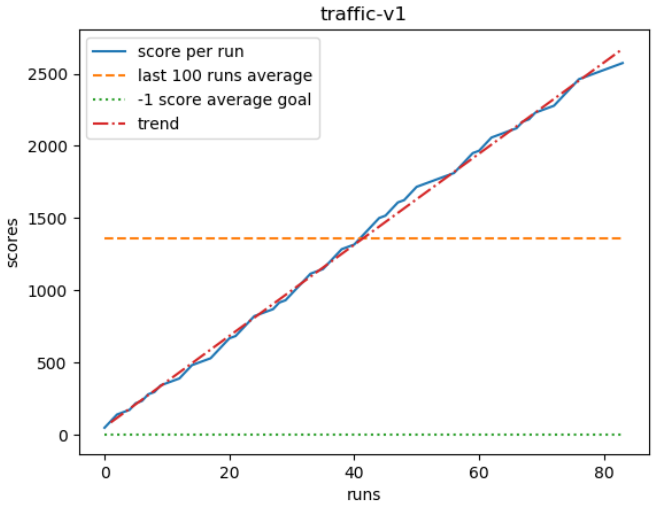

Fig. 5.

The logger will be updated to plot multiple trials with their own lines in order to better compare the performance as the sequence of trials are completed.

Also, updates to plot more metrics such as average max Q value, episode return, and loss are currently in progress.

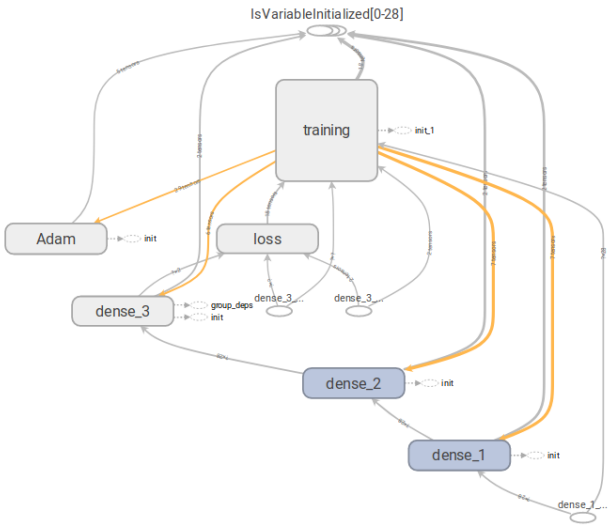

Fig. 6. The architecture of the neural network used for DQN.

The next update to the architecture will be a convolutional neural architecture that will take a pre-processed image of the simulator and learn to extract the important features rather than explicitly declaring them manually. This approach follows the one taken by the DeepMind team to achieve human-level performance on the majority of Atari games included in the Arcade Learning Environment (and are also compatible in OpenAI's gym).

VI. Future Work

A. Multi-agent Extensions

We've developed a traffic simulator that allows us to test a traffic signal control agent in multiple environments. In addition to evaluating our algorithm to the state of practice simulators, we will investigate how multi-agent reinforcement learning algorithms can train a traffic signal agent to adapt to different events that cause traffic. For example, events such as cars stalling, accidents, and natural disasters like flooding. Our simulator gives us control over parameters that can simulate these conditions. We can specify environments with multiple roads, intersections, traffic lights, and specify if zones aren't accessible to drivers. Thus, given an environment where an event has taken place that causes traffic, the traffic signal controller can coordinate routes to circumvent traffic-heavy areas.

B. Flow, RLib, and SUMO

An alternative to continuing development of a simulator, along with creating, testing, and iterating over learning algorithms, is to use this preliminary work as a step to using other traffic simulation and management software currently used in research.

One such program is the open-sourced Flow library developed by the Mobile Sensing Lab at the University of California Berkeley. The program they have developed is well documented, has integration with other tools such as OpenAI's

gym and AimSun's API's for real-world road simulation, and was developed with high-performing and distributional reinforcement learning frameworks such as RLib. One of the many opportunities Flow could provide is to model sections of interest of real cities for scalable multi-agent reinforcement learning algorithms.

Another popular program in the traffic reinforcement learning research community is to use the Institute of Transportation Systems at the German Aerospace Centers' Simulation of Urban Mobility (SUMO) to provide an API interface to a widely adopted traffic and road simulation environment.

VII. Conclusions

We've developed a traffic simulator that allows us to test a traffic signal control agent in multiple environments. This preliminary work is a single step towards properly evaluating our algorithm to the state of practice simulators, such as SUMO and Flow.

The next step forward is to extend the traffic simulator to support a network of multiple intersections, as well as, the ability to dynamically adjust the driving conditions of distinct lanes. These set of features will allow us to be able to test multi-agent reinforcement learning algorithms in this specific environment.

Additionally, we plan on developing more advanced features, such as: wider range of driving behaviors, support for multi-laned roads (which will expand the set of legal statuses), and advanced traffic signals such as protected left turns.

Ultimately, we aim to investigate how multi-agent reinforcement learning algorithms can train a traffic signal agent to adapt to different events that cause traffic. For example, events such as cars stalling, accidents, and natural disasters like flooding.

References

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage and Anil Anthony Bharath. *A Brief Survey of Deep Reinforcement Learning*. arXiv e-prints

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*. arXiv e-prints

[3] Xiaoyuan Liang and Xusheng Du *Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks*. arXiv e-prints

[4] Seyed Sajad Mousavi, Michael Schukat and Enda Howley *Traffic light control using deep policy-gradient and value-function-based reinforcement learning*. arXiv e-prints

[5] Juntao Gao, Yulong Shen, Jia Liu, Minoru Ito and Norio Shiratori *Adaptive Traffic Signal Control: Deep Reinforcement Learning Algorithm with Experience Replay and Target Network*. arXiv e-prints

[6] Marco Wiering *Multi-Agent Reinforcement Learning for Traffic Light Control*. arXiv e-prints

[7] Richard Sutton and Andrew Barto *Reinforcement Learning: An Introduction, 2nd edition*. MIT Press.