Coordinated Deep Reinforcement Learners for Traffic Light Control

Elise van der Pol
University of Amsterdam
`e.e.vanderpol@uva.nl`

Frans A. Oliehoek
University of Amsterdam, University of Liverpool
`frans.oliehoek@liverpool.ac.uk`

Abstract

This paper investigates learning control policies for traffic lights. We introduce a new reward function for the traffic light control problem, and propose the combination of the popular Deep Q-learning algorithm with a coordination algorithm for a scalable approach to controlling coordinating traffic lights, without requiring the simplifying assumptions made in earlier work. We show that this approach reduces travel times compared to earlier work on reinforcement learning methods for traffic light control and investigate possible causes of instability in the single-agent case.

1 Introduction

The world-wide cost of traffic congestion is huge. For instance, in the EU this cost is estimated to be 1% of its GDP [2]. In this paper we focus on techniques to improve the control of traffic lights, which may reduce traffic congestion, thus saving time and money and reducing environmental pollution [19]. In particular, we build on approaches that phrase the problem of selecting the configurations of traffic lights at an intersection (i.e., which directions get green) as a reinforcement learning (RL) problem [23, 8]. To evaluate the effectiveness of the approach, we use SUMO [7], an open source traffic simulator.

A difficult aspect in applying RL to traffic light control is the selection of features: the number of states, each of which describing the exact situation around an intersection, is huge. This implies that a form of featurization and subsequent function approximation is needed to even represent the value function for these settings. However, it is not a priori clear what the right features are in these setting. Recent years, however, have seen the development of RL methods, such as Deep Q-learning (DQN), that use deep learning techniques [12, 18], such as convolutional networks [9] to side-step the need for manual feature extraction. Such a DQN approach to traffic control was also shown to be potentially promising [15]. This paper extends this line of work, making the following contributions. We make minor modifications to the the single-intersection DQN approach of [15] and investigate effective formulations of the reward function. In order to improve the stability of the training process, we test the effect of a number of recent techniques developed in the deep learning community in context of traffic control [22, 5, 17]. Moreover, we propose an approach to coordinate multiple intersections trained with these techniques. We base ourselves on the idea of 'transfer planning' [13], which tries and find a solution (a Q-function) for smaller source problems involving few agents (in this paper we use DQN on a 2-agent source problem) and uses this solution to select actions in a coordinated fashion in the larger target problem (using max-plus coordination [6, 8]).

This paper presents a reward function for the traffic light control problem, and empirically shows that DQN encounters good policies, but suffers from oscillations in the training process. Finally, we show that using DQN in a coordination algorithm is a promising approach to multi-agent deep reinforcement learning.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

2 Background

In a reinforcement learning setting, an agent learns to map an environmental state s to an optimal action a, such that long-term reward is maximized. On each time step t, the agent receives information about the current state s_t, takes an action a, ends up in state s_{t+1} and receives a reward r_t. The learning agent needs to find a policy π such that the discounted cumulative reward $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$ is maximized, where $0 < \gamma \leq 1$ is a discount factor. The value of a current state, action-pair is the Q-value, which is estimated in the Q-learning algorithm by iterative Bellman updates $Q_{t+1}(s, a) = Q_t(s, a) + \alpha[y_t - Q_t(s, a)]$ where the target $y_t = r + \gamma \max_{a'} Q_t(s,' a')$

In Deep Q-learning (DQN), deep neural networks are used as function approximators that map from states to Q-values, instead of estimating the Q-value of each state, action-pair separately. Using these function approximators allows the use of a larger and/or continuous space of states, and the use of images as state representations [12]. However, while the regular, tabular Q-learning algorithm is guaranteed to converge in the limit (infinity), there are no similar guarantees for Q-learning with function approximation. This is caused by at least two reinforcement learning-specific issues that the machine learning algorithms used in function approximation are not well-equipped to deal with: 1) the data set is not i.i.d.: sequential samples are heavily correlated and the data distribution is non-stationary, and 2) as the agent learns, the targets move. While reinforcement learning algorithms are designed to deal with these problems, regular machine learning algorithms tend to assume i.i.d. data distributions and stationary targets.

Two approaches have been used to stabilize the DQN algorithm: The first is experience replay [10, 14], where the sampled data points - transitions $< s, a, r, s' >$ - are stored in memory, and at training time, a batch of these data points is sampled uniformly (or according to TD-error, as in prioritized experience replay[17]) and used in backpropagation. The second solution is target network freezing [12], where the Q-value estimation is split into two different networks, a *value* network to estimate the Q-value of the current state, action-pair, $Q(s, a)$, and a *target* to compute the targets y_t. That is, the target network is used to estimate $\max_{a'} Q(s', a')$. By freezing the target network for a period of time, the targets are partially stabilized.

Earlier work on applying Deep Reinforcement Learning to the multi-agent case has focused on developing communication protocols [3] or the difference in learned behavior between cooperative and competitive agents in two-player games [20].

3 Deep Reinforcement Learning for Traffic Light Control

We apply the DQN framework to the problem of selecting optimal light configurations for traffic intersections. The SUMO traffic simulator [7] is used to run experiments. For details on the experimental setup, see [21].

3.1 Model

We describe the traffic light control problem using the components of a Markov Decision Process, where an agent responds to an environmental state $s \in \mathcal{S}$, takes one of the possible actions $a \in \mathcal{A}$, with some transition probability $p(s'|s, a)$ ends up in state s' and receives reward signal $r(s, a, s')$.

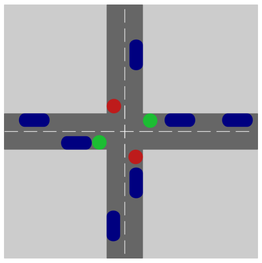
States We represent the state around an intersection using an image-like representation as in Figure 1b. Following earlier work [15], the state is a binary matrix of the positions of vehicles on the lanes controlled by a traffic light - see Figure 1c - with the addition of information about the current light configuration. As such, a convolutional neural network should be able to recognize traffic jams. In the current model, the representation of traffic lights is an arbitrary mapping to numbers representing light colors. Ideally, the traffic light information would be an extra layer to the state space, with binary features for each traffic light color. However, this increases the size of the state space significantly, and leads to memory problems for large replay memory sizes, as well as slower computation.

Actions On each time step, the action an agent takes is a choice between two different configurations of traffic light settings. Essentially, the agent selects which lanes get a green light.

Transitions The transitions from s_t to s_{t+1} are implicitly defined by SUMO and depend on a_t (the configuration of traffic lights on step t), and the cars in the simulation.

Parameter	Untuned	Fine-tuned
Optimizer	ADAM	ADAM
Replay memory size	50,000	50,000
Experience sampling	$\tau = 0.0$	$\tau = 0.5$
Learning rate (α)	0.001	0.00025
Batch size	32	32
Exploration rate (ϵ)	0.1	0.1
Discount factor (γ)	0.99	0.99
Freeze interval	30,000	30,000
State matrix size	84×84	168×168
State matrix frames	1	2

(a) Settings for the untuned and fine-tuned DQN agent



(b) Traffic situation

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.2 & 0.8 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0.8 & 0.2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(c) Simplified example of state representation in 8×8 matrix.

Figure 1:

3.2 Reward specification

Defining a feedback signal for the traffic light control problem is not clear-cut. A good metric would be to aim to reduce travel times. However, the average travel time of a vehicle cannot be computed until it has completed its route, which leads to the problem of extremely delayed rewards. We have investigated two other measures that could be used as a proxy for the average travel time, such as vehicle delay or waiting time. However, using the delay d of a vehicle, defined as $1 - \frac{\text{vehicle speed}}{\text{allowed speed}}$, as a reward signal does not properly penalize traffic jams, since one jammed lane and one maximum speed lane result in similar reward as two half-speed lanes. On the other hand, using the vehicle's waiting time w, assigning a penalty of -0.5 for a single time step of waiting and -1.0 for two or more, as a feedback signal causes an agent to favour *flickering*, constantly toggling between green and red lights, so that vehicles are never halted. So, we investigate combinations of these measures, and include penalties for teleports j (which indicate crashes and/or jams in SUMO), emergency stops e (decelerations of more than $4.5m/s^2$) and add a boolean measure c, indicating whether the light configuration has changed (to prevent flickering). We then perform a rough grid search on a test problem, and select the best combination of weights, which for the test case resulted in an average travel time of 262.32 with a standard deviation of around 75 after $60,000$ training steps (worst: average travel time of 413.5).

The final reward signal r_t at each time step is given by iterating over all vehicles around the traffic light intersection and computing the penalties, where i is the vehicle index:

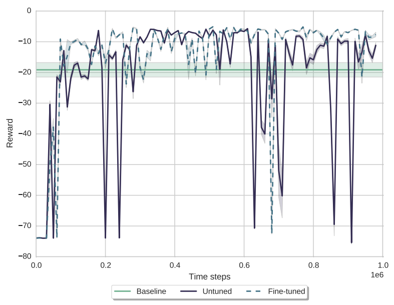
$$r_t = -0.1c - 0.1\sum_{i=1}^{N} j_i - 0.2\sum_{i=1}^{N} e_i - 0.3\sum_{i=1}^{N} d_i - 0.3\sum_{i=1}^{N} w_i \tag{1}$$

where N is the number of vehicles on the lanes the agent controls.

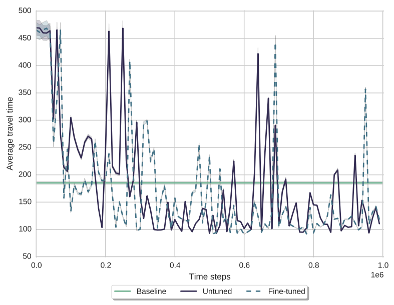
3.3 Learning stability

Using the above state description and reward function we investigated the performance of an 'untuned' DQN algorithm, which uses parameters typically found in the literature. The results are shown in Figure 2, which displays the reward and average travel time found during policy evaluation for the DQN agent, after every 10,000 time steps of training. This clearly shows that while the DQN algorithm encounters good policies quickly, the training curve oscillates heavily. This is a problem because it may prevent the algorithm from learning even better policies. As a baseline, we use a linear regressor with manually defined features (for details, see [21]). While DQN finds policies with higher average rewards than the linear agent with manual features, the reward drops off every now and then into a performance dip. The system corrects, though, and no serious divergence happens. This instability may be caused by *catastrophic forgetting* [11], a phenomenon in neural networks where learning to solve a new task causes the system to forget earlier learned structure. On the other hand, convergence is not guaranteed for function approximation in Q-learning, and oscillation is a known problem. Finally, the lack of stability may be related to the specific problem of traffic, which is a decidedly different problem from Atari games such as Pong. While earlier work shows great results on some Atari games, other work [17] (Appendix B, Figure 7) shows similar stability issues with the DQN algorithm for specific games.

We test a number of different parameter settings, such as replay memory size, optimization algorithm and temperature in prioritized experience replay, and compare these to the untuned agent, whose parameter settings can be found in Figure 1a. The results are shown in Table 1. Generally, adding more information to the state representation (more position frames, larger matrices) helps the agent find better policies. Moreover, using prioritized experience replay with a balanced τ increases the average reward, indicating less negative outliers. This balanced sampling ensures the use of informative samples, while at the same time preventing the loss of earlier learned structure by repeating 'regular' experience. Finally, we use the best combination of parameter settings to train a fine-tuned DQN agent (see Figure 1a for parameter settings), whose performance is also displayed in Figure 2.



(a) Average reward and standard error.



(b) Average travel time and standard error.

Figure 2: Behavior of the greedy policy for the untuned and fine-tuned DQN agents. Compared to best-performing linear agent at training time.

Parameter	New value	Best reward	Average (last 100k)	Median (last 100k)
Untuned Settings		-5.73	-18.22	-11.45
Frames	2	-5.23	-18.44	-8.19
Frames	3	-5.60	-6.88	-7.47
Frames	4	-5.18	-7.16	**-7.25**
Matrix size	42×42	-5.73	-46.65	-10.22
Matrix size	168×168	**-4.93**	-10.46	-7.44
Memory size	$10,000$	-5.00	-15.05	-8.34
Memory size	$100,000$	-5.81	-8.56	-15.27
Freeze interval	$10,000$	-5.68	-10.28	-9.13
Freeze interval	$50,000$	-5.88	-11.25	-10.22
Experience replay τ	0.5	-5.48	**-6.85**	-7.73
Experience replay τ	1.0	-5.20	-19.51	-8.12
Learning rate	0.00025	-5.18	-13.10	-10.56
Optimizer	SGD	-73.76	-73.88	-73.90
Optimizer	RMSProp	-11.33	-18.89	-32.29
Double DQN	On	-73.73	-73.89	-73.89
Batch normalization	On	-5.59	-66.98	-51.42

Table 1: Parameter changes relative to untuned settings (first row), and the maximum reward found, the average reward and the median reward during the last 100k training steps (with steps of 10k).

Since the Double DQN (DDQN) algorithm performed very well on the Atari benchmark [22], we also test its performance on the traffic problem. However, DDQN appears to get stuck in a local minimum, where the agent maps every state to the same Q-values. At training time, this results in rapid action switching, as on every update the 'best' action changes. However, during policy evaluation, every state maps to the same action, and as a result, some lanes face a red light indefinitely. We also experimented with the use of Batch Normalization[5], since this increases stability and allows the use of higher learning rates in regular deep learning approaches. However, similarly to earlier work [16] we find that Batch Normalization causes divergence, possibly due to the non-stationarity of the underlying data distribution. More research is needed to address the issues we encountered with these algorithms.

4 Coordinated Deep Reinforcement Learners

We extend the single-agent DQN solution to the multi-agent case by making use of *transfer planning* [13] and the *max-plus coordination* algorithm [6], and evaluate the resulting approach on different traffic scenarios.

4.1 Approach

To coordinate between multiple agents, we follow earlier work [4, 6] and factorize the global Q-function as a linear combination of local suproblems: $\hat{Q}(s,a) = \sum_e Q_e(s_e, a_e)$, where e corresponds to a subset of neighbouring agents.

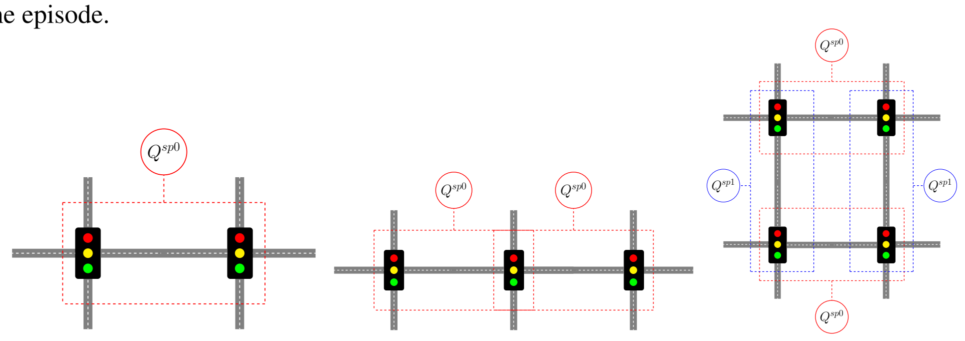
We then use the max-plus [6] coordination algorithm to optimize the joint global action over the entire coordination graph. Note that while max-plus is guaranteed to find the optimal solution in acyclical graphs in a finite number of iterations, it has no similar guarantees for cyclical graphs [1].

In contrast to the aforementioned earlier approaches [4, 6], the functions Q_e are found using a variant of *transfer planning* [13]. In transfer planning, we learn a Q-function for a subproblem of a larger multi-agent problem. Training on the source problem results in an approximation of its Q-function. Provided that the source problem and other subproblems are similar, we can then re-use the source problem's Q-function for each subproblem in the larger multi-agent problem, rather than training a Q-function for each separate subproblem. In other words, unlike earlier work [4, 6], transfer planning does not attempt to minimize the global approximation error of $\hat{Q}$. This transfer planning approach circumvents two problems present in multi-agent reinforcement learning. The first is the non-stationarity in the environment introduced by multiple agents learning and acting simultaneously. By training on a source problem, the environmental dynamics do not change during learning. The second is the cost of training many agents simultaneously. Because the source problems are independent, they can be solved independently (e.g. sequentially). Moreover, we exploit symmetries of our source problems, further reducing the computational cost.

In particular, we train a DQN agent on the two agent source problem in Figure 3a to get Q^{sp0}, and a rotated version to get Q^{sp1}, then use transfer planning to solve the multi-agent problems in Figure 3b and 3c.

4.2 Evaluation

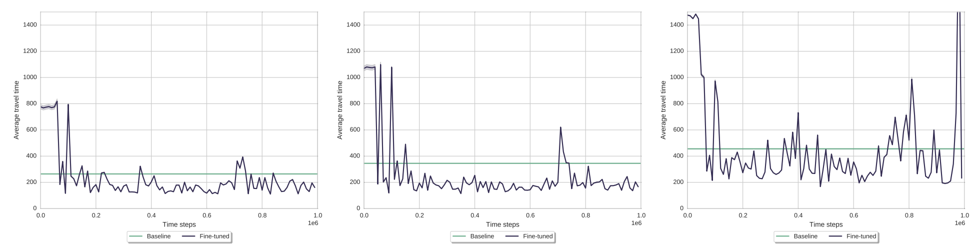
As a baseline, we use an earlier algorithm by Wiering [23] and its multi-agent extension by Kuyer et al [8]. Wiering decomposes the state into a linear combination of vehicle states. The reward function assigns a penalty for each stationary car in the current time step. In particular, we use Wiering's algorithm to learn policies for the two-agent scenario, then use transfer planning to combine these with max-plus to get an algorithm similar to Kuyer's. To get the baseline, we use the best performing version of the Wiering/Kuyer algorithm at training time and evaluate it. Since the Wiering algorithm uses a different reward function, we can only compare on the basis of average travel time at the end of the episode.



(a) Two agents (b) Three agents (c) Four agents

Figure 3: Three traffic scenarios. The numbers index the Q-function's source problem.

Figure 4a shows the training curve of the two-agent source problem. Figures 4b and 4c show the results of evaluating the two-agent source Q_e after each 10,000 training steps in the three-agent and four-agent scenario, respectively. These graphs show that the DQN approach outperforms the Wiering/Kuyer approach most of the time, but due to instability, it sometimes underperforms, especially in the four-agent case. The latter is not unexpected: since the DQN algorithm results in an approximation of the Q-value function, and max-plus has no guarantees for cyclical graphs, there are two sources of error that may stack.



(a) Two agent scenario (b) Three agent scenario (c) Four agent scenario

Figure 4: Transfer planning with fine-tuned settings. Compared to best-performing Wiering/Kuyer agent at training time.

Manual inspection of the policies found by using the DQN approach shows some interesting behavior[1]. For one, the policies found for the three-agent chain are smooth, with clear traffic waves, whereas the Kuyer baseline resorts to rapid switching of traffic lights, resulting in a lot of starting and stopping for vehicles. However, there is a point where the convolutional filters no longer properly activate: the filters show no sign of vehicles, even though they are still present on the state matrix. As a result, the agent behaves as though there are no vehicles, and no longer switches lights, resulting in a traffic jam.

5 Discussion

Traffic is unlike Atari games for multiple reasons: for one, the consequences of taking suboptimal actions can be felt for much longer in very busy traffic scenarios, and especially so in multi-agent scenarios. In busy scenarios, it is very hard to return to a more desirable state from congestion, and a few suboptimal actions can already lead to a jam. Compare this to, for example, the Atari game Pong, where a suboptimal action results in a small penalty and a fresh start.
Moreover, traffic light agents suffer significantly from missing information: a low matrix resolution may result in some vehicles being obscured behind the traffic lights. If a vehicle is missed, or if the convolutional filters fail to activate, the agent will keep the vehicle stuck forever. Thus, in the scenario of learning for traffic light agents, a good state representation is important.

Overall, combining DQN with transfer planning is promising, but more work is needed to ensure the reliability of the approach.

6 Conclusion

This paper presented the use of the DQN algorithm with transfer planning as a promising and scalable multi-agent approach to deep reinforcement learning. By using transfer planning, it avoids problems present in multi-agent reinforcement learning, and allows for faster and more scalable learning. It outperforms earlier work on multi-agent traffic light control, but the DQN algorithm may oscillate, a problem also found in earlier work on deep reinforcement learning. More research is needed to fence off circumstances where DQN is not stable, and to find approaches that make it more reliable.

Acknowledgments

Research supported (in part) by NWO Innovational Research Incentives Scheme Veni #639.021.336. The Tesla K40 used for this research was donated by the NVIDIA Corporation.

References

[1] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] Commission of the European communities. *White paper-European transport policy for 2010: time to decide*. Office for Official Publications of the European Communities, 2001.

[1]Watch the video: `http://fransoliehoek.net/trafficvideo`

[3] Jakob N Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *arXiv preprint arXiv:1605.06676*, 2016.

[4] Carlos Guestrin, Michail Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *ICML*, volume 2, pages 227–234, 2002.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] Jelle R Kok and Nikos Vlassis. Using the max-plus algorithm for multiagent decision making in coordination graphs. In *Robot Soccer World Cup*, pages 1–12. Springer, 2005.

[7] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent development and applications of sumo–simulation of urban mobility. *International Journal On Advances in Systems and Measurements*, 5(3&4), 2012.

[8] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 656–671. Springer, 2008.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[11] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. *Psychology of learning and motivation*, 24:109–165, 1989.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[13] Frans A Oliehoek, Shimon Whiteson, and Matthijs TJ Spaan. Approximate solutions for factored Dec-POMDPs with many agents. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 563–570. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[14] Martin Riedmiller. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *Machine Learning: ECML 2005*, pages 317–328. Springer, 2005.

[15] Tobias Rijken. DeepLight: Deep reinforcement learning for signalised traffic control, 2015.

[16] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

[17] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *ICLR 2016*, 2016.

[18] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[19] Stephen F Smith, Gregory J Barlow, Xiao-Feng Xie, and Zachary B Rubinstein. Surtrac: Scalable urban traffic control. In *Transportation Research Board 92nd Annual Meeting*, number 13-0315, 2013.

[20] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *arXiv preprint arXiv:1511.08779*, 2015.

[21] Elise van der Pol. Deep reinforcement learning for coordination in traffic light control. Master's thesis, University of Amsterdam, 2016.

[22] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. *CoRR, abs/1509.06461*, 2015.

[23] Marco Wiering et al. Multi-agent reinforcement learning for traffic light control. In *ICML*, 2000.