The Cost of OPS in Reinforcement Learning

Yao Fu

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-128
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-128.html

May 14, 2021

The Cost of OPS in Reinforcement Learning

by

Yao Fu

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Pieter Abbeel, Chair
Professor John DeNero

Spring 2021

The Cost of OPS in Reinforcement Learning

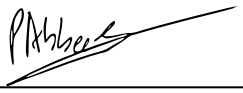
by Yao Fu

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:



Professor Pieter Abbeel
Research Advisor

5/12/2021

(Date)

* * * * * * *

Professor John DeNero
Second Reader

5/14/21

(Date)

The Cost of OPS in Reinforcement Learning

Abstract

The Cost of OPS in Reinforcement Learning

by

Yao Fu

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Pieter Abbeel, Chair

It is typical to ignore the cost of computing an observation or action in the perception-action loop, such that the agent is free to sense the environment and prepare its decision at length before time steps forward. Decision making and dynamics of the environment are treated as *synchronous*. Yet, the need to act and react efficiently is a basic constraint in natural environments that shapes the behavior of animals. We consider a setting in which the environment is *asynchronous*, and the computational components of decision making such as observation, prediction, and action selection involve associated costs. The costs of operations affect the policy by inducing a need to trade off between them, and can be incorporated in the learning setting as an intrinsic reward function. As a first attempt, we develop a simple hierarchical approach that adaptively chooses between OPs – explicitly **o**bserving or implicitly **p**redicting – in order to update its hidden state, and analyze emergent strategies on a number of environments involving partial observability and stochastic dynamics.

Contents

Contents i

List of Figures ii

1 Introduction 1

2 Approach 3
- 2.1 Preliminaries: 'Synchronous' RL . . . 3
- 2.2 Operations and their Costs . . . 3
- 2.3 A Simple Hierarchical Model . . . 4
- 2.4 Objectives . . . 4

3 Experiments 6
- 3.1 Dynamic Obstacles . . . 6
- 3.2 Multi-Room Navigation . . . 7
- 3.3 Pixels . . . 8

4 Related Work 12

5 Conclusion 13

Bibliography 14

List of Figures

2.1 Computation graph of our proposed method. 4

3.1 An 8x8 dynamic obstacle environments. The partially observable view of the agent is highlighted. The red triangle is the current position of the agent. The goal is to reach the green goal square while avoiding colliding with any blue obstacle. 7

3.2 Top: Histogram of agent's distance to the nearest obstacle when it observes vs predicts. Bottom: proportion of prediction. The policy predicts more often when the nearest obstacle is further away from the agent. 7

3.4 An N4S5 environment. The arrow indicates the position and the direction of the agent. The agent should navigates through all the rooms to reach the green goal position. 7

3.3 Comparison of using different bonus coefficients λ. All the experiments start with no bonus and we start to add bonus reward after the agent is able to solve the task. Adding reward bonus significantly boosts the proportion of prediction. However, if the bonus is too large, π_g will converge to always predict and the task will fail. 8

3.5 Comparison of performance and proportion of prediction with different bonus coefficients. Adding reward bonus significantly boost the proportion of prediction, while still keeping the overall performance of the policy. However, if the bonus is too large, the gating policy will predict too much, which degrades performance. 9

3.6 Eight example episodes of N4S5(Left) and N6S6(Right). The figures show the agent's path towards the green goal square, in which the arrow indicates the direction and its color indicates its decisions at the position: Red-Observe, Blue-Predict, Pink-Both. The agent predicts very often when it is inside a room and mostly observes when it is in front of a door. It usually predicts to open the door and observes after that to get the information of the new room, therefore the arrow right before the door usually appears pink. 9

3.7 Comparison of our method with and without bonus reward for prediction in terms of proportion of prediction(Left) and the cumulative reward(Right). Each step in the plot corresponds to 8k environment steps. Adding bonus reward after the agent solves the task significantly boosts the proportion of prediction, while keeping the performance. . 10

3.8 Four episodes collected by our model. The images above show when the image observe(Left) and predict(Right). In these four episodes, the agent predicts about 73.8% of the time. It is able to "predict" a sequence of enemies' actions based on the limited information it gets from the environment. 11

Acknowledgments

I would like to thank my advisors, Professor Pieter Abbeel and Professor Lerrel Pinto at NYU, for their support, advice, and the opportunity to work in the Robot Learning Lab. I am also extremely grateful to Allan Jabri for his mentorship, guidance, and assistance. This experience has helped me learn a lot!

Chapter 1

Introduction

Reinforcement learning is an exceedingly general paradigm for formalizing decision-making, in a manner that allows for abstracting away complications faced by agents in natural environments. Paired with powerful function approximators, this flexibility has led to successful application to a range of challenging settings from game-playing to continuous control, often in simulation. This progress has been fueled by consolidation of experimentation interfaces and benchmarks, with a focus on developing general-purpose policy optimization algorithms applicable across environments, at the cost of standardization. While there is growing interest in addressing challenges of the real-world [3, 2], most of these efforts focus on how realistic settings involve complications, and how these complications can be neutralized. In this work, we ask whether aspects of natural environments – rather than be mere complications to be abstracted away – might serve as important incentives for development of behaviour.

One aspect of natural environments that is typically overlooked is the cost – in time and energy – of making a decision. While it is common to treat the perception-action loop as proceeding synchronously with environment dynamics, natural environments do not wait for action – they are *asynchronous*. Efficient decision-making is a fundamental aspect of evolutionary fitness, for example, when it comes to fleeing from predators, trapping prey, or overcoming chaotic dynamics. Decision making can been optimized by adaptive computation: sensing the environment might not be so necessary if it can be easily predicted; frequently encountered tasks might increasingly rely less on planning and become more reactive; and time and energy that is freed up can be dedicated to preparing for the future. These emergent strategies are by no means a direct byproduct of environment asynchrony, but they are shaped by it. Indeed, the idea that intelligent animals like humans behave optimally is unrealistic; rather, humans decision making is subject to bounded resources, plagued by cognitive biases, and reduced to a so-called *bounded rationality* [9]. While the idea of conditioning computation on context is not new, our interest is in how the costs of different pathways of computation can play a role in learning.

In this work, we split the perception-action loop into computational components with associated costs. As a first attempt, we consider two pathways for perception – an explicit observation pathway and implicit prediction pathway – which are followed by action selection. We will say that

the explicit observation pathway is more *expensive* than the prediction pathway. To incorporate operation cost into the learning setting, we express it as an intrinsic reward function. We experiment in a number of environments with varying aspects of stochasticity and partial observability, and analyze how agents learn to adaptively trade off observation and prediction while learning to solve tasks.

Chapter 2

Approach

In the following, we first describe how operation cost transforms the RL setting, and then describe a simple hierarchical model equipped to condition its operation on context.

2.1 Preliminaries: 'Synchronous' RL

Consider the typical setting of a discrete-time finite-horizon discounted Markov decision process (MDP) by a tuple $M = (S, A, P, r, s_0, \gamma, T)$, with state space S, action space A, transition probability distribution $P : S \times A \times S \to [0, 1]$, reward function $r : S \times A \to R$, initial state distribution $S_0 : S \to [0, 1]$, discount factor $\gamma \in [0, 1]$, and horizon T.

As usual, the goal of policy search is to find a policy $\pi_\theta : S \times A \to R+$ which maximizes the expected discounted return, $E_\tau[\sum_{t=0}^{T} r(s_t, a_t)]$, where $\tau = (s_0, a_0, ...)$ denotes the whole trajectory generated by applying the policy $a_t \sim \pi_\theta(a_t|s_t)$, from an initial state s_0 under transition function P.

2.2 Operations and their Costs

Operation cost is inherently policy-specific: we need to first decompose the computation performed by the overall policy π in order to attribute costs. For simplicity, here we will say that the policy can be split into perception and action selection. That is, given an input s_t, the policy computes an intermediate hidden state $h_t \in R^D$, from which an action distribution is produced a_t. We will take particular interest in the formation of h_t, assuming that this is where much computation is performed.

Consider alternative computational pathways $f_i : S \to R^D$ for producing the hidden state. For each pathway, associate cost parameters $(c_i^{\texttt{time}}, c_i^{\texttt{energy}})$, where the former represents the temporal cost (i.e. a *delay*) and the latter represents an energy cost (i.e. a reward penalty). These costs can be viewed as transforming the effective MDP by supplementing the reward function with an

additional intrinsic reward and altering the transition function to reflect delays. Concretely, this can be achieved by augmenting the action space to include pathway selection and augmenting the state space to mark which pathway is chosen. In essence, the resulting MDP encapsulates an *inner* decision making process of making a decision.

2.3 A Simple Hierarchical Model

We now describe a concrete instantiation of the framework. For simplicity, our model considers two perception pathways:

- An **encoder** $\phi : S \to R^D$, which encodes the observation of state into a latent space, forming the hidden state directly. Let $c_\phi^{\texttt{time}} = k, c_\phi^{\texttt{energy}} = k$.
- A **predictor** $f : R^D \times A \to R^D$, which predicts a hidden state given only the hidden latent state and action. Let $c_\phi^{\texttt{time}} = 1, c_\phi^{\texttt{energy}} = 1$.

Given these alternative perception pathways, for which encoding is k times more expensive than predicting, the policy π is formed by:

- The **gate** $\pi_g : R^D \to \{0, 1\}$, which chooses the perception pathway conditioned on the previous hidden state.
- The **actor** $\pi_a : R^D \to A$, which computes an action from the updated hidden state.

Let h_t be the hidden state, initialized to $\mathbf{0}$. At every time-step, the agent computes the following.

$$\begin{aligned} \textbf{Gate:} \quad & g_t \sim \pi_g(\cdot|h_{t-1}) \\ \textbf{Observe:} \quad & z_t = \phi(o_t) \\ \textbf{Predict:} \quad & \hat{z}_t = f(h_{t-1}, a_{t-1}) \\ \textbf{Update:} \quad & h_t = g_t * z_t + (1 - g_t) * \hat{z}_t \\ \textbf{Act:} \quad & a_t = \pi_2(\cdot|h_t) \end{aligned}$$

Figure 2.1 depicts this resulting perception-action loop. Note that the dashed box delineates a loop capable of producing a sequence of actions without constant access to the ground-truth state.

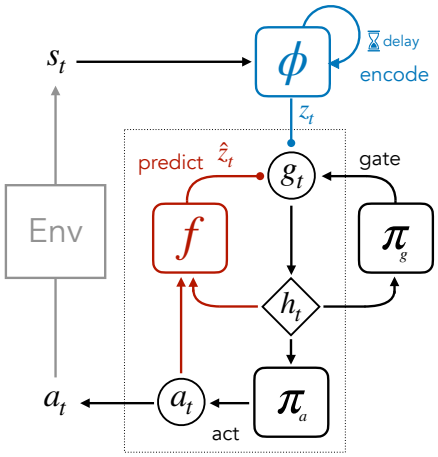


Figure 2.1: Computation graph of our proposed method.

2.4 Objectives

Reward functions

$$r_{\texttt{ops}}(t) = g_t$$
$$r_g(s_t) = r_{\texttt{ext}}(s_t) + \lambda r_{\texttt{ops}}(t)$$

where λ weighs intrinsic reward reflecting the cost of operation and $r_{\texttt{ext}}(s_t)$ is the reward returned by the environment. We use $r_{\texttt{ext}}(s_t)$ as reward to optimize π_2 and use $r_g(s_t)$ to optimize the gating policy g_t.

Chapter 3

Experiments

We experiment with the proposed model in a range of environments that involve stochastic dynamics and partial observability:

- Toy, illustrative examples: Gym Mini-grid [1]
- Pixels: Vizdoom [11]

In the experiments below, we aim to examine three questions: 1) Does our new reward function enable the agent to sometimes predict instead of always observe? 2) Is the implicitly trained predictor f useful for solving the task? 3) How does π_g choose to predict or observe?

3.1 Dynamic Obstacles

We run our method in a gridworld environment with randomly moving obstacles to see how our model balances between capturing the movement of the obstacles and minimizing the cost of operation. The dynamic obstacles environment is a room with randomly moving obstacles. As shown in figure 3.1, the agent starts at the top-left corner of the room and aims to reach the goal square at the bottom-right corner. At the same time, it should avoid colliding with any obstacle. It is rewarded for moving closer to the goal and a large penalty will be subtracted if the agent dies. The environment is partially observable: the agent only sees a 7x7 box.

We experiment in a 8x8 room with 4 obstacles. Figure 3.3 shows the comparison of performance using bonus rewards. The agent starts with randomly choosing to predict or observe with equal probabilities at the beginning. Without any bonus reward, the agent quickly learns to observe all the time. After 120 epochs, when the agent is able to solve the task, we start adding $r_{\texttt{ops}}$ as a part of the reward function to encourage the agent to do some predictions. With an appropriate bonus, the agent is able to substitute half of the observations with predictions while still safely reaching the goal, which greatly reduces the total cost.

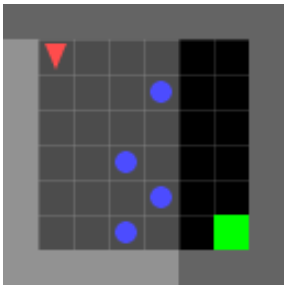


Figure 3.1: An 8x8 dynamic obstacle environments. The partially observable view of the agent is highlighted. The red triangle is the current position of the agent. The goal is to reach the green goal square while avoiding colliding with any blue obstacle.

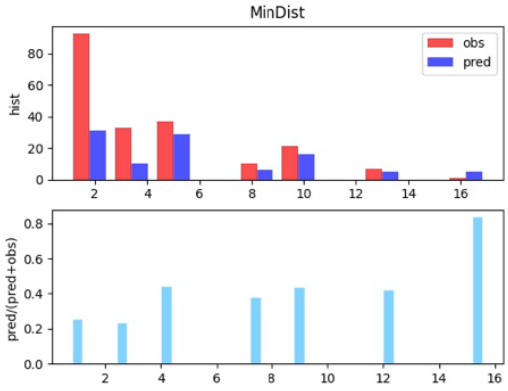


Figure 3.2: Top: Histogram of agent's distance to the nearest obstacle when it observes vs predicts. Bottom: proportion of prediction. The policy predicts more often when the nearest obstacle is further away from the agent.

.

How does the agent decide to predict or observe in this case? We collect 8 episodes using a trained model to find the relationship of the agent's decisions with its distance to the obstacles. Figure 3.2 shows that correlation exists between the agent's distance to the closest obstacle and its decision. When the agent is further away from the obstacles, it is less possible to be attacked, therefore the model predicts more often.

3.2 Multi-Room Navigation

In navigation tasks, the agent only gets limited new information every timestep due to the fact that its original field of view and the new field of view usually overlap a lot. Therefore, it is not necessary to keep observing. We experiment our model in the gym minigrid partially-observable multi-room environments to see if the agent is able to learn to only observe every time it's totally out of the field it sees last time. A multi-room environment contains a set of rooms with random size and the agent must open a closed door in order to enter a new room. The goal of the agent is to navigate

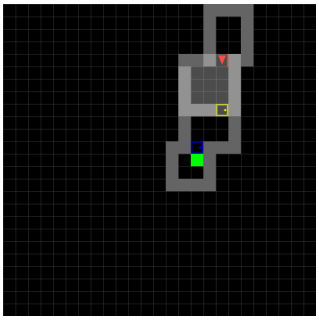


Figure 3.4: An N4S5 environment. The arrow indicates the position and the direction of the agent. The agent should navigates through all the rooms to reach the green goal position.

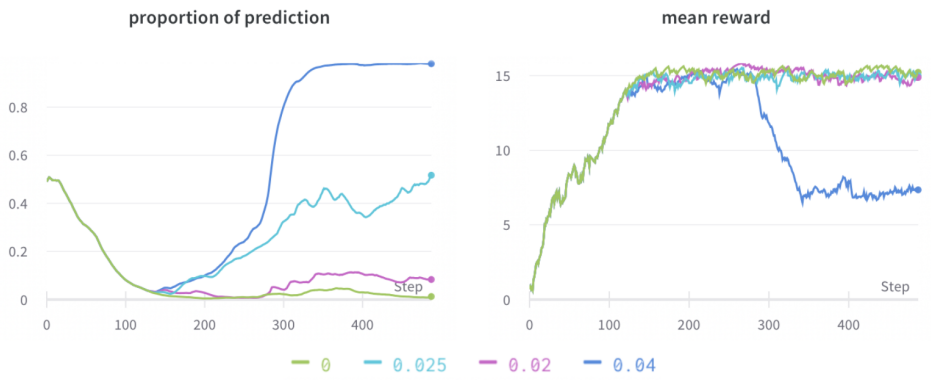


Figure 3.3: Comparison of using different bonus coefficients λ. All the experiments start with no bonus and we start to add bonus reward after the agent is able to solve the task. Adding reward bonus significantly boosts the proportion of prediction. However, if the bonus is too large, π_g will converge to always predict and the task will fail.

through the rooms to reach the goal, which is placed in the last room. The agent is only allowed to see a 7x7 box and it can't see though the walls nor a closed door. Therefore, we expect the agent to only observe when it enters a new room or when it arrives at somewhere that is too far from the door of the current room.

We ran our model on two environments: N4S5(four rooms, the max size of each room is 5x5) and N6S6(six rooms, the max size of each room is 6x6). Figure 3.5 shows how our model works with different λ. In both environments, the higher the bonus is, the more it is going to predict. The agent is able to reach the goal with appropriate λ. We also show eight episodes for both environments in figure 3.2, indicating the sequence of actions the agent takes to get to the goal. The agent observes more often after it opens a door and predicts more often when it is inside a room, which matches our expectation.

3.3 Pixels

Pixel-based environments are a more realistic example because encoding an observation is time and energy consuming. In some environments, the agent should react quickly to overcome the change of the environment. Thus, predicting is necessary to save time. Vizdoom environments with monsters is such an example. The agent is placed at the center of a large circle. Monsters are spawned along the wall and will approach the agent to attack him. The observation is the agent's view and the agent is rewarded every time it kills a monster. The goal of the agent is to kill as many monsters as possible with limited number of ammo and at the same time avoid the attack of the monsters.

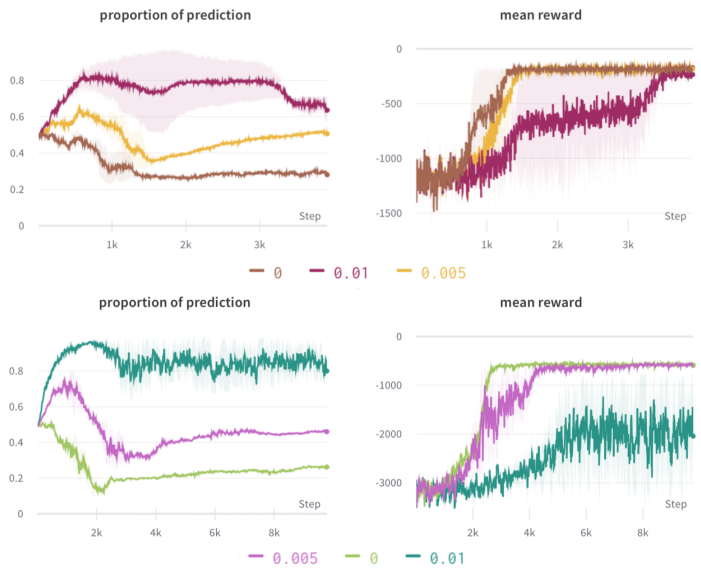


Figure 3.5: Comparison of performance and proportion of prediction with different bonus coefficients. Adding reward bonus significantly boost the proportion of prediction, while still keeping the overall performance of the policy. However, if the bonus is too large, the gating policy will predict too much, which degrades performance.

Figure 3.6: Eight example episodes of N4S5(Left) and N6S6(Right). The figures show the agent's path towards the green goal square, in which the arrow indicates the direction and its color indicates its decisions at the position: Red-Observe, Blue-Predict, Pink-Both. The agent predicts very often when it is inside a room and mostly observes when it is in front of a door. It usually predicts to open the door and observes after that to get the information of the new room, therefore the arrow right before the door usually appears pink.

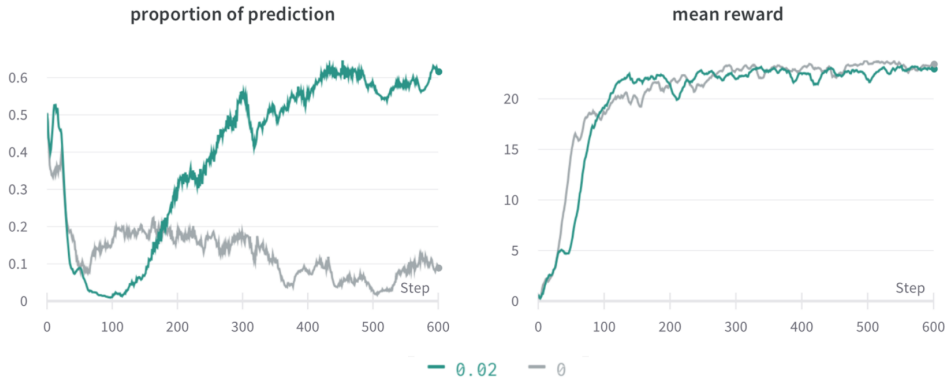


Figure 3.7: Comparison of our method with and without bonus reward for prediction in terms of proportion of prediction(Left) and the cumulative reward(Right). Each step in the plot corresponds to 8k environment steps. Adding bonus reward after the agent solves the task significantly boosts the proportion of prediction, while keeping the performance.

We ran our method on this environment to test if our model is able to find a balance between killing the most monster and saving the cost of encoding images by predicting the movement of the monsters. Figure 3.7 shows how adding $r_{\texttt{ops}}$ encourages the agent to predict more often. The selected hyper-parameters boost the proportion of prediction to above 60%, which significantly reduces the total cost to decode observations. In figure 3.8, we also collect 4 episodes and stack all the cases in which the agent observes vs predicts. Notice that the movement of the monsters is not fully predictable because they changes their moving direction every several steps. As a result, the agent has to observe periodically. However, from the visualization we can see that the agent is able to "predict" a large proportion of the monsters' movement using only limited observations.

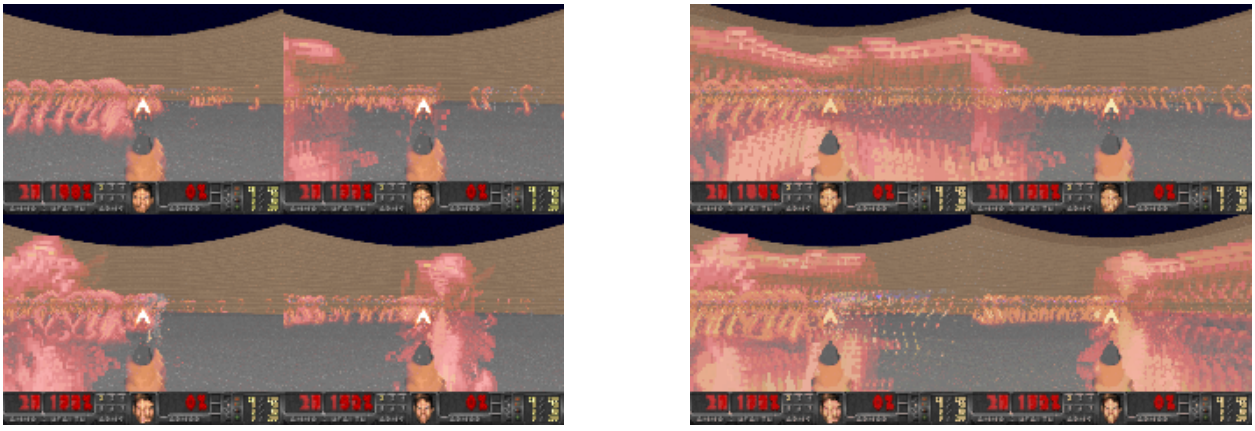


Figure 3.8: Four episodes collected by our model. The images above show when the image observe(Left) and predict(Right). In these four episodes, the agent predicts about 73.8% of the time. It is able to ”predict” a sequence of enemies’ actions based on the limited information it gets from the environment.

Chapter 4

Related Work

Model-based reinforcement learning. Model-based reinforcement learning methods are applied to reduce sample complexity by learning a model of the environment it interacts with. In Dyna-Style algorithms [10, 6, 5], a prediction model is trained explicitly using the collect data from the environment, and then it generates imagined data to train the policy. However, in [4], the author propose a new model training method, in which the model not trained explicitly to predict a next observation, instead, they randomly do observational dropout and train an RNN to fill in observation gaps. Similarly, our model is also implicitly trained using an RNN. However, instead of dropping observations with a preset probability, our agent determines to substitute the observation with prediction or not based on its memory.

Conditional Computation. Adapting or conditioning computation on input has been widely explored. In [8], the authors propose a model that consists of many experts and a a trainable gating network, which chooses a part of the experts to process each input. Conditioning computation on the input has also been shown to be an effective inductive bias for incorporating side-information in an expressive yet simple way [7]. Other approaches adapt computation on the basis task difficulty, allocating more compute for challenging examples, with a focus on resource allocation. In our work, we explore settings in which conditioning computation to follow pathways of alternative cost is necessary to maximize a reward function.

Real-world challenges of RL. There has been a recent interest in settings Real-world challenges of RL. In [3], the authors proposed a set of nine unique challenges that should be considered when applying RL in real world, including potential system delays in real systems. They experimentally show that observation, action, and reward delays greatly influence the agent's performance. In our work, we try to minimize the influence of observation delay, but trying to dynamically drop observations and use predictions to generate actions.

Chapter 5

Conclusion

In this work, we propose a model with two perception pathways: an encoder that encodes observations directly and a predictor that uses the past hidden states and actions to predict the latent of the current state. We assume that in real world, encoding an observation is more experiment than predicting in terms of the cost of time and energy. Our model uses a gating policy π_g to choose to observe or predict and encourages predicting by adding bonus reward to the predictions. It is able to find a balance between observation and prediction and make sure the policy solves the tasks. For future work, we want to consider more realistic cases where the cost of time and energy directly influence the performance of the policy, for example, environments with delayed observations, in which it takes more than one step for the agent to get an observation but it takes less time to predict. Therefore we expect the model to learn to predict as an emergent behavior even without reward bonus.

Bibliography

[1] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. *Minimalistic Gridworld Environment for OpenAI Gym*. https://github.com/maximecb/gym-minigrid. 2018.

[2] Gabriel Dulac-Arnold, Daniel Mankowitz, and Todd Hester. *Challenges of Real-World Reinforcement Learning*. 2019. arXiv: 1904.12901 [cs.LG].

[3] Gabriel Dulac-Arnold et al. *An empirical investigation of the challenges of real-world reinforcement learning*. 2020. arXiv: 2003.11881 [cs.LG].

[4] C. Daniel Freeman, Luke Metz, and David Ha. *Learning to Predict Without Looking Ahead: World Models Without Forward Prediction*. 2019. arXiv: 1910.13038 [cs.NE].

[5] Michael Janner et al. *When to Trust Your Model: Model-Based Policy Optimization*. 2019. arXiv: 1906.08253 [cs.LG].

[6] Yuping Luo et al. *Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees*. 2019. arXiv: 1807.03858 [cs.LG].

[7] Ethan Perez et al. *FiLM: Visual Reasoning with a General Conditioning Layer*. 2017. arXiv: 1709.07871 [cs.CV].

[8] Noam Shazeer et al. *Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer*. 2017. arXiv: 1701.06538 [cs.LG].

[9] Herbert A Simon. "Theories of bounded rationality". In: (1972).

[10] Richard S. Sutton. "Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming". In: *In Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann, 1990, pp. 216–224.

[11] Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. "ViZDoom Competitions: Playing Doom from Pixels". In: *IEEE Transactions on Games* (2018).