An Ensemble Method for Traffic Light Management

Evangelos Mathioudis

A Dissertation

Presented to the Faculty

of University of Nicosia

in Candidacy for the Degree

of Master of Science

Recommended for Acceptance

by the Department of

Computer Science

June, 2022

Abstract

Traffic congestion costs in the EU are estimated to be 1% of the EU's GDP, and good solutions for traffic light control may reduce traffic congestion, saving time and money and reducing the environmental pollution. Reinforcement learning relies on reward signals from the environment to learn policies to determine the optimal actions on traffic signal control. However, classical reinforcement learning approaches fail to scale with large state spaces. Thus, we focus on deep reinforcement learning applications. In addition, applications of ensemble methods on reinforcement learning show improvements in performance. To this end, in this work, we experiment with applications of ensemble methods on traffic light management of a single intersection. Our results reveal a potential solution to this issue. Compared to state-of-the-art algorithms, our method improves the performance of its contributor algorithms, especially on the total waiting time overall lanes.

Acknowledgements

I am deeply grateful to my supervisor Dr. Ioannis Katakis for his continued guidance and valuable assistance through this work. I learned a lot through this process.

I am also grateful to all professors of the Master of Science in Data Science, Dr. Athena Stassopoulou, Dr. Ioannis Partalas, Dr. Demetris Trihinas, Dr. Michalis Agathokleous, Dr. Nectarios Papanicolaou and Dr. Portides for their support and suggestions throughout this journey.

Words cannot express my gratitude to my wife Pelagia for her unstoppable assistance, for continuously motivating me to keep moving forward, and to my kids Vasileia, Manouil, and Eleftheria. I could not have managed to complete this journey without you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Traffic Congestion costs a lot of money to countries. Apart from the waste fuel, it is responsible for the increase of harmful emissions e.g carbon oxides that can be harmful to human health. Additionally, it relates to intersection management on main or busy roads during peak hours or on different days. By improving traffic conditions countries may increase their cities' efficiency, improve the economy, and make people's daily life easy. One way to tackle the traffic congestion issue is to control traffic signals intelligently. The goal of traffic signal control is to facilitate the safe and efficient movement of vehicles at an intersection.

In most countries, traditional traffic control uses historical real data to define a pre-timed control with fixed intervals for switches on signal phases. In such cases, we find a cycle-based signal plan with fixed-time cycles without adjustments to the traffic demand. Unfortunately, this approach is not able to handle short-term changes in traffic conditions as it doesn't consider real-time circumstances. Another strategy type of traffic signal control is the actuated control. In this kind of control data from sensors may influence phase intervals. Adaptive Traffic Control Systems (ATCS) is considered an effective strategy to deal with traffic congestion. ATCS combines hardware and software to control traffic signal timings. An ATCS adapts to traffic mainly by using data from sensors located on the roads. A downside of this is that sensors have a small area of control thus, the data they gather

usually don't represent the real traffic of the road.

Recently, researchers start to investigate how to apply reinforcement learning (RL) techniques to traffic signal control. The superior performance of RL techniques over traditional transportation approaches is reported on several works [35] [29] [23]. The biggest advantage of RL is that it directly learns how to take the next actions by observing the feedback from the environment after previous actions. On the other hand, the representation of the environment and how to design the connection between environment and decision are issues that prohibit normal reinforcement learning to be applied to dynamically adjust the traffic signals. Additionally, in complex road-networks observation spaces increase thus the complexity in a traditional reinforcement learning system grows exponentially.

Over the last decade, deep learning has gained recognition and many researchers have turned to this field of research. Applications of deep learning on reinforcement learning field lead to agents able to outperform humans in games [21] [25]. This new field is called Deep Reinforcement Learning (DRL). As a result of the excellent performance of this new field of deep reinforcement learning, researchers have applied similar techniques to solve important problems such as traffic congestion. Applying deep reinforcement learning to traffic light control systems seems a promising solution. DRL aids traditional reinforcement learning to approximate nonlinear functions from complex datasets. In this case, an agent learns how to dynamically switch green lights at crossroads and adapts to traffic situations.

Traffic Light Control - Terminology

In this section, we give a brief explanation of common terms used in literature and what we are using in this work.

Approach: An approach corresponds to a roadway that leads to an intersection. The part of the roadway that brings vehicles to the intersection is called *incoming approach* on the other hand, the section of the roadway that leads the vehicles away from the intersection is called *outgoing approach*.

Lanes: An approach consists of a set of lanes. There are different types of lanes similar to

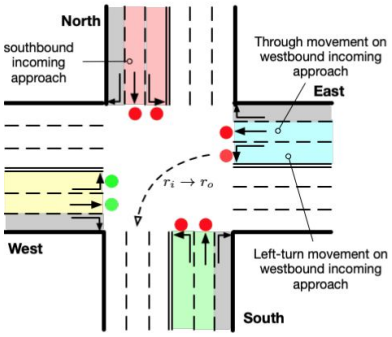


Figure 1.1: Road Structure and Traffic Movement - IEEE ITSC'20

approaches. Consequently, we have the *incoming (approaching/ entering) lanes* and the *outgoing (receiving/ exiting) lanes*.

Traffic movement: Traffic movement is defined as the flow of vehicles moving from the incoming approach to the outgoing. There are three types of traffic movement the *left turn*, *through* and the *right turn*.

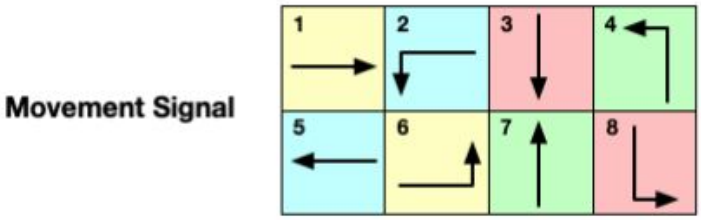
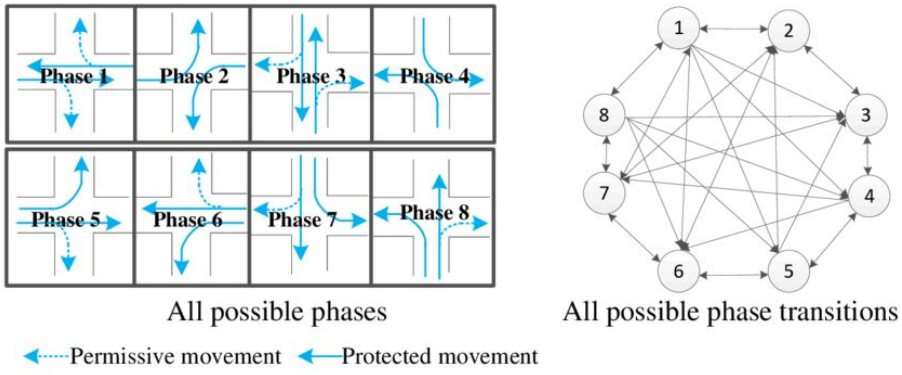


Figure 1.2: Movement Signals - IEEE ITSC'20

Movement Signal: A traffic movement defines a movement signal. A green movement signal

means that the traffic movement is allowed whereas a red indicates that the traffic movement is prohibited. Fig.1.2 presents eight different movement signals and their related permitted movements.

Phase and **Phase interval**: A phase is a combination of movement signals. Fig.1.3 represents the possible phases on a standard four-approach intersection. The period during which all the signal indications remain constant is called phase interval.



Source: Solving simultaneous route guidance and traffic signal optimization problem using space- phase-time hypernetwork

Figure 1.3: Possible signal phase and phasing sequence in a standard four-approach intersection

Phase Sequence and **Signal plan**: Phase sequence is a sequence of phases that defines a set of phases and their order of changes. For a single intersection, a signal plan is the sequence of phases and their corresponding starting time.

A road intersection consists of multiple lanes and traffic lights in multiple directions. When vehicles enter and leave the intersection in multiple directions a traffic light control system should be able to efficiently adapt the time intervals of different phases on each traffic signal to maintain a good traffic flow. On a large scale like city level with multiple intersections, the traffic signal control system should optimize the efficiency of the intersections' flows by dynamically changing the phase of each traffic light considering the whole intersection network.

In this work, we introduce a new approach to traffic signal control. To the best of our knowledge, it is the first time to use an ensemble method to handle the traffic lights in a single intersection. We

use Deep Q-Network [21], Advantage Actor-Critic [19] and Proximal Policy Optimization [24] algorithms to handle the traffic lights of single intersection using the SUMO-RL [1] framework. In supervised learning ensemble methods have been more effective than a single classifier and they lead to better accuracy. Methods like *Bagging* and *Boosting* have been used to train multiple classifiers. In reinforcement learning ensemble methods have been used to either represent and learn value functions or to combine different policies into a final policy for the agent.

In our work we use *Majority Voting (MV)*, *Rank Voting (RV)*, and *Boltzmann Multiplication (BM)* to teach our *ensemble agent* which movement signal to pick at each step. Additionally, we create a transformed version of the RV scheme. Because, PPO outperforms the other two algorithms we develop an ensemble with only two agents an A2C and a DQN.

Based on our results, the most promising method is the RV. On the RV method, we assign a weight to each action which corresponds to the action probability each agent calculates given the state of the environment. With the RV method, we see improvement in the total time delay across all the lanes of the road network.

1.1.1 Outline

In Chapter 2 we introduce the necessary background for Reinforcement Learning and Deep Reinforcement Learning. Additionally, we briefly discuss the most important theory for the reader to know related to the algorithms we use in our experiments.

The following chapter (Chapter 3) discusses the related traffic signal control work. We focus on how previous works represent states and actions. We describe the different types of rewards that have been used and we introduce some of the simulators we came up with.

Next in chapter 4, we describe how we generate data in Simulation of Urban MObility software, and in chapter 5 we discuss our methodology on a single intersection using the ensemble method. In chapter 6 we present our results.

Finally, we close this work by presenting our conclusions and providing suggestions for future works.

Chapter 2

Background

In this chapter, we introduce the necessary background knowledge for the reader to be able to follow the rest work. We initiate our discussion for Reinforcement Learning by talking about topics such as *Markov Decision Process* and *Q-learning*. Then, we briefly describe the algorithms we use to conduct our experiments on traffic signal control at a single intersection.

2.1 Reinforcement learning

Reinforcement learning (RL) relies on the concept of learning through trial and error. An agent learns to perform tasks by interacting with an environment. The agent will take an action a_t based on the environment's current state s_t and the environment will return a reward signal r_{t+1} to the agent and change to the next state s_{t+1}. Figure 2.1 depicts how a reinforcement learning system works. Through this process, the agent acquires knowledge to solve the environment by making the best decision at each time step regarding the environment's state based on a policy that aims to maximize the agent's rewards.

2.1.1 Markov Decision Process

In reinforcement learning, an agent interacts with an environment and learns to decide what action to take next based on its current state. Following the Markov Decision Process (MDP), environments

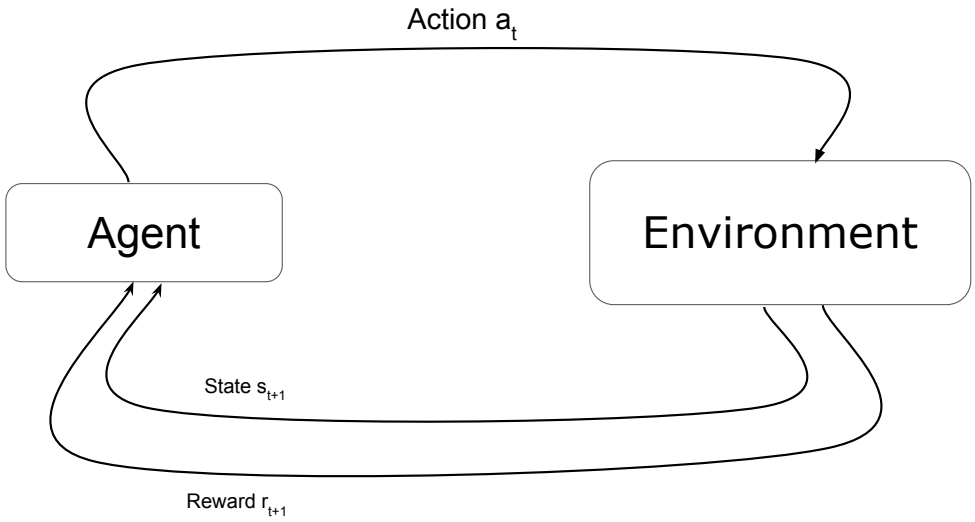


Figure 2.1: Reinforcement Learning Process

in RL are described by a batch of possible actions, a set of states or observations, a reward function that maps states or state-action pairs to feedback, and a transition function.

- $\mathcal{S}$ is the state space,
- $\mathcal{A}$ is the space of all possible actions,
- $\mathcal{P}(s_t, a, s_{t+1})$ is the transition function. The transition function is the probability of moving from a state s_t to a state s_{t+1} by taking an action a,
- $\mathcal{R}(s, a)$ is the reward function

The agent aims to maximize its *cumulative reward* also called *expected return* G. Rewards come in a sequence of $s_0 a_0 r_1$, represent the feedback in reinforcement learning, and they are the only way to let the agent knows if it is doing well.

In reinforcement learning to solve a problem, we need to define accurately and a priori what the states or observations, actions, and rewards or feedback should be. Each of these elements has a key role in the process of reinforcement learning, and our solution is dependent on these.

State and observation spaces. They are two very similar concepts. The state s refers to a complete description of the state of the world in the context that there is no hidden information.

For example, on a chessboard at every time-step we know the exact position of each unit. On the other hand, observation o describes a partially observed environment meaning that we don't have access to all information about our environment e.g. in the Sonic game, the agent sees only the environment that is close to him.

Reward. A reward is feedback given to the agent from the environment after taking action. Based on this, decides which actions should take next to maximize its cumulative reward at the end of each episode. It is a scalar, can be positive or negative, and can be delayed or instantaneously. The agent aims to maximize the cumulative reward i.e $G_t = R_{t+1} + R_{t+2} + \dots$. In general, future rewards are less important. Thus, we use a discount factor γ, where $\gamma \in 0, 1$, for future rewards. Eq.2.1 represents the discounted cumulative reward [27].

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

Actions and Action space. An action space refers to all possible actions the agent can take in an environment. There are two types of action spaces: i) Discrete action spaces and ii) Continuous action spaces. A discrete action space means that we have a finite number of actions e.g. on an Atari video game, there is a finite number of actions like "move left" and "move right". Additionally, there are multi-discrete action spaces. In this case, the multi-discrete action space consists of discrete action spaces individually with its number of actions. On the other hand, in continuous action spaces, there are an infinite number of actions e.g. self-driving car agents, where the agent can steer the wheel at different angles like 15^o or 9.6^o or 22.66^o. Well-defined action space is important, especially in cases where we need to build our customer environments.

The agent needs to learn to choose the best action to maximize the cumulative expected reward given a state. It does so by learning a policy π. A policy is a function that maps a given state to action or expected value. Our goal is to find the optimal policy π^* that maximizes the cumulative expected reward. There are two types of training: **Policy-based methods** and **Value-based methods**.

Policy-based methods. The agent learns a policy function that maps a state to the best action for the given state. There are two types of policy: deterministic and stochastic. Deterministic policy means that given a state, the policy always returns the best action, whereas a stochastic policy means

that a given state returns a probability distribution over actions.

Value-based methods. The agent learns a value function that maps each state to the expected value of being at this state. In this case, the policy assigns a value to each state, and the agent moves to the state with the highest value. The value for each state is defined in Eq.2.2.

$$v_\pi(s) = \mathcal{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \quad | \quad S_t = s] \tag{2.2}$$

Additionally, in value-based methods we have a function that maps a state-action pair to a value. The Eq.2.3 describes the expected return in the case the agent starts from the state *s* takes the action a and never changes his policy π.

$$Q_\pi(s, a) = \mathcal{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \quad | \quad S_t = s, A_t = a] \tag{2.3}$$

The difference between policy-based methods and value-based methods is that in the policy-based methods, we find the optimal policy by training the policy. On the other hand, in the second case, we are led to an optimal policy by finding the optimal value function or the so-called Q-value.

$$\pi^*(s) = \arg\max_\pi Q^*(s, a) \tag{2.4}$$

2.1.2 Q-learning

Q-learning is a *model-free* and *off-policy* reinforcement learning algorithm. *Model-free* means that the agent directly predicts the value of taking an action a in state s i.e the state-action pair's Q-value. In Q-learning, the agent uses a table that consists of all the Q-values for every state-action pair. The agent updates these Q-values based on Bellman's equation 2.5.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_\alpha Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.5}$$

An *off-policy* algorithm means that it uses a different policy to calculate Q-values and another to make decisions over actions. Thus, even if Q-learning will use the optimal action a to update the Q-values, it is possible to take a sub-optimal action a_{t+1} to explore the environment further. The Q-learning algorithm is described in Algorithm 1

Algorithm 1 Q-learning

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
for each episode **do**
    Initialize $\mathcal{S}$
    for each step in episode **do**
        $a = \pi(s)$ // Choose action a for state s using policy derived from Q
        Take action a
        Update Q-value using 2.5
        Update $\mathcal{S} \leftarrow \mathcal{S}'$
    end for
end for

Q-learning, compared to *SARSA* learns a more optimal solution faster, but it explores more than SARSA. Depending on the task we want to solve may not be desirable. One of the Q-learning downsides is that it does not scale well on large state spaces. To tackle this issue, we use neural networks to approximate the Q-values.

2.2 Deep Reinforcement Learning

In recent years, deep learning has experienced growth, mainly due to the progress in GPUs additionally to the increase in computing power. Additionally, the large volume of data we have access to in recent years also contributed to this. Deep learning has helped researchers develop machines that succeed in difficult tasks that previously humans outperformed them, such as natural language processing, speech recognition, and image recognition.

Classic reinforcement learning techniques suffer from the curse of dimensionality of the action and state space. Deep reinforcement learning (DRL) is the combination of neural networks with reinforcement learning to find an approximation of the Q-value the $Q_\theta(s, a)$. DeepMind developed the algorithm DQN [21] back in 2015. They used neural networks to approximate Q-values. The DQN comes from the Deep Q-Network and exploits the Q-learning by using neural networks.

2.2.1 Deep Q-learning

In contrast to Q-learning, which we explained in section 2.1.2 in DQN, we use neural networks to approximate Q-values. The DQN receives state s as input and outputs Q-values for all possible actions. We depict this difference in Fig.2.2. In deep Q-learning, we have experience stored in a replay memory database $\mathcal{D}$. The agent decides which action a_t to take based on the predicted Q-values, and receives a reward r_t and a new state s_{t+1}. The experience he gains from picking this action is stored in the replay memory database as a tuple of the form (s_t, a_t, r_t, s_{t+1}). Then, the agent uses a batch of these experiences to train the networks. We present the pseudo-code for the deep Q-learning in Algorithm 2.

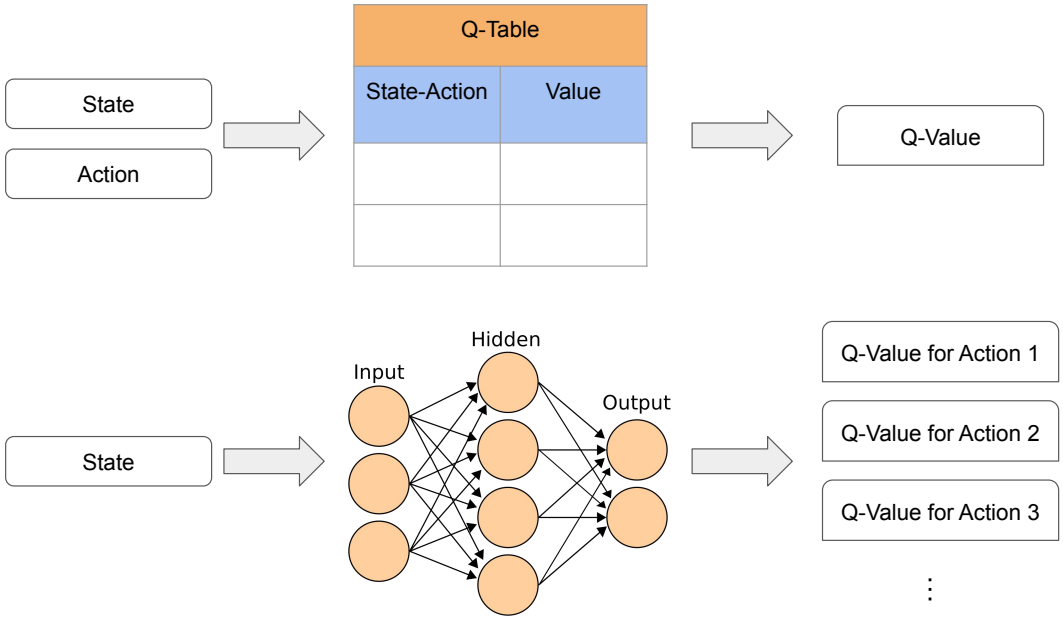


Figure 2.2: Difference between Q-learning and DQN

The DQN algorithm uses the loss function as defined below in Eq.2.6 which is the mean squared error of the difference in the predicted value, $Q(s, a; \theta_i)$ and the target value, $(r+\gamma \max_{a'} Q(s', a'; \theta_{i-1}))$. This difference also called "Temporal Difference (TD) Error".

$$Li(\theta_i) = E_{s,a\sim\rho(.)}\Big[\big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\big)^2\Big] \tag{2.6}$$

Algorithm 2 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function $\mathcal{Q}$ with random weights
for episode=1, M **do**
  Initialize sequence $s_1 = x_1$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a \mathcal{Q}^*(\phi(s_t, a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random mini-batch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} \mathcal{Q}(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_i - \mathcal{Q}(\phi_j, a_j; \theta))^2$ according to equation 2.7
  end for
end for

Moving Targets

An important issue for deep Q-learning is the non-stationary target values. Eq.2.7 defines the gradient used to update our weights. An important note here is that the predicted Q-value and the target Q-value use the same weights. Thus, while we update the weights for the predicted Q-value to drive it closer to the target Q-value, we also move the target Q-value with the result to chase the target value. The result is to have oscillations during the training phase.

$$\nabla_{\theta_i} Li(\theta_i) = E_{s,a\sim\rho(.);s'\sim\mathcal{E}}\Big[\Big(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)\Big)\nabla_{\theta_i} Q(s, a; \theta_i)\Big] \tag{2.7}$$

To deal with the "Moving Target" issue, Deepmind used two different networks, the main DQN and a second one called "Target Network" to update periodically every N step the weights for the target Q-value. The weights for the target Q-value remain fixed most of the time in training.

Experience Replay

An important feature of the DQN algorithm is the *experience replay* method to train itself. In *experience replay* the agent stores in replay memory $\mathcal{D}$ tuples of (s_t, a_t, r_t, s_{t+1}). On every step,

it samples a mini-batch of these from the replay memory $\mathcal{D}$ and uses them to update the weights of the DQN that predicts the Q-value. This method helps us deal with correlations between similar states that result in the same action regularly, and thus, the agent will not lurch in training.

Prioritized Experience Replay

In DQN, we use the experience replay mechanism to replay to the agent random past experiences periodically and prevent him from forgetting them. But, there would be experiences more important than others. If this is the case, we should prioritize them to be replayed.

Instead of randomly sampling experiences from the replay buffer, we sample using priorities. We define the *priority* as the magnitude of the temporal difference error. Eq.2.8 depicts the priority concept where ϵ is a constant to avoid zero probability. The idea behind prioritized experience replay is to focus on predictions that diverge a lot from the target.

$$p = |\delta| + \epsilon \tag{2.8}$$

To avoid replay experiences with the highest TD-error, prioritized experience replay adds some stochasticity computing a sampling probability for every experience using a Boltzmann distribution as displayed in eq.2.9.

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{2.9}$$

where $\alpha \in (0, 1)$ is the *temperature* parameter used to balance between completely greedy prioritization to uniform sampling.

2.2.2 Advantage Actor Critic

Advantage Actor-Critic (A2C) is a hybrid algorithm in reinforcement learning in the sense that it combines two different types of algorithms, policy-based and value-based. Policy-based algorithms mean that we train the policy to find the best action from a specific state, whereas value-based algorithms learn to select actions according to the predicted state or action value. Before we go deeper into A2C, we will briefly describe the Actor-Critic methods and Advantage Functions.

Actor-Critic Methods

Actor-Critic methods exploit the benefits of the policy-based and value-based algorithms by using two neural networks. One to quantify the value of the taken action (value-based) called Critic and another to return the best action for the given state (policy-based) called Actor.

The Actor controls how the agent behaves. It does so by taking as input a state and returning the best action for this state. On the other hand, the Critic evaluates the Actor's choice by computing the Q-value of the chosen action.

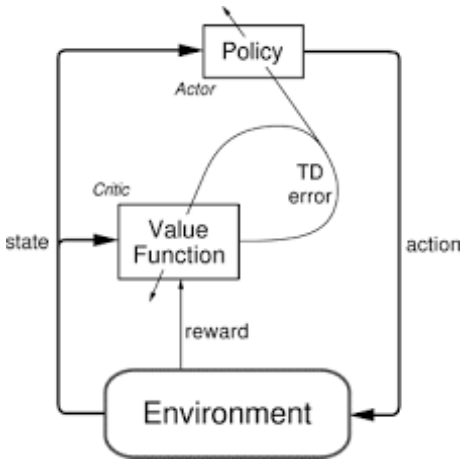


Figure 2.3: Actor-Critic Framework Richard S. Sutton, 1998

To give an example, consider yourself playing a game with a friend. You are the actor, and your friend is the critic. At the beginning of the game, both of you have no experience with what is a good action or a good state. So you start by taking random actions, and your friend gives you feedback. As you play, both gain experience. Your friend will learn to evaluate your actions better, and you will use this information to make better decisions over which actions to take based on the given state.

Advantage Functions

In contrast to Q-value for action a from a state s that tell us how good is to be to that state, *advantage function* describes how much better will be to take an action a from state s against a randomly chosen action according to the same policy $\pi(\cdot|s)$ if we stick to this policy forever. The Eq.2.10 defines the

advantage functions.

$$A^{\pi}(s,\alpha) = Q^{\pi}(s,\alpha) - V^{\pi}(s) \tag{2.10}$$

The A2C architecture is the same as in the Actor-Critic method. Given a state s from the environment, the actor outputs the policy π_θ for this state. The policy π_θ is a vector of probabilities for each action. The Critic, on the other hand, returns the value of that state V_s. Additionally, the A2C leverages advantage functions to deal with the high variability. We describe the Advantage Actor-Critic in the Algorithm 3.

Algorithm 3 Advantage Actor-Critic learner

Initialize the actor π_{theta} and the critic V with random weights.
Observe the initial state s_0
for $t = 0, T_{total}$ **do**
  Initialize empty episode minibatch
  for $\kappa = 0, n$ **do**
    Select a action a_κ using the actor π_{theta}.
    Perform the action a_κ and observe the next state $s_{\kappa+1}$ and the reward $r_{\kappa+1}$.
    Store $(s_\kappa, a_\kappa, r_{\kappa+1})$ in the episode minibatch
  end for
  if $s_n \quad is \quad not \quad terminal$ **then**
    Set $R = V_(s_n)$ with the critic
  else
    $R = 0$
  end if
  Reset gradient $d\theta$ and $d\phi$ to 0
  for $\kappa = n-1, 0$ **do** // Backwards iteration over the episode
    Update $R = \sum_\kappa r_\kappa + \gamma R$
    Accumulate the policy gradient using the critic:

$$d\theta \leftarrow d\theta + \nabla_\theta log \pi_\theta(s_\kappa, a_\kappa)(R - V(s_\kappa)) \tag{2.11}$$

    Accumulate the critic gradient:

$$d\phi \leftarrow d\phi + \nabla_\phi (R - V(s_\kappa))^2 \tag{2.12}$$

  end for
  Update the actor and critic with the accumulated gradients using gradient descent or similar:

$$\theta \leftarrow \theta + \eta d\theta \quad \phi \leftarrow \phi + \eta d\phi \tag{2.13}$$

end for

2.2.3 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy gradient method introduced in 2017 [24]. As we have mentioned before, a policy is a function that maps states to actions or expected values. Policy gradient methods rely on optimizing parameterized policies concerning the expected return by gradient descent.

PPO, as compared to Trust-Region Policy Optimization (TRPO), is easier to implement, has a more simple clipped surrogate objective function, and uses only a first-order optimization. Even though its simple objective function, PPO has better sample efficiency in many control tasks than TRPO.

PPO compared against several methods that are considered to be efficient for continuous problems and outperformed them on almost every continuous control environment [24]. In addition, it performed well in Arcade Learning Environments, including Atari games, against fine-tuned implementations of A2C and ACER.

As a policy gradient method PPO tries to ensure that the updated policy does not converge too much from the old policy. This is done by clipping the update region to a small range of values as is shown in Eq.2.14.

$$L^{CLIP}(\theta) = \hat{E}_t\Big[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t\Big] \tag{2.14}$$

Importance Sampling

The term $r_t(\theta)$ is the probability ratio, same as in TRPO also called *importance sampling*, is defined in Eq.2.15 and computes how far away are the two policies. So in PPO, we have two policies the new one, which we want to predict, and the old one, which we use to collect data. *importance sampling* helps us to evaluate the predicted policy with the collected samples from the old one and to improve sample efficiency.

$$r_t(\theta) = \frac{\pi_\theta(\alpha_t|s_t)}{\pi_{\theta_{old}}(\alpha_t|s_t)} \tag{2.15}$$

As we saw in Eq.2.14 by using this ratio, we clip the estimated advantage in case the new policy diverges a lot from the old one. Thus, for an action that is close to the new policy, we clip the objective function to prevent overdoing it. On the other hand, if an action is more likely under the old policy, the objective function is flattened out to avoid doing the update again.

Below we represent the algorithm for the PPO with Actor-Critic style as is written in [24]. As we can see, the old policy will run for a small batch of observations and then we will apply the surrogate loss to this mini-batch of samples. Finally, we update the old policy with the new one.

Algorithm 4 PPO, Actor-Critic Style

for $iteration = 1, 2, \ldots$ **do**
  for $actor = 1, 2, \ldots$ **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T time-steps
    Compute advantage estimates $\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L w.r.t θ, with K epochs and mini-batch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

2.2.4 Ensemble Learning

Ensemble methods aim to enhance the model's final performance. In supervised learning, we use ensemble methods such as bagging, boosting, and stacking. These are leveraged to combine multiple preferably diverse classifiers using a voting scheme. On the other hand, in RL, ensemble methods represent and learn the value function. Known ensemble methods in RL are Majority Voting, Rank Voting, Mixture Model, Boltzmann Multiplication, and Boltzmann Addition.

Ensemble methods in RL have been used in stock trading [37] on maze tasks [36], and on classical control tasks [6] [15]. In [37] Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C), and Deep-Deterministic Policy Gradient (DDPG) combined to develop an ensemble method to make predictions on stock trading. They validate each algorithm on a 3-month validation rolling window. After validation, they pick the algorithm with the best Sharpe Ratio to make predictions and trade on the next quarter.

In [36] they combined *Q-learning*, *Sarsa*, *Actor-Critic*, *QV-learning*, and *Actor-Critic Learning*

Automata in four different ensemble methods *Majority Voting (MV)*, *Rank Voting (RV)*, *Boltzmann Multiplication (BM)*, and *Boltzmann Addition (BA)*. In [6] algorithms like *Deep Q-Networks*, *Deep Sarsa*, *Double Deep Q-Networks* have been combined. Ensemble methods that used were Majority Voting, Rank Voting, Boltzmann Multiplication and Boltzmann Addition. Below, we present the mentioned ensemble methods as described in [36].

Majority Voting

The MV method calculates the preference values of n different RL algorithms as shown below:

$$p_t(s_t, a[i]) = \sum_{j=1}^{n} I(a[i], a_t^j) \tag{2.16}$$

where $I(x, y)$ is the indicator function that returns 1 if $x = y$ and 0 otherwise. The predicted action is the most voted action across the agent. To leave space for exploration, we do not always select the most preferred action.

Rank Voting

The RV method assigns weights on the action probabilities based on action rank. Consequently, the most probable action could be weighted m times, where m is the total number of action, the second most probable $m - 1$ and so on. The preference values are given by:

$$p_t(s_t, a[i]) = \sum_{j} \pi_t^j(a[i]) \tag{2.17}$$

Boltzmann Multiplication & Addition

In BM, we multiply all the action probabilities we receive from the policies. The ensemble produces preference values given by

$$p_t(s_t, a[i]) = \prod_{j} \pi_t^j(s_t, a[i]) \tag{2.18}$$

A possible issue here is that in the case an agent predicts zero probably for an action, then it will set the preference value for this action to zero. The last method is the BA. In BA, we sum the action probabilities, and the preference values are set by

$$p_t(s_t, a[i]) = \sum_j \pi_t^j(s_t, a[i]) \tag{2.19}$$

In the presented work, we use a slightly different versions of the MV, RV, and BM methods. Our methods are presented in the section 5.

Chapter 3

Related Work

We focus our study on applications of deep reinforcement learning methods in traffic systems control. We identify different aspects that previous studies differ from each other. For example, we categorize previous works regarding the type of intersection each method applied. Additionally, they differ in the definition of the state, action, and reward of the system and use different neural network architectures.

3.1 States

In Reinforcement Learning we need to represent accurately the state of the system. The more accurate the representation the better learning will have in the system. Consequently, many different state representations have been used in traffic signal control.

In [10] state is represented as a raw RGB image which is fed onto a Deep Neural Network consisting of Convolutional Networks. The same representation is used in [22] in which the state is the current view of the intersection in the graphical simulator and is represented as a vector of raw pixel values. Additionally, in [17] they use a photo of the controlled intersection. In [31] we come across to the *cell occupancy* state representation. In this case, each incoming lane is separated into multiple cells. For each cell, we assign the value 1 if it is occupied by a vehicle otherwise we set it to 0. This kind of representation almost ensembles an image data input. [29] and [26] use the discrete traffic state encoding (DTSE) method to represent the state. The DTSE method is another

image-like state representation.

Another type of state representation is feature-based form. In [23] queue length and the average velocity on the lane are used to form the feature-based representation. Additionally, in [7] the state representation is a combination of the cumulative delay for the first vehicle and the total number of approaching vehicles. In [5] the state is represented as a vector with values of the traffic condition i.e. the number of vehicles in the intersection (or pressure as defined in this work) and the current traffic signal phase which is a 12 element vector.

In [35] state is represented as a combination of a feature vector containing information of queue length, number of vehicles, and updated waiting time and a discrete traffic state encoding using an image of vehicles' position, current phase, and next phase.

3.2 Actions

The state representation is important for the system to decide what action should take next. A common intersection consists of four approaches. An approach is referred to as a roadway that leads to an intersection. The part of the roadway that brings vehicles to the intersection is called the incoming approach and the part of the roadway that leads the vehicles away from the intersection is called the outgoing approach. Each approach is controlled by green, yellow and red movement signals. For a single intersection, the most common action selections are: (1) choosing one of the possible green phases and (2) the binary selection of keeping the same phase or not.

Intersections in the real world usually have the following possible green phases: North-South Green (NSG), East-West Green (EWG), North-South Advance Left-Green (NSLG), and East-West Advanced Left-Green (EWLG). When we set the action selection to choose one of the possible green phases the agent will select one of these four green phases at each time t [5]. This phase will apply to the system after the yellow and red transitions. Yellow and red phase definitions are also important for successful agent learning and safety traffic [13]. In some cases to make things simpler only two green phases were considered for the experiments [29] [16] [22].

In case we have set the binary selection action the agent will decide whether or not it will keep the current green phase the intervals are pre-defined and at each time t [35] [10] [23] [26] [31].

In addition, the agent will change to the next phase in a pre-fixed cycle. If the agent decides to change the phase then the yellow and the red transition will occur before the change to secure a safe traffic flow [13]. A slightly modified version of the binary selection action we find in [17] where the team uses a high dimension Markov Decision Process to decide how to update the phase in the next cycle. In this case, the agent decides to keep the same phase modified by ±5secs or switch to another phase. In [7] they use a predefined set of all possible actions for each intersection and the agent chooses one of them for a pre-fixed interval.

3.3 Rewards

While states are usually represented as feature vectors or matrices and actions most commonly as discrete values reward regardless of the problem we try to tackle are always scalar values. In our case rewards are scalar values related to the traffic data. Some common types of rewards we found in the literature are *waiting time*, *queue length*, and *cumulative delay*.

Some studies use only one feature as a reward. In [31] and [22] an instant reward is generated for each state-action thus, it represents the *accumulated system delay between two actions*. Similar to this we find in [17] the *cumulative waiting difference between two cycles*. In [26] [23] the *total waiting time* for all vehicles is considered as reward for the system. Whenever we use *delay* or *waiting time* of vehicles the reward is the negative value of this feature. Thus, the agent tries to maximize it, meaning that its goal is to get it as close to zero as possible. A more unusual reward is defined in [5]. They use the *pressure* on the intersection meaning that they take as a reward the difference between the number of vehicles waiting in all the incoming lanes and the number of vehicles moving in the outgoing lanes. A very simple reward of $+1$ for every vehicle passed through the intersection we find in [10].

On the other hand, there are cases where multiple features are used to define the system reward. In [7] reward is a combination of the *queue length along each incoming lane* plus the *wait time for each vehicle*. In [16] along with *queue length* they also use as reward the *traffic delay*. [35] uses a more complicated reward. They use the weighted sum of the *queue length*, the sum of the *delay* over all the lanes, the sum of the *updated waiting time* overall approaching lanes, the *indicator of*

light switches, the *total number of vehicles* passed the intersection, *total travel time* of vehicles. In [29] we find another weighted sum of the *crashes and/or jams* in the simulator, the *emergency stops* and whether or not the phase was changed.

3.4 Simulation Environments

In reinforcement learning, we are using environments to train our agents. Environments help researchers to conduct multiple experiments in parallel if needed multiple times without the risk of destroying the equipment. Over time scientists have developed software to simulate these environments. Related to the TLCSs there have been developed multiple simulation environments. Most studies use the open-source traffic simulator Simulation Urban MObility (SUMO) [2]. SUMO provides an interactive environment and researchers using Python can handle and modify the traffic circumstances. Another well known simulator is PARAMICS [4]. [9] is a simulator for those who prefer MATLAB to interact with the simulation. In [10] a custom simulator is built using Unity3d. Recently, in [35] they used the Cityflow [38] which is supposed to be faster than SUMO, especially in large scale scenarios.

In Table3.4 we summarize all previous works related to traffic signal control research. We summarize the state representation, the type of reward they used in each paper, the simulation environment, and the type of data they use.

Table 3.1: Traffic Signal Control Review

Papers	State Representation	Reward	Simulation Environment	Data
[10]	RGB images	+1 for every vehicle passed through the intersection	Custom Traffic Simulator - Unity3d	Synthetic

[22]	RGB images	accumulated system delay between two actions	SUMO	Synthetic
[17]	RGB images	cumulative waiting difference between two cycles	SUMO	Synthetic
[31]	Cell Occupancy	accumulated system delay between two actions	VISSIM	Synthetic
[26]	Discrete Traffic State Encoding (DTSE)	total waiting time	SUMO	Synthetic
[29]	Discrete Traffic State Encoding (DTSE)	the weighted sum of the crashes and/or jams in the simulator, the emergency stops, whether or not the phase was changed	SUMO	Synthetic
[23]	Feature-based (queue length, avg velocity)	total waiting time	SUMO	Synthetic

[7]	Feature-based (cumulative delay for the first vehicle, total number of approaching vehicles)	queue length along each incoming lane, wait time for each vehicle	SUMO	Synthetic
[5]	Feature-based (signal pressure, current traffic signal phase)	pressure	CityFlow	Synthetic & Real-world Data
[35]	Feature-based (queue length, number of vehicles, updated waiting time, DTSE uses an image of vehicles' position, current phase, next phase)	the weighted sum of the queue length, the sum of the delay over all the lanes, the sum of the updated waiting time overall approaching lanes, the indicator of light switches, the total number of vehicles passed the intersection, the total travel time of vehicles	SUMO	Synthetic & Real-world Data
[16]		queue length, traffic delay	PARAMICS	Synthetic

Chapter 4

Data

In the last few years, there has been an increase in traffic data we collect from different sources. Traffic surveillance cameras and loop induction sensors are some of the sensors used to monitor real-time traffic conditions. There are several sources[1] [2] in which we can find real-world traffic datasets. Despite of the existence of these real-world datasets it is common in the literature to start the experiments with synthetic data that comes primarily from the simulation environment.

In this work, we use the Simulation of Urban MObility (SUMO) to conduct all our experiments. SUMO provides a very handful API and a GUI view to model your traffic scenarios. We can develop any type road-network structure either manually or by using OpenStreetMap to build real-world road networks. Additionally, we can define our traffic light plans or load pre-defined based on real-world traffic scenarios.

We conduct all of our experiments on a single intersection with four approaches with directions $\mathcal{N} \rightleftharpoons \mathcal{S}$ and $\mathcal{E} \rightleftharpoons \mathcal{W}$. On each approach, we incorporate two lanes, and the possible traffic movements are left-turn, through, and right turn. U-turns are omitted everywhere on the road-network and right turns are optional (see Fig.4.1).

The total length of each approach is $300\,m$ and the intersection is managed by a single traffic light. The traffic light plan is as follows: green phase correspond to $\mathcal{N} \rightleftharpoons \mathcal{S}$ with optional right turns lasts $33\,s$ it is represented as $GGrrrrGGrrrr$ and followed by a yellow phase of $2\,s$ with

[1] https://traffic-signal-control.github.io/#open-datasets
[2] https://sumo.dlr.de/docs/Data/Scenarios.html

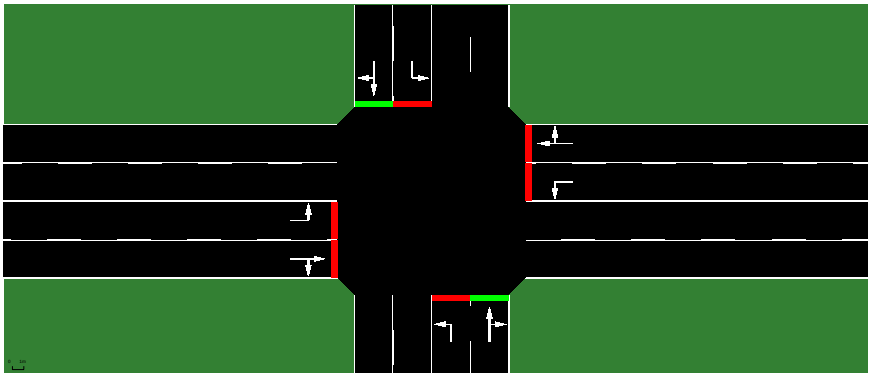


Figure 4.1: Traffic Movements in a single intersection consisting of two lanes per approach.

representation $yyrrrryyrrrr$ and a red phase of $6\,s$ represented by the sequence $rrGrrrrrGrrr$. During the red phase we have a green phase for left-turns on directions $\mathcal{N} \rightleftharpoons \mathcal{E}$ and $\mathcal{S} \rightleftharpoons \mathcal{W}$. Then, a green phase in the direction of $\mathcal{W} \rightleftharpoons \mathcal{E}$ is activated with duration $33\,s$ followed again by the same yellow and red phases. The total cycle traffic plan is $86\,s$.

In SUMO, we can also modify the demand data. The demand data correspond to how many vehicles drive through the network over time. Each vehicle in SUMO relates to a route. Each route defines the starting position, the end position, and all the edges the vehicle will go through. We can also configure the time step at which each vehicle will appear in the network, and thus, we can define dynamic traffic conditions during rush hours during the day. Demand data is generated artificially. We define 12 different *route types* for each direction e.g. for $\mathcal{N}$ we create three routes, one for each of the rest three directions (and we repeat the same process for the rest directions). Then, we configure the traffic volume per route. We set a range from $50 \quad to \quad 350 \quad vehicles \quad per \quad hour$ for each route in the road network. The total simulation time is $100000\,s$ and at every $25000\,s$ we change the traffic conditions in each direction.

Chapter 5

Methodology

Reinforcement learning is a repeatable process in which an agent tries to learn how to act accordingly to an environment. The agent relies on the interaction with the environment to learn a policy that maps states or states-action pairs to an expected value. Classical RL algorithms do not scale with large state and action spaces. To overcome this issue, we use neural networks to approximate value functions using nonlinear representations. Due to the nature of the problems RL tries to solve, we use digital environments to simulate our solutions. This way we save money, time and of course equipment. In specfic cases like traffic simulation, we work on safety without the risk of hurting people if the solution doesn't work fine.

Simulation Environment

In this research we used the *SUMO-RL* [1] framework to conduct our experiments. SUMO-RL provides an interface to create reinforcement learning environments with SUMO [2] for Traffic Signal Control (TSC). Moreover, it provides ready to use road-networks with configuration for demand data and traffic lights plans.

We conducted experiments on a single intersection with four approaches with directions $\mathcal{N} \rightleftharpoons \mathcal{S}$ and $\mathcal{W} \rightleftharpoons \mathcal{E}$. Each approach consists of two lanes which means that we have eight incoming and eight outgoing lanes. Traffic movement signals are defined as shown in Fig.5.1. The demand data have been described in chapter 4.

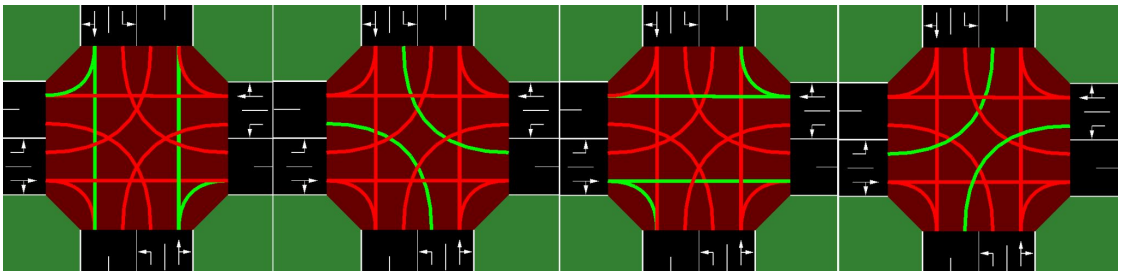


Figure 5.1: Possible Actions in a Single Intersection. Each signal phase permits non-contradicting movements e.g. $\mathcal{N} \rightarrow \mathcal{S}$ with right turn and the non-contradicting direction to $\mathcal{S} \rightarrow \mathcal{N}$ with right turn.

States - Observations

The observation space is a vector on $R^{\#greenPhases+2*\#lanes}$, with state representation $obs = [phase\_one\_hot, min\_green, lane\_1\_density, ..., lane\_n\_dens, lane\_1\_queue, ..., lane\_n\_q]$. The action space is defined as a 4 Discrete action space. Each action defines which green phase is going to be open for the next $delta\_time(dt)$ seconds which by default is $5\,s$. Additionally, we define a $yellow\_dt$ equal to $2\,s$ a min_dt for green phase and a max_dt. The max_dt is defined to avoid continuously select the same action. The total running time in the simulation is set to $100000\,s$. A $dt = 5\,s$ means that a time-step in training phase corresponds to $5\,s$ of simulation.

Rewards

We define a reward function as the change in cumulative vehicle delay $r_{t+1} = D_{a_{t+1}} - D_{a_t}$. The cumulative vehicle delay equals the vehicles' total waiting time over all the lanes. Besides rewards, we keep track of total waiting time in the road network for all vehicles. We aim to minimize the

time delay resulting in the agent receiving a negative reward. But the agent's goal is to maximize its rewards, thus minimizing the time delay.

Algorithms' Configuration

To conduct our experiments, we use three state-of-the-art algorithms in Reinforcement Learning. Previous works on traffic light management have been used Deep Q-Network [29] [35] [16] [5] and reported a better performance over baseline agents such as Fixed-time. In addition, DQN from literature is a well-known algorithm for its great performance in several games. We also use A2C and PPO algorithms. PPO relies on the A2C algorithm. Additionally, PPO has better sample efficiency than TRPO and a good performance on Arcade Learning Environments.

The SUMO configuration for all simulations is the same. Thus, at each $25000\,s$ we change the direction from $\mathcal{N} \rightleftharpoons \mathcal{S}$ to $\mathcal{W} \rightleftharpoons \mathcal{E}$ and vice versa on the traffic volume. Additionally, we use a $sumo - seed = 42$ in evaluation thus all algorithms will have the same simulation conditions. For PPO and A2C, we also use *vectorized environments* which means that we stack multiple independent environments into a single environment. Consequently, an RL agent is trained on n environments per step instead of one. Because of this, actions passed to the environment are now a vector (of dimension n). It is the same for observations, rewards, and end of episode signals.

We initiate our experiments by training each algorithm with default hyper-parameters on the single intersection for 110000 time-steps, which correspond to 5 simulations of $100000\,s$, and then we evaluate each algorithm for five more episodes on simulation. PPO outperforms the other two algorithms, seems more stable, and adapts well to traffic changes.

Next, we train each algorithm for $1.5\,million$ time-steps, which correspond to 74 simulations of $100,000\,s$ with slightly changed hyperparameters as shown in Table5.1 and evaluate each algorithm for five episodes. In this case, PPO performs again better than others, but we see progress on the other two.

Finally, we train only the DQN algorithm for $2\,million$ time-steps with tuned hyperparameters and the A2C for $1.5\,million$ time-steps with Adam optimizer and evaluate them for 5 episodes. Settings are shown in Table5.2.

Table 5.1: Settings for A2C, DQN, and PPO

A2C		DQN		PPO	
Hyperparameter	**Value**	**Hyperparameter**	**Value**	**Hyperparameter**	**Value**
Learning Rate α	0.001	Learning Rate α	0.001	Learning Rate α	0.001
Discount Factor γ	0.99	Discount Factor γ	0.99	Discount Factor γ	0.99
Optimizer	RMS	Batch Size	32	Batch Size	64
Normalized Advantages	False	Exploration Rate	0.01		
		Replay Memory Size	50000		

Table 5.2: Settings for A2C, DQN

A2C	
Hyperparameter	**Value**
Learning Rate α	0.001
Discount Factor γ	0.99
Optimizer	Adam
Normalized Advantages	False

DQN	
Hyperparameter	**Value**
Learning Rate α	0.001
Discount Factor γ	0.99
Batch Size	32
Exploration Rate	0.005
Replay Memory Size	50000

Average Voting

Because DQN and A2C are weaker than PPO, we combined their policies to build an ensemble method for better predictions. We pick the A2C with Adam optimizer version, and for the DQN, we choose the one trained for $2\,million$ time-steps. Our first approach is an average voting scheme and uses *Boltzmann Probabilities* given by the eq.5.1 below,

$$\pi_t(s_t, a[i]) = \frac{e^{p_t(s_t,a[i])/\tau}}{\sum_k e^{p_t(s_t,a[k])/\tau}} \tag{5.1}$$

where τ is a temperature parameter, and p_t is the probability distribution from each algorithm. The derived policy from the average voting scheme is to take for each action the average of the action probabilities we get from the agents for a state s and select the action with the highest average vote.

Transformed Rank Voting

Next, we use a transformed *rank voting* ensemble as shown in Fig5.2. We feed the agents with a state-reward pair (r_t, s_t) and each algorithm calculates q-values for each action. Then, we provide these values to the ensemble method, stack them on an array, and pick the maximum probability

from each column where each column corresponds to an action. We get an array of maximum probabilities for each action across the agents. Finally, the predicted action is the one with the highest probability.

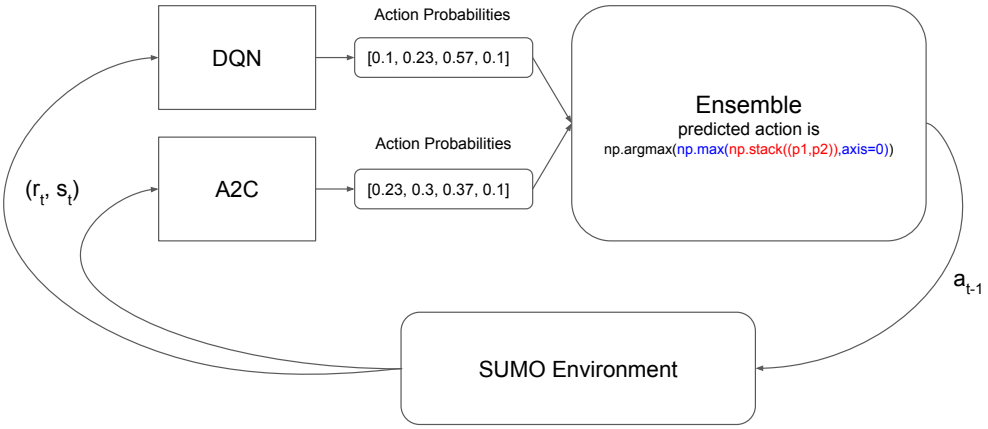


Figure 5.2: Transformed Rank Voting. The predicted action is the action with highest q-value.

Soft Voting

Next, we work with a *soft voting* scheme. In this case, we provide to the agents a pair of (r_t, s_t) and we sum up the predicted action probabilities from the agents. We then decide the action with the highest value. The soft voting scheme is shown below in Fig.5.3

In this case, we leave enough space for exploration. As we show in Fig.5.3 in cases where the agents are not very confident about the best action to select, the predicted action might be an action that is not the best action per agent.

Majority Voting

Majority voting is a standard ensemble method used in supervised learning and, as we mentioned before, in reinforcement learning. In supervised learning, the MV method comes in two ways the

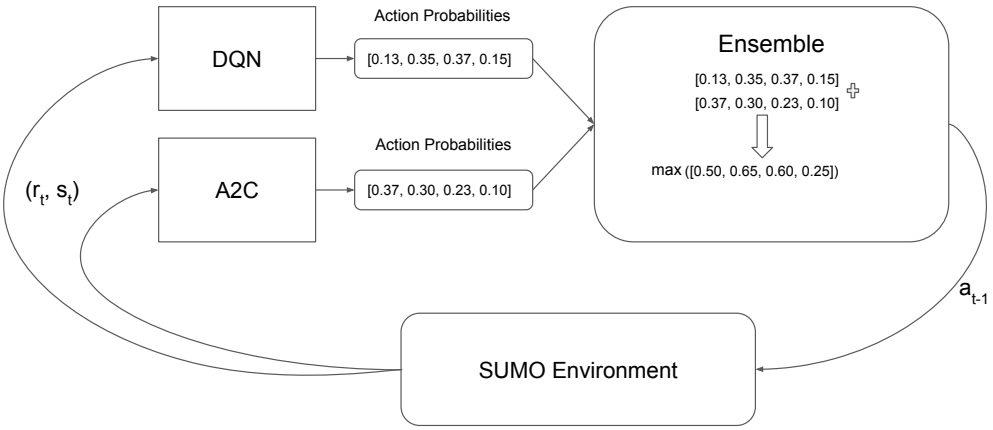


Figure 5.3: Soft Voting Scheme

hard voting and *soft voting*. We describe the latter above. We implement the MV method in a hybrid way because we use only two agents. To exploit the experience from the learned policies, we choose the most voted action across the agents (hard voting). But, when the agents predict different best actions we use the soft voting scheme to select one of them.

On the other hand, when we want to follow a less greedy strategy and explore the environment, we use action probabilities to randomly select an action from each agent and then to determine an action we follow the above approach.

Chapter 6

Results and Discussions

In this chapter, we discuss the results of our experiments. As we have already mentioned, we conduct our experiments on SUMO in a single intersection environment. Initiating our research, we use a DQN, an A2C, and a PPO, all with default hyperparameters. We evaluate each of them for five episodes in SUMO with episode duration $100,000\,s$. We use a moving average of $100\,s$ and plot the average of the total waiting time over all episodes. Results are shown below in Fig.6.1

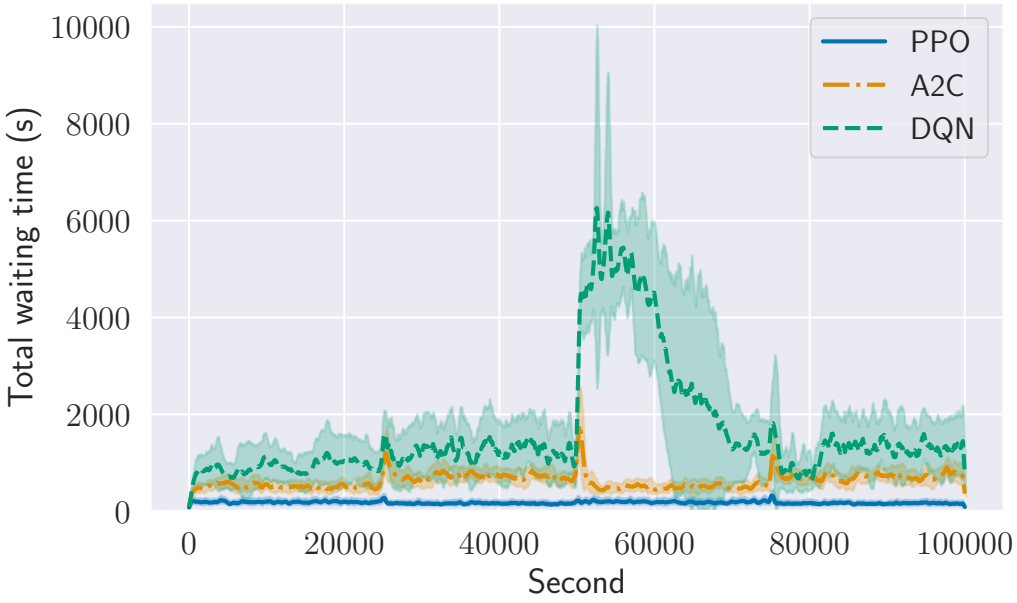


Figure 6.1: Comparing PPO, A2C and DQN on single intersection

A reminder here is that at every $25,000\,s$ we modify the traffic conditions. Thus, from $0 - 25,000$ the traffic demand is increased on the direction of the $\mathcal{N} \rightleftharpoons \mathcal{S}$ and again from $50,000 - 75,000\,s$. PPO performs way better than DQN and A2C. It seems to adapt easier to the traffic conditions and does not oscillates. On the other hand, DQN has the worst performance. We notice a high pick at around $50,000\,s$ when the traffic demand changes for the second time, and it is actually like restarting the simulation. There are a lot of wiggles and oscillations. A2C has a better performance than DQN. It presents picks when the traffic volume changes directions, but it recovers fast enough and keeps the total waiting time low.

In Fig.6.2 we plot the average rewards over the 5 episodes. By comparing the average rewards for each algorithm, we see that they have almost the same performance. DQN is again the more unstable, but it maximizes its rewards. It is important to keep in mind that comparing only the rewards is not always the best way to decide on which agent is better. In our case, even if all algorithms have maximized their rewards, they fail in the ultimate goal, to minimize the total waiting time.

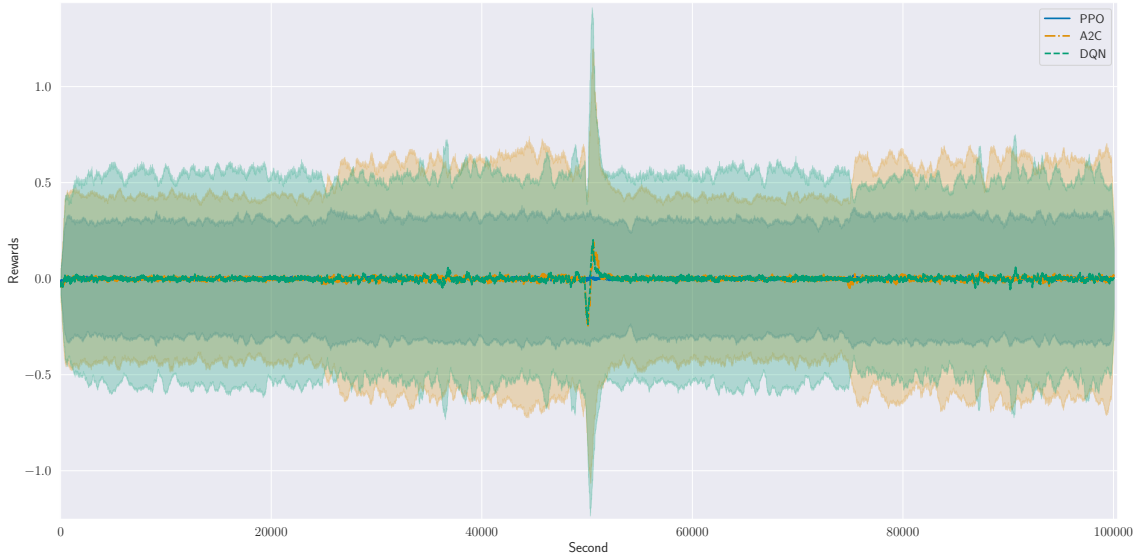


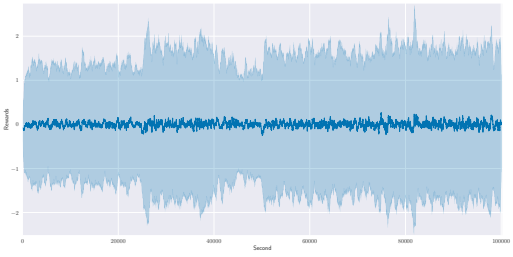
Figure 6.2: Comparison of rewards for PPO, A2C and DQN on single intersection

6.1 Ensemble Methods Results

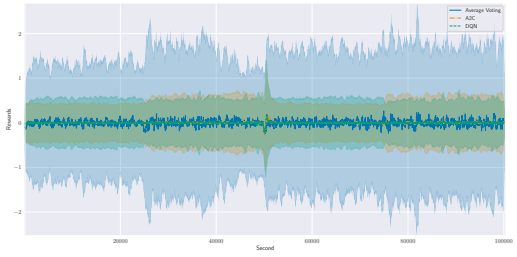
Ensemble methods in supervised learning are a promising approach to dealing with classification problems. Inspired by the *wisdom of the crowd* they show better accuracy and faster learning compared to each contributor algorithm. In RL, ensemble methods mostly combine learned policies to predict an action. The environment gives the ensemble an observation o_t, and each agent that composes the ensemble makes a prediction. All agents calculate a probability distribution for every possible action. In the following parts, we discuss the results from the ensemble of the two weaker algorithms, which are A2C and DQN.

6.1.1 Average Voting Ensemble - Results

In the average voting ensemble method, we use Boltzmann probabilities. We combine probabilities distributions given by the A2C and DQN to predict the best action. The average voting method regarding the rewards performs very well, as shown in Fig.6.3. Comparing our ensemble to each algorithm separately, we see that our method wiggles more and it appears to have more variance across all episodes (see Fig.6.3b.



(a) Average rewards per episode for Average Voting method



(b) Average Voting method rewards compared to A2C and DQN rewards

Figure 6.3

But, concerning the total waiting time, the ensemble method performance is poor. In Fig.6.4 we see that the ensemble method suffers from the changes in traffic conditions and does not adapt to them. At $50,000\,s$ when the system returns to its initial conditions, the ensemble climbs close to $3,800\,s$ in total waiting time.

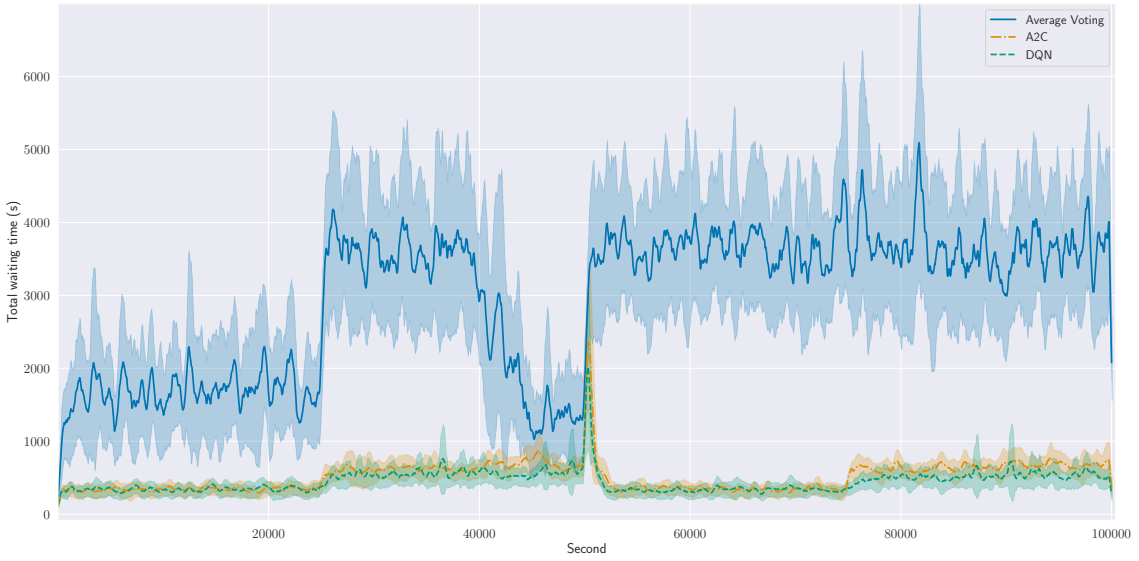


Figure 6.4: Average Voting performance compared to A2C and DQN.

6.1.2 Soft Voting - Results

We move on with a soft voting scheme. In this case, all agents provide the ensemble with a probability distribution for every action. We stack them and calculate the sum of action probabilities. A note here is that after the addition we do not refer to probabilities anymore but rather to ”weights”. Finally, we pick the action with the largest weight. Thus, the ensemble predicts the action with the biggest cumulative weight across the agents. This approach uses more exploration than the others as the selected action it is possible to be the second-best action for the agents.

Figure 6.5: Soft Voting Ensemble compared to A2C and DQN.

Fig.6.5 depicts the performance of our ensemble compared to the other two algorithms. It seems that the ensemble method performs better regarding the total waiting time. It is more stable in the intervals where the traffic conditions are constant and adapts better to the traffic changes (see at $t \simeq 50,000\,s$ when the system returns to its initial conditions). The ensemble predicts actions that handle better the traffic demand across all the lanes, which means shorter traffic delays. In addition, in Fig6.6 we plot the average rewards over all episodes. Our method maximizes its rewards instantaneously, and so do its component agents.

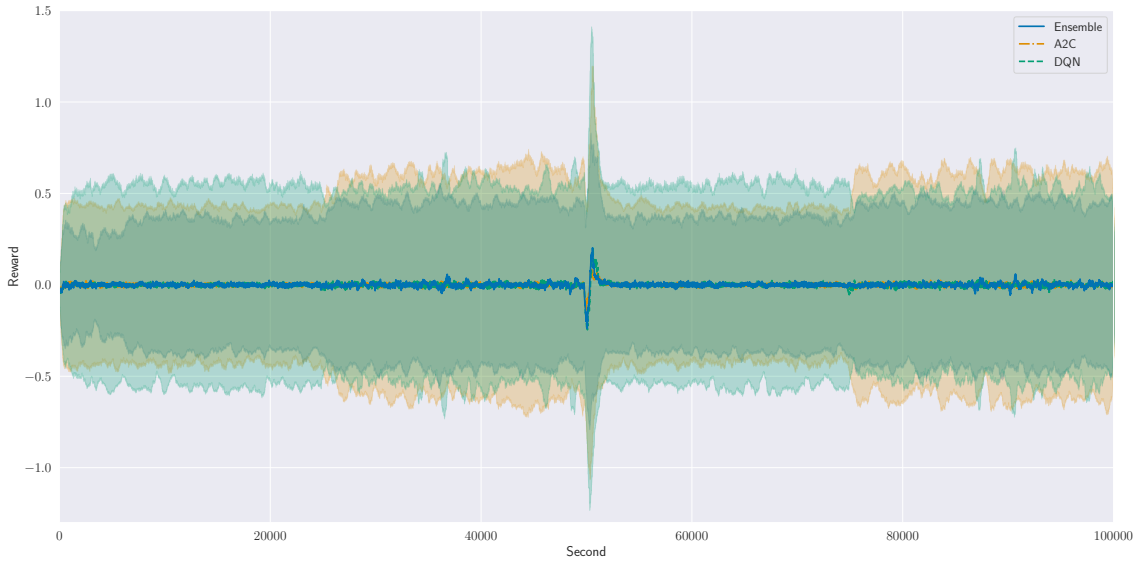


Figure 6.6: Soft Voting Rewards on a single intersection.

6.1.3 Transformed Rank Voting - Results

Additional to the previous ensembles, we experiment with a transformed rank voting. In this case, the ensemble takes in action probabilities from the agents, stacks them, and creates a new vector of weights instead by taking the maximum probability for each action across the agents. Then, the ensemble predicts the action with the largest weight. With this approach, we do not leave enough space for exploration, but we follow a more greedy strategy and exploit the learned policies by selecting more often the most confident actions. Fig6.7 represents how the algorithms A2C, DQN, and our new ensemble method perform on a single intersection.

The ensemble method takes action that handles pretty fine the traffic conditions most of the time. But, there is an interval where the ensemble fails to adapt to the traffic changes. In this interval, it seems the ensemble does more exploration than exploitation.

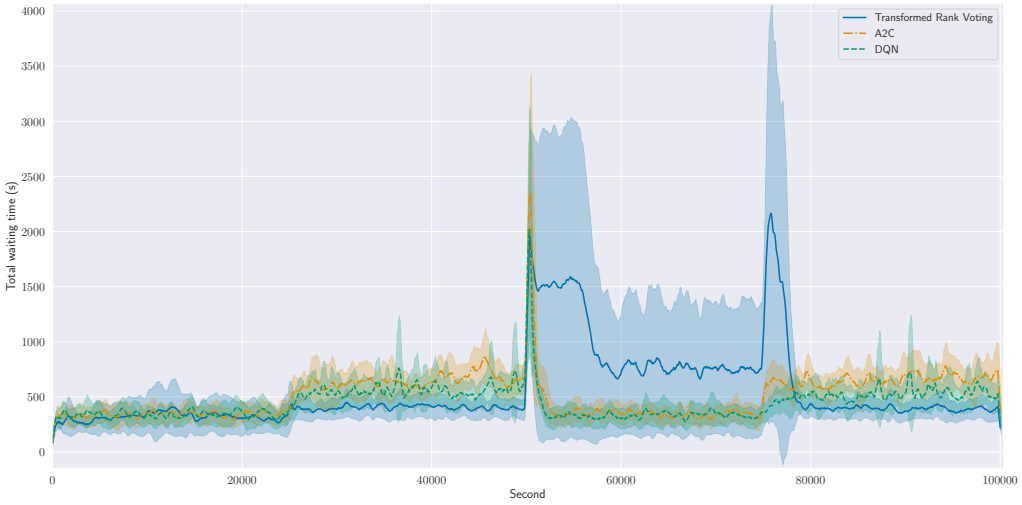


Figure 6.7: Transformed Rank Voting on single intersection

By comparing the soft voting and the transformed rank voting methods (Fig.6.8 we see that the soft voting scheme is less prone to traffic demand changes. It seems that recovers faster after modifications on traffic and keeps the total waiting time at decent levels.

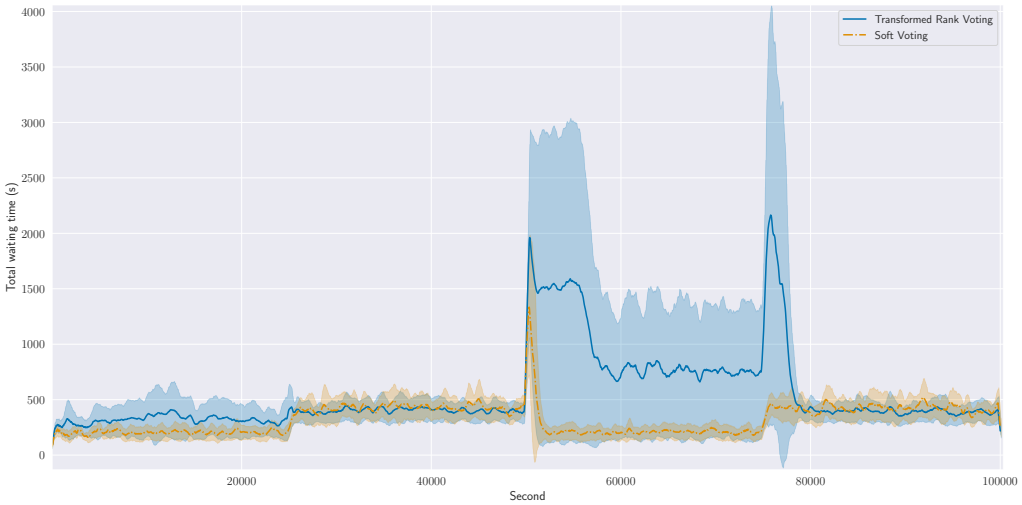


Figure 6.8: Soft Voting compared to Transformed Rank Voting

Furthermore, we also try to create an ensemble where the A2C agent uses the RMS optimizer. By comparing these two ensembles, the one with the Adam and the other with the RMS optimizer, we notice that when the A2C uses the RMS optimizer this ensemble does not perform well. The graphs look similar, but the blue line is frequently on higher values of total time delay.

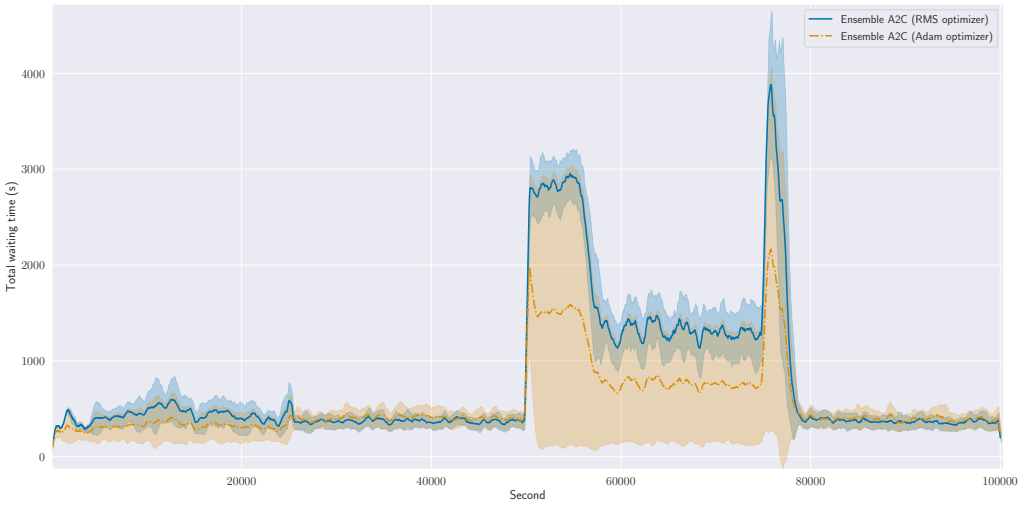


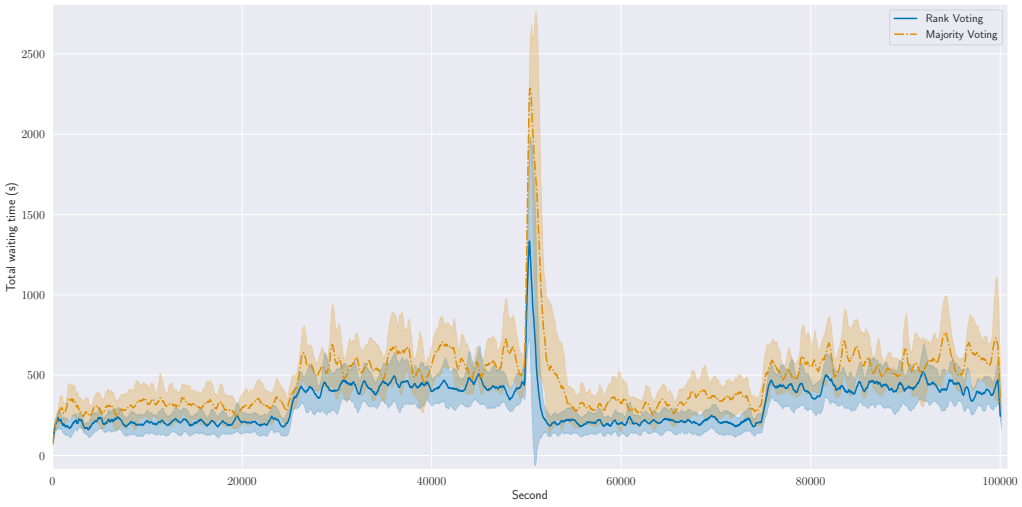
Figure 6.9: Comparison of two ensembles. The ensemble with the blue line consists of a DQN and a A2C with RMS optimizer whereas the ensemble with the orange line consists of a DQN and a A2C with Adam optimizer.

6.1.4 Majority Voting

As the last ensemble method, we use a majority voting scheme. Majority voting means that we always pick the action with the most votes (in our case $max\_votes = 2$). Because we only have two agents, the agents are possible to disagree on the best action. In such a case, we follow the same approach as in 6.1.2. As Fig.6.10a presents, the ensemble method does not perform better than its component agents, but it performs almost the same. A reason for this could be the number of agents, we use only two thus we exploit the learned policies without any exploration.

In Fig6.10b we plot the soft voting and MV methods. Compared to the soft voting method the MV is less stable, adapts slower to the traffic demand changes and

(a) Majority Voting against A2C and DQN



(b) Majority Voting compared to Soft Voting

Figure 6.10

Chapter 7

Conclusion and Future Work

7.1 Conclusions

This thesis presents a new application of the ensembles methods on traffic light management with deep reinforcement learning. Ensemble methods have been heavily used in supervised learning, mainly in classification problems, because they perform better than any constituent learning algorithms alone. Significant improvements appear when the algorithms significantly differ in how they make predictions.

In reinforcement learning, ensemble methods have been used for stock trading and playing games. In contrast to supervised learning, RL ensembles incorporate the learned policies from the agents and combine them to determine the best action for the current state. Similar to supervised learning, when we combine agents with different learning methods we tend to get the most significant improvements in the final performance.

In this work, we develop an ensemble using two state-of-the-art algorithms, the A2C and the DQN which have the worst performance compared to the PPO. Fig.7.1 reveals PPO exceptional performance compared to the others algorithms trained for $110,000$ time-steps.

DQN is an off-policy value-based algorithm and uses neural networks to approximate the Q-values for state-action pairs. On the other hand, A2C is a hybrid algorithm that combines value-based and policy-based methods to determine the selection of the following action.

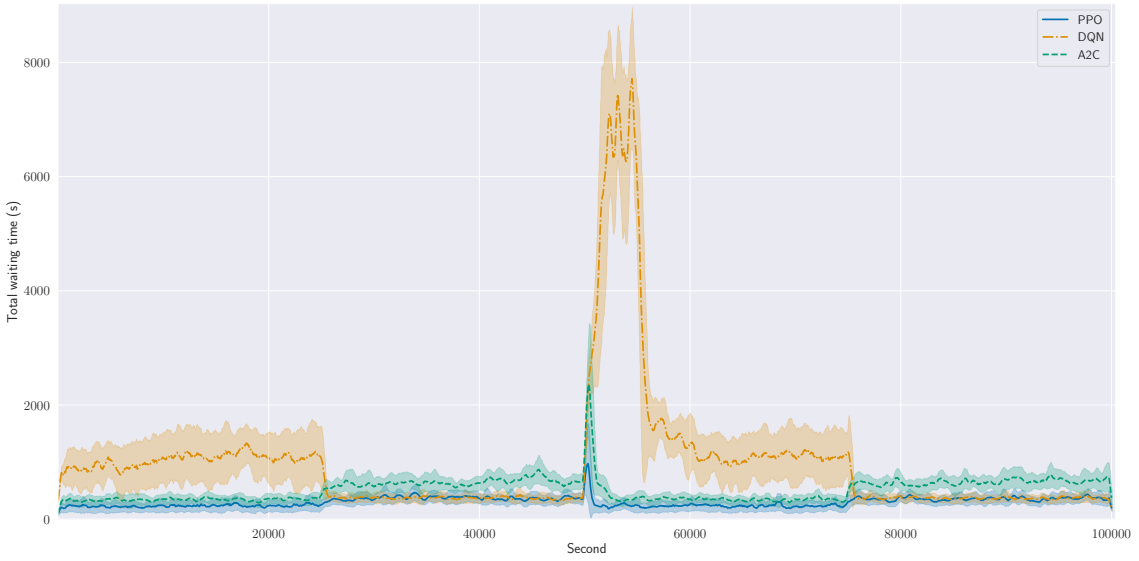


Figure 7.1: PPO, DQN and A2C trained for 110000 time-steps.

Ensemble's performance does not rely only on the type of algorithms but also on the voting scheme we choose to combine the agents. Thus, our experiments show that the voting method performed better the soft voting. Soft voting possibly performs better because it follows a balanced strategy between exploration and exploitation. In contrast, majority voting follows a greedy approach and exploits the knowledge gained from the training phase.

In addition to the previously mentioned experiments, we also experiment with an ensemble with three algorithms. We use as a third algorithm the PPO which outperformed the other two on the single intersection scenario. As Fig.7.2 demonstrates the ensemble performance is worst than PPO's performance. This result might happen because PPO outperforms A2C and DQN. Moreover, the PPO mechanism relies on the Actor-Critic method that is similar to A2C, meaning that PPO's contribution to the ensemble is not significant.

Concluding, our experiments reveal that ensemble methods can manage better traffic signal control system on a single intersection compared to compose agents. Not only they perform well concerning the total rewards but also they manage to keep the total vehicle time delays at lower level of the two algorithms.

Figure 7.2: Soft voting using A2C, DQN and PPO. The ensemble performs the same or worst than PPO.

7.2 Future Work

There is more room for improvement on the presented work. In the future, we aim to work extensively in the direction of developing an ensemble method that we will be able to train. As we see in our experiments, there are intervals in which an algorithm performs better than the other and intervals in which agents have the same behavior. We hope that the ensemble in the training phase will learn to assign weights on the agents regarding the simulation period and the traffic conditions, concluding with a better policy.

Furthermore, we plan to add more agents to the previously mentioned approaches. Specifically, we want to use a big batch of agents that barely perform better than a fixed-time traffic signal plan and see how the ensemble performs. On the same principle, we plan to combine state-of-the-art algorithms on traffic signal control systems such as Intellilight [35] ,and MPLight [5].

It would be interesting to see how the ensemble method will scale to large road networks with more than one intersection. In these scenarios, we use cooperative multi-agent systems the goal of these at every time step is to find an optimal joint action.

References

[1] L. N. Alegre. SUMO-RL. `https://github.com/LucasAlegre/sumo-rl`, 2019.

[2] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. SUMO – Simulation of Urban MObility. page 7.

[3] S. L. Brunton and J. N. Kutz. Data Driven Science & Engineering. page 76, 2021.

[4] G. Cameron and G. I. D. Duncan. Paramics—parallel microscopic simulation of road traffic. *The Journal of Supercomputing*, 10:25–53, 2004.

[5] C. Chen, H. Wei, N. Xu, G. Zheng, M. Yang, Y. Xiong, K. Xu, and Z. Li. Toward A Thousand Lights: Decentralized Deep Reinforcement Learning for Large-Scale Traffic Signal Control. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3414–3421, Apr. 2020.

[6] X.-l. Chen, L. Cao, C.-x. Li, Z.-x. Xu, and J. Lai. Ensemble Network Architecture for Deep Reinforcement Learning. *Mathematical Problems in Engineering*, 2018:1–6, 2018.

[7] T. Chu, J. Wang, L. Codeca, and Z. Li. Multi-Agent Deep Reinforcement Learning for Large-Scale Traffic Signal Control. *IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS*, 21(3):10, 2020.

[8] S. El-Tantawy and B. Abdulhai. Multi-Agent Reinforcement Learning for Integrated Network of Adaptive Traffic Signal Controllers (MARLIN-ATSC). page 8.

[9] M. Fellendorf and P. Vortisch. Microscopic Traffic Flow Simulator VISSIM. In J. Barceló, editor, *Fundamentals of Traffic Simulation*, volume 145, pages 63–93. Springer New York,

New York, NY, 2010. Series Title: International Series in Operations Research & Management Science.

[10] D. Garg, M. Chli, and G. Vogiatzis. Deep Reinforcement Learning for Autonomous Traffic Light Control. In *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, pages 214–218, Singapore, Sept. 2018. IEEE.

[11] W. Genders and S. Razavi. Using a Deep Reinforcement Learning Agent for Traffic Signal Control. *arXiv:1611.01142 [cs]*, Nov. 2016. arXiv: 1611.01142.

[12] W. Genders and S. Razavi. Asynchronous *n* -step Q-learning adaptive traffic signal control. *Journal of Intelligent Transportation Systems*, 23(4):319–331, July 2019.

[13] A. Haydari and Y. Yilmaz. Deep Reinforcement Learning for Intelligent Transportation Systems: A Survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(1):11–32, Jan. 2022.

[14] L. Kuyer, S. Whiteson, B. Bakker, and N. Vlassis. Multiagent Reinforcement Learning for Urban Traffic Control Using Coordination Graphs. In W. Daelemans, B. Goethals, and K. Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5211, pages 656–671. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISSN: 0302-9743, 1611-3349 Series Title: Lecture Notes in Computer Science.

[15] K. Lee, M. Laskin, A. Srinivas, and P. Abbeel. SUNRISE: A Simple Unified Framework for Ensemble Learning in Deep Reinforcement Learning, June 2021. Number: arXiv:2007.04938 arXiv:2007.04938 [cs, stat].

[16] L. Li, Y. Lv, and F.-Y. Wang. Traffic signal timing via deep reinforcement learning. 3(3):8, 2016.

[17] X. Liang, X. Du, G. Wang, and Z. Han. A Deep Reinforcement Learning Network for Traffic Light Cycle Control. *IEEE Transactions on Vehicular Technology*, 68(2):1243–1253, Feb. 2019.

[18] P. Mannion, J. Duggan, and E. Howley. An Experimental Review of Reinforcement Learning Algorithms for Adaptive Traffic Signal Control. In T. L. McCluskey, A. Kotsialos, J. P. Müller, F. Klügl, O. Rana, and R. Schumann, editors, *Autonomic Road Transport Support Systems*, pages 47–66. Springer International Publishing, Cham, 2016.

[19] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning, June 2016. Number: arXiv:1602.01783 arXiv:1602.01783 [cs].

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing Atari with Deep Reinforcement Learning. *arXiv:1312.5602 [cs]*, Dec. 2013. arXiv: 1312.5602.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[22] S. S. Mousavi, M. Schukat, and E. Howley. Traffic Light Control Using Deep Policy-Gradient and Value-Function Based Reinforcement Learning. *arXiv:1704.08883 [cs]*, May 2017. arXiv: 1704.08883.

[23] T. Nishi, K. Otaki, K. Hayakawa, and T. Yoshimura. Traffic Signal Control Based on Reinforcement Learning with Graph Convolutional Neural Nets. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 877–883, Maui, HI, Nov. 2018. IEEE.

[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms, Aug. 2017. Number: arXiv:1707.06347 arXiv:1707.06347 [cs].

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis.

Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan. 2016.

[26] L. Song and W. Fan. Traffic Signal Control Under Mixed Traffic With Connected and Automated Vehicles: A Transfer-Based Deep Reinforcement Learning Approach. *IEEE Access*, 9:145228–145237, 2021.

[27] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[28] B. Van de Vyvere, P. Colpaert, E. Mannens, and R. Verborgh. Open traffic lights: a strategy for publishing and preserving traffic lights data. In *Companion Proceedings of The 2019 World Wide Web Conference*, pages 966–971, San Francisco USA, May 2019. ACM.

[29] E. van der Pol and F. A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. 2016.

[30] P. Varaiya. The Max-Pressure Controller for Arbitrary Networks of Signalized Intersections. page 40.

[31] C. Wan and M. Hwang. Value-based deep reinforcement learning for adaptive isolated intersection signal control. *IET Intelligent Transport Systems*, 12(9):1005–1010, Nov. 2018.

[32] H. Wei, C. Chen, G. Zheng, K. Wu, V. Gayah, K. Xu, and Z. Li. PressLight: Learning Max Pressure Control to Coordinate Traffic Signals in Arterial Network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1290–1298, Anchorage AK USA, July 2019. ACM.

[33] H. Wei, N. Xu, H. Zhang, G. Zheng, X. Zang, C. Chen, W. Zhang, Y. Zhu, K. Xu, and Z. Li. CoLight: Learning Network-level Cooperation for Traffic Signal Control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1913–1922, Beijing China, Nov. 2019. ACM.

[34] H. Wei, G. Zheng, V. Gayah, and Z. Li. A Survey on Traffic Signal Control Methods. 2019. Publisher: arXiv Version Number: 3.

[35] H. Wei, G. Zheng, H. Yao, and Z. Li. IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2496–2505, London United Kingdom, July 2018. ACM.

[36] M. Wiering and H. van Hasselt. Ensemble Algorithms in Reinforcement Learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 38(4):930–936, Aug. 2008.

[37] H. Yang, X.-Y. Liu, S. Zhong, and A. Walid. Deep Reinforcement Learning for Automated Stock Trading: An Ensemble Strategy. *SSRN Electronic Journal*, 2020.

[38] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. Li. CityFlow: A Multi-Agent Reinforcement Learning Environment for Large Scale City Traffic Scenario. In *The World Wide Web Conference*, pages 3620–3624, San Francisco CA USA, May 2019. ACM.

[39] G. Zheng, Y. Xiong, X. Zang, J. Feng, H. Wei, H. Zhang, Y. Li, K. Xu, and Z. Li. Learning Phase Competition for Traffic Signal Control. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1963–1972, Beijing China, Nov. 2019. ACM.