MONASH University
Engineering

Deep Reinforcement Learning for Intelligent Traffic Management

Christopher Adis, Kai Andrews, Evan Tan
October 2021

Department of Electrical and Computer Systems Engineering

Introduction

Traffic control and congestion is a prominent issue faced in urban areas where the number of road users continues to increase. This has negative impacts consisting of environmental pollution caused by longer commute times, and economic impacts to create larger traffic networks as well as social impacts for road users who spend extended periods of time in traffic.

The use of intelligent traffic control systems is becoming increasingly popular in ‘smart cities’ where there is readily available traffic data from sensors throughout traffic networks **[2]**. To reduce congestion and improve traffic flow, traffic control systems can be optimised by adopting deep reinforcement learning (RL) to train agents that act as a human traffic controller, bringing multiple benefits such as reduced labour cost,increased reliability and the ability to dynamically adapt to changing traffic conditions in real time to reduce congestion. However it is difficult to train the RL agent to control intersections effectively therefore careful selection of neural network, optimization algorithm and reward function are required.

In our project, the use of a single agent at a single intersection with 16 lanes is explored through an open source traffic simulator, Simulation of Urban MObility (SUMO). A custom neural network architecture was implemented and investigated, specifically a Multi-Layer Perceptron (MLP). This approach was compared using mean travel time against a traditional traffic controller as a baseline, where the traditional controller simply selects next phases sequentially with set durations of 60 seconds plus 2 seconds transition time for yellow lights. The network input is the observation space, consisting of the normalized density and queue length, a 4D phase tensor indicating the current traffic light phase, as well as a boolean flag which indicates whether the minimum green time has been met. The network output is the next light phase at the intersection. A policy gradient algorithm, Proximal Policy Optimisation (PPO), in particular the PPO-Clip variant, is used to optimise the agent’s policy and maximise reward. Different reward functions were crafted based on achieving the objective of minimising queue length and wait time, by using cumulative waiting time from all lanes, as well as an urgency metric.

Related Work

Existing approaches vectorize all lanes across an intersection, considering the number of waiting vehicles or queue length as the observation space **[4]**. This approach poses problems as the level of congestion varies according to the length of the queue, and if the length of lanes change the network would need to be retrained if the numbers of cars were not normalized according to each lane’s capacity. Furthermore Bouktif et al. **[4]** combine the use of discrete and continuous action spaces, to combine the best of both policy-based and value-based and allow the agent to control the phase pattern for the lights and the duration of each phase respectively to achieve higher performance than where just one action space is used. **[4]** uses Q-learning to optimize a multi-pass deep Q-network (MP-DQN) - consisting of 2 neural networks to select the phase and the duration of the traffic lights. Joo et al. **[5]** use a Q-learning algorithm to learn the optimal action-value function, $Q(\cdot)$. The spatial-temporal nature of traffic is also considered in this paper to create a model that can dynamically handle traffic demands such as side roads, tidal traffic flows. [5] also focuses on creating an intelligent controller that is general enough to be applied to various intersection configurations and traffic demands.

In summary, both approaches discussed consider the specifics of traffic control such as choosing to control both the light phase and duration or considering the non-uniform demands for different lanes. A notable feature of both approaches is that MP-DQN and Q-learning are both off-policy approaches, making them efficient in the reuse of old data but not guaranteed to produce stable results if the policy being followed is not sufficiently similar to the policy being evaluated. In contrast, this report will explore the use of an on-policy approach, PPO which evaluates the same policy used to make decisions. Generally, the advantage of this is that the objective (in this case wait time) is being directly optimized and stability is greater.

Dataset & Environment

With reinforcement learning, the dataset cannot be simply defined as in image classification and object detection tasks. The environment could be thought of as the dataset, where the environment is constantly changing whenever the agent enters new states by sampling from some action space. Consequently, the agent will observe different states after exploring different sets of actions. The SUMO simulation package allows for highly controllable and customisable traffic environments. The environment used for this project consisted of a single intersection with 3 lanes and 1 turning lane in both directions. This setup was chosen as it has sufficient complexity to be optimized, and is representative of real-world configurations. The traffic behaviour was modeled using a normal distribution to ensure a continuous stream of traffic passes through the intersection. This was achieved by inserting a car into the simulation every 1.538 timesteps with a 75% chance of going straight and a 25% chance of turning. Source and destination are randomly assigned in each case.

The TraCI API was used to interface between the agent and access and interact with the SUMO environment. The data read from the SUMO environment was the number of halted cars in the environment, waiting time, the traffic density and the current phase of the traffic lights.

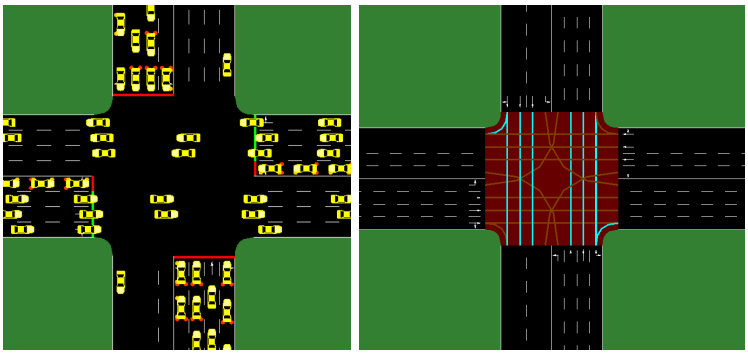


Fig. 1: Intersection setup for SUMO, where the agent controls the phases

Experiments & Methodology

Experiment Architecture

The github repository SUMO-RL **[6]** is used as an interface between the RL architecture implemented and the SUMO environment. It also provides compatibility with RL libraries RLlib and Stable Baselines3. SUMO-RL allows for easy customisation, this was used to apply the RL implementations to the traffic optimisation task by modifying the observation space, reward function, network hyperparameters and implementing the action space. Multiple networks are trained and compared with variations of the above parameters to investigate the optimal configuration.

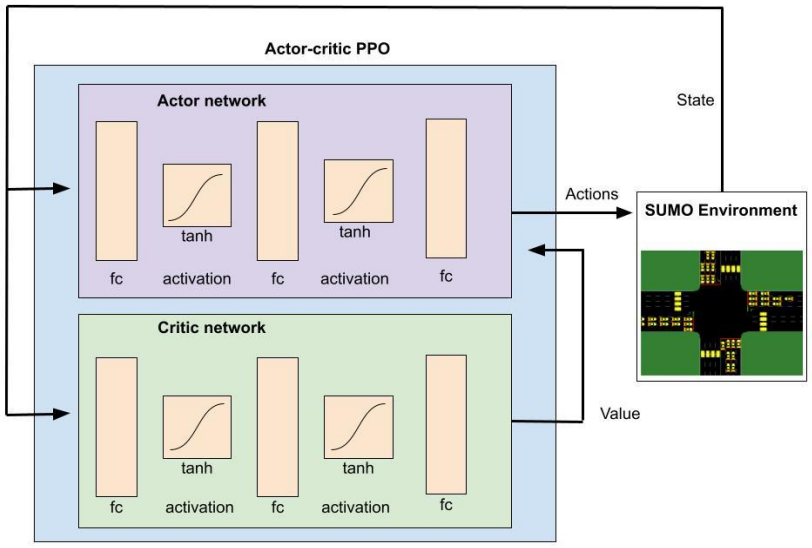


Fig 2. Our actor-critic network architecture

Observation & State space

For the observation space in the intersection environment a state vector is defined as the current phase of the lights, the density of each lane and the queue in each lane. The current phase is a one-hot coded vector relating to each traffic light in the intersection, the density is the number of cars divided by the size of the lanes and the queue length is the number of cars in each lane.

Action Space

The agent's action space consists of the 4 green light phases for the intersection, choosing whether the North South or East West lanes will receive a green light. It selects a phase to set every decision timestep, equating to every 7 simulation timesteps. If the phase chosen is the same as the previous phase, the transition phase using yellow lights is skipped.

Network used - MLP

Using an MLP architecture, the effect of having shared inputs and non-shared inputs to the actor-critic networks is explored. Non-shared layers were used as it was thought to improve stability of the policy. For the critic, the following architecture is used, where each tuple represents the input and outputs of each

layer **(37, 256) -> Tanh -> (256,256) -> Tanh -> (256, action-size)**. Tanh activation functions were chosen due to being centered at 0, compared to 0.5 for Rectified Linear Unit (ReLU) activations. For the actor, the architecture used is **(feature shape, 64) -> Tanh -> (64,64) -> Tanh -> (64, 1)**. These dimensions were used for both the shared and non-shared actor-critic networks.

For the shared actor-critic, the feature extractor is a fully-connected layer with ReLU activations, where the layer is of dimensionality **(37, 256)**, where 37 corresponds to the number of dimensions in the observation space.

The critic network has the same depth as the actor network but has 4 times larger width.

Optimizer - Proximal Policy Optimization

The role of the optimiser in RL is to update the policy for actions taken by the agent. In this experiment the algorithm used is PPO-Clip (a popular variant of PPO). PPO is an on-policy algorithm, it updates the same policy online using fresh data, compared to an off-policy algorithm which uses some replay buffer or memory. The advantage of this is that it directly optimises the objective of minimising waiting time. As PPO is an actor-critic method it consists of an actor network that uses the policy to make an action and a critic network that takes the state and produces a value (an estimate of how good the current state is).

Clip range - in PPO is a function that limits how much the policy can update. This improves stability as it prevents an action with a very different probability from the old policy causing an update that is too large (and possibly moving away from a region that provides a good advantage estimate). The clipping value used is **0.25** to ensure updates are not too large. When the advantage estimate is positive or negative, updates are more conservative according to the clipping value to prevent policy collapse via greedy updates. This is also under guidance by the study, “What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study”, by the Brain Team **[7]**.

Gamma - is the discount factor (between 0 and 1). It is a measure of how much we value future rewards. For traffic control the future reward is important, hence the value of gamma is set as **0.99** (it is common practice to set gamma close to 1). This value was also guided from the conclusions and recommendation given by the study by the Brain Team [7].

Number of steps - steps the environment is run for per update. Set to **256** to capture enough relevant simulation timesteps.

Entropy coefficient - is used as a regularizer to prevent convergence on one action. This can occur when one action is more probable than all others (entropy is at a minimum). The entropy coefficient is set to **0.09**

Learning rate - is the amount the weights of the network are updated. It is set as **1e-4**

Max gradient norm - is used to scale the error gradient (in conjunction with clipping) to prevent large updates and instability. It is set as **0.9**

GAE lambda - factor to reduce variance of the generalized advantage estimate (GAE) and can improve stability. It is set as **0.9**. This is the recommended value concluded in the study by the Brain Team [7]. Performance variation from varying this value was outweighed by the choices of reward and network structure.

Number of PPO Epochs - is set as **5**

Reward Function

Formulation

Waiting Time Reward

Reward function provides feedback through positive and negative rewards, which in turn tells the agent how good or bad it is currently doing in achieving its objective - to maximize cumulative rewards. The reward function is crafted for the agent to learn to decrease mean waiting and travel time through the intersection. The reward is maximised by optimising the agent's policy to choose appropriate actions. The reward function initially used in the training of the agent is defined as waiting time reward, which is the change in cumulative waiting time of all vehicles across all lanes between 2 simulation timesteps, which is a potential-based reward and is **highly flawed** for this problem - when waiting time is already low, the model would not receive high rewards in the following timesteps.

$$R(s_t, a_t) = \sum_{l=1}^{L} \sum_{c=1}^{C} \left(w_{t-1} - w_t \right)$$

Fig 3. Original reward function, considering the difference between current accumulated waiting time and that at the previous time step, where L is the number of lanes, and C is the number of cars per lane.

Urgency Reward

An important aspect of RL is defining an optimal reward function that maximises the objective. Hence a reward function that attempts to consider the specifics of traffic control optimisation was investigated and implemented, it is defined as:

$$R(s_t, a_t) = \sum_{l=1}^{L} \sum_{c=1}^{C} U_l \odot O_l$$

Fig. 4: New reward function based on urgency, where L is the number of lanes

$$O_l = 2\left(\frac{\overline{V_{cars}}}{V_{max}} - 0.5 \right)$$

Fig. 5: Throughput per lane, considering the mean speed of all cars on the maximum allowable speed

We used expert knowledge of the field of traffic management in order to tune and optimize our implementation. In this case we use expert information to formulate the reward function that attempts to improve upon the existing potential-based reward.

Consider a lane with high density, if all cars are constantly moving, however not coming to a complete stop the waiting time would be low, but the congestion would still be present across the intersection. In the field of traffic management, traffic controllers are designed so each controller minimises its negative effect of the flow of vehicles moving throughout the wider city network. The urgency metric, $\boldsymbol{U_l}$ in Fig. 4 is derived by multiplying normalized queue length by normalized lane density, to represent the actual number of cars that need to be "flushed" from that lane, better representing congestion in the network. Thus, we are providing a well guided reward function for the desired actions as well as being in an efficient form for training. This is because if normalized density is 1.0, but queue length is 0.0, there are actually zero cars waiting and there is no urgency in ensuring the lights turn green for that particular lane, proving the correct magnitude and sign of the reward in this situation. Urgency is combined with another metric, throughput, $\boldsymbol{O_l}$. Throughput measures the average speed of cars in a lane compared to the maximum allowable speed, set to 13.89m/s in the simulator, and is in the range [-1, 1] as seen in Fig. 5 above. By awarding positive rewards where the average speed normalized, the positive reward feedback tells the agent that higher throughput is better. The sign changes at the point where average speed is half of the maximum allowable speed, discouraging the agent from selecting those actions that lead to congestion.

Validation

While during the training and evaluation progresses information from the environment was also logged each time an observation was taken. To help craft the reward function, all of the most relevant metrics which are accessible through the TraCI API were logged, where at each time an observation is taken metrics are recorded per lane of the intersection, including the mean speed of the vehicles in that lane, the occupancy which given as the area of the entire lane which is occupied by vehicles, the number of vehicles currently in each lane, the number of vehicles currently halted in each lane where a vehicle being halted is defined as having a speed less than 0.1m/s. Another metric which was logged is derived from the accumulated waiting times assigned to each vehicle in the simulation. This measure keeps track of the amount of timesteps out of the previous 100 simulation timesteps in which the particular vehicle has spent at a speed less than 0.1 m/s, which is then aggregated per lane of the intersection. In assisting with crafting a reward function, these vector values were summed for visuation to display the state of the entire intersection. This logged data allowed for rapid validation of the feasibility of concepts and aided in timing, as the observations of previous training sessions would be fed into a prototype reward function, to check whether the magnitude and sign of the reward function values matched the performance of the simulation as seen from the key indicators.

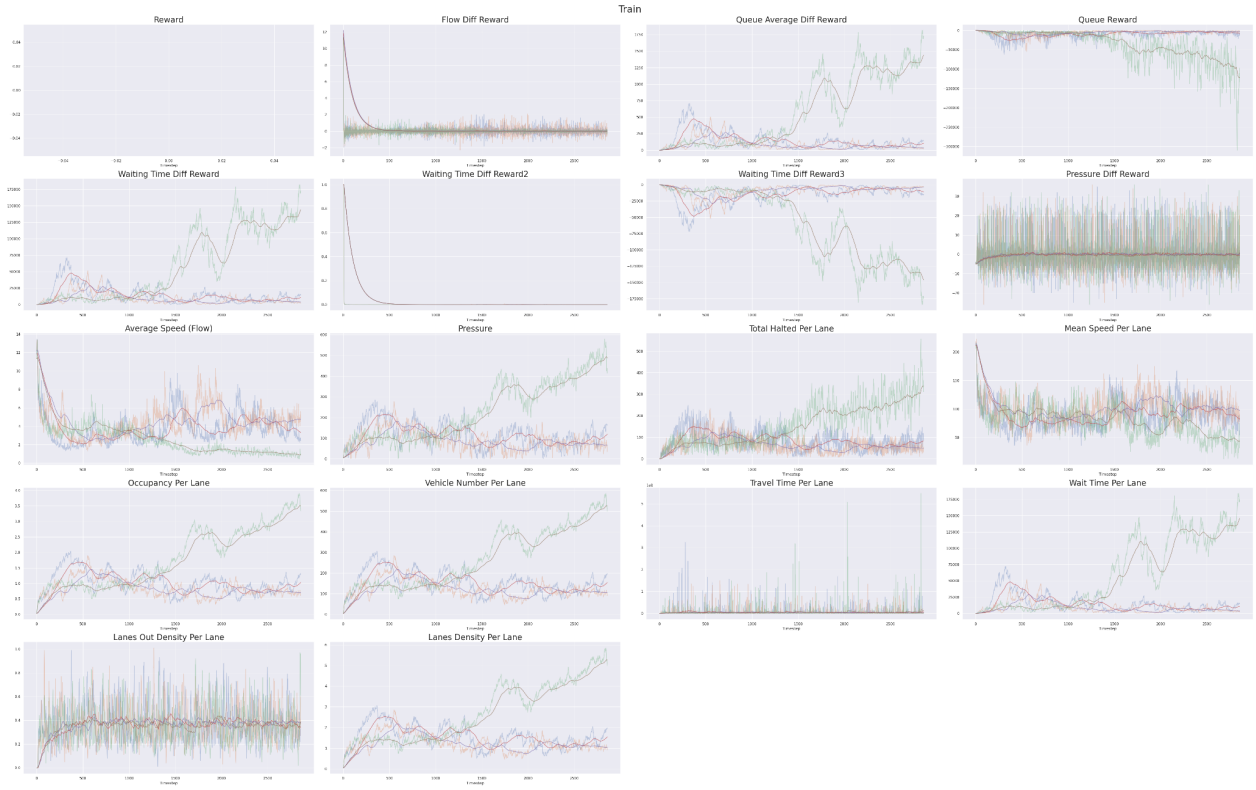


Fig. 6: Custom Dashboard created for monitoring training and validating reward function ideas. This dashboard has been evaluated over a single timestep.

Each figure in this dashboard shows given and derived metrics at each timestep of a previous training session, for a single episode. Each plot contains information from three concurrent environments where each is also attributed to a smoothed line derived from the noisy data, an exponential moving average weight of 0.99. Here we notice that performance in one out the the three environments is much worse then the others, shown by higher wait times, higher queue lengths and lower average speeds. This diversity was useful in preliminary testing different reward functions under varying scenarios.

Crafting the reward function is tricky for a traffic intersection, as the agent does not reach terminal states for long periods of time unlike OpenAI's LunarLander-v2, where every episode ends whenever the agent lands the ship successfully or crashes it.

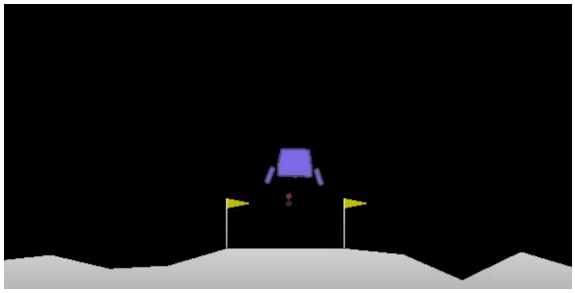


Fig. 6: OpenAI Gym's LunarLander-v2

Training

Training was performed using multiple environments to improve training times. The evaluation interval was every episode (1.5e4 simulation timesteps). The best model was checkpointed and updated when performance improved. A learning rate of 1e-4 was used with a rollout fragment length of 256 and a SGD minibatch size of 256.

Performance metrics and benchmarks

A benchmark setup was used to evaluate the performance of the RL traffic controller. The benchmark consists of a traffic controller with timed phase durations in a fixed pattern. The performance of the benchmark was measured using average speed, average wait time and queue length and compared to the performance of the RL controller.

Analysis & discussion

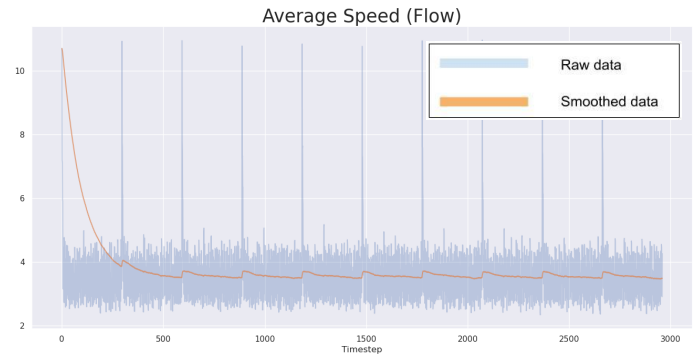
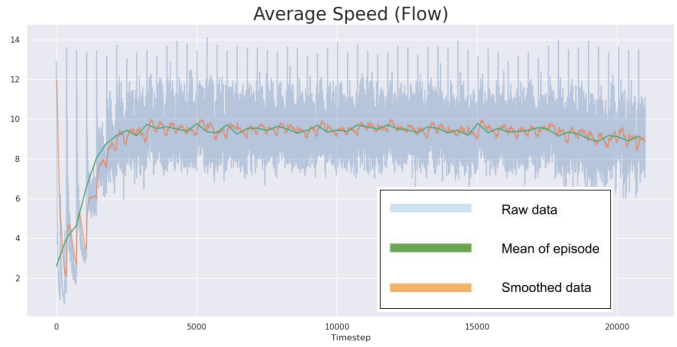
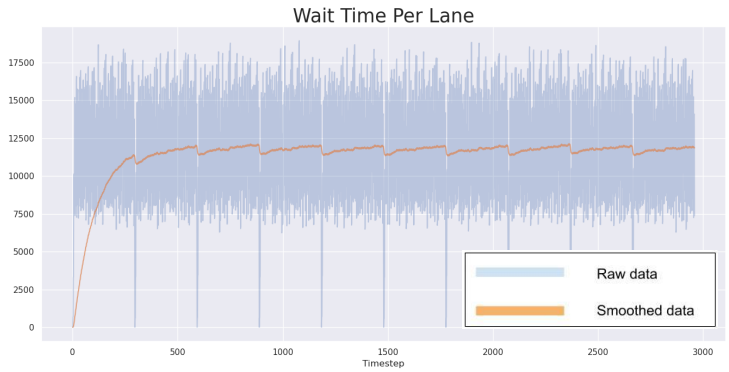
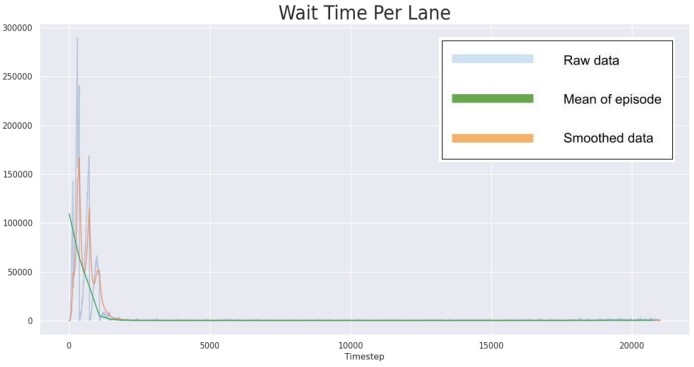
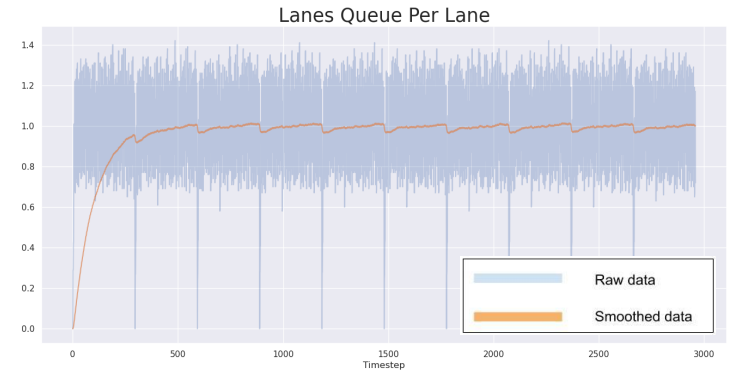
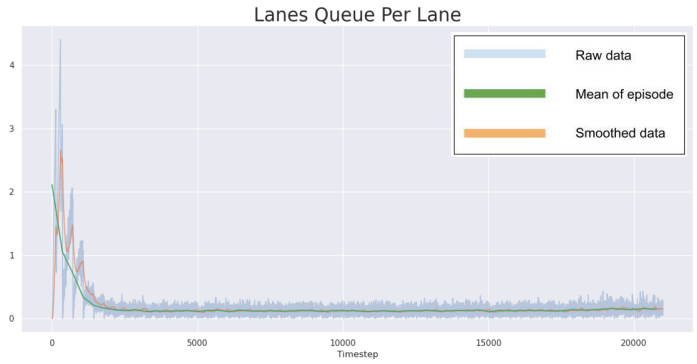
Fixed Timer Agent - Performance over 1 episode	RL Agent, Evaluation Performance over training
	
	
	

Table 1: Comparison of fixed timer agent (left) and RL agent (right) over training

Comparison of traffic controllers

Controller	Waiting time (s)	Average speed (m/s)
Baseline	10771.36	3.91
RL agent - waiting time reward	400.66	8.88
RL agent - urgency reward	211.62	9.78

Table 2: Comparison of different RL agents against the baseline

Discussion

In the above plots the average wait time and average speed were calculated with the chosen model, over an entire episode. Shown on the left above is the performance of the fixed timer agent, where its performance is shown over a single episode. The plots for speed, queue length and wait time show that the time based approach is consistent as it reaches a steady state for all 3 parameters.

Shown on the right is the progress of the training by visualising the performance of agents in the evaluation sessions undertaken throughout the training process. Where the raw and smoothed data is shown as well as the average scores for each episode, shown in green. While the RL agent performs worse than the fixed timer at the beginning of training, it quickly outperforms it as training progresses.

The table above shows the comparison of the timed controller, RL agent with a simple waiting time reward function and RL agent with the urgency reward function discussed above. The RL agent with waiting time reward function far outperformed the fixed timer baseline with a waiting time reduction of 96.3% and a 127.1% increase in average speed. This is expected as the fixed controller cannot adapt to the current loads faced by the intersection, where the direction of demand varies dynamically. However, the implementation of the custom reward function based on traffic urgency and considering the specifics of traffic control performs far better, reducing waiting time by 47.2%, and increasing average speed by 10.1% for the simple waiting time reward implementation.

These implementations have access to these types of information about the current situation, where they are given the opportunity to spend more of the share of available time giving access to the directions of traffic would benefit the problem of reducing the waiting time the most. It is important to note that in real world applications of traffic controllers there are also other human oriented metrics which need to be considered. As the most efficient solution may cause frustration to drivers. Where a lane which is not very much in demand should still be given priority even if it is not the most efficient solution. As seen in our demo, these factors are also handled by our implementation, due to our designed reward function. When our reward function is used the performance is increased further as we have applied aspects of traffic management which align with the goals of the problem.

Conclusions & Future Work

The experiments in this report have applied a RL implementation to a traffic control system for determining the optimal phase of traffic lights at an intersection. PPO has been used to maximise the agents objective of minimising mean wait time and travel time through an intersection. An MLP has been used to select the action to be performed by the agent and to estimate the value of the agent's current state. The network was trained in SUMO to simulate controlled traffic conditions. The performance evaluation of the RL traffic controller against a benchmark using timed phases showed a significant improvement in both waiting time and average speed. The implementation of a custom reward function also showed significant improvement over the simple waiting time reward that was initially implemented.

Recommended future work includes the implementation of a Long short term memory (LSTM) RNN to replace the MLP. The advantage of using an LSTM is that it has both long term and short term memory functions due to the feedback loop incorporated. This memory may provide benefits for the traffic optimisation problem as dependencies that span over long periods of time can be learnt. This may result in an agent more capable of optimising traffic flow over large time intervals, resulting in a more efficient intersection overall. Other extensions on this work could involve training an agent on various intersection environments and traffic flow distributions. This could result in an agent that is more general and could be applied to a wide variety of real world traffic optimisation scenarios.

References

[1] X. Liang, X. Du, G. Wang, and Z. Han, “Deep Reinforcement Learning for Traffic Light Control in Vehicular Networks,” *IEEE Trans. Veh. Technol.*, vol. 68, no. 2, pp. 1243–1253, Feb. 2019, doi: 10.1109/TVT.2018.2890726.

[2] K. Su, J. Li, en H. Fu, “Smart city and the applications”, 2011 International Conference on Electronics, Communications and Control (ICECC), bll 1028–1031, 2011.

[3] M. Guo, P. Wang, C.-Y. Chan, and S. Askary, “A Reinforcement Learning Approach for Intelligent Traffic Signal Control at Urban Intersections,” in 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Oct. 2019, pp. 4242–4247. doi: 10.1109/ITSC.2019.8917268.

[4] S. Bouktif, A. Cheniki, and A. Ouni, “Traffic Signal Control Using Hybrid Action Space Deep Reinforcement Learning,” Sensors, vol. 21, no. 7, Art. no. 7, Jan. 2021, doi: 10.3390/s21072302.

[5] Joo, H., Ahmed, S. and Lim, Y., 2020. Traffic signal control for smart cities using reinforcement learning. Computer Communications, 154, pp.324-330. doi: 10.1016/j.comcom.2020.03.005

[6] Lucas N. Alegre, “SUMO-RL”, GitHub repository. GitHub, 2019. https://github.com/LucasAlegre/sumo-rl

[7] M. Andrychowicz *et al.*, “What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study,” *arXiv:2006.05990 [cs, stat]*, Jun. 2020, Accessed: Jan. 04, 2022. [Online]. Available: http://arxiv.org/abs/2006.05990