Reinforcement Learning in the Presence of Sensing Costs

by

Tzu-Yun Ariel Shann

B.Sc., National Tsing Hua University, 2017

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

April 2022

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Reinforcement Learning in the Presence of Sensing Costs

submitted by **Tzu-Yun Ariel Shann** in partial fulfillment of the requirements for the degree of **Master of Science** in **Computer Science**.

Examining Committee:

Leonid Sigal, Associate Professor, Computer Science, UBC
Supervisor

Michiel van de Panne, Professor, Computer Science, UBC
Co-Supervisor

Jeff Clune, Associate Professor, Computer Science, UBC
Supervisory Committee Member

Abstract

In recent years, reinforcement learning (RL) has become an increasingly popular framework for formalizing decision-making problems. Despite its popularity, the use of RL has remained relatively limited in challenging real-world scenarios, due to various unrealistic assumptions made about the environment, such as assuming sufficiently accurate models to train on in simulation, or no significant delays between the execution of an action and receiving the next observation. Such assumptions unavoidably make RL algorithms suffer from poor generalization. In this work, we aim to take a closer look at how incorporating realistic constraints impact the behaviour of RL agents. In particular, we consider the cost in time and energy of making observations and taking a decision, which is an important aspect of natural environments that is typically overlooked in a traditional RL setup. As a first attempt, we propose to explicitly incorporate the cost of sensing the environment into the RL training loop, and analyze the emerging behaviours of the agent on a suite of simulated gridworld environments.

Lay Summary

Reinforcement learning (RL) is a popular framework to model decision making, where an agent learns by interacting with the environment. In a typical RL setup, at each time step, the agent receives an observation, decides on an action, and receives feedback from the environment. However, RL remains relatively limited in real-world scenarios, as natural complications such as delays in time or costs in energy are typically ignored in standard setups. In this work, we explore the possibility of exploiting these complications as learning cues. By explicitly incorporating sensing costs into the RL problem, we show that it allows the agent to learn to make "smarter" decision, using less time and less resources. We argue that the problem setting we explore is particularly suitable for real-world challenges, where efficiency is crucial.

Preface

The presented work is originally and entirely done by the author, Tzu-Yun Ariel Shann, at the University of British Columbia, under the supervision of Prof. Leonid Sigal and Prof. Michiel van de Panne.

Model design, code implementation, experiments and writing are done by the author alone, with useful and constructive feedback from Prof. Leonid Sigal and Prof. Michiel van de Panne. Specifically, Prof. Leonid Sigal kindly shared his advice on model design, as well as debugging tips. Prof. Michiel van de Panne provided his expertise in reinforcement learning and shared insights along every step. Both of them also helped with revising the draft of this thesis.

This study does not involve any human subjects and is done entirely by me, hence no Ethics Board approval was required.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a general paradigm for solving sequential decision-making problems, where an agent interacts with an environment in a sequence of actions, observations and rewards. Unlike supervised learning wherein explicit supervision is provided, an RL algorithm must instead learn from feedback in the form of rewards, which is frequently sparse, noisy and delayed, and in a trial-and-error fashion. An optimal policy can be learned by maximizing expected cumulative rewards. However, traditional RL methods are limited to tabular settings, and thus are infeasible to deal with high-dimensional or continuous state spaces, limiting generalization. In the last few years, equipped with deep neural networks as function approximators and through representation learning, RL has been successfully applied to more complex environments, e.g., using pixel observations, and has achieved remarkable success in a wide range of sequential decision making problems, such as game-playing [15, 22], robotics [13], self-driving cars [12], and natural language processing [27, 29].

Despite the progress in various domains, there are still a number of limitations and challenges that hinder the applicability of RL. It has been shown that existing RL algorithms exhibit high sensitivity to the choice of hyperparameters [14]. Retuning these hyperparameters may thus be necessary for every new task, which can become expensive both in terms of time and computational costs. This is especially undesirable in real-world scenarios where sample efficiency is crucial. Another major barrier to broader applicability comes from the design of reward functions–

many real-world tasks have complex reward structures that humans understand via high-level concepts, but it may be difficult or prohibitively expensive to design such a reward function by hand. For example, consider a driving task in a busy city, where humans know the importance of the high-level concept of safety. However, there are many underlying objectives we need to trade off, such as maintaining safe distance, not changing lane too often, staying far from any pedestrians, and so on, which will be hard to specify manually for an agent. Moreover, current RL algorithms have been shown to suffer from poor sample efficiency– it can take tens to hundreds of millions of samples to train an RL agent, which makes many practical applications infeasible, especially in real-world control problems where data collection is expensive.

Thus far, most RL successes depend on the availability of a simulator, which makes a series of assumptions that are not always satisfied in practice. One prominent difference between such simulations and real sensorimotor systems lies in time delays, in either the sensing of the state, the actuators, or the reward feedback [1, 8]. Indeed, such sensorimotor delays are inherent in many systems, in both humans and animals, and our ability to move or control our actions is constrained by this time-sensitive relationship between sensations and motor outputs. Consider the scenario where an animal is trying to flee from its predator– the time required to perceive a stimulus (predator) and to produce a response (escape) is crucial to its survival. Thus, such sensorimotor delays have been shown to impact motion strategies. For example, additional mechanisms, such as sensorimotor prediction, using "efficient copies" of motor commands may be needed to compensate for sensorimotor delays. Apart from the different dynamics, the observation discrepancy between simulated environments and the real world also makes it challenging for RL to generalize [18].

In this work, however, instead of trying to find a way to be robust to these natural complications, we are instead interested in **whether they might serve as important constraints for an agent when learning to act optimally in the environments?**

An essential aspect of natural environments that is typically overlooked is the cost in time and energy of making an observation. In RL, it is commonly assumed that the observation of the next state comes instantaneously and at no cost, which

is generally not true in practice. Furthermore, while it is possible to sense the environment at every single timestep, it might not be necessary for optimal performance if the environment can already be easily predicted. Instead, it might be better to conserve resources in preparation of future events.

Specifically, we consider the scenario where there is a cost associated with making an observation. At each timestep, the agent can either choose to observe at a cost, or to skip sensing the environment (at no cost). We use frame-skipping as a proxy of not observing, and implement a two-part policy, where there is an actor for action selection and another policy to determine the number of skips. To effectively incorporate sensing costs into the problem, we express it as an intrinsic reward function, in which choosing to skip is a less costly operation. Under this assumption, we hypothesize the optimal strategy for the agent is to skip more (save resources) when the environment can be easily predicted, and alternatively, to choose to pay the cost of frequent observations in regions of uncertainty. We experiment in a variety of grid-world environments with varying degrees of difficulty and stochasticity, and analyze how agents learn to observe only when it is necessary, thus making fewer decision steps compared to traditional RL baselines.

Chapter 2

Related Work

2.1 Reinforcement learning

Reinforcement learning (RL) is a popular formulation for solving sequential decision making problems [24]. In the RL setup, an agent observes a state S_t from its environment $\mathscr{E}$ at timestep t, and then chooses an action A_t in state S_t based on its policy π_θ. Once the action is executed, the environment transitions to a new state S_{t+1} and provides feedback in the form of a reward R_{t+1}. The agent thus learns to alter its behaviours in response to the rewards received. This perception-action-learning loop is illustrated in Figure 2.1.

Formally, the environment $\mathscr{E}$ is usually characterized by a Markov decision process (MDP):

$$M = (S, A, P, r, s_0, \gamma, T),$$

with state space S, action space A, transition probability $P : S \times A \times S \to [0,1]$, reward function $r : S \times A \to R$, initial state distribution s_0, discount factor $\gamma \in [0,1)$, and horizon T. The goal of the agent is to learn a policy $\pi_\theta : S \times A \to R+$ such that it maximizes the expected return:

$$G_\tau = \mathbf{E}_\tau \left[\sum_{t=0}^{T} \gamma^k r(S_t, A_t) \right],$$

where τ denotes the trajectory generated by π_θ parameterized by θ.

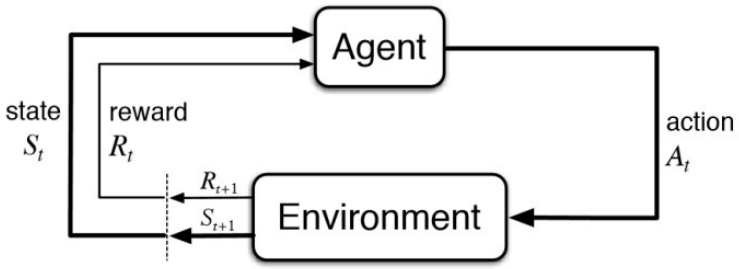


Figure 2.1: A typical perception-action-learning loop of reinforcement learning. At each timestep, the agent perceives a state S_t, and chooses an action A_t according to its current policy. The environment then transitions into a next state S_{t+1} and provides a reward R_{t+1}.

Recently, with the breakthrough of deep learning, there is also a renewed interest in RL. A plethora of deep RL algorithms and techniques have been proposed to tackle more challenging problems [13, 15, 22]. We discuss distinctions between some general categories of RL algorithms in the following.

Model-based and model-free methods. In model-based approaches, a predictive model of the environment is used to learn the control policy. While in model-free methods, the modeling step is bypassed and a control policy is learned directly.

On-policy and off-policy methods. In on-policy learning, the algorithm is trained using samples produced by the current policy, whereas off-policy methods are independent of the policy used to generate behaviours.

Value-based and policy-based methods. Value-based methods rely on the value function– the value (expected return) of being in a given state. The policy is implicit and can be derived by choosing the action that results in the best value. On the other hand, policy-based approaches do not maintain a value function, but have an explicit representation of the policy and directly searches for an optimal policy. It is possible to combine value functions with an explicit policy, which results in actor-critic methods, where the "actor" (policy) learns by using feedback from the "critic" (value function).

2.1.1 Q-learning

In this work, we focus primarily on Q-learning, which is a model-free, off-policy and value-based RL algorithm. Q-learning is based on Q-value, or state-action value, which is the expected return of a state-action pair (s_t, a_t) at timestep t:

$$Q_\pi(s,a) = \mathbf{E}_\pi \left[G_t \mid S_t = s, A_t = a \right].$$

The best policy can be found by choosing a greedily at every state, i.e. $\arg\max_a Q_\pi(s,a)$. To actually learn Q_π, we exploit the Markov property and define the function as a Bellman equation [4], which has the following recursive form:

$$Q_\pi(S_t, A_t) = \mathbf{E}_{S_{t+1}} \left[R_{t+1} + \gamma Q_\pi(S_{t+1}, \pi(S_{t+1})) \right].$$

It can then be learned iteratively by:

$$Q_\pi(S_t, A_t) \leftarrow Q_\pi(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in A} Q_\pi(S_{t+1}, a) - Q_\pi(S_t, A_t) \right],$$

where α is the learning rate.

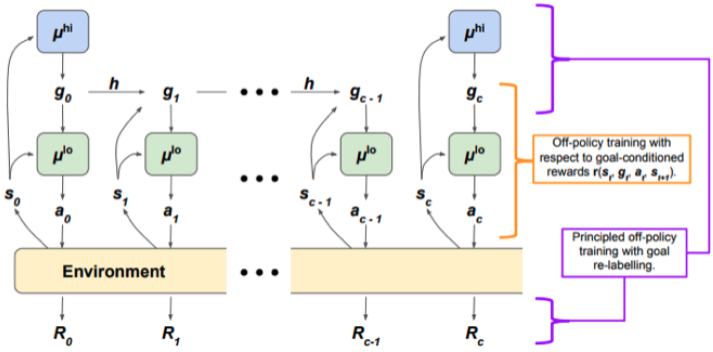
Recently, Deep Q-Networks [16] (DQN) makes use of machine learning models to approximate Q values, which brings together classic RL algorithms and deep learning. With some innovative mechanisms, such as experience replay that allows more efficient use of previous experience, DQN is shown to improve and stabilize Q-learning.

2.2 Temporal abstraction

Learning and operating over different levels of temporal abstraction is a long-standing challenge in RL, especially for tasks that require long-term planning. A common framework for temporal abstractions is the options framework [19, 25], which involves two levels of abstractions over actions. The bottom level is called an *option*, which is a sub-policy with a termination condition. It takes in observations, and outputs actions until the termination condition is met. The top level, on the other hand, is a policy over options, which picks an option to follow until its termination. There is a large body of works on techniques for learning options. How-

ever, many existing methods make assumptions about the task structures, and/or the skills needed to solve the tasks, thus they may require intermediate supervisory signals such as pseudo-rewards [10, 21] or pre-defined subgoals [6]. To avoid dependence on prior knowledge and manual design in the task space, some methods of automatic options discovery have also been proposed to learn these abstractions from scratch [28].

Recently, many works on hierarchical RL have made a departure from the options framework [17, 26]. Instead of having a policy over options as the top level, these approaches learn the higher-level policy that produces meaningful and explicit goals for the bottom level to achieve (Figure 2.2).



1. Collect experience $s_t, g_t, a_t, R_t, \ldots$.
2. Train μ^{lo} with experience transitions $(s_t, g_t, a_t, r_t, s_{t+1}, g_{t+1})$ using g_t as additional state observation and reward given by goal-conditioned function $r_t = r(s_t, g_t, a_t, s_{t+1}) = -||s_t + g_t - s_{t+1}||_2$.
3. Train μ^{hi} on temporally-extended experience $(s_t, \tilde{g}_t, \sum R_{t:t+c-1}, s_{t+c})$, where $\tilde{g}_t$ is relabelled high-level action to maximize probability of past low-level actions $a_{t:t+c-1}$.
4. Repeat.

Figure 2.2: An instance of recent hierarchical RL methods. In hierarchical RL, the lower-level policy (μ^{lo}) interacts directly with the environment. The high-level policy (μ^{hi}) instructs the low-level policy via high-level actions or goals. Figure borrowed from [17].

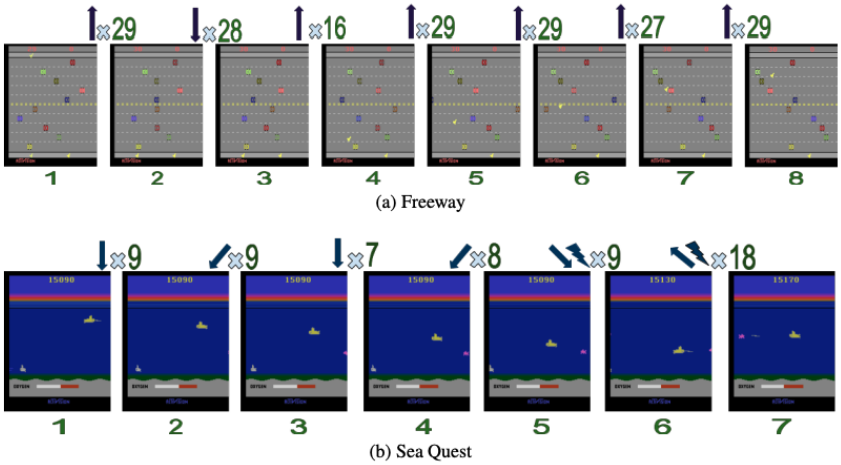


Figure 2.3: An schematic overview of dynamic-frame skipping in FiGAR, borrowed from [20]. The arrows represent the action taken between frames, the thunderbolt indicates the firing action, and the numbers indicates the number of time steps for which the action was repeated.

2.3 Frame-skipping

Frame-skipping is a technique commonly used in deep RL algorithms, where a chosen action is repeated for a fixed number of timesteps k. If a_t represents the action taken at timestep t, then $a_1 = a_2 = \cdots = a_k$, $a_{k+1} = a_{k+2} = \cdots = a_{2k}$ and so on. This allows deep RL algorithms to compute the action once every k timesteps, reducing computational cost [9, 20], and can also be viewed as an alternative and naive way to achieving temporal abstractions. On the popular Atari benchmark [2], frame-skipping is first applied with the intent to alleviate the computation load caused by sensing pixel observations. However, subsequent research has shown that better performance can be achieved by skipping up to 180 frames in some games [5]. In the past, the number of frame skips has largely been static, but recently there are a few works that explore dynamic frame skips and show that it leads to significant improvement in some simulated environments [23].

FiGAR [20] is one generic framework that enables deep RL algorithms to learn temporal abstractions in the form of temporally extended macro-actions (Figure 2.3). Instead of setting a pre-specified k, FiGAR decouples the policy into two

independent components: one for choosing the action, and the other for selecting the number of frame skips. The authors have shown that FiGAR can be used to improve current deep RL algorithms, and learns better control policies by discovering optimal sequences of temporally elogated macro-actions.

2.4 RL with State-sensing Costs

Standard RL algorithms assume that the observation of the next state comes instantaneously and at no cost. However, this is generally not true in natural environments, where making an observation often has its associated cost in both time and energy. In such scenarios, efficient decision making is especially crucial, and sensing the environment might not be necessary if it can already be easily predicted. It may be a better strategy to save energy whenever possible instead for future uses. While costs are inherent in real-world systems, to the best of our knowledge, there has been little work that explores the roles of such sensing costs in the scope of RL.

Some related papers in this direction view the problem through the lens of active learning [3, 11]. Bellinger et al. [3] study active RL (ARL), a modification to traditional RL. As in active learning which aims to select only informative samples to improve the network, an ARL agent *chooses when to observe the reward signal (sample)*. If the agent chooses to observe the reward, it has to pay the "query cost". The agent's goal is to maximize total reward minus total query cost, instead of learning the reward function. In a similar vein, Krueger et al. [11] consider an MDP with one or multiple classes of observations, each with their explicit associated costs. They propose the active measure RL (AMRL) framework, where the agent chooses which class to observe. However, their current experiments are limited to only one class of observation.

The closest related work to our own is [7]. The authors consider a setting in which the environment is asynchronous, and operations such as observation, prediction and action selection have their associated costs, which necessitates a trade-off between them while learning to solve tasks. As illustrated in Figure 2.4, there are two pathways for perception: an encoder ϕ that encodes observations directly, and a predictor f that predicts the latent representation of the current state given

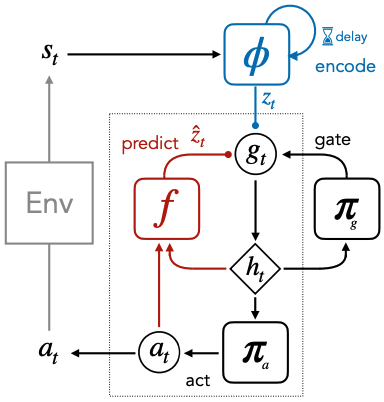


Figure 2.4: The computation graph borrowed from [7]. As depicted in the graph, there are two pathways for perception: one that goes through the encoder ϕ (marked in blue), and the other that passes through the predictor f (marked in red). The two pathways are assumed to have different costs, and the gating network π_g determines which pathway should be followed.

the past hidden state and action. It is assumed that making an observation is more costly than predicting, and this "observation cost" is incorporated by including an intrinsic reward in the learning setting.

In contrast to these works, our work focuses on learning a policy that predicts **when to observe what**. Specifically, instead of predicting whether to observe per timestep, ours directly outputs a number of timesteps that determines when and what to observe next. In other words, our agent makes *fewer decisions*, and is more efficient in terms of computation. Ours also does not require training an extra encoder, although it is possible to incorporate one as well.

Chapter 3

Method

The objective of this thesis is to take a look at how real-world constraints play a role in policy learning. In particular, we consider the sensing costs– one that the agent has to pay in order to sense the environment. Under this setting, the agent's goal is to maximize its accumulated rewards minus the costs. Thus, the agent has to learn to observe only at states where observations are critical for succeeding the task. We propose strategies to realize such a problem setup, including the use of frame-skipping as a proxy for not making observations, and defining an intrinsic reward function to incorporate the notion of sensing costs into the RL problem.

In this section, we begin by describing how frame-skipping can be viewed as a proxy for not making observations. We then show how the cost is incorporated into the problem setup through the use of intrinsic rewards. And finally, we introduce a specific model instantiation for learning not only which action to take, but also *when to make a new decision*.

3.1 Frame-skipping as a Proxy

While frame-skipping is commonly used in RL algorithms, we view it from the perspective of temporal abstraction, and argue that repeating the same action regardless of real environment states is analogous to not making an observation. Hence, to provide the agent with the flexibility of choosing when to observe, we propose to employ frame-skipping as a proxy, which is simple to implement yet effective.

3.2 Observation and its Cost

We consider the scenario where there is a sensing cost c associated with making an observation. In such scenarios, the agent's goal is to maximize its accumulated reward, while simultaneously minimizing the cost. To incorporate c, we express it in the form of intrinsic rewards:

$$r_{int}(t) = \begin{cases} 0, & \text{if observing at timestep } t \\ c, & \text{otherwise} \end{cases} \tag{3.1}$$

where c is a hyperparameter. Hence, the total reward function can be expressed as:

$$r = r_{ext}(s_t) + r_{int}(t), \tag{3.2}$$

where r_{ext} is the external reward returned by the environment.

3.3 The Model

Our model consists of the two policies below:

- The **behaviour policy** $\pi_a : S \to A$, which is a standard RL policy that computes the optimal action given the current state.
- The **skip policy** $\pi_k : S \times A \to R$, which outputs the number of timesteps $k = \{1, 2, \ldots, k_{max}\}$ to repeat the previous action, where k_{max} is a hyperparameter. We set $k_{max} = 7$ in most of our experiments.

A computation graph of our model is given in Figure 3.1. While we focus exclusively on Q-learning for policy learning in this work, our framework can in principle be used to extend any RL algorithms.

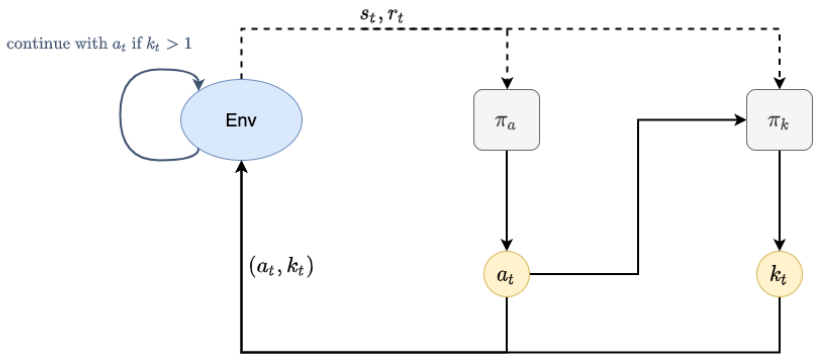


Figure 3.1: Computation graph for our model. The agent consists of 2 policies: π_a and π_k, which operates sequentially. π_a first determines the action given the current state, and π_k outputs the skip number given current state and action. Dashed line indicates that the agent may skip making decisions at some time steps. In case π_k predicts to execute the same action over multiple timesteps, the previous predicted action will be reused (indicated by the blue loop).

We summarize how our method operates as follows:

1. In the very first state s_0, the actor π_a predicts an action a_0, and the action repetition policy π_k predicts the number of timesteps k_0 to repeat a_0.

2. From timestep 0 until k_0, the agent executes a_0, regardless of intermediate states s_j, and receives an intrinsic reward $r_{\texttt{int}}$ during intermediate timesteps.

3. At timestep k_0, the agent again decides a new action a_{k_0} to execute, as well as the number of timesteps k_{k_0} to repeat it.

An illustrative example can be found in Figure 3.2. In this example, the behaviour policy π_a outputs the action `UP`, and the skip policy π_k outputs 3 as the number of skips. As a result, the agent will perform `UP` for 3 consecutive timesteps, without sensing the actual states s_1 and s_2 in intermediate timesteps t_1 and t_2.

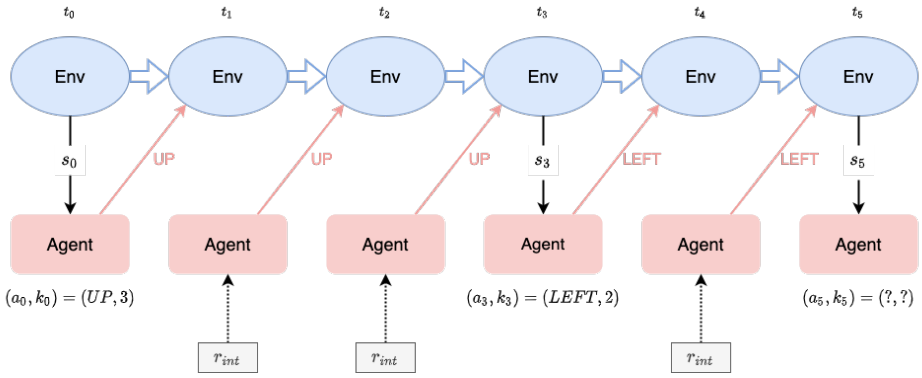


Figure 3.2: A systematic overview of our method. In this example, at timestep t_0, the behaviour policy π_a outputs an action `UP`, and the skip policy π_k chooses the number of skips 3. Thus, the action `UP` is executed 3 times, regardless of intermediate states s_1 and s_2. Since no observation is made at timesteps t_1 and t_2, an intrinsic reward $r_{\texttt{int}}$ is awarded to the agent. At timestep t_3, the agent again makes an observation and determines new action-skip pair (a_3, k_3).

Chapter 4

Results

In this chapter, we aim to look into the following questions through our experiments:

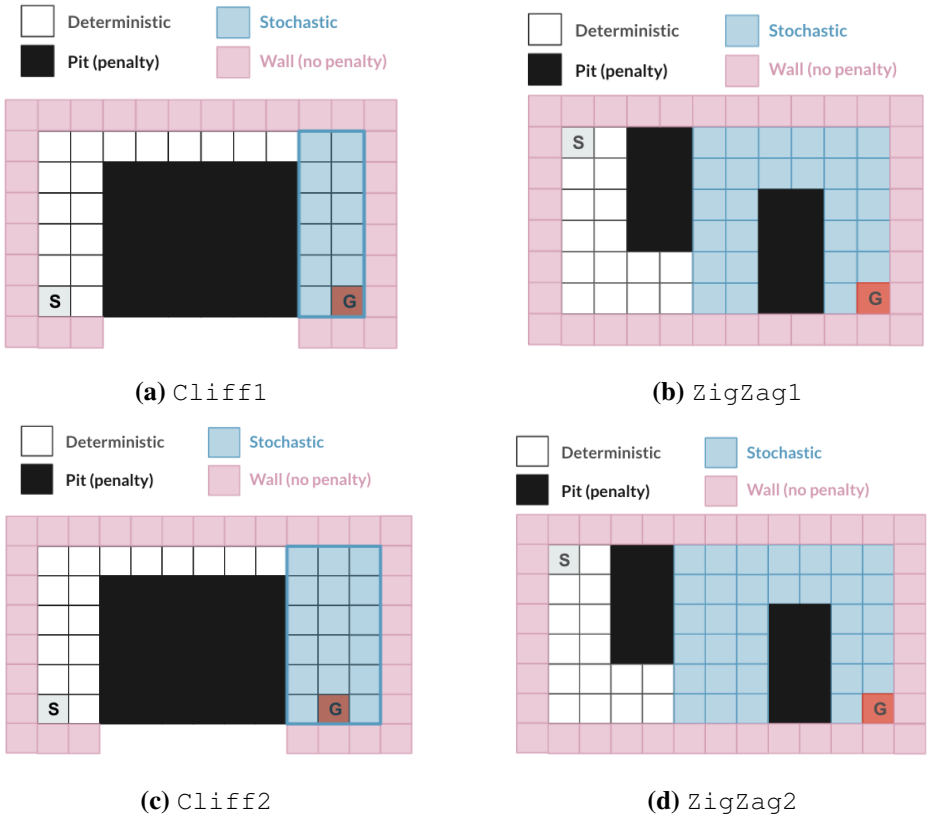
1. How does the sensing cost affect the agent's behaviour, especially in the face of uncertainty.
2. Is the proposed framework sensitive to the choice of hyperparameters, in particular the choices of intrinsic rewards and the number of skips.

For the first question, we hypothesize that incorporating the sensing cost will encourage the agent to only make observations at states that are most critical to completing the task. For example, in a navigation task, it might require more frequent observations at areas that are cluttered or at corridor turns, but less so at regions that can be easily predicted.

4.1 Setup

4.1.1 Environment

We benchmark on a variety of classic grid-world environments, as illustrated in Figure 4.1. The agent's overall goal is to navigate from the start location (**S**) to the goal location (**G**), without falling into the pits (black cells). At each timestep, the agent can choose among four available actions: `UP`, `DOWN`, `LEFT`, and



(a) `Cliff1` **(b)** `ZigZag1`

(c) `Cliff2` **(d)** `ZigZag2`

Figure 4.1: Some example environments.

`RIGHT`. A negative reward of -1 is given if the agent falls into the pit, and the episode immediately terminates. A positive reward of 1 is awarded once the agent arrives at the fixed goal location. There is no penalty if the agent bumps into the wall, and the maximum timesteps allowed per episode is set to 100 unless otherwise specified.

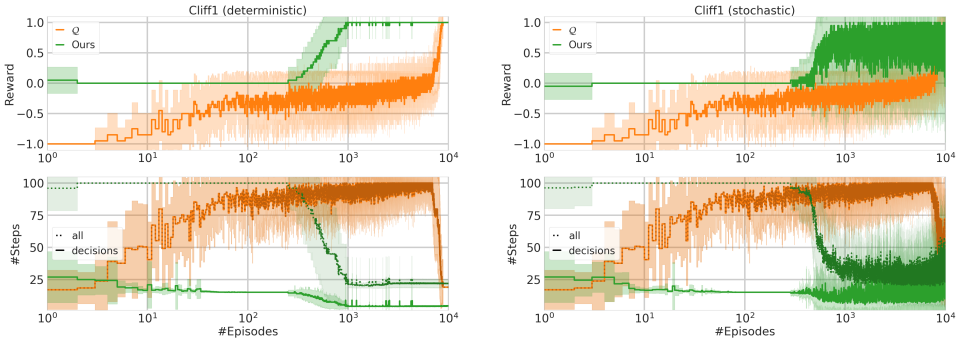
To test the proposed framework, we specifically include environments that are strictly deterministic, as well as ones that are (partially) stochastic. As an example, in Figure 4.1, only the right regions with blue cells are stochastic. When the agent is in one of these cells, there is 10% chance that a random action may be executed instead of the chosen one.

4.1.2 Evaluation

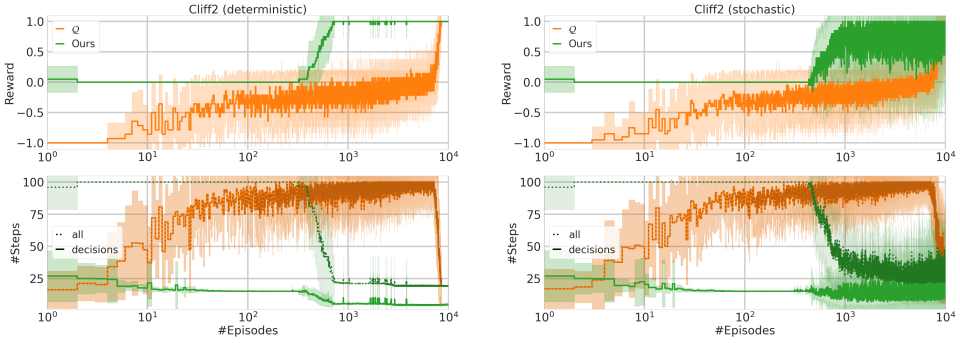
Throughout this thesis, all the experimental results are averaged results over 20 random seeds, unless stated otherwise. We highlight the effectiveness of our method using a mixture of different evaluation criteria. Quantitatively, we report the learning curves, mean rewards, and the mean number of steps taken, averaged over 20 random seeds. For qualitative results, we directly visualize the agent's trajectories on the map, as well as state visitation counts to show how frequent the agent visits a specific state. We also describe how the sensing cost plays a role in policy learning in terms of learned behaviours.

4.1.3 Implementation Details

Unless otherwise specified, we set the maximum number of skips to be 7 in all our experiments. Both ours and the baseline model are trained using the same ε-greedy strategy, where ε is linearly decayed from 1.0 to 0.0 over all episodes. We use $\gamma = 1$ for all our experiments. In tabular cases, the Q-table for π_a and π_k is of sizes $(|S|, |A|)$ and $(|S| \times |A|, |K|)$ respectively.



(a) `Cliff1` (deterministic) **(b)** `Cliff1` (stochastic)



(c) `Cliff2` (deterministic) **(d)** `Cliff2` (stochastic)

Figure 4.2: Learning curves in `cliff`, averaged over 20 random seeds. (a) Ours (green) learns significantly faster than the Q learning baseline (orange).

4.2 Cliff

Figure. 4.2 illustrates the learning curves in the `cliff` environment. It can be seen that ours not only learns significantly faster than the vanilla Q-learning baseline (i.e., $k = 1$), but also requires fewer decision steps to reach the goal, regardless of whether the environment is strictly deterministic or partially stochastic. Similar observations can be made in Table 4.2b.

In Figure 4.3, we provide visualization of the trajectories taken by the agent in both deterministic and stochastic settings. In Figure 4.3b, we can observe that the agent learns to maximize #skips. It takes the action `UP` with the skip number of 5, and then decides on a new action again at the top-left corner of the map, with

Table 4.1: Results in `cliff` environments. We report (a) the mean reward and (b) the number of steps after 10k training episodes averaged over 20 random seeds.

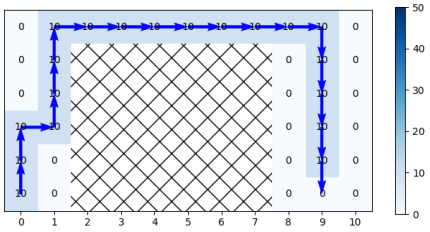
(a) Average rewards

	Q	Ours
`cliff1`		
deterministic	0.148	0.942
stochastic	0.039	0.637
`cliff2`		
deterministic	0.161	0.948
stochastic	0.087	0.702

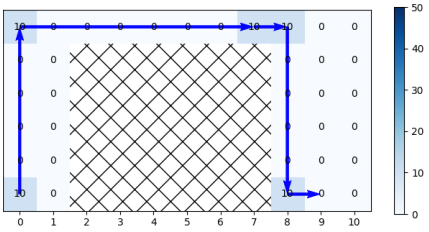
(b) Average steps

	Q	Ours
`cliff1`		
deterministic	80.430	4.800
stochastic	88.081	15.801
`cliff2`		
deterministic	80.164	5.167
stochastic	85.682	14.408

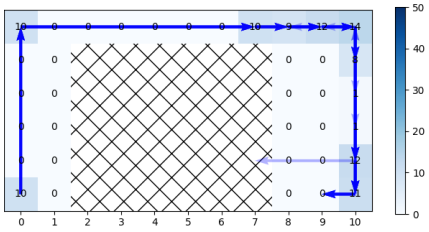
a skip number of 7 this time. Compared to the baseline model in Figure 4.3a, it can be seen that a new decision of which action to take is only made at certain states, instead of every step. Meanwhile, the agent appears to be more "cautious" in Figure 4.3c, it opts to take smaller steps, especially in the stochastic region, and stays in the farthest column away from the pits.



(a) Baseline ($k_{max} = 1$)



(b) Ours ($k_{max} = 7$, deterministic)



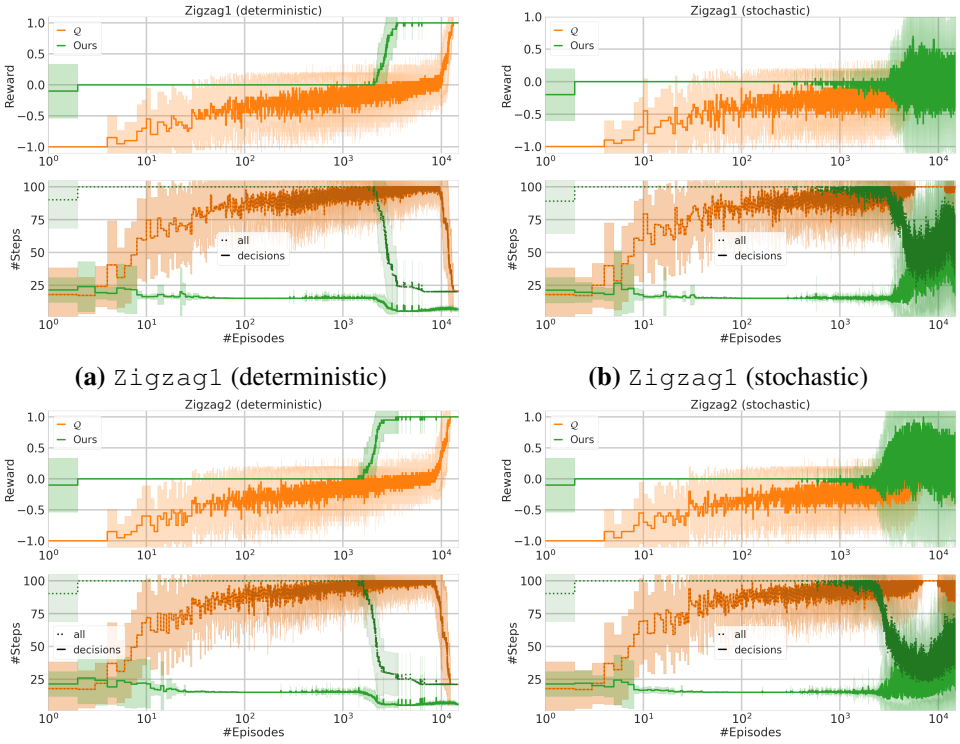
(c) Ours ($k_{max} = 7$, stochastic)

Figure 4.3: Visualization results in `cliff2`. The arrows indicate the trajectories taken by the agent, over the length of 10 episodes. The more opaque it is, the more frequent the agent takes the same step. The number in each grid represents the state visitation count.

4.3 ZigZag

As can be seen in Figure 4.4, our method once again learns significantly faster than the baseline model, consistently reaching an average reward of 0 nearly immediately after training starts, regardless of environments. Similarly, in terms of the number of decision steps, it can be seen in Table 4.2 that ours requires fewer steps compared to the Q learning baseline. However, we would also like to point out that both methods fail to learn a meaningful policy at the end of training in the stochastic versions of `Zigzag` environments (Figure 4.4b and 4.4d).

In Figure 4.5, we visualize the behaviours of the agent by plotting the trajec-



(a) `Zigzag1` (deterministic) **(b)** `Zigzag1` (stochastic)

(c) `Zigzag2` (deterministic) **(d)** `Zigzag2` (stochastic)

Figure 4.4: Learning curves in `zigzag`, averaged over 20 random seeds. (a) Ours (green) learns significantly faster than the Q learning baseline (orange).

tories over 10 episodes. From Figures 4.5a and 4.5b, we can observe that ours succeeds to learn to take the minimum number of decision steps, while still managing to complete the task. Compared to the baseline that requires 21 steps to reach the goal, ours only needs 5, which is a significant improvement in terms of efficiency. Even in stochastic settings where both methods fail to learn the optimal policy, ours still manages to use fewer steps (Figure 4.5c).

Table 4.2: Results in `zigzag` environments. We report (a) the mean reward and (b) the number of steps after 10k training episodes averaged over 20 random seeds.

(a) Average rewards

	Q	Ours
`zigzag1`		
deterministic	0.191	0.820
stochastic	-0.069	-0.000
`zigzag2`		
deterministic	0.265	0.858
stochastic	-0.050	0.247

(b) Average steps

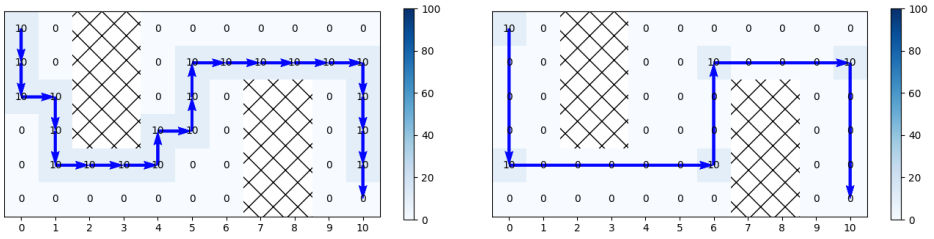
	Q	Ours
`zigzag1`		
deterministic	80.025	7.909
stochastic	97.564	34.985
`zigzag2`		
deterministic	75.970	7.481
stochastic	96.098	25.208

4.4 Analysis on Hyperparameters

In this section, we look into the effects of hyperparameters on our model. We experiment on the `cliff2` environment, with the maximum environmental steps set at 100 and for 10k episodes. Similar to previous experiments, all results shown here are averaged over 20 random seeds.

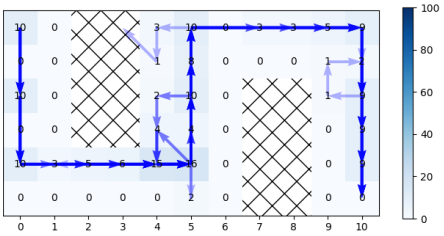
Intrinsic rewards. As shown in Figure 4.6a, setting different intrinsic rewards does not seem to have a noticeable effect performance-wise, as all of the models are able to achieve similar average rewards at the end of training. However, we would also like to point out it does effect how fast the agent learns.

Number of skips. Unlike intrinsic rewards, setting an appropriate maximum number of skips has a much more prominent effect on learning. In Figure 4.6b, it can be seen that even with #skips set at a small number of 3 (marked in brown), it enables the agent to learn significantly faster than the no-skipping baseline (marked in green). Furthermore, as we increase #skips, we can observe a certain level of speed-up.



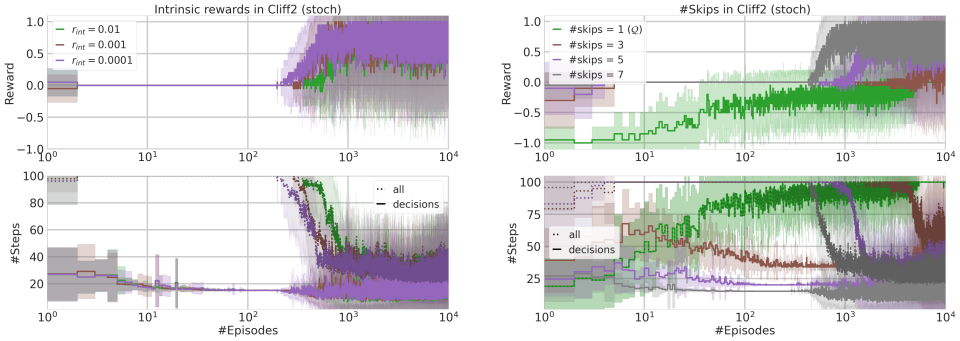
(a) Ba4seline ($k_{max} = 1$)

(b) Ours ($k_{max} = 7$, deterministic)



(c) Ours ($k_{max} = 7$, stochastic)

Figure 4.5: Visualization results in `zigzag2`. The arrows indicate the trajectories taken by the agent, as observed during 10 episodes. The more opaque it is, the more frequent the agent takes the same step. The number in each grid represents the state visitation count.



(a) Intrinsic rewards. **(b)** Number of skips.

Figure 4.6: Effects of hyperparameters. We show the learning curves (a) using different intrinsic rewards, and (b) using different number of maximum skips.

Chapter 5

Conclusions

In this thesis, we propose to explicitly incorporate the sensing cost– one that the agent has to pay to observe the true state of the environment– into the reinforcement learning training loop. The agent's objective is thus to make as few observations as possible, while still managing to succeed the task. To achieve this, we describe a general framework with two major components, an intrinsic reward function to represent the sensing cost, as well as a two-part policy that uses frame-skipping as a proxy for not making observations. We benchmark on a suite of gridworld environments, and highlight the effectiveness of our proposed framework by both quantitative and qualitative results. We find that the agent is able to learn more efficiently compared to the vanilla Q-learning baseline. Moreover, we show that the agent learns to trade off accuracy and cost by only observing the environment at important states, without any prior knowledge about the environment.

5.1 Limitations and Future Work

The proposed method does not come without limitations, and there remain many directions for future work.

We only explore how the sensing costs influence decision making. However, we focus mostly on when to observe, but there is more to be done on what to observe. For example, is it possible to let the agent choose how many bits of information it needs every time step. In an alternative setting, if there are multiple classes

of observations with different associated costs, how would the agent trade off costs and which class to observe? Furthermore, it may be worthwhile to incorporate memory into control policy to allow for an "internal model" of the dynamics, as postulated in the motor control literature. We would also like to extend the current framework to continuous states and actions. Last but not least, it would be interesting to understand how the results might be predictive of human behaviour in similar settings.

Bibliography

[1] J. A. Arjona-Medina, M. Gillhofer, M. Widrich, T. Unterthiner, J. Brandstetter, and S. Hochreiter. Rudder: Return decomposition for delayed rewards. *arXiv preprint arXiv:1806.07857*, 2018. → page 2

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013. → page 8

[3] C. Bellinger, R. Coles, M. Crowley, and I. Tamblyn. Active measure reinforcement learning for observation cost minimization. *arXiv preprint arXiv:2005.12697*, 2020. → page 9

[4] R. E. Bellman and S. E. Dreyfus. *Applied dynamic programming*. Princeton university press, 2015. → page 6

[5] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen. Frame skip is a powerful parameter for learning to play atari. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015. → page 8

[6] T. G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13: 227–303, 2000. → page 7

[7] Y. Fu. The cost of OPS in reinforcement learning. 2021. → pages ix, 9, 10

[8] T. Hester and P. Stone. Texplore: real-time sample-efficient reinforcement learning for robots. *Machine learning*, 90(3):385–429, 2013. → page 2

[9] S. Kalyanakrishnan, S. Aravindan, V. Bagdawat, V. Bhatt, H. Goka, A. Gupta, K. Krishna, and V. Piratla. An analysis of frame-skipping in reinforcement learning. *arXiv preprint arXiv:2102.03718*, 2021. → page 8

[10] G. Konidaris and A. Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. *Advances in neural information processing systems*, 22:1015–1023, 2009. → page 7

[11] D. Krueger, J. Leike, O. Evans, and J. Salvatier. Active reinforcement learning: Observing rewards at a cost. *arXiv preprint arXiv:2011.06709*, 2020. → page 9

[12] X. Liang, T. Wang, L. Yang, and E. Xing. CIRL: Controllable imitative reinforcement learning for vision-based self-driving. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 584–599, 2018. → page 1

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015. → pages 1, 5

[14] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra. Benchmarking reinforcement learning algorithms on real-world robots. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 561–591. PMLR, 29–31 Oct 2018. URL https://proceedings.mlr.press/v87/mahmood18a.html. → page 1

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. → pages 1, 5

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518 (7540):529–533, 2015. → page 6

[17] O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. *arXiv preprint arXiv:1805.08296*, 2018. → pages ix, 7

[18] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018. → page 2

[19] D. Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000. → page 6

[20] S. Sharma, A. Srinivas, and B. Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017. → pages ix, 8

[21] D. Silver and K. Ciosek. Compositional planning using optimal option models. *arXiv preprint arXiv:1206.6473*, 2012. → page 7

[22] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016. → pages 1, 5

[23] A. Srinivas, S. Sharma, and B. Ravindran. Dynamic frame skip deep q network. *arXiv preprint arXiv:1605.05365*, 2016. → page 8

[24] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018. → page 4

[25] R. S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999. → page 6

[26] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pages 3540–3549. PMLR, 2017. → page 7

[27] W. Xiong, T. Hoang, and W. Y. Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. *arXiv preprint arXiv:1707.06690*, 2017. → page 1

[28] J. Zhang, H. Yu, and W. Xu. Hierarchical reinforcement learning by discovering intrinsic options. *arXiv preprint arXiv:2101.06521*, 2021. → page 7

[29] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017. → page 1