State-of-the-Art Deep Reinforcement Learning Technical Report

Parth Jaggi, Xiaoyu Wang and Nicolas Carrara.

August 2020

1 Introduction

Sequential decision making, commonly formalized as Markov Decision Process (MDP) optimization, is a key challenge in artificial intelligence. Reinforcement Learning is one of the available solutions for sequential decision making. In this report, we aim to explore the different algorithms in reinforcement learning that have achieved exceptional performance on varying tasks. We also categorize the algorithms into different sections based on their properties. Lastly we also discuss the different configurations of a traffic signal controller, which would affect its compatibility with different reinforcement learning methods.

2 Algorithms

2.1 Model-Free, Single-Agent, Discrete-Action Algorithms

Deep Q-Networks (DQNs)

Please refer to Section 4.1 of Coordinated Traffic Control report.

Improved DQNs

After the DQN was first proposed, many researchers dedicated on improving vanilla DQNs in many approaches. (Hessel et al., 2017) summarizes and compares the related extensions:

- Double Q-learning:

  Back in 1993, (Thrun and Schwartz, 1993) have shown that using function approximation method in reinforcement learning causes systematic overestimation of Q-values. DQN with deep neural networks also has this issue. Double Q-learning (Hasselt, 2010) alleviates this issue by decoupling the maximization and estimation processes. Double DQN utilizes this trick and evaluates the squared TD-error of a sample in this form:

$$\left(r_t + \gamma \max_a Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)\right)^2, \tag{1}$$

where θ and θ' represent the parameters of the action-evaluation Q-network and bootstrapping target-network respectively. Double Q-learning improves DQN's performance by reducing the overestimation and is broadly used.

- Dueling network:

  The dueling network replaces the last layer of a value-based DRL network to factorize Q-values into two parts: state values and state-dependent action advantages (Wang et al., 2016). This factorizing method evades unnecessary estimations.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha), \tag{2}$$

  where θ represents the parameters of the rest part of the DRL network except the dueling layers. α and β are the parameters of the two streams corresponding to state values and action advantages in the dueling layers.

  We further restrict the advantage function in Equation 2 to solve the unidentifiable issue:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha) \right), \tag{3}$$

  where $\mathcal{A}$ is the set of available actions.

- Prioritized replay:

  In a DQN with experience replay mechanism, the sample is chosen uniformly. However, those samples with higher TD-error are more important to learning and should be sampled more frequently. (Schaul et al., 2015) ranks samples based on their TD-errors in the last replay to assign those samples higher priority, and samples from the prioritized replay buffer with stochastic sampling to address the issues greedy prioritization has.

- Multi-step learning:

  Q-learning with one-step TD exploring faces high bias issue. n-step TD method standing between one-step TD method and Monte Carlo method introduces multi-step return (reward) to accelerate the learning (Sutton, 1988). The n-step return is defined as:

$$R_t^{(n)} = \sum_{k=0}^{n-1} \gamma^k r_{t+k}. \tag{4}$$

  A DQN with multi-step learning has this alternative loss:

$$\left(R_t^{(n)} + \gamma \max_a Q(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta) \right)^2. \tag{5}$$

- Distributional RL:

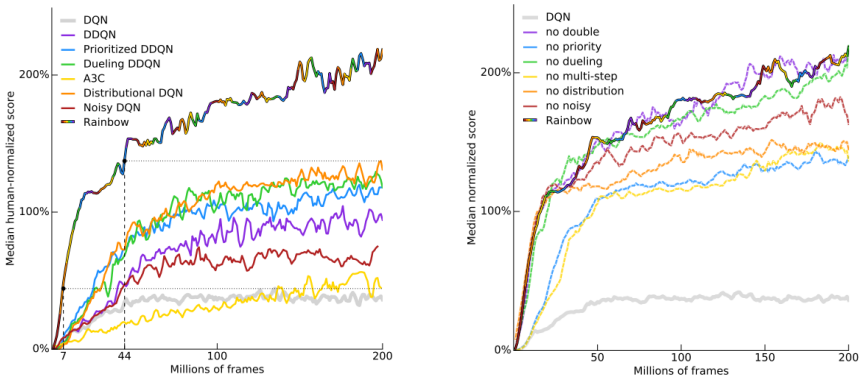
  Common value-based RL methods provide the estimation of state-action pairs value which ignores detailed information. (Bellemare et al., 2017) discusses the advantage of using value distribution and proposed the distributional Q-learning method. This work formalizes the value distribution with probability masses, introduces a variant of Bellman's equation with distributional Bellman operator, and minimizing the Kullbeck-Leibler divergence between the value distribution and the target distribution. Experiment results demonstrates the performance and importance of using value distribution in approximate reinforcement learning.

- Noisy net:

  Finally, the noisy net is an alternate of exploring in RL (Fortunato et al., 2017). Different from the ϵ-greedy policy, the noisy net introduces noise into fully connected layer to enable the exploration. In detail, a noisy linear layer replaces the standard formulation $\mathbf{y} = W \cdot \mathbf{x} + \mathbf{b}$ with:

$$\mathbf{y} = (W + W_{noisy} \odot \epsilon^w)\mathbf{x} + (\mathbf{b} + \mathbf{b}_{noisy} \odot \epsilon^b), \tag{6}$$

  where ϵ^w and ϵ^b are random variables. W_{noisy} and $\mathbf{b}_{noisy}$ are the weights corresponding to the noise streams. $\odot$ is the element-wise multiplication operator.



(a) Performance comparison across DQNs with each of the extensions. The rainbow-colored curve represents the DQN with all 6 extensions.

(b) Performance comparison across the rainbow DQN and its ablations. The dashed curves represent the DQNs without a certain extension.

Figure 1: Median human-normalized performance across 57 Atari games (Hessel et al., 2017)

(Hessel et al., 2017) tests and analyses the contribution of each extension technique. The experiment results across 57 Atari games are shown in Figure 1. The Rainbow is a DQN combining all 6 extensions mentioned above and

obtains the best performance. In the ablation study shown in Figure 1(b), removing multi-step learning and prioritized replay influences the performance of integrated DQNs the most, which means the two contribute much.

Ape-X Architecture

In a traditional RL algorithm, a single process takes both data generation and model training tasks, which is inefficient. In order to take the advantage of distributed computing, (Horgan et al., 2018) proposed the ApeX architecture making RL agents could access to order of magnitude more data than before. Ape-X decomposes the two key procedures in RL - collecting data and learning on samples - and runs them asynchronously. A group of actors play multiple episodes in parallel to generate massive data samples; while a specific learner trains the model based on the most important samples with the prioritized replay technique.

Despite all the workers (actors and the learner) work in an asynchronous manner, they share information periodically to coordinate the learning: actors update their network parameters periodically from the learner to make better action evaluation and save the evaluated samples with absolute TD-error priority to the replay buffer, the learner train the network based on stochastic sampling and calls actors to update samples' priorities.

The Ape-X architecture fits either value-based RL methods (DQN) or policy gradient methods (DDPG). Experiments using hundreds of CPU cores for actors and one GPU for learner demonstrate the proposed architecture's scalability and capability. As shown in Figure 2, the Ape-X DQN outperforms the single-thread Rainbow DQN in both effectiveness and efficiency.

2.2 Model-Free, Single-Agent Algorithms

Policy Gradient Methods

Policy gradient is an approach to solve reinforcement learning problems. The policy gradient methods target at modeling and optimizing the policy directly to obtain the optimal rewards. The policy is usually modeled with a parameterized function respect to θ, $\pi_\theta(a|s)$. Here we would discuss the off-policy policy gradient method, because it has advantages of better sample efficiency and better exploration. Behavior policy which collects the samples is a known policy, labelled as $\beta(a|s)$. And the objective function sums up the reward over the state distribution defined by the above behavior policy as:

$$J(\theta) = \sum_{s\in\mathcal{S}} d^\beta(s) \sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a|s) = \mathbb{E}_{s\sim d^\beta}\big[\sum_{a\in\mathcal{A}} Q^\pi(s,a)\pi_\theta(a|s)\big] \quad (7)$$

where $d^\beta(s)$ is the stationary distribution over the policy β. And we can rewrite the gradient as:

$$\nabla_\theta J(\theta) = \mathbb{E}_\beta\Big[\frac{\pi_\theta(a|s)}{\beta(a|s)} Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)\Big] \quad (8)$$

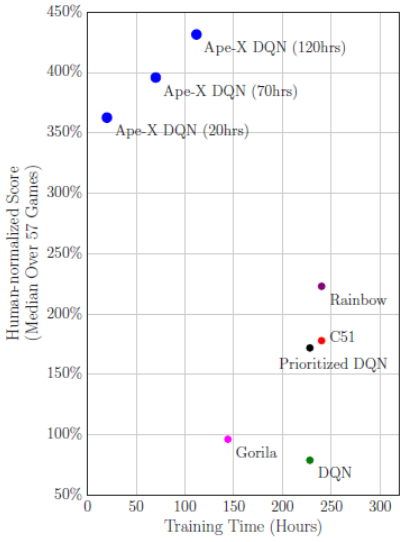


Figure 2: Comparing Ape-X DQN with other RL methods across 57 Atari games (Horgan et al., 2018)

where $\frac{\pi_\theta(a|s)}{\beta(a|s)}$ is the importance weight. In summary, when applying policy gradient in the off-policy setting, we can simple adjust it with a weighted sum and the weight is the ratio of the target policy to the behavior policy, $\frac{\pi_\theta(a|s)}{\beta(a|s)}$.

There are many other variations of policy gradient methods, and some of them we would discuss below.

Asynchronous Advantage Actor-Critic (A3C)

Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016) is a classic policy gradient method with a focus on parallel training. In A3C, there is a single critic that learns the value function and multiple actors that are trained in parallel. The actors get synced with global parameters periodically.

The critic learns the state-value function, and the loss function is the mean squared error, $J_v(w) = (G_t - V_w(s))^2$ and gradient descent is applied to find the optimal w. The state-value function is used as the baseline in the policy gradient update. Even though A3C is the asynchronous version of Advantage Actor-Critic (A2C) (Mnih et al., 2016), recent works (Wu et al., 2017) has shown that A2C is more efficient in GPU utilization and also works better with larger batch sizes than A3C.

Proximal Policy Optimization (PPO)

Proximal policy optimization (PPO) (Schulman et al., 2017) extends the policy gradient, which is an on-policy method, with importance sampling technique to increase the sample efficiency. The importance sampling estimates a variable

by using samples from another distribution:

$$E_{x\sim p}\left[f(x)\right] = E_{x\sim q}\left[f(x)\cdot\frac{p(x)}{q(x)}\right]. \tag{9}$$

By using the importance sampling, the policy gradient corresponding to a trajectory sample generated by another policy is expressed as:

$$\nabla\bar{R}_\theta = E_{\tau\sim p_{\theta'}(\tau)}\left[\frac{p_\theta(\tau)}{p_{\theta'}(\tau)}\cdot R(\tau)\cdot\nabla\log p_\theta(\tau)\right]. \tag{10}$$

Similarly, after sampling one-step samples from trajectories, the gradient is:

$$E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}\cdot\frac{p_\theta(s_t)}{p_{\theta'}(s_t)}\cdot A^{\theta'}(s_t|a_t)\cdot\nabla\log p_\theta(a_t|s_t)\right]. \tag{11}$$

In the importance sampling, large deviation between distributions p and q creates a large difference in the variance of the estimation. Hence, PPO introduces KL-divergence as a regularization term to restrict that the policy θ' should be similar to the policy θ. This limitation makes $p_\theta(s_t)$ similar to $p_{\theta'}(s_t)$. And the objective function is defined as:

$$J^{\theta'}_{\text{PPO}}(\theta) = J^{\theta'}(\theta) - \beta\cdot\text{KL}(\theta,\theta'), \tag{12}$$

$$J^{\theta'}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta'}}\left[\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}\cdot A^{\theta'}(s_t|a_t)\right]. \tag{13}$$

Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) could be seen as the precursor of PPO. It also uses the importance sampling to alleviate the low sample efficiency issue that the vanilla policy gradient has. Different from the PPO, TRPO is built on the approximation to the theoretically-justified procedures to solve the policy gradient's step-length issue and guaranteed to improve the policy monotonically. In formal, TRPO has a similar objective function as PPO:

$$J^{\theta'}_{\text{TRPO}}(\theta) = J^{\theta'}(\theta), \text{KL}(\theta,\theta') < \delta. \tag{14}$$

Compared with the objective function of PPO, TRPO has the KL-divergence term as the constraint, which makes the optimization harder to solve.

Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradients (DDPG) (Lillicrap et al., 2015) is a method standing between the policy gradient (producing action distribution) and DQNs (producing discrete actions). Different from the original policy gradient, the word "deterministic" in DDPG means that it only produces deterministic continuous actions. DDPG takes the actor-critic structure as we introduced in the policy gradient to handle the continuous action space. Meanwhile, DDPG

also utilizes the experience replay and double Q-network techniques in DQNs to approximate the Q-values efficiently.

A DDPG consists of 4 neural networks: actor (target) networks and critic (target) networks. The two target networks perform similarly to the target network in a DQN to alleviate the overestimation issue. The actor network parameterizes the current policy by $\mu(s|\theta^{\mu})$ and is trained by the policy gradient method. The critic network $Q(s, a|\theta^{Q})$ performs as a DQN to evaluate the current policy's value and is learned following the Bellman's equation.

Distributed Distributional Deep Deterministic Policy Gradients (D4PG)

Distributed Distributional Deep Deterministic Policy Gradients (D4PG) (Barth-Maron et al., 2018) is an extension of the DDPG method combining multiple DRL tricks. The D4PG performs well in dealing with problems with continuous action space.

- *Distributed* means the distributed computing architecture (similar to the Ape-X). Multiple actors (not the actor in DDPG) in the D4PG interact with environments, collect data, evaluate samples by TD-error priority, and save data to the experience replay buffer. The single learner with actor-critic structure samples data from the prioritized replay buffer and update the four network following the DDPG's workflow.
- *Distributional* means the distributional RL technique discussed in Section 2.1. D4PG implements the distributional value function at the critic side and uses the cross-entropy to measure the distance between distributions rather than the KL-divergence used in DQNs.
- Besides, the D4PG also introduced the multi-step TD return technique to reduce the variance in network updating.

This work also compares the D4PG and related methods with PPO. Experiment results (Figure 3) shows that the D4PG with all extensions performs the best and even the D3PG (distributed DDPG) without prioritized replay could outperform the PPO as well.

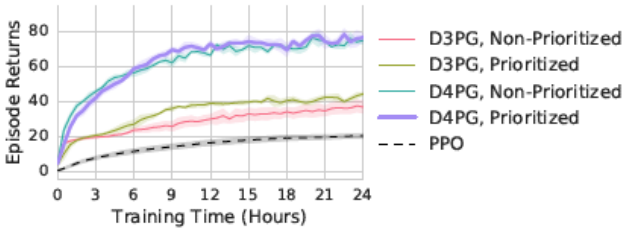


Figure 3: Comparing D4PG with PPO (Barth-Maron et al., 2018)

2.3 Model-Free, Multi-Agent Algorithms

Please refer to Section 5 of Coordinated Traffic Control report.

2.4 Model-Based, Single-Agent Algorithms

In this section we would discuss a couple of recent model-based papers that have competed quite well with the model-free algorithms. For some time it has been accepted that model-free methods performance is superior especially in domains with high-dimensional observations. The following works challenge that and perform above or at the level of top model-free algorithms.

Learning Latent Dynamics for Planning from Pixels (PlaNet)

Planning has been very successful for control tasks with known environment dynamics, for example games like AlphaGo. To leverage planning in unknown environments, the agent needs to learn the dynamics from interactions with the world. Deep Planning Network (Hafner et al., 2019a) (PlaNet) attempts to learn a world model that is accurate enough for planning, however this has been a long-standing challenge, especially in image-based domains.

High performance can only be achieved if the dynamics model can accurately predict the future rewards for multiple time steps. Planet uses a latent dynamics model with both deterministic and stochastic transition components. Planet solves wide variety of tasks with contact dynamics, partial observability, and sparse rewards with only pixel observations which exceed the difficulty of tasks that were previously solved by planning methods. Planet also uses substantially fewer episodes while matching the performance of strong model-free algorithms.

PlaNet is able to achieve such performance through the following:

- **Planning in Latent Spaces**: Dynamics model is learnt over latent space, which enables efficient planning.
- **Recurrent State Space Model**: The latent dynamics model has both deterministic and stochastic components, that are crucial for high planning performance.
- **Latent Overshooting**: Generalization of standard variational bounds to include multi-step predictions.

Planning is performed using the cross-entropy method (Rubinstein, 1997) (CEM), which is a population-based optimization algorithm that infers distribution over action sequences that maximize the objective. Results discussed in the following section on Dreamer.

Dream to Control (Dreamer)

Deep learning has allowed the learning of world models from high-dimensional observations, but there are still different ways of driving behaviors from these

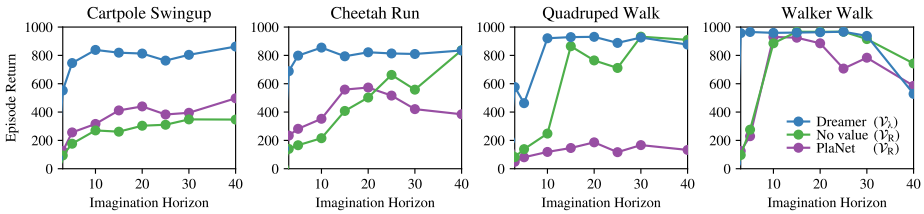


Figure 4: Effect of imagination horizons. Comparing performance of Dreamer, Dreamer without value prediction, and PlaNet. Dreamer with value prediction is more robust to changes in horizon length.

models. Dreamer (Hafner et al., 2019b) extends on the work done in PlaNet, by learning behaviors by propagating analytical gradients of learned state values, through trajectories imagined in compact state space of a learned world model. Dreamer's core contributions include:

- **Latent Imagination for learning long-horizon behaviors**: Model-based agents have tendency to be short-sighted if they use limited imagination horizon. By predicting both action and state values, and propagating analytical gradients through latent dynamics allows Dreamer to address this limitation.
- **Performance in visual control tasks**: Using same hyper parameters for all tasks, Dreamer exceeds previous model-free and model-based agents in terms of computation time, data-efficiency and final performance.

In Figure 4 we can see the comparison between Dreamer, Dreamer without value estimation and PlaNet implementations. We also see that using the value estimation allows Dreamer to be more robust to changes in the imagination horizon and highlights the performance differential between PlaNet and Dreamer. The model components inside Dreamer are namely:

$$\begin{aligned}
&\text{Representation model:} && p_\theta(s_t|s_{t-1}, a_{t-1}, o_t) \\
&\text{Transition model:} && q_\theta(s_t|s_{t-1}, a_{t-1}) \\
&\text{Reward model:} && q_\theta(r_t|s_t) \\
&\text{Action model:} && q_\phi(a_t|s_t) \\
&\text{Value model:} && v_\psi(s_t)
\end{aligned} \tag{15}$$

Here p is used for distributions that generate samples in the real environment and q for their approximations that enable latent imagination.

Value estimation To learn the action and value models, we need to estimate the state values of imagined trajectories $\{s_\tau, a_\tau, r_\tau\}_{\tau=t}^{t+H}$. These trajectories branch off of the model states s_t of sequence batches drawn from the agent's

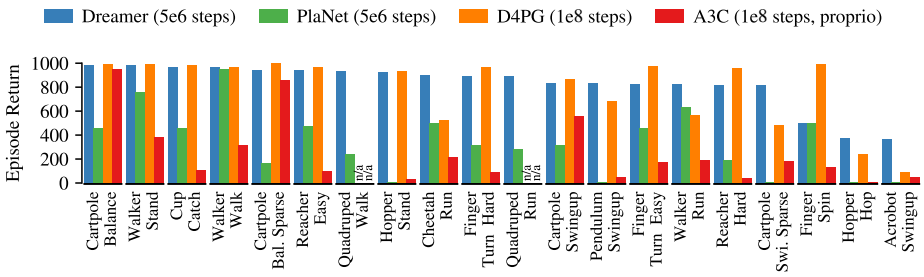


Figure 5: Performance comparison to existing methods.

dataset of experience and predict forward for the imagination horizon H using actions sampled from the action model. State values can be estimated in multiple ways that trade off bias and variance (Sutton and Barto, 2018),

$$\mathrm{V}_{\mathrm{R}}(s_\tau) \doteq \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{n=\tau}^{t+H} r_n\right), \tag{16}$$

$$\mathrm{V}_{\mathrm{N}}^{k}(s_\tau) \doteq \mathrm{E}_{q_\theta, q_\phi}\left(\sum_{n=\tau}^{h-1} \gamma^{n-\tau} r_n + \gamma^{h-\tau} v_\psi(s_h)\right) \quad \text{with} \quad h = \min(\tau + k, t + H), \tag{17}$$

$$\mathrm{V}_{\lambda}(s_\tau) \doteq (1-\lambda) \sum_{n=1}^{H-1} \lambda^{n-1} \mathrm{V}_{\mathrm{N}}^{n}(s_\tau) + \lambda^{H-1} \mathrm{V}_{\mathrm{N}}^{H}(s_\tau), \tag{18}$$

where the expectations are estimated under the imagined trajectories. V_{R} simply sums the rewards from τ until the horizon and ignores rewards beyond it. This allows learning the action model without a value model, an ablation we compare to in our experiments. $\mathrm{V}_{\mathrm{N}}^{k}$ estimates rewards beyond k steps with the learned value model. Dreamer uses V_{λ}, an exponentially-weighted average of the estimates for different k to balance bias and variance.

Comparing with other existing methods, we see in Figure 5 that Dreamer inherits the data-efficiency of PlaNet while exceeding the asymptotic performance of the best model-free agents. After 5×10^6 environment steps, Dreamer reaches an average performance of 823 across tasks, compared to PlaNet at 332 and the top model-free D4PG agent at 786 after 10^8 steps.

3 Traffic Controller Configurations

3.1 State Spaces

Please refer to Section 2.2 of Coordinated Traffic Control report for the generous discussion about the traffic state space. In this section, we define two DRL specific state spaces. The discrete traffic state encoding (DTSE) is designed to describe the detailed traffic states including each vehicle's speed and

position (Genders and Razavi, 2016). Since the DTSE relies on detailed information, it is also called long-range detection and is hard to obtain in the real world with existing technology. The temporal DTSE (TDTSE) is proposed to alleviate the real-world accessible issue (Shabestary and Abdulhai, 2018). The TDTSE representation only needs the data provided by the inductance loop detectors, which is broadly used by many traffic agencies all around the world. The TDTSE representation is more reasonable than the DTSE, but contains lesser information.

Discrete Traffic State Encoding (DTSE)

The DTSE meshes the road into matrices, where each row represents a lane of the road link and each column represents a certain position of the lane, as shown in Figure 6. Hence, for each approach, we can create two matrices corresponding to the existence and velocities of vehicles. For all the approaches, we stack all the matrices together to form two longer matrices. And we treat these two matrices as two channels of a tensor. For example, we have an intersection with total n lanes from all approaches. We gather the traffic states from 200m away from the stop bars and divide the lanes into 5m cells. Then we get a three-dimensional tensor with size (n, 40, 2) which has a similar form as an RGB based image. The advantage of this approach is that we can use the ability and convenience of convolutional neural networks (CNNs) to handle the input easily. However, the drawbacks of this representation are also prominent as we discussed before. The data it requires is hard to access in the real world. And it does not contain any historical information.

(a)

(b)

0	0	0	0	0	0	0	0	1	0	0	1
1	0	0	0	1	0	0	0	1	0	1	1
0	1	0	0	0	0	1	0	0	1	0	1

(c)

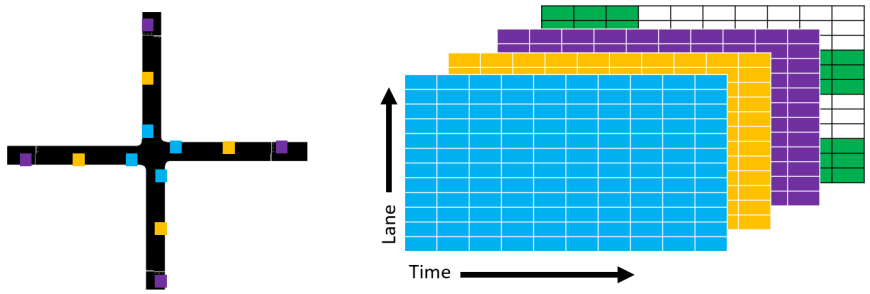
0	0	0	0	0	0	0	0	0.2	0	0	0
0.8	0	0	0	0.5	0	0	0	0	0	0	0
0	0.6	0	0	0	0	0.4	0	0	0	0	0

Figure 6: The DTSE representation. (a) The snapshot of the approach at an intersection. (b) The positional matrix. (c) The velocity matrix.

Temporal Discrete Traffic State Encoding (TDTSE)

The TDTSE uses the information from fixed loop detectors. Take the same intersection with n lanes as an example, we put a fixed number m = 3 groups of loop detectors in each lane as shown in Figure 7(a). In order to build a tensor, we introduce the time axis into the TDTSE which doesn't exist in the DTSE representation. Thus, in the TDTSE, each row still represents the lane; while each column represents the time step. Since we have m sets of detectors, set

the length of history to h, stacking all m matrices together generates a tensor with size (n, h, m). In addition, we also pass the current and historical phases into the state space by adding another channel representing the green phases as shown in Figure 7(b).



(a) An intersection with 3 groups of loop detectors

(b) The TDTSE tensor

Figure 7: The TDTSE representation

3.2 Action Spaces

There are many different types of action-spaces that are available to us, and these all would be part of the hyper parameter search when we are looking for the best performing software for traffic control. Some of the categories are:

1. **Extend-Change**
   (a) Default:
      i. Consists of 2 actions: extend and change.
   (b) Default with fast-forward:
      i. Consists of 2 actions: extend and change.
      ii. Agent skips the unactionable time steps.
   (c) Default with progression information:
      i. Consists of 2 actions: extend and change.
      ii. Agent is provided with information regarding where it exists in the phase-cycle which helps effectively ignore unactionable time steps.
   (d) Default with No-Operation:
      i. Consists of 3 actions: extend, change and no-operation (no-op).
      ii. Agent is also provided with a action-mask, which forces the agent to choose the no-op action during unactionable time steps.
2. **Phase-Select**

(a) Default:

i. Consists of n actions, where n is the number of green-phases.

ii. Agent is also provided with a action-mask, which forces agent to choose only from allowed phases.

iii. Action-mask also forces order of phases. Action-mask can be removed to simulate the case of variable phase sequence.

3. **Cycle-Based**

(a) Variable Phasing plan, Variable Cycle time, Variable Phase distribution

(b) Fixed Phasing plan, Variable Cycle time, Variable Phase distribution

(c) Phase-Split: Fixed Phasing plan, Fixed Cycle time, Variable Phase distribution

3.3 Rewards

Please refer to Section 2.4 of Coordinated Traffic Control report.

3.4 Evaluation Metrics

Please refer to Section 2.4 of Coordinated Traffic Control report.

4 Summary

We present this technical report about the State-of-the-art Deep Reinforcement Learning methods. We divide the algorithms in the literature into different categories based on their properties such as model-free or model-based, single-agent or multi-agent, and discrete action or continuous action. This delineation also allows us to create separate action-space configurations for the environment. And depending upon the action-space configuration chosen, there are a limited number of compatible algorithms that one can test against. This allows for easier testing in future and ease in finding the optimal traffic signal controller.

References

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. In *Proceedings of the 1993 Connectionist Models Summer School Hillsdale, NJ. Lawrence Erlbaum*, 1993.

Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003, 2016.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. *arXiv preprint arXiv:1707.06887*, 2017.

Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado Van Hasselt, and David Silver. Distributed prioritized experience replay. *arXiv preprint arXiv:1803.00933*, 2018.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, Dhruva Tb, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. *arXiv preprint arXiv:1804.08617*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565, 2019a.

Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.

Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*, 2019b.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Wade Genders and Saiedeh Razavi. Using a deep reinforcement learning agent for traffic signal control. *arXiv preprint arXiv:1611.01142*, 2016.

Soheil Mohamad Alizadeh Shabestary and Baher Abdulhai. Deep learning vs. discrete reinforcement learning for adaptive traffic signal control. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 286–293. IEEE, 2018.