Queen Mary University of London

Accelerating Deep Reinforcement Learning via Action Advising

Ercüment İlhan

Primary Supervisor: Diego Perez-Liebana

Secondary Supervisor: Jeremy Gow

Submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy in Computer Science

in the

School of Electronic Engineering and Computer Science

December 15, 2022

Abstract

Deep Reinforcement Learning (RL) algorithms can solve complex sequential decision-making tasks successfully. However, they suffer from the major drawbacks of having poor sample efficiency and long training times, which can often be tackled by knowledge reuse. Action advising is a promising knowledge exchange mechanism that adopts the teacher-student paradigm to leverage some legacy knowledge through a budget-limited number of interactions in the form of action advice between peers. In this thesis, we studied action advising techniques, particularly in Deep RL domain, both in single-agent and multi-agent scenarios. We proposed a heuristic-based jointly-initiated action advising method that is suitable for multi-agent Deep RL setting, for the first time in literature. By adopting Random Network Distillation (RND), we devised a measurement for agents to assess their confidence in any given state to initiate the teacher-student dynamics with no prior role assumptions. We also used RND as an advice novelty metric to construct more robust student-initiated advice query strategies in single-agent Deep RL. Moreover, we addressed the absence of advice utilisation mechanisms beyond collection by employing a behavioural cloning module to imitate the teacher's advice. We also proposed a method to automatically tune the relevant hyperparameters of these components on the fly to make our action advising algorithms capable of adapting to any domain with minimal human intervention. Finally, we extended our advice reuse via imitation technique to construct a unified student-initiated approach that addresses both advice collection and advice utilisation problems. The experiments we conducted in a range of Deep RL domains showed that our proposal provides significant contributions. Our Deep RL-compatible action advising techniques managed to achieve a state-of-the-art level of performance. Furthermore, we demonstrated that their practical attributes render domain adaptation and implementation processes straightforward, which is an important progression towards being able to apply action advising in real-world problems.

Keywords: Deep Reinforcement Learning, Multi-Agent Reinforcement Learning, Deep Q-Network, Action Advising, Knowledge Reuse, Teacher-Student Framework

Acknowledgements

First and foremost, I would like to thank my primary supervisor Diego Pérez-Liébana for believing in me and providing me with this invaluable Ph.D. opportunity here at Queen Mary University of London. As well as being a great researcher role model for me, he has always been very supportive and has given me the freedom of exploring my own ideas which I really appreciate. I would also like to thank my second supervisor Jeremy Gow for his valuable feedback on the publications we have co-authored. I also wish to thank John Woodward for being my independent assessor to evaluate my progression constructively at multiple stages of this course.

I am deeply indebted to Matthew E. Taylor for showing interest in my work and presenting me the opportunity to collaborate with his research group at University of Alberta. His support has really made the final months of this Ph.D. endurable for me.

My time in London would not have been the same without my dear colleague Alvaro Ovalle Castañeda. I am very glad to have known him both as a great friend and a passionate researcher, with whom I have had countless amusing memories and influential scientific discussions.

I am extremely grateful to my dear parents İclal, Ergun and my dear brother Kerem for all the love, encouragement and support they have been providing me all this time. No matter how far we have ended up being from each other, they have always made me feel like I have them around as if I am back at home with them. I also can not go without commemorating my grandmother Naciye and my uncle Cengiz, who were among my biggest supporters from the beginning and would surely be proud to see me achieve this Ph.D. degree.

Finally, I would like to extend my sincere thanks to Nattaruedee for bringing further joy to my life and supporting me with her best by being like a family to me whenever I needed it.

And of course, I gratefully acknowledge the Ph.D. scholarship granted to me by Queen Mary University of London, which made this study possible.

Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial Intelligence
AIR	Advice Imitation & Reuse
ANA	Advice Novelty-Based Advising
AR	Early Advising with Advice Reuse via Imitation
AR+A	AR with Automatic Threshold Tuning
AR+A+E	AR+A with Extended Reuse
AUC	Area Under the Curve
BC	Behavioural Cloning
DAgger	Dataset Aggregation
Dec-POMDP	Decentralised Partially Observable Markov Decision Process
DQfD	Deep Q-Learning from Demonstrations
DQN	Deep Q-Network
EA	Early Advising
JAL	Joint Action Learners
LfD	Learning from Demonstrations
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent System
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MG	Markov Games
NA	No Advising
PER	Prioritised Experience Replay
POMDP	Partially Observable Markov Decision Process
RA	Random Advising
ReLU	Rectified Linear Unit

RHEA . . . . Rolling Horizon Evolutionary Algorithm
RL . . . . . Reinforcement Learning
RND . . . . . Random Network Distillation
RtG . . . . . Reach the Goal
SNA . . . . . State Novelty-Based Advising
UA . . . . . Uncertainty-Based Advising
UMAP . . . . Uniform Manifold Approximation and Projection

List of Symbols

$\mathcal{S}$	Finite Set of states.
$\mathcal{A}$	Finite Set of actions.
$\mathcal{R}$	Reward function.
$\mathcal{T}$	Set of conditional state transition probabilities.
$\mathcal{N}$	Finite set of agents.
$\mathbf{\Omega}$	Finite set of joint observations.
$\mathcal{O}$	Set of conditional observation probabilities.
G_t	Sum of discounted future rewards obtained from timestep t.
γ	Discount factor.
$V(s)$	State value of state s.
$Q(s, a)$	State-action value of state s and action a.
$A(s, a)$	State-action advantage value of state s and action a.
π	Decision-making policy.
π^*	Optimal decision-making policy.
π_S	Decision-making policy of student.
π_T	Decision-making policy of teacher.
ε	Termination criterion.
Δ	Termination criterion comparison term.
α	Learning rate.
t_{max}	Number of training steps.
ϵ	ϵ-greedy exploration rate.
$\mathcal{U}$	Uniform distribution.
θ	RL algorithm approximator weights.
$\mathcal{L}_i(\theta_i)$	Loss function with respect to θ at learning step i.
$\mathcal{B}$	Minibatch of samples.
$\mathcal{D}$	DQN replay memory.
$N_{\mathcal{D}}$	DQN replay memory capacity.

$M_{\mathcal{D}}$ . . . DQN replay memory size to start learning.
T_{target} . . DQN target network update period.
T_{train} . . DQN training period.
G . . . . RND target network.
$\hat{G}_{\omega}$ . . . . RND predictor network parameterised with ω.
ω . . . . RND predictor network weights.
χ . . . . Dropout rate.
$\mathcal{C}$ . . . . BC/LfD samples buffer.
H_{η} . . . . BC network parameterised with η.
η . . . . . BC network weights.
$N_{\mathcal{C}}$ . . . . Number of samples needed in $\mathcal{C}$ to trigger BC training.
K_{BC} . . . Number of initial BC training iterations.
K_{BCP} . . Number of periodic BC training iterations.
ϵ_{reuse} . . Probability to enable episodic advice-reusing.
$\epsilon_{reuse-init}$ Initial value of probability to enable episodic advice-reusing.
$\epsilon_{reuse-final}$ Final value of probability to enable episodic advice-reusing.
T_{ϵ} . . . . Number of timesteps to decay $\epsilon_{reuse-init}$ value to $\epsilon_{reuse-final}$.
$H_{\eta}^{u}(s)$ . . Epistemic uncertainty of BC model H_{η} in state s.
$T_{imitation}$. Number of timesteps needed to trigger periodic BC training.
$\mathcal{H}$ . . . . Uncertainty values set for the automatic threshold tuning process.
$\rho_{\mathcal{H}}$ . . . . Proportion cutoff point in $\mathcal{H}$ to be set as the reuse threshold.
$I(s)$ . . . State importance of state s.
$I_V(s)$ . . Variance-based state importance of state s.
$I_D(s)$ . . Absolute deviation-based state importance of state s.
$U(s)$ . . . Epistemic/proxy uncertainty for state s.
b_{ask} . . . Advice-asking budget of the student.
b_{give} . . . Advice-giving budget of the teacher.
τ_{ask} . . . State uncertainty threshold to request advice.
τ_{give} . . . State uncertainty threshold to provide advice.
τ_{imp} . . . State importance threshold to request/provide advice.
τ_{reuse} . . Advice-reusing threshold.
Υ_s . . . . Student's state confidence of state s.
Ψ_s . . . . Teacher's state confidence of state s.
v_a . . . . Scaling hyperparameter to determine the advice-asking probability with Υ.
v_g . . . . Scaling hyperparameter to determine the advice-giving probability with Ψ.
$p_{ask}(s)$ . . Advice-asking probability (derived from $U(s)$) for state s.

λ . . . . Advice-giving probability scaling term.
λ_{UA} . . . Advice-giving probability scaling term of UA.
λ_{SNA} . . Advice-giving probability scaling term of SNA.
λ_{ANA} . . Advice-giving probability scaling term of ANA.

Chapter 1

Introduction

Today, a vast amount of real-world domains ranging from finance to surveillance benefit from having automation through machines that can exhibit intelligent behaviour. These kinds of capabilities granted to the machines are referred to as Artificial Intelligence (AI) and are especially relevant in accomplishing sequential decision-making tasks. For instance, we build self-driving cars to eliminate human efforts and errors in controlling vehicles, or we require robotic manipulation for handling various industrial tasks. Despite its potential, building AI for such practical applications is not straightforward. A significant portion of the domains are complex and involve non-deterministic, non-stationary dynamics, i.e. the outcomes of the interactions made within them may be time-varying and unpredictable. Therefore, manually defined behaviours are not feasible options for the agents that are responsible to drive decision-making within these domains; it is critical for them to be able to adapt continually to unseen circumstances and changing conditions. Because of these reasons, Reinforcement Learning (RL) (Sutton & Barto 2018), the machine learning paradigm of self-improvement via experience acquired through trial-and-error interactions, is considered to be a key AI approach to building intelligent sequential decision-making agents.

RL has a long history of development with a rapidly increasing research effort devoted to it in recent years. It has made its most significant breakthroughs of reaching

super-human level progressively in difficult domains like Atari games (Mnih et al. 2015), the game of Go (Silver et al. 2016), StarCraft II (Vinyals et al. 2019) and DotA II (Berner et al. 2019) in the last few years aided by the emergence of Deep Learning (Goodfellow et al. 2016) in parallel to the availability of high computational power and immense amount of data. Consequently, RL has obtained its popular name of Deep RL to refer to its deep learning variants. In addition to its remarkable performance, Deep RL's end-to-end structure and general applicability make it a desirable sequential decision-making approach for various real-world problems (Popova et al. 2018, Levine et al. 2018, Bellemare et al. 2020). Nevertheless, these applications are still considered to be limited. This is due to the Deep RL's costly requirements of long training times and large numbers of samples to be acquired through environment interactions, which are often highlighted as the well-known drawbacks of it (Arulkumaran et al. 2017).

The challenge of accelerating Deep RL is tackled in multiple aspects by different approaches. The most prominent of these can be listed as follows: learning the environment dynamics explicitly, i.e. model-based RL (Hafner et al. 2020); learning to learn faster by using the previous learning experience, i.e. Meta RL (Finn et al. 2017); improving techniques to make agents more proficient at exploring their environment mechanics and collecting samples that will yield high-reward strategies (Taïga et al. 2019) and finally, knowledge reuse (or transfer learning) (Zhu, Lin & Zhou 2020) which stands for the process of leveraging some legacy knowledge obtained from a previous learning session to improve the learning process in the current task. We now shift our focus to the latter concept which forms the core idea of this thesis.

The ability to utilise the knowledge obtained from prior experience when learning a novel task is an important aspect of intelligence. As humans, we demonstrate remarkable performance when learning and adapting to new concepts. For example, in the case of playing video games, we rapidly learn and develop a strong understanding by utilising our prior knowledge in visuals and physics (Dubey et al. 2018). Driven by this motivation, a considerable amount of methods to leverage past knowledge in RL are proposed to this date (Taylor & Stone 2009). More recently, these techniques have

also been applied in Deep RL with promising results in complex domains (Rusu et al. 2016, Teh et al. 2017, Zhu, Lin & Zhou 2020). As the results of these studies suggest, these methods are still open to development, and they offer many different avenues for different scenarios to be studied.

A common method to preserve past knowledge for reusing is to store either the encountered experience (Ho & Ermon 2016, Hester et al. 2018, Kumar et al. 2019) or the trained agent policies themselves (Kurenkov et al. 2019). However, in some situations, it may not be possible to access the relevant RL task prior to the training to generate such datasets of experience. This may occur when we are to deploy an agent in an environment with unpredictable characteristics. Furthermore, the previous policies to be leveraged may not be transferable for unlimited access, e.g., in the case of humans or other agents without fixed policies. These issues render the aforementioned knowledge reuse approaches infeasible. As an alternative to these conditions, the learning agent(s) may instead have access to some interactive peers, be it, humans or other agents, during the training time either in a multi-agent setting or in an isolated single-agent environment. This presents a whole new set of opportunities to tackle the learning inefficiency via peer-to-peer knowledge reuse (da Silva, Warnell, Costa & Stone 2020) even without any previous datasets or policies at hand.

Action advising (Torrey & Taylor 2013) is a flexible peer-to-peer knowledge exchange framework that is designed for this exact problem setting. Essentially, the learning agent (student) receives advice in the form of actions from a more knowledgeable peer (teacher) to speed up its learning with the aid of a guided exploration process. However, the maximum number of teacher-student interactions to be made is limited with a budget in this framework to resemble the practical limitations of communication and attention in the real-world domains. For this reason, the peer that is responsible to drive the advice exchange interactions needs to do so carefully in order to make the most efficient use of the available budget. This notion forms the core challenge of action advising and distinguishes it from other similar approaches like Policy Reuse (Kurenkov et al. 2019) that has no limitations in policy interactions. Since the action advising

paradigm targets very specific scenarios, it stands at a critical point of being a solution to these kinds of problem settings. For instance, having an agent deployed remotely to some previously unknown environment to be trained with the assistance of a human expert over a costly communication channel can be a very relevant application case for action advising. A system with multiple agents that learn in a decentralised fashion with occasional inter-agent communication can also make use of action advising to accelerate collective learning efficiently. These make action advising an important subject to be researched. However, due to not being an extensively applicable framework in a wide range of domains, it has not been studied as much as the other RL acceleration paradigms. Consequently, all the related work conducted prior to the beginning of this Ph.D. study remained to be limited to the classical (non-Deep) RL domains. In this thesis, we study action advising approaches, particularly in Deep RL to address the present research gaps.

The following sections first describe the problem specifications we target along with the research questions we tackle (Section 1.1). Then, the scientific contributions we made in this thesis with their respective publications are listed and detailed (Section 1.2).

1.1 Problem Statement

The main goal of this Ph.D. research is to study action advising methods in order to accelerate Deep RL. We aim to build modular techniques to make the most of the limited teacher-student interaction budgets without modifying the underlying Deep RL algorithms at all. In terms of environment and learning conditions, we are interested in either fully cooperative decentralised multi-agent scenarios or single-agent tasks in general, both in partial and fully observable forms where peer-to-peer communication is allowed but limited with a set budget.

We consider cooperative decentralised multi-agent learning as one of the most relevant application cases of Deep Multi-agent RL (MARL) with action advising because of the presence of multiple agents that can learn mutually by communicating with each other.

This can easily be utilised as a peer-to-peer knowledge exchange setup. Furthermore, these agents usually tend to learn in different parts of the environment and therefore end up with heterogeneously distributed knowledge among them. Action advising can be a very efficient tool to take advantage of this condition by sharing diverse knowledge among the agents to speed up their learning. Finally, another property of such domains that makes action advising a suitable framework is the commonly encountered setbacks in the inter-agent communication channels such as limited bandwidth and time-varying behaviour. Therefore, it is important to treat communication as a valuable resource to make the most out of it while it is available, as it is done in action advising.

Even though the described MARL scenario can be an ultimate application case for action advising, the learning processes in such multi-agent systems can be very complicated to study. For this reason, in addition to confirming and showcasing the potential effectiveness of the action advising framework in MARL, we are also interested in studying action advising algorithms extensively in single-agent domains to develop more principled approaches in simpler settings that can be scaled up later. Nevertheless, single-agent Deep RL can also gain substantial benefits from action advising in practice. It can either be in a scenario where multiple agents learn in their isolated environments and still can interact with each other over a limited communication channel, or in a human-agent setting where the agent's ongoing learning can be aided by a human with a limited attention span.

We believe that the outcomes of this Ph.D. regarding action advising and related approaches will have contributions and practical uses both in single-agent and multi-agent Deep RL scenarios. The research questions we have tackled over the course of this Ph.D. study are as follows:

[RQ1] **Primary Question**: To what extent can we accelerate Deep RL via (budget-limited) action advising with one or more knowledgeable peer(s)?

It is vital to address this question because the outcomes will open up new avenues of knowledge reuse opportunities in Deep RL to speed up learning significantly,

especially in the specific problem scenarios where there is no possibility of pre-generating useful datasets or transferring the previous policies to the learning agent for unlimited access.

[RQ2] How can we scale the state-of-the-art action advising approaches from classical RL to Deep RL/MARL domains?

This can be considered as the first step of establishing action advising methods in Deep RL by extending the state-of-the-art from the classical RL domains. The answer(s) to this question will let us identify some of the action advising challenges that are present in the Deep RL domains.

[RQ3] What can be an efficient heuristic to perform student-initiated action advising that is also robust to teacher absence conditions in Deep RL?

In addition to developing functional action advising approaches in Deep RL, it is also important to understand the non-ideal situations regarding the teacher's presence and competence to identify the shortcomings of these approaches to be alleviated. One obvious case in the action advising framework is the assumption of teacher availability from the beginning of the learning sessions which may not always hold in practical scenarios. Analysing and alleviating this will help us build more robust techniques.

[RQ4] Can imprecise usage of action advising budget hamper Deep RL performance?

As it is highlighted in the previous studies in the classical RL domains (Clouse 1996), the negative effects of excessive amounts of advice are a matter to be paid attention to. Thus, we want to find out if this finding also holds in Deep RL.

[RQ5] How can we further utilise the collected advice by memorising and reusing them in Deep RL domains?

The objective of the action advising strategies is mainly about determining the most impactful moments to collect advice to maximise budget efficiency. Yet,

how the advice are leveraged beyond collection also plays a critical role in the overall learning process. Being able to reuse the previous advice can definitely be a promising way to strengthen this aspect.

[RQ6] How can we use the advice memorisation techniques to build more efficient advice collection strategies?

We want to find out how we can integrate our findings into devising a unified action advising algorithm that is capable of utilising the advice efficiently and also use this ability to drive the advice collection. This helps us compose the developments we have achieved over the course of this study into a final algorithm.

1.2 Contributions

The main scientific contributions we have made as a part of this Ph.D. study are as follows with their respective publications and the chapters they are covered in:

- **E. İlhan,** J. Gow, and D. Pérez-Liébana, "Teaching on a Budget in Multi-Agent Deep Reinforcement Learning," *2019 IEEE Conference on Games (CoG)*, London, United Kingdom, August 20–23, 2019.

  [Chapter 4]

  In this first study, we extended the idea of peer-to-peer knowledge exchange via action advising with no predefined roles from multi-agent tabular RL (da Silva et al. 2017) to the domain of non-linear function approximation with agents employing Deep RL algorithms as their task policies. We employed Random Network Distillation (RND)(Burda et al. 2018), originally a state novelty measurement to aid exploration techniques, as a proxy of tabular state counting to Deep RL to facilitate self-assessment of agents' confidences. This way, the agents can switch between teacher-student roles and initiate advice exchange interactions whenever they decide it is appropriate. By doing so, we demonstrated the effectiveness of a

heuristic-based action advising algorithm in a Deep RL/MARL setting for the first time. This study addresses the research questions [RQ1] and [RQ2].

- **E. İlhan,** J. Gow, and D. Pérez-Liébana, "Student-Initiated Action Advising via Advice Novelty," in IEEE Transactions on Games, vol. 14, no. 3, pp. 522-532, September 2022, doi: 10.1109/TG.2021.3113644.

  [Chapter 5]

  In this work, we investigated the drawbacks of the existing action advising methods in the single-agent Deep RL domain and analysed them in an extensive set of experiments with comparisons. We highlighted the significance of being able to handle a teacher that joins the communication loop at a later time during the training rather than being available from the beginning. We also showed how over-advising can even hamper the performance of an off-policy Deep RL algorithm. Finally, we proposed an approach that utilises RND to measure the novelty of the advice instead of the states themselves. This is achieved by updating RND exclusively by the advised states instead of those encountered in Deep RL model updates. Our approach overcomes the drawbacks that are present in the previous action advising techniques as well as performing better than or at least on par with the best ones. This study addresses the research questions [RQ1], [RQ3] and [RQ4].

- **E. İlhan,** J. Gow, and D. Pérez-Liébana, "Action Advising with Advice Imitation in Deep Reinforcement Learning," *Proceedings of the 20th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2021)*, Online, May 3–7, 2021, IFAAMAS.

  [Chapter 6]

  We addressed the absence of further advice utilisation by reusing advice in Deep RL domains, which is especially crucial in practical settings considering the importance of making the most of a small amount of budget. To do so,

we presented an approach to enable the student agent to partially imitate the teacher's policy through previously acquired advice to reuse them directly in its exploration steps without any interventions in the Deep RL mechanism itself. In particular, we employ a Behavioural Cloning (BC) module to imitate the teacher policy and use dropout regularisation in this model to have a notion of epistemic uncertainty to keep track of which state-advice pairs are actually collected to reuse them accurately. This study addresses the research questions [RQ1] and [RQ5].

- **E. İlhan,** J. Gow, and D. Pérez-Liébana, "Learning on a Budget via Teacher Imitation," *2021 IEEE Conference on Games (CoG)*, Copenhagen, Denmark, August 17-20, 2021.

  [Chapter 7]

  We extended the idea of advice reusing via teacher imitation to construct a unified student-initiated approach in single-agent Deep RL that addresses both advice collection and advice utilisation problems. This is achieved by involving the imitated teacher model's epistemic uncertainty estimations in the advice collection decisions. This method makes more efficient use of the budget in terms of collecting samples that are more useful to build a broader imitation model, e.g., constructing a more diverse state-advice dataset. We also proposed a technique to automatically tune the relevant hyperparameters of these components on the fly to make the action advising methods able to adapt to any task with minimal human intervention. This contribution provides an important advantage for the scenarios when the agent needs to be deployed with minimal domain knowledge or even blindly. This study addresses the research questions [RQ1] and [RQ6].

Additionally, the following study was also conducted during this Ph.D. with no direct contributions to this thesis:

- D. Pérez-Liébana, R. D. Gaina, O. Drageset, **E. İlhan,** M. Balla, and S. M. Lucas,

"Analysis of Statistical Forward Planning Methods in Pommerman," *Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2019*, Atlanta, Georgia, 2019, pp. 66-72.

We developed a Java version of the Pommerman Framework with forward environment models that are available to the agents. By using this framework, we experimented with two prominent Statistical Forward Planning methods, namely Monte Carlo Tree Search (MCTS) and Rolling Horizon Evolutionary Algorithm (RHEA). We provided findings and insights on how the agents behave and explained their performances. Results show that MCTS is a better performer than RHEA in this particular domain in several different settings. We also identified what could be promising improvements, such as improved opponent modelling, further algorithm tuning and assumptions for the partial observability cases.

1.3 Organisation of the Thesis

The remainder of this thesis is structured as follows:

- Chapter 2 provides the background information and prior work on the relevant topics of RL, Deep RL, and the approaches to leverage prior knowledge with an emphasis on the action advising framework.
- Chapter 3 describes the game domains we employed in our experiments across the thesis.
- Chapter 4 covers our first study titled "Teaching on a Budget in Multi-Agent Deep Reinforcement Learning" (Ilhan et al. 2019).
- Chapter 5 is based on our second work "Action Advising via Advice Novelty" (Ilhan et al. 2022) is explained in details.
- Chapter 6 describes our third piece of work "Action Advising with Advice Imitation in Deep Reinforcement Learning" (Ilhan et al. 2021*a*).

- Chapter 7 covers our final piece of work "Learning on a Budget via Teacher Imitation" (Ilhan et al. 2021*b*).

- Chapter 8 finalises the thesis with some concluding remarks and potential future work directions.

Chapter 2

Background

This chapter provides the background knowledge required to understand the technical aspects of the approaches presented in this thesis.

2.1 Markov Decision Processes

The process of taking a series of actions in order to achieve the desired goal in a domain is referred to as sequential decision-making. This kind of problem setting is generally formulated by using a formalisation called Markov Decision Process (MDP).

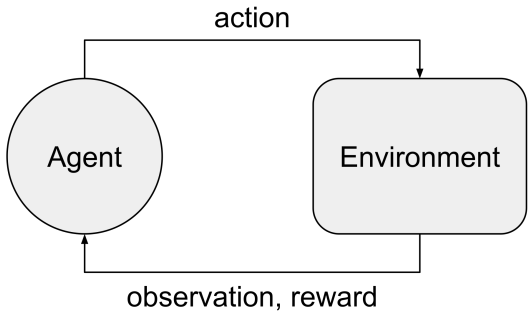


Figure 2.1: Agent-environment interaction.

In MDP, an *agent* makes decisions in an *environment* by receiving observations and rewards, and executing actions sequentially in a cycle over multiple time steps

$t = 0, 1, 2, \cdots$ as it is shown in Figure 2.1. The environment is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function and $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a set of conditional state transition probabilities denoted as $\mathcal{T}(s, a, s') = P(s' \mid s, a)$. At every timestep t in a state $s \in \mathcal{S}$, the agent executes an action $a \in \mathcal{A}$ to advance its state to $s' \in \mathcal{S}$ with probability $P(s' \mid s, a)$ and obtain a reward $r = \mathcal{R}(s, a)$. It is worth mentioning that it is common to see the reward function defined in the forms of $\mathcal{R}\colon \mathcal{S} \to \mathbb{R}$ (action-independent) and $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ (next state-dependent). In this thesis we consider rewards to be determined deterministically by the current state and the action executed in it, regardless of the next state the agent ends up advancing to.

Actions to be taken by the agent in a state s_t at every timestep t are determined by the agent's policy $\pi\colon \mathcal{S} \times \mathcal{A} \to [0, 1]$ which is a probability distribution over actions given states. We denote the action probability distribution of π for a given state s with $\pi(a \mid s)$. If the policy is purely deterministic, it is possible to define it as $\pi\colon \mathcal{S} \to \mathcal{A}$ that maps states to actions which is denoted as $\pi(s)$. We use either form to model action selection depending on the use cases and problem formulations across the thesis.

A key property that directly affects the decision-making formulation in MDPs is referred to as the Markov property. According to this, the transitions in the process depend only on the present state and the executed action without any dependence on history. This assumes that the state perceived by the agent contains all the relevant information for decision-making. In this case, the conditional state transition probability for a state s_t and action a_t that determines the next state s_{t+1} satisfies the following:

$$P(s_{t+1} \mid s_t, a_t) = P(s_{t+1} \mid s_1, s_2, \ldots, s_t, a_t). \tag{2.1}$$

With the aid of this, the optimal agent policy can be a function of the current state, as we describe in the remainder of this section.

The objective of the agent is to maximise the expected sum of discounted future rewards G_t (also referred to as *return*) obtained from any timestep t, over a horizon T

where $\gamma \in [0, 1]$ is the discount factor:

$$G_t = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-1} r_{t+T} = \sum_{k=0}^{T-1} \gamma^k r_{t+k+1}. \tag{2.2}$$

The discount factor here is responsible for determining the importance of future rewards. For instance, if γ is set as 0, then, the agent will only be concerned with maximising the immediate rewards; whereas setting it as 1 will make the rewards at every future timestep equally valuable. In practice, γ is usually set to be a value close to 1, e.g. 0.99, to ensure that the infinite sum in Equation 2.2 (when $T = \infty$) has a finite value.

A common strategy to define optimal agent behaviour involves computing the values of states and actions to drive the agents' decisions by selecting the most valuable ones. The value of a state s under a stochastic policy π is defined as follows:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma V^\pi(s')\big]. \tag{2.3}$$

It is also possible to decompose $V^\pi(s)$ into the state-action values $Q^\pi(s, a)$ as follows:

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) Q^\pi(s, a), \tag{2.4}$$

where

$$Q^\pi(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma V^\pi(s')\big]. \tag{2.5}$$

With the aid of these equations, the agent's objective can be defined as obtaining the optimal policy $\pi^* = \arg\max_\pi V^\pi(s)$. Similarly, $\pi^* = \arg\max_\pi Q^\pi(s, a)$ can also be constructed using the state-action value function. Here, the optimal state value and state-action value functions can be defined as $V^*(s) = \max_\pi V^\pi(s)$ and $Q^*(s, a) = \max_\pi Q^\pi(s, a)$, respectively. Considering the fact that the agent's ultimate goal is to follow the optimal policy based on the optimal state and state-action value functions V^* and Q^*, it is possible to formulate the state-action values independently from the

policy as follows:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma V^*(s') \big], \tag{2.6}$$

$$Q^*(s, a) = \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \big]. \tag{2.7}$$

As it can be seen, Equations 2.6 and 2.7, which are also referred to as Bellman equations, involve computing the values of the successor states. Thus, computing these is done by bootstrapping over the previous estimations which are corrected over multiple iterations via dynamic programming. It should be noted that by definition, the optimal policy for an MDP is deterministic as it can be seen in these equations too. When it comes to some MDP extensions (Section 2.1.1), it is possible that the optimal policies can also be stochastic ones, e.g. competitive multi-agent scenarios. Regardless, we adopt the stochastic notation for generality.

When the complete model of an MDP ($\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}$ variables) is known, it is trivial to find the optimal policy by solving these Bellman equations. First, given a policy π (stochastic or deterministic), it is possible to compute the value of each state by performing Policy Evaluation (Algorithm 1). Furthermore, when we have the computed state values V, we can use these to construct a better policy π. For the case of deterministic policies, this can be done by following the Policy Construction algorithm (Algorithm 2). Clearly, these two algorithms can be executed subsequently to improve each other. Policy Iteration (Algorithm 3) is the algorithm that combines these two procedures into a complete policy construction strategy. As an alternative to these, Value Iteration (Algorithm 4) or State-Action Value Iteration (Algorithm 5) algorithms that integrates Policy Evaluation and Policy Improvement into a single step effectively can be incorporated to solve an MDP. All these algorithms require full access to the MDP components, e.g., iterating through every s in $\mathcal{S}$. If any of the MDP components are unknown, however, the problem formulation changes drastically as it is covered later in Section 2.2.

Algorithm 1 Policy Evaluation

Input: Policy π, termination criterion threshold ε

Initialise $V(s)$ arbitrarily (usually as 0) for every $s \in \mathcal{S}$
$\Delta \leftarrow$ an arbitrary value greater than ε, e.g. $\varepsilon + 1$
while $\Delta > \epsilon$ **do**
  for all $s \in \mathcal{S}$ **do**
    $V_{old} \leftarrow V(s)$
    $V(s) \leftarrow \sum_{a' \in \mathcal{A}} \pi(a' \mid s) \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a', s') \big[\mathcal{R}(s, a') + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |V_{old} - V(s)|)$
  end for
end while
return V

Algorithm 2 Policy Construction

Input: Policy π, value function V

$policy\_stable \leftarrow True$
for all $s \in \mathcal{S}$ **do**
  $a_{old} \leftarrow \pi(s)$
  $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma V(s')\big]$
  if $a_{old} \neq \pi(s)$ **then**
    $policy\_stable \leftarrow False$
  end if
end for
return π, $policy\_stable$

Algorithm 3 Policy Iteration

Input: Policy π, termination criterion threshold ε

$policy\_stable \leftarrow False$
while $policy\_stable$ is $False$ **do**
  $V \leftarrow$ PolicyEvaluation(π, ε)
  π, $policy\_stable \leftarrow$ PolicyConstruction(π, V)
end while
return π

Algorithm 4 State Value Iteration

Input: Termination criterion threshold ε

Initialise $V(s)$ arbitrarily (usually as 0) for every $s \in \mathcal{S}$
$\Delta \leftarrow$ an arbitrary value greater than ε, e.g. $\varepsilon + 1$
while $\Delta > \varepsilon$ **do**
  for all $s \in \mathcal{S}$ **do**
    $V_{old} \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |V_{old} - V(s)|)$
  end for
end while
return $\pi(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma V(s')\big]$

Algorithm 5 State-Action Value Iteration

Input: Termination criteria threshold ε

Initialise $Q(s, a)$ arbitrarily (usually as 0) for every $s \in \mathcal{S}$ and $a \in \mathcal{A}$
$\Delta \leftarrow$ an arbitrary value greater than ε, e.g. $\varepsilon + 1$
while $\Delta > \varepsilon$ **do**
  for all $(s, a) \in \{\mathcal{S} \times \mathcal{A}\}$ **do**
    $Q_{old} \leftarrow Q(s, a)$
    $Q(s, a) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a, s') \big[\mathcal{R}(s, a) + \gamma \max_{a' \in \mathcal{A}} Q(s', a')\big]$
    $\Delta \leftarrow \max(\Delta, |Q_{old} - Q(s, a)|)$
  end for
end while
return $\pi(s) = \arg\max_{a \in \mathcal{A}} Q(s, a)$

2.1.1 Markov Games

In some problem domains, there may be more than one agent interacting within the environment. This special case of a decision-making system is referred to as Multi-Agent System (MAS). Since it is assumed that only a single agent is active within the environment by definition in typical MDPs, MAS renders the standard MDP formalisation unsuitable to model its problems. Therefore, Markov Games (MG) (or

Stochastic Games) is proposed by Littman (1994) as an extension of MDPs to provide a more general framework that supports multiple agents.

In MG, the tuple that defines an environment takes the form of $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{R}, \mathcal{T}, \mathcal{N} \rangle$ where $\mathcal{N} = \{0, 1, \ldots, , n\}$ is a finite set of agents, $\mathcal{S}$ is a finite set of states, $\boldsymbol{\mathcal{A}}$: $\mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^n$ is a finite set of actions, $\mathcal{R}$: $\mathcal{S} \times \boldsymbol{\mathcal{A}} \rightarrow \mathbb{R}^n = (r^1, r^2, \ldots, r^n)$ is a reward function, $\mathcal{T}$: $\mathcal{S} \times \boldsymbol{\mathcal{A}} \times \mathcal{S} \rightarrow [0, 1]$ is a state transition probability function. As a consequence of these changes, the fundamental state value function in Equation 2.3 also transforms into the following, from perspective of an agent i:

$$V_i^{\boldsymbol{\pi}}(s) = \sum_{\boldsymbol{a} \in \mathcal{A}} \Big[\boldsymbol{\pi}(\boldsymbol{a} \mid s) \sum_{s' \in \mathcal{S}} \mathcal{T}(s, a^i, \boldsymbol{a^{-i}}, s') \big[\mathcal{R}(s, a^i, \boldsymbol{a^{-i}}) + \gamma V_i^{\boldsymbol{\pi}}(s') \big] \Big], \tag{2.8}$$

where $\boldsymbol{-i}$ denote the set of other agents (shorthand notation of $\mathcal{N} \setminus \{i\}$). According to this equation, the optimal policy π_i^* of agent i which is the best response to the other agents' policies then can be expressed as follows:

$$\begin{aligned} \pi_i^*(a^i \mid s, \boldsymbol{\pi_{-i}}) &= \arg\max_{\pi_i} V_i^{(\pi_i, \boldsymbol{\pi_{-i}})}(s) \\ &= \arg\max_{\pi_i} \sum_{\boldsymbol{a} \in A} \pi_i(a^i \mid s) \boldsymbol{\pi_{-i}}(\boldsymbol{a^{-i}} \mid s) \sum_{s' \in S} \mathcal{T}(s, a^i, \boldsymbol{a^{-i}}, s') \big[\mathcal{R}(s, a^i, \boldsymbol{a^{-i}}) \\ &\qquad + \gamma V_i^{(\pi_i, \boldsymbol{\pi_{-i}})}(s') \big]. \end{aligned} \tag{2.9}$$

As it can be seen, π_i^* now depends on the joint policy $\boldsymbol{\pi_{-i}}(\boldsymbol{a^{-i}} \mid s)$ which is determined by the joint action of other agents. This term introduces non-stationarity since it changes as the strategies of the corresponding agents change, and agent i has no control over it. This makes the solutions far more challenging to be achieved, as also detailed later in Section 2.2.1. It should also be noted that $\pi_i(a^i \mid s)$ and $\boldsymbol{\pi_{-i}}(\boldsymbol{a^{-i}} \mid s)$ are independent probability distributions at the time of execution even though they might have been derived by incorporating the information regarding each other's previous timestep versions.

2.1.2 Decentralised Partially Observable Markov Decision Processes

Another generalisation of MDPs, namely, Decentralised Partially Observable Markov Decision Process (Dec-POMDP)(Oliehoek & Amato 2016) extends the MG framework in two ways: partial observability and decentralisation. Partial observability is the case where the agent can't directly access the full state s but instead perceives an observation (also referred to as state) generated by an observation function. This variant is especially relevant for the realistic domains where the full states are usually inaccessible. Decentralisation is the concept that is related to the MAS, which refers to the case where the agents are operated by their individual controllers rather than having one centralised controller. A Dec-POMDP is defined by a tuple $\langle \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{R}, \mathcal{T}, \mathcal{N}, \boldsymbol{\Omega}, \mathcal{O} \rangle$ where $\mathcal{S}$ is a finite set of states, $\boldsymbol{\mathcal{A}} = \times_{i \in \mathcal{I}} \mathcal{A}^i$ is a finite set of joint actions, $\mathcal{R}$ is a reward function, $\mathcal{T}$ is a state transition probabilities, $\mathcal{N}$ is a finite set of agents as in Section 2.1.1; additionally, $\boldsymbol{\Omega} = \times_{i \in \mathcal{I}} \Omega^i$ is the finite of set of joint observations, $\mathcal{O}$ is the set of conditional observation probabilities. At each timestep t in a state s, each agent i perceives observation o^i determined by $\mathcal{O}(\boldsymbol{o} \mid s', \boldsymbol{a})$, where $\boldsymbol{a} = \langle a^1, \cdots, a^n \rangle$ is the joint action that caused the state transition from s to s' according to $\mathcal{T}(s' \mid \boldsymbol{a}, s)$, and receives reward r^i determined by $\mathcal{R}(s, \boldsymbol{a}_t)$. In the fully-cooperative case, every agent gets the same shared reward as $r^i = \cdots = r^n$. This formalisation is especially useful and necessary when we deal with a multi-agent problem where the agents are in cooperation and are responsible to act in a decentralised fashion while only getting local observations. In Chapter 4, we model our multi-agent problem domain as a Dec-POMDP.

2.2 Reinforcement Learning

When the complete model of an MDP is accessible, it is trivial to obtain the optimal policy as it is covered in Section 2.1. However, in a wide range of sequential decision-making problems, the environment components, e.g., $\mathcal{T}$, $\mathcal{S}$, $\mathcal{R}$, are usually unknown. In

these cases, the agent is expected to develop its policy by learning through trial-and-error interactions within the environment. This paradigm is referred to as Reinforcement Learning (RL) which, as a term, stands for both the class of problems and their solutions (Sutton & Barto 2018).

In RL, the state space, consequences of actions, state transitions and even the rewards are discovered progressively on the fly. Due to this, the ground-truth state and state-action values are no longer directly available to define an optimal policy. Therefore, RL methods that utilise these values rely on the estimations that are constantly updated through the collected samples instead. Even though there are also policy-based RL techniques that directly derive the policy itself via samples, e.g. policy gradients, we base our approaches in this thesis on the value-based class of them; therefore, we briefly describe only the most relevant value-based algorithms in the RL sections.

One of the fundamental and most widely used value-based RL algorithms is Q-learning (Watkins 1989) which can be seen as an RL equivalent of State-Action Value Iteration (Algorithm 5). Q-learning tries estimate the state-action values $Q(s, a)$ (which is a decomposition of value function $V(s)$ into actions) through repeated interactions by utilising the update rule given in Equation 2.10 where α is the learning rate. Then, these values are used to specify the agent's policy. A complete breakdown of this algorithm is shown in Algorithm 6 with its corresponding policy shown in Algorithm 7.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big]. \tag{2.10}$$

An important notion in RL is the exploration-exploitation trade-off. Since the agent lacks the knowledge about $\mathcal{S}, \mathcal{T}, \mathcal{R}$, it is responsible to explore these components at the same it is improving its policy to maximise its cumulative rewards. Therefore, RL algorithms also deal with the *exploration* problem in parallel, differently from the dynamic programming methods. For instance, in a typical policy often incorporated by Q-learning (Algorithm 7), the agent adopts a straightforward way to achieve this by alternating between random (exploration) and greedy (exploitation) actions. However,

Algorithm 6 Q-Learning

Input: Number of training steps t_{max}, learning rate $\alpha \in (0, 1]$.

Initialise $Q(s, a)$ arbitrarily (usually as 0) for every $s \in \mathcal{S}$ and $a \in \mathcal{A}$
for training steps $t \in \{1, 2, \dots t_{max}\}$ **do**
  Get observation s_t from *Environment* if it is reset
  $a_t \leftarrow \pi(s_t)$ ▷ Get action from policy, e.g., ϵ-greedy, for the current state s_t
  Execute a_t and obtain r_{t+1}, s_{t+1} from *Environment*
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\big]$
  $s_t \leftarrow s_{t+1}$
end for

Algorithm 7 ϵ-greedy Policy

Input: Action set $\mathcal{A}$, Q-value table (or Q-values approximator) Q, state s, exploration rate ϵ.

$u \sim \mathcal{U}(0, 1)$ ▷ Draw a number uniformly at random
if $u < \epsilon$ **then**
  return a random action drawn uniformly from $\mathcal{A}$ ▷ Exploratory action
else
  return $\arg\max_{a \in \mathcal{A}} Q(s, a)$ ▷ Greedy action
end if

as the problems and the RL algorithms become more complicated, the exploration strategies also need to be more sophisticated than this as we cover in the later sections.

2.2.1 Multi-Agent Reinforcement Learning

As it is described in Section 2.1.1, MAS introduces some additional challenges in the sequential decision-making problems. Linked to these differences in the foundation, the branch of RL that deals with MAS, namely, Multi-agent Reinforcement Learning (MARL), needs to overcome some exclusive challenges that have arisen due to the presence of multiple agents, as well as the present ones in single-agent RL that are further intensified. The most significant ones of these setbacks can be summarised as follows:

1. **Non-stationarity:** The state transition and reward functions depend on the

joint action of all agents. Since the environment is no longer stationary from an agent's perspective, the *Markov property* which is the key assumption of most single-agent RL algorithms, is invalidated. Therefore, the algorithms in MARL now have to deal with the non-stationarity-induced moving target problem.

2. **Curse of dimensionality:** MAS consist of multiple agents to take into calculations, result in a joint action space $\boldsymbol{\mathcal{A}}$: $\mathcal{A}^1 \times \mathcal{A}^2 \times \cdots \times \mathcal{A}^n$ which grows exponentially with the number of agents. These directly affect computational complexity. Additionally, as reported in Lowe et al. (2017), the probability of taking a gradient step in the correct direction in policy gradient algorithms decreases exponentially with the number of agents.

3. **Game-theoretic effects:** Interaction of multiple agents is accompanied by interesting mechanics. Due to the dynamic nature of strategies, convergence to equilibrium can not be relied on; therefore, environments need to be analysed with techniques like Evolutionary Game Theory (Bloembergen et al. 2015). In addition, even in conflict-of-interest situations, cooperation with competitors or vice versa may be crucial to obtain short or long-term benefits. Lastly, since the rewards are often correlated and can not be maximised individually, defining goals in MARL tend to be problematic.

4. **Multi-agent credit assignment:** The long-studied *credit assignment problem* in sequential decision-making and RL has some additional aspects in MARL. For example, in typical Q-learning, the values are stored for individual actions, yet they are actually determined by the joint action. This makes it difficult to figure out which agent's action is responsible for an outcome. This is referred to as *action shadowing* (Fulda & Ventura 2007). Furthermore, in *lazy agent problem* (Sunehag et al. 2017) that emerges in cooperative settings, agents sometimes become lazy and discouraged to explore, leading to a failure in learning as a consequence of having one successful active agent.

5. **Exploration-exploitation:** In MARL, agents now have to explore other agents' strategies as well as the environment dynamics through their learning process. Moreover, these strategies are not stationary, requiring them to be dealt with constantly in terms of exploration.

MARL research mainly revolves around overcoming these hindrances. Due to them being such a large variety, the approaches in MARL are also very diverse. For this reason, even the surveys in this subject (Shoham et al. 2003, Busoniu et al. 2008, Bloembergen et al. 2015, Hernandez-Leal et al. 2017, 2018) differ greatly in the way they analyse and categorise them. As pointed out in these studies, the primary subjects MARL research currently deals with can be listed as follows: the combination of deep learning with MARL, learning to communicate, emergent behaviour, opponent modelling, knowledge reuse, heterogeneous systems, analysis of emergent behaviour, tracking versus convergence, dynamic number of agents. Despite being dominated by Deep RL (described in Section 2.3.1) based techniques, it is not easy to identify state-of-the-art approaches because of the differences in the types of environment and training conditions followed in these. In this regard, in order to better understand what a MARL algorithm can address, it is helpful to determine which intersection of the following categorisations they fall:

- **Environment Objective:**

  [Covered in Bloembergen et al. (2015), Hernandez-Leal et al. (2018).]

  - **Fully cooperative:** The agents try to achieve a common goal and the reward functions are the same for every agent ($r^1 = r^2 = ... = r^n$).
  - **Fully competitive:** This scenario is also referred to as *zero-sum* where the agents have exactly opposite goals ($r^1 = -r^2$ for 2-agent case).
  - **Mixed:** The environment is neither fully cooperative nor fully competitive.

- **Environment Observability:**

  [Covered in Hernandez-Leal et al. (2017).]

  - **Full:** Agents can observe the complete state of the environment.

  - **Partial:** Agents only get a partial information o of the actual state s, determined by an emission function $\mathcal{O}(s)$.

- **Inter-agent communication:**

  [Covered in Hernandez-Leal et al. (2018).]

  - **Present:** Agents may explicitly exchange information with each other. This may have various specifications in terms of channels, dictionary size, bandwidth, permissions and noise.
  - **Absent:** Agents may not communicate.

- **Learning approach**:

  [Covered in Shoham et al. (2003), Bloembergen et al. (2015).]

  - **Independent Learners:** The agents learn independently by ignoring the presence of other agents and treating them as a component of a non-stationary environment. This way, they learn and act as they would in a single-agent environment.
  - **Joint Action Learners (JAL):** The agents model other agents' strategies explicitly. Specifically, they are aware of the presence of other entities and they incorporate this knowledge explicitly in their learning dynamics similarly to Equation 2.9.

- **Training and execution**:

  [Covered in Hernandez-Leal et al. (2018).]

  - **Centralised:** Agents are linked at the computational level and are managed together. This way, they can be trained by taking advantage of collective knowledge such as common value functions, e.g., $V(s)$, $Q(s, a)$.
  - **Decentralised:** Agents have no links between each other; their RL models are completely independent. The only way they can convey knowledge between each other is through their behaviour or communication (if the environment allows it).

The approaches that can exploit non-independent learning dynamics tend to be more successful with the aid of centralised training with decentralised execution (Rashid et al. 2018, Lowe et al. 2017, Foerster, Farquhar, Afouras, Nardelli & Whiteson 2017). However, such algorithms face problems when it comes to practical applications as it is not always possible to maintain the environment conditions in training. And it is a desired property for agents to be able to keep learning even when they are deployed in a decentralised way. Therefore, there is also a significant amount of interest in independent learners with decentralised learning (Tampuu et al. 2015, Leibo et al. 2017, Foerster, Nardelli, Farquhar, Torr, Kohli & Whiteson 2017) (or semi-independent decentralised learners with slight modifications (Omidshafiei et al. 2017)) since they exhibit promising empirical results despite of their shortcomings in performance in comparison with centralised methods. In our studies that deal with MARL (Chapter 4), we also follow the independent-learners approach.

2.3 Deep Reinforcement Learning

In the previous sections, the RL problem and one of the fundamental RL algorithms, Q-learning (Algorithm 6) are described. A key property of these classical RL techniques is that the state and state-action values are stored in a look-up table, which holds a specific entry for every state and state-action pair. While this seems sufficient, when the state space becomes more complex (10^{18} in checkers, $\approx 10^{50}$ in chess (Allis et al. 1994) for example), these approaches become infeasible due to two major drawbacks. Firstly, such large numbers of states make it impractical to store and look up the values. Secondly, in a tabular approach, every state is evaluated independently no matter how similar they are to a previously encountered state. As result, knowledge acquired from one state can't be generalised to others. This is a significant setback in learning with large state spaces and results in policies that have no way to deal with even slight state differences. Due to these, a family of RL algorithms were developed to address these setbacks via function approximation through the features generated from states.

According to these, the values such as $V(s)$ and $Q(s,a)$ are approximated as $\hat{V}_\theta(s)$ and $\hat{Q}_\theta(s,a)$ with parameters θ that is relevant to the selected function approximator.

To this date, various forms of techniques are applied, ranging from linear function approximation methods over features like coarse coding, tile coding, and radial basis functions to more complex, non-linear function approximation methods like artificial neural networks (Sutton & Barto 2018). Even though linear function approximation helps with generalising between similar states, its poor representational capacity has remained as a limitation for the algorithms. Moreover, generating the features for linear function approximation often requires manual labour and domain knowledge, which hinders the general applicability of the developed techniques. This has been especially problematic in domains with complex sensory inputs such as high-dimensional visual streams from a camera.

By the emergence of deep learning with the introduction of Convolutional Neural Networks and its groundbreaking success in visual tasks (Krizhevsky et al. 2012), deep neural networks (Goodfellow et al. 2016) became an attractive choice of non-linear function approximation for RL. Eventually, they have been successfully integrated with RL in the pioneering study of Deep Q-Network (DQN) (Mnih et al. 2013). This has led to the creation of the popular RL subfield: Deep RL. Today, the majority of RL research is concentrated on Deep RL to develop algorithms that employ deep neural networks as their function approximators due to their strong generalisation properties and removing the need for hand-specified (and often misspecified) input features. As a result of being studied extensively, the Deep RL family consists of many successful techniques devised for different problem conditions. In this thesis, we employ DQN among these due to its simplicity, customisability, and off-policy learning capability, i.e. being able to be trained with samples generated by a different policy. The next section describes DQN and its enhancements in detail.

2.3.1 Deep Q-Network

Deep Q-Network (DQN) (Mnih et al. 2013) is the scaled-up, Deep RL version of Q-learning (Algorithm 6) with non-linear function approximation that is suitable for complex domains, which attempts to obtain the optimal policy by learning state-action values approximation in an end-to-end fashion. In essence, it utilises a (deep) neural network with weights θ to map an input state s to $Q(s, a)$[1] by minimising the following loss term $\mathcal{L}_i(\theta_i)$ via stochastic gradient descent over randomly sampled transitions, where i refers to the current training step and $\mathcal{D}$ denotes the set of stored transitions $\langle s, a, r, s' \rangle$:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \big[r + \gamma \max_{a' \in \mathcal{A}} Q_{\overline{\theta}}(s', a') - Q_\theta(s, a) \big]. \tag{2.11}$$

As it is hinted by the components of this loss term, DQN operates in a different way than its counterpart Q-learning to maintain functionality and stability of learning in the domain of Deep RL. First of all, instead of using the encountered transitions to update the $Q(s, a)$ estimations instantaneously, these transitions are stored as $\langle s, a, r, s' \rangle$ tuples in a limited size first in-first out buffer $\mathcal{D}$ (also referred to as replay memory) to perform model updates via random batch of samples. Not only do such off-policy updates improve sample efficiency, but also help break the non-i.i.d. property of the samples to make the model's convergence easier. This is a common practice followed when training neural network models and is referred to as experience replay in the specific context of RL. Secondly, DQN keeps the network weights θ in a separate copy network as $\overline{\theta}$ that is updated periodically by copying the θ over and is used as a reference point in the loss function to stabilise learning. Without this tweak, the moving target problem in RL can be very problematic as it renders the model learning non-stationary. A simplified breakdown of DQN is given in Algorithm 8.

Following its breakthrough, DQN has been enhanced with some extension to alleviate its shortcomings and to make it compatible with more challenging RL tasks. The most

[1] For brevity, we use $Q(.)$ and $V(.)$ instead of $\hat{Q}_\theta(.)$ and $\hat{V}_\theta(.)$ to represent the approximated values in the context of Deep RL. Sometimes we may also use $Q_\theta(.)$ and $V_\theta(.)$ to explicitly specify the underlying approximation parameters.

Algorithm 8 Deep Q-Learning

Input: Number of learning steps t_{max}, learning rate α, target network update period T_{target}, training period T_{train}, replay memory capacity $N_{\mathcal{D}}$, replay memory size to start learning $M_{\mathcal{D}}$.

Initialise empty replay memory $\mathcal{D}$ with capacity $N_{\mathcal{D}}$ to store state transition tuples
Initialise online deep Q-network with weights θ randomly
Initialise target deep Q-network with weights $\overline{\theta}$ copied from θ
for training steps $t \in \{1, 2, \dots t_{max}\}$ **do**
  Get observation s_t from *Environment* if *Environment* is reset
  $a_t \leftarrow \pi_\theta(s_t)$ ▷ Get action from policy, e.g., ϵ-greedy (Algorithm 7)
  Execute a_t and obtain r_{t+1}, s_{t+1} from *Environment*
  Store $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ in $\mathcal{D}$
  Remove $\langle s_{t-N_{\mathcal{D}}}, a_{t-N_{\mathcal{D}}}, r_{t+1-N_{\mathcal{D}}}, s_{t+1-N_{\mathcal{D}}} \rangle$ from $\mathcal{D}$ if $|\mathcal{D}| > N_{\mathcal{D}}$
  if $\mathcal{D} \geq M_{\mathcal{D}}$ **and** $t \mod T_{train} = 0$ **then**
    Draw a minibatch of transitions $\mathcal{B}$ from $\mathcal{D}$
    Perform a gradient descent step with $\mathcal{B}$ to minimise Equation 2.11 (at α rate)
  end if
  $\overline{\theta} \leftarrow \theta$ if it is target network updating period ($t \mod T_{target} = 0$)
  $s_t \leftarrow s_{t+1}$
end for

prominent of these modifications are combined and studied under the name of Rainbow DQN (Hessel et al. 2018). We employed DQN as our main deep RL algorithm in every chapter of this study. The main reasons that motivated this decision of ours are as follows:

- It is an off-policy RL algorithm. This allows us to incorporate the transitions that are generated by a different policy in the learning process.
- It is a fundamental, simple yet efficient algorithm. This makes the impact of our modifications clearer and easier to be analysed. Furthermore, we believe that providing improvements for a simpler algorithm can make it applicable to a wider range of algorithms.
- It is well-studied in the literature with many useful modifications that suit different use cases. Among these, we employ Double Q-learning (van Hasselt et al. 2016),

Dueling Networks (Wang et al. 2016), Noisy Networks (Fortunato et al. 2018) and Prioritised Experience Replay (Schaul et al. 2016) in the different parts of this thesis, which we believe are the most impactful ones. These are described in detail in the remainder of this section.

Double DQN

In the default version of DQN, the *max* operator in Equation 2.11 (and also in the default version of Q-learning, Equation 2.10) relies on the same Q-values whether it is for selecting or evaluating an action. If there happens the be a case of overestimation in the process of building these Q-values, such selections amplify this negative effect even further. Double Q-learning (van Hasselt et al. 2016) algorithm alleviates this issue by modifying the DQN loss function to take the following form:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}\big[r + \gamma Q_{\overline{\theta}}(s', \arg\max_{a' \in \mathcal{A}} Q_\theta(s', a')) - Q_\theta(s, a)\big]. \tag{2.12}$$

As it can be seen, the *max* operation is now split in order to allow the greedy policy estimation to be done with the current values whereas the target network weights are still responsible for a fair evaluation of this policy. This modification brings significant performance improvements with very minimal trade-offs.

Dueling Networks

Another significant improvement DQN has had is achieved by employing a special network architecture designed for the value-based Deep RL algorithms, namely Dueling Networks (Wang et al. 2016). In this model, the final Q-values are constructed with the aid of two streams of input from that are separated in the previous layer as advantage and value[2]. This decomposition can be written in the following form:

$$Q_\theta(s, a) = V_{\theta,\theta_V}(s) + A_{\theta,\theta_A}(s, a), \tag{2.13}$$

[2]This is also visually illustrated in our network architectures in Figures 4.3, 5.1 and 6.1

where θ_V and θ_A are the weights belonging to the outputs of the value stream that produces state values (denoted by V) and the advantage stream that produces state-action advantage values (denoted by A), respectively. However, in this given form, it is impossible to recover unique V and A values for a given set of Q values. To address this, Equation 2.13 is transformed into the following form to ensure that the advantage is zero for the selected action:

$$Q_\theta(s,a) = V_{\theta,\theta_V}(s) + \left(A_{\theta,\theta_A}(s,a) - \max_{a' \in \mathcal{A}} A_{\theta,\theta_A}(s,a') \right). \tag{2.14}$$

Alternatively, the following form of the equation is proposed with more stable optimisation properties as it is reported in Wang et al. (2016), which then became the established version of Dueling Networks:

$$Q_\theta(s,a) = V_{\theta,\theta_V}(s) + \left(A_{\theta,\theta_A}(s,a) - \frac{1}{|\mathcal{A}|} \sum_{a' \in \mathcal{A}} A_{\theta,\theta_A}(s,a') \right). \tag{2.15}$$

This modification only applies to the network structure itself without any additional changes in the Deep RL algorithm. Therefore, it is also an extension of DQN that is preferred over the default version in the majority of the problem cases.

Noisy Networks

Noisy Networks (NoisyNets) (Fortunato et al. 2018) is an eminent exploration technique devised for Deep RL algorithms, which is also employed in Rainbow DQN (Hessel et al. 2018) as the default exploration strategy. In principle, NoisyNets is a modified linear layer with noise perturbations with the noisy weights $\theta \coloneqq \mu + \Sigma \odot \varepsilon$. Here, $\zeta \coloneqq (\mu, \Sigma)$ denote the learnable parameters, ε denote the zero-mean noise and $\odot$ represent element-wise multiplication. This modification causes the Deep RL loss to take the expectation form $\mathcal{L}(\zeta) = \mathbb{E}\big[\mathcal{L}(\theta)\big]$ over ε. Linked to this, a linear layer that is

defined as $y = wx + b$ now takes the following form:

$$y = (\mu_w + \sigma_w \odot \varepsilon_w)x + \mu_b + \sigma_b \odot \varepsilon_b, \tag{2.16}$$

where the input is $x \in \mathbb{R}^p$, the output is $y \in \mathbb{R}^q$, the learnable parameters are $\mu_w \in \mathbb{R}^{q \times p}$, $\sigma_w \in \mathbb{R}^{q \times p}$, $x \in \mathbb{R}^p$, $\mu_b \in \mathbb{R}^q$, $\sigma_b \in \mathbb{R}^q$, and the noise parameters are $\varepsilon_w \in \mathbb{R}^{q \times p}$, $\varepsilon_b \in \mathbb{R}^q$ for input size of p and output size of q. As the learning progresses, the noisy components are learnt to be ignored in a state-conditional trend. This gives the agent an implicitly driven exploration ability that diminishes for the states that are seen more often.

The exploration behaviour and the algorithm performance are defined by the specifications of the noisy layers. In terms of noise distributions, two options are presented:

- **Independent Gaussian noise:** Every weight and bias has independently generated noise from a unit Gaussian distribution resulting in $pq + q$ number of noise variables for each noisy later.
- **Factorised Gaussian noise:** Number of noise variables are reduced to $p + q$ by factorising $\varepsilon^w_{i,j}$ and ε^b_j as $\varepsilon^w_{i,j} = f(\varepsilon_i)f(\varepsilon_j)$ and $\varepsilon^b_j = f(\varepsilon_j)$. The real-valued function f is set as $f(x) = \text{sgn}(x)/\sqrt{|x|}$ in Fortunato et al. (2018). This is also the variant that is used in the experiments with Dueling Networks enhanced DQN.

Another important benefit of having NoisyNets is being able to obtain a form of uncertainty estimation by using the predictive variance of a noisy layer as described in (Chen et al. 2018). This is computed for action a and state s in the final layer that outputs $Q(s, a)$ values as follows:

$$\begin{aligned} Var[Q_\theta(s,a)] &= Var[w_a \phi(s)] + Var[b_a] \\ &= \phi(s)^\intercal \text{diag}({\sigma_{w_a}}^2)\phi(s) + {\sigma_{b_a}}^2, \end{aligned} \tag{2.17}$$

where $\phi(s)$ is the latent features generated from state s to be fed this layer, w_a and b_a are the weight and bias terms, respectively. The uncertainty in state s is then obtained

by taking the predictive variance of the action with the largest Q-value as follows:

$$U(s) = Var[Q(s, \arg\max_{a \in \mathcal{A}} Q(s, a))]. \tag{2.18}$$

Due to these two important advantages of providing advanced exploration capabilities and access to uncertainty estimations, we utilise NoisyNets in Chapter 4 (particularly for exploration) and Chapter 5 (particularly for uncertainty estimations). However, we also found that the addition of NoisyNets makes the experimental process significantly slower and more difficult to analyse after; therefore, it is left out in Chapters 6 and 7 where we do not necessarily need to benefit from these advantages.

Prioritised Experience Replay

In Deep RL environments with sparse rewards, it may take a significantly longer amount of time until the encountered rewards are incorporated into the learning due to the delayed experience replay dynamics of DQN. Consequently, DQN keeps doing its updates with outdated targets that slow down learning. In addition, sometimes, an infrequent transition may be much more useful to learn from regardless of having a reward associated with it. To alleviate these issues, Schaul et al. (2016) proposed Prioritised Experience Replay (PER) as a customised experience replay technique.

In PER, the transitions are stored in the replay memory with a priority value p_t computed proportionally to their temporal-difference error where ϱ defines the shape of the distribution:

$$p_t \propto \left| r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') - Q_\theta(s, a) \right|^{\varrho}. \tag{2.19}$$

Then, transitions are sampled from this buffer during learning with respect to their priorities (new samples are given maximum priority since they have no recorded error value). PER in general provides very significant learning speed benefits, but there are also situations it falls behind the default uniform replay memory sampling. Furthermore, non-uniform sampling from the replay memory influences the way the samples are

utilised in learning which interferes with our proposed techniques by making them either far more stronger or far more weaker. This makes it difficult to see the actual impact of the modifications. Therefore, we considered PER only in our first study (Chapter 4) in this thesis.

2.3.2 Random Network Distillation

Random Network Distillation (RND) (Burda et al. 2018) is a technique built to provide intrinsic curiosity rewards for RL agents to enhance their exploration capabilities in environments with sparse rewards. For this purpose, RND aims to assess the novelty of the encountered states. It involves the usage of two neural networks alongside the actual task-level RL algorithm, namely *target* and *predictor* networks, denoted by differentiable functions G and $\hat{G}$ respectively. These networks are identical in structure, which is defined arbitrarily, and are able to map state observations to embeddings as in $G\colon \mathcal{S} \to \mathbb{R}^k$ and $\hat{G}\colon \mathcal{S} \to \mathbb{R}^k$. They are initialised with different random weights; therefore, they produce different embeddings even for identical inputs in their initial state. Over the course of training, samples used in the task-level RL algorithm are used to train predictor network $\hat{G}$ using gradient descent to minimise the mean squared error $\|\hat{G}_\omega(s) - G(s)\|^2$. By doing so, the predictor network becomes more accurate at matching the target network's outputs for the samples it observes more, which is referred to as distilling a randomly initialised network. The error between target and predictor are expected to be higher for the type of states that are seen less frequently and it acts as a natural indicator of state novelty, which is used to augment the environment reward to provide the agent intrinsic motivation to explore.

Burda et al. (2018) highlights four types of causes behind the error between the predictor and the target networks:

1. **Epistemic Uncertainty:** This is caused by the insufficient number of training samples seen by the predictor; incorporating more samples in the learning makes this smaller.

2. **Aleatoric Uncertainty:** Stochasticity in the target function produces this error, e.g. stochastic environment dynamics when attempting to predict forward model dynamics.
3. **Model Misspecification:** This kind of error arises when the predictor model is incapable of matching the target function's complexity.
4. **Learning Dynamics:** Any kind of setback in the predictor model's optimisation process itself can cause this error.

Since the predictor model is set to be deterministic and identical to the target network (hence equally capable), #2 and #3 don't contribute to the error. Therefore, RND can be a good measurement of epistemic uncertainty which is the main factor to drive the exploration. This property motivated us to use RND scores as a measurement of novelty.

In order to be able to use RND reliably, it is especially important to have observations normalised due to having the target network's parameters frozen which renders it incapable to adapt to different scales. This consequently may result in extremely smaller variances in the embedding outputs without any valuable information. To tackle this issue, an observation normalisation scheme is suggested. According to this, every dimension of observation is whitened by subtracting the running mean and then dividing by the running standard deviation. Finally, the resulting observation is clipped to be in a certain range ($[-5, 5]$ by default). To do so, several steps of random behaviour are executed to collect observations to construct the running mean and the running standard deviations. This normalisation process is applied for both the target and predictor networks, independently from the actual RL algorithm's observation preprocessing. In this thesis, we employ RND for state novelty measurement as a proxy of tabular state counting instead of incorporating it as an extra reward in learning.

2.3.3 Dropout

Dropout (Srivastava et al. 2014) is a simple yet powerful regularisation method developed to prevent neural networks from overfitting. Its working principle is based on involving some random noise in the hidden layers of the networks. A neural network layer with the feed-forward operation can be described as $\boldsymbol{y} = f(\boldsymbol{w}\boldsymbol{x} + b)$, where the output is $\boldsymbol{y} \in \mathbb{R}^q$, the input is $\boldsymbol{x} \in \mathbb{R}^p$, the network weights for this particular layer are $\boldsymbol{w} \in \mathbb{R}^{q \times p}$ and $b \in \mathbb{R}^q$, f is any activation function, for input size of p and output size of q. In a layer with dropout, this equation takes the form of $\boldsymbol{y} = f(\boldsymbol{w}\tilde{\boldsymbol{x}} + b)$ where $\tilde{\boldsymbol{x}} = \boldsymbol{d} * \boldsymbol{x}$ represent randomly dropped out input which is determined by $d \sim Bernoulli(\chi)$. Hence, the learning process gets to be regularised with this random noise which is re-determined in every forward pass. The value χ controls the rate of dropout and is responsible for the regularisation strength. During the test time, χ is set as 0 hence disabling the dropout mechanism.

In addition to its regularisation capability, dropout can also be used to estimate the epistemic uncertainty of a neural network model, as investigated in Gal & Ghahramani (2016). For any particular input, performing forward passes multiple times yield different outputs due to the dropout-induced stochasticity, which can be treated as an approximation of probabilistic deep Gaussian process. Following this idea, the variance in these output values can therefore be interpreted as a representation of the model's uncertainty. Finally, since these forward passes can be performed concurrently, this approach provides a practically viable option to evaluate the uncertainty in deep learning models.

In Deep RL, the learning objective is to *overfit* to the task at hand during the training because it is assumed that the task will remain the same later unless there is a generalisation objective. Furthermore, Deep RL itself contains enough variance that provides natural regularisation to the learning process. Therefore, the conventional regularisation methods are usually refrained from in Deep RL models except for some special cases (Liu et al. 2021). In our studies, we also follow this idea and do not employ

any regularisation in the Deep RL algorithms themselves. Yet, we use them in our side techniques to have access to uncertainty estimations as we describe in Chapters 6 and 7.

2.4 Learning from Prior Knowledge

In this section, we describe the most widely adopted paradigms for leveraging prior knowledge into RL with a particular focus on those that are closely related to our core framework, *action advising*, which is described in detail in the final section along with the previously proposed algorithms.

2.4.1 Imitation Learning

Imitation learning is the process of learning some expert behaviour through presented demonstrations without any access to the environment rewards themselves. Even though the agent can be allowed to interact with the environment during this, it is assumed that no reward signals are available. This difference distinguishes imitation learning from the typical RL problem and makes it more similar to supervised learning. Imitation learning is especially useful in situations where it is more difficult to specify reward functions than to record/provide some expert demonstration.

The problem formulation is based on the MDP formalisation where the provided expert demonstrations dataset that contains state-action pairs is represented as $\mathcal{C} = \{\langle s_0, a_0\rangle, \langle s_1, a_1\rangle, \dots\}$. There are two main approaches in imitation learning:

- **Direct (Behaviour Cloning):** The expert policy is imitated through supervised learning via the samples in $\mathcal{C}$ by considering them as the input features and the labels.
- **Indirect (Inverse RL):** Goal or reward function of the expert policy is learned through the samples in $\mathcal{C}$; then, a policy is derived with this information.

In some of our studies (Chapters 6 and 7), we make use of the *direct* variant in its simplest form, Behaviour Cloning (BC) (Pomerleau 1991). In the case of Deep

RL, the expert policy is typically modelled with a function approximator with enough complexity, e.g. a neural network H_η with weights ω. An appropriate loss function such as $\mathcal{L}(\eta) = \sum_{\langle s,a \rangle \in \mathcal{C}} - \log H_\eta(a \mid s)$ is minimised to train this model by treating the state-action pairs in $\mathcal{C}$ as i.i.d. samples. Consequently, a state-conditional generative model is obtained that is capable of imitating the expert actions for the demonstrated states. In practice, however, this approach is unreliable to be used as a task policy as it is. This is because the agent often encounters states that are not contained in the provided dataset, and therefore, ends up exhibiting sub-optimal behaviour in these states which leads to further divergence in the trajectories. However, adopting the idea in this most basic form is sufficient in our study as it provides us with the adequate functionality of generating actions correctly for the states we ensure H_η is trained with.

In order to address the problem of distribution mismatch, an algorithm with a slightly modified training procedure, namely Dataset Aggregation (DAgger) (Ross et al. 2011) was proposed. In principle, it adopts the idea of typical behavioural cloning. However, it relies on the assumption that there is an expert in the loop that can be queried on demand during training. By using this advantage, the algorithm requests the expert labels (actions) for the unlabelled states, e.g. out-of-distribution ones, whenever they are encountered. At every iteration, the policy is rolled out to encounter new data and then is re-trained with the aggregated dataset to have better state coverage. Eventually, the imitated policy becomes an accurate approximation of the expert policy. This process is summarised in Algorithm 9.

2.4.2 Learning from Demonstrations

As its name suggests, Learning from Demonstrations (LfD) (Schaal 1996) idea is about utilising a dataset of previously recorded demonstrations to facilitate the agent's learning process. However, differently from Imitation Learning, LfD does this by also making use of the reward signals present in the RL problem. For this reason, it is considered to be a combination of Imitation Learning and RL.

Algorithm 9 Dataset Aggregation (DAgger)

Input: Number of training steps t_{max}, trajectory length T, expert with policy π^*.
1: $\mathcal{C} \leftarrow \emptyset$ ▷ Initialise empty demonstration dataset
2: Initialise initial policy π_1
3: **for** training steps $t \in \{1, 2, \dots t_{max}\}$ **do**
4: Sample T-step trajectory by executing π_i
5: Generate expert dataset $\mathcal{C}_i = \{\langle s_0, \pi^*(s_0)\rangle, \dots\}$ of the sampled states in the trajectory
6: $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_i$ ▷ Aggregate datasets
7: Train π_{i+1} with $\mathcal{C}$
8: **end for**

A fundamental LfD technique that is designed for Deep RL is called Deep Q-Learning from Demonstrations (DQfD) (Hester et al. 2018). They propose an LfD framework on top of DQN as the baseline algorithm. Specifically, a DQN agent is trained as it would be normally, but its learning is assisted by pre-populating its replay memory with the demonstration samples. During learning, these samples are incorporated efficiently with two additional changes in the base RL algorithm. Firstly, the agent is pre-trained for several steps before interacting with the environment. Secondly, the RL algorithm's loss function is customised with some additional terms. These modifications make DQfD different and more successful than its predecessors that only fill up the replay memory with previously collected data without any further treatment to them (Lipton et al. 2018). The DQfD loss $\mathcal{L}_{DQfD}(\theta)$ is formulated as follows:

$$\mathcal{L}_{DQfD}(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}}\big[\mathcal{J}_{DQ}(\theta) + \lambda_1 \mathcal{J}_n(\theta) + \lambda_2 \mathcal{J}_E(\theta) + \lambda_3 \mathcal{J}_{L2}(\theta)\big], \tag{2.20}$$

where,

$$\mathcal{J}_{DQ} = r + \gamma \max_{a'} Q_{\bar{\theta}}(s', a') - Q_\theta(s, a), \tag{2.21}$$

$$\mathcal{J}_n = r_t + \gamma r_{t+1} + \cdots + \gamma^{n-1} r_{t+n-1} + \gamma^n \max_{a \in \mathcal{A}} Q_\theta(s_{t+n}, a), \tag{2.22}$$

$$\mathcal{J}_E = \max_{a \in \mathcal{A}} [Q_\theta(s, a) + l(a_E, a)] - Q_\theta(s, a_E). \tag{2.23}$$

The term $\mathcal{J}_{DQ}$ is the default DQN loss (Equation 2.11). $\mathcal{J}_n$ is the n-step returns loss that was found to be especially beneficial in the pre-training phase by propagating the expert's trajectory. $\mathcal{J}_E$ is the large margin margin classification loss (Piot et al. 2014), where a_E is the demonstrator action taken in state s, $l(a_E, a)$ is a margin function that takes value of 0 when $a = a_E$ and a positive value otherwise. This loss term causes the values of the actions different from the demonstrator's choice to be at least a margin smaller than those of the demonstrator, which enforces imitation and helps ground the values of the unseen actions. Finally, $\mathcal{J}_{L2}$ is the regularisation term that is included to prevent the model from overfitting during the pre-training phase with a relatively small number of samples. The constants λ_1, λ_2 and λ_3 are hyperparameters that control the importance of their corresponding loss terms by weighting them in the aggregation to form the total loss $\mathcal{L}(\theta)$ in Equation 2.20. For instance, setting λ_1 to be greater will make the agent optimise for the n-step return loss $\mathcal{J}_n$ more than before as it will have more impact on the total loss $\mathcal{L}_{DQfD}$ and vice versa. Yet, setting these hyperparameters is non-trivial as every loss term can be on different scales.

2.4.3 Action Advising

Action advising (Torrey & Taylor 2013) is a peer-to-peer knowledge transfer framework that utilises the teacher-student paradigm to accelerate RL process. In its simplest form, an RL agent that is learning to perform a task (student) is monitored by a more knowledgeable peer (teacher) that tries to guide the student's learning with the advice it provides in the form of actions. Requiring only a common understanding over a communication protocol and a set of actions makes action advising one of the most flexible among the interactive knowledge exchange approaches (da Silva, Warnell, Costa & Stone 2020).

An important aspect of action advising is the limitation in the number of teacher-student interactions via a budget. This notion is incorporated considering the real-world challenges of limited inter-agent communication and human attention span; neither

can be expected to last over the course of a long training session. Therefore, the peer that is in charge of initiating the advice exchange interactions needs to be aware of this limitation and do so only in the most appropriate moments to make efficient use of the available budget. In the initially proposed version (Torrey & Taylor 2013), where the teacher constantly observes the student give advice (teacher-initiated), the teacher takes this responsibility; whereas in other forms, i.e. jointly-initiated and student-initiated (to be covered later), either of the peers can be in charge of these decisions. In addition to the budget limitation itself, the fact that the execution of a particular advised action can have different effects in the learning dynamics in different stages of learning requires the advice exchanges to be timed carefully. Therefore, the main goal of the action advising algorithms is to answer the question of "*when to ask for/to provide an advice?*" in the best way possible to address these altogether.

The aforementioned characteristics of action advising distinguish it from the similar learning paradigms we have covered previously. First of all, it does not require any pre-generated datasets of previous experience like Imitation Learning and DQfD. Thus, given that there is a peer to be interacted with, action advising can deal with problem settings where the environment properties are unknown until the time the learning agent is deployed for learning, or when it is costly and difficult to generate useful environment interactions to be provided to the agent. Secondly, the budget limitation in the interactions makes action advising significantly different from the approaches that also consider an interactive expert to be in the training loop to assist the student, such as DAgger. Due to these, action advising is capable of dealing with a unique set of problems, unlike any other similar technique.

There have been various action advising methods presented prior to the date we started conducting our study. While some of them rely on simple heuristics to distribute the advice, there are also more advanced ones that employ methods such as Meta RL. In the remainder of this section, we present a literature survey of the action advising approaches both in classical RL and Deep RL.

2.5 Action Advising in Classical RL

The idea of peer-to-peer knowledge sharing through actions has its roots in classical single-agent RL. Clouse (1996) is one of the first studies to investigate the benefits of integrating a competent agent's decisions in a learner's process. To do so, they combined RL with *apprentice learning* where the learning agent is made to mimic a trainer agent's behaviour by replicating its action decisions, in a form of student-initiated advising, in which a learning agent is assisted by an expert agent whenever it asks for advice about its decisions. For this purpose, a metric to assess the importance of a state s (as confidence from the student's perspective) is formulated as follows:

$$I(s) = \max_{a' \in \mathcal{A}} Q(s, a') - \min_{a' \in \mathcal{A}} Q(s, a'). \tag{2.24}$$

This measurement is compared with a threshold value k to determine whether a state s satisfies the condition $I(s) < \tau_{imp}$ for the student to request advice. In addition to showing that getting feedback from the *trainer* even at random times provides a performance boost, when to ask for advice is claimed to be critical.

Later, Torrey & Taylor (2013) proposed teacher-student action advising framework for RL for the first time. In this framework, an expert teacher agent with policy π_T constantly monitors a student agent's learning process and provides it action advice as $\pi_T(s)$ (which are the teacher's *best* actions for any given state s) whenever they are decided necessary. This study has introduced the notion of advice budget limitation considering the practical concerns regarding attention and communication. According to this, the teacher starts with a budget of b_{give} which is decreased by 1 every time it provides a piece of advice. When b_{give} becomes 0, the teacher can no longer give advice. Therefore, addressing the challenge of *when* to advise became more crucial with the introduction of this particular setting. They attempted to solve this problem by devising the following heuristic approaches:

- **Early Advising:** By intuition, the agents are thought to benefit more from

teaching earlier in the learning process; this forms the main motivation of this heuristic. The teacher provides advice to the student for every state from the beginning until it runs out of budget b_{give}.

- **Importance Advising:** Even though earlier states are usually very impactful in learning, not every single of them may worth to be given advice. Therefore, a budget may be better spent on the states that are determined to be more important. In this method, the importance of a state s is computed similarly to Clouse (1996), but this time from the teacher's perspective:

$$I(s) = \max_{a' \in \mathcal{A}} Q_T(s, a') - \min_{a' \in \mathcal{A}} Q_T(s, a'). \tag{2.25}$$

  This formula makes use of the teacher's fully-learned Q-values (Q_T) and is considered to be a better indicator of the state's importance. Advice is given whenever there is an available budget b_{give} and $I(s)$ greater than the pre-determined importance threshold τ_{imp}.

- **Mistake Correcting:** The budget efficiency of Importance Advising can be further improved by making the advice exchange be performed only when there is a disagreement in the selected action to be executed by the student and the teacher. Therefore, the student first announces its action-to-be-executed a. Then, if there is a budget b_{give} available and the conditions of $\pi_T(s) \neq a$ and $I(s) > \tau_{imp}$ are satisfied, the teacher proceeds with the advice. The extra communication steps make this heuristic less practical than the previous ones.

- **Predictive Advising:** A way to improve Mistake Correcting to make it easier to be employed would be removing the requirement for the student to broadcast its intended action. In this heuristic, this is achieved by training a model of the student's policy to predict its actions with a supervised learning model $\hat{\pi}_S$. For any state s the student is in, the teacher predicts its action a as $a = \hat{\pi}_S(s)$. Then, if there is an available budget b_{give} and the conditions $\pi_T(s) \neq a$ and $I(s) > \tau_{imp}$

hold, the student is given an action advice. Even though this approach removes the need for additional communication step, it suffers from some challenges. The supervised learning model should be capable of representing the student's policy efficiently; it needs to be trained incrementally; it should handle the noisy data generated by the student during its exploration stage; finally, it needs to take into account the fact that student's policy is also non-stationary.

The results of this work obtained in RL domains with linear function approximation settings have outlined the significance of action advising and have opened new research avenues by inspiring follow-up studies.

An extended version of this work (Taylor et al. 2014) later introduced two new state importance metrics of variance-based importance:

$$I_V(s) = \frac{1}{|\mathcal{A}|} \sum_{i=1}^{|\mathcal{A}|} (Q_T(s, a_i) - \overline{Q_T(s, a)})^2, \tag{2.26}$$

and absolute deviation-based importance:

$$I_D(s) = \frac{1}{|\mathcal{A}|} \sum_{i=1}^{|\mathcal{A}|} |Q_T(s, a_i) - \overline{Q_T(s, a)}|, \tag{2.27}$$

where $\overline{Q_T(s, a)}$ is the mean of the teacher's Q-values for a state s. The methods in both studies (Torrey & Taylor 2013, Taylor et al. 2014) were tested in more complex environments again with non-linear function approximation and obtained promising results.

Following these, Zimmer et al. (2014) treated the optimal way of spending the budget in terms of acceleration in student's learning performance as a RL problem and attempted to solve it from the teacher's perspective. By running a learning session multiple times, the teacher learns (via Meta RL) the most appropriate times to give advice based on the student's state in environment as well as its learning state. Moreover, a technique is proposed for the student to make the most of advice it acquires;

however, this requires modifications to be made in student's internal RL algorithm. Even though the results show how efficient these techniques can be, there are some significant shortcomings of this kind of approach. First of all, running a learning session multiple times changes the problem setting majorly. In the real world where there are no simulations and no previous access to an unseen task, this is clearly inapplicable. Secondly, the teacher's Meta RL reward mechanism was made to work by employing detailed information regarding the task properties, such as maximum obtainable reward and timesteps until the goal. This is also difficult information to obtain in realistic and more complex tasks. Therefore, Meta RL class of methods like this one tends to be more tailored for a small subset of domains.

In Amir et al. (2016), the action advising framework, that was originally proposed as a teacher-initiated, was converted into a jointly-initiated version. As opposed to the previous methods in which advising occurs only with the student (Clouse 1996) or the teacher (Torrey & Taylor 2013) initiation, the interactions are performed with the agreement of both sides. Jointly-initiated action advising makes use of two heuristics selected for the student and the teacher, and advice exchange is done if both conditions are satisfied. For instance, when both the student and the teacher employ the state importance heuristic, the student in a state s first checks whether it is uncertain ($I_{student}(s) > \tau_{ask}$ where τ_{ask} is the student's threshold). If this condition is satisfied, then the student forwards its request to the teacher, who then checks its importance metric to determine ($I_{teacher}(s) > \tau_{give}$ where τ_{give} is the teacher's threshold) whether it is suitable to provide advice. This way, the need for the student agent to be constantly monitored is discarded, which makes the action advising techniques more feasible to be incorporated, especially in the human-agent interaction settings. However, teachers are still required to have competent, fixed policies.

Zhan et al. (2016) extended the teacher-student framework to take advantage of getting advice from multiple teachers by combining the advice using a voting-based selection. Moreover, the case of getting suboptimal advice when the student surpasses the teacher's performance is also addressed. Even though this improvement relaxed the

requirement of expert teachers, it still assumes them to follow fixed policies that are good enough to provide advice. This study also provided theoretical analyses of action advising in these conditions, however, the findings are limited to very simple tabular environments.

Fachantidis et al. (2019) made further investigations in learning-to-teach concept based on Zimmer et al. (2014). They distinguished the qualities of being an expert in the task from being good at teaching, claiming that the best performers are not necessarily the best teachers. As well as learning *when* to advise, teachers are made to learn *what* to advise to make the student learn as quickly as possible. This work shares the same drawbacks with Zimmer et al. (2014) in terms of being limited to the Meta RL suitable problem cases.

All of these previously mentioned studies are based on agents that operate in isolated single-agent environments. In MARL, multiple agents simultaneously learn in the same environment while affecting each others' policies, rendering it non-stationary. Due to this, even when we have agents with good policies, they can no longer be guaranteed to follow a fixed policy. Moreover, the assumption of teacher and student roles within agents is not straightforward to hold in MARL, especially when there is a large number of agents. Despite being a challenging domain, these properties of MARL reflect a more natural and realistic application case for peer-to-peer knowledge sharing.

The application of action advising methods in MARL is a challenging and fairly new subject. da Silva et al. (2017) was the first to propose a teacher-student framework that is suitable for multi-agent settings where agents are a part of the environment from each other's perspective and they learn simultaneously. They extended the heuristics from Torrey & Taylor (2013) by introducing several metrics based on the number of state visits to measure confidence in a given state, in order to make it possible for the agents to do self-assessment in order to overcome the challenge of having no fixed roles of student and teacher via executing jointly-initiated advice exchanging (as in Zhan et al. (2016)). In every state s, agents determine their advice-asking and advice-giving

probabilities as follows:

$$P_{ask}(s, \Upsilon) = (1 + v_a)^{-\Upsilon(s)}, \tag{2.28}$$

$$P_{give}(s, \Psi) = 1 - (1 + v_g)^{-\Psi(s)}. \tag{2.29}$$

Here, Υ and Ψ are the confidence evaluation functions; v_a and v_g are the scaling hyperparameters. With the advantage of being designed for non-Deep RL domains, the confidence functions are designed to utilise state counters:

$$\Upsilon(s) = \sqrt{n_{visits}(s)}, \tag{2.30}$$

$$\Psi(s) = \log_2 n_{visits}(s). \tag{2.31}$$

During training, agents interact with each other based on the probabilities computed according to these formulas. Advice from multiple peers is fused with majority voting. They tested their methods in Half Field Offense environment (Hausknecht et al. 2016) using a cooperative team of agents utilising SARSA(λ) with linear function approximation as task-level policies; and showed that the proposed modifications accelerated team-wide learning. However, the complexity of the non-linear function approximation was kept limited as they rely on state visit counters. They also did not develop any mechanisms for agent selection for advice requests, as well as for the fusion of advice.

In Omidshafiei et al. (2018), teaching in MARL is approached as an advising-level Meta RL problem as in Zimmer et al. (2014), Fachantidis et al. (2019). They proposed a centralised training and decentralised execution procedure using multi-agent actor-critic Deep RL (*learning to coordinate and teach reinforcement*) for teaching-level policies and tabular Q-learning for the agent's task-level policies. They demonstrated the efficiency of the method in a set of different environments including a repeated game, and two different gridworld environments, namely hallway game and room domain. One important aspect of this is highlighting that optimal action given by an expert agent is not always the best possible advice a learning agent can take to accelerate its learning. Despite its promising results, this work has some shortcomings which

are being limited to 2 agents only, using tabular and linear representations in simple environments, and requiring multiple, centralised learning sessions before its execution. Moreover, since advising occurs bidirectionally without any fixed roles, the agents learn these behaviours at the same time and therefore end up actually being tuned for each other only.

Differently from the previous work on this subject, Zhu, Cai, Leung & Hu (2020) addressed the issue of utilising the collected advice further. They consider the cases where the communication may be noisy to rely on a constant stream of advice and the budget may be too limited. Furthermore, it is also highlighted that the student agent usually needs to execute the same advice multiple times until it is actually learned, especially when there is stochasticity in the environment. As a solution to these, they proposed several techniques to reuse the previously collected advice. The state-advice pairs are stored in a look-up table; then, they are reused based on the selected reuse heuristic which can be one of the following:

- **Q-change per Step:** This is applicable when the student uses Q-learning. The state-advice pairs are kept in the look-up table to be reused whenever they are available as long as their execution leads to an increment (determined by a threshold) in their corresponding Q-values.
- **Reusing Budget:** Separate reusing budgets are set for the states which are decreased every time advice reusing occurs. Advice are reused for any state as long as there is an available reusing budget for it.
- **Decay Reusing Probability:** Advice reusing happens according to a state-dependent probability function that is based on state visit counts. These probability values are also decayed after every reuse.

These were found to be very effective to make more of the advice budget in the experiments conducted in both single-agent and multi-agent environments. Reusing not only lets the student execute the advice multiple times regardless of the budget without consuming it, but also can potentially help the student spend it more efficiently with

the aid of a previous state-advice look-up table. Nevertheless, the techniques introduced in this study are applicable to a limited set of non-Deep RL domains only.

2.6 Action Advising in Deep RL

The adoption of action advising techniques in the Deep RL domain has begun only recently. Following Omidshafiei et al. (2018), Kim et al. (2020) extended the idea of learning to efficiently give and take advice in Omidshafiei et al. (2018) to agents that use deep neural networks in their task-level policies and act in environment with continuous state and action spaces. In order to do so, they employed Hierarchical RL (Nachum et al. 2018) and took advantage of some algorithm-specific properties to tackle teacher credit assignment problem in deep RL action advising scenarios. Despite being effective in forming teacher-student interactions bidirectionally without any fixed roles, these learning-to-teach approaches share the drawbacks of requiring the agents to be trained over multiple, centralised learning sessions due to the Meta RL mechanisms. This may also undesirably cause the agents to end up being tuned for each other and face difficulties when paired with different peers.

Another line of work in Deep RL involves applying action advising methods by following heuristics without any pre-training, which forms the specific group of approaches our study belongs. In (Chen et al. 2018), the DQfD algorithm was combined with active learning by making the student agent query for demonstrations itself in a very similar way to action advising. To do so, they make use of some specialised network architectures that make it possible for the student agent to measure its state uncertainty, and be able to use this estimation to determine the most appropriate times and states to request advice for.

More recently, at the time this Ph.D. study was being conducted, da Silva, Hernandez-Leal, Kartal & Taylor (2020) proposed a heuristic-based student-initiated action advising algorithm for Deep RL. They utilised uncertainty measurements obtained via multiple neural network heads in the student's model to time the advice requests in single-agent

domains, similarly to (Chen et al. 2018); however, they don't employ a special loss function as in DQfD. The uncertainty calculation for a state s is performed as follows:

$$U(s) = \frac{\sum_{\forall a \in \mathcal{A}} var(\boldsymbol{Q}(s,a))}{|\mathcal{A}|}. \tag{2.32}$$

Here, $\boldsymbol{Q}(s,a)$ denotes the matrix of h different (determined by the number of heads h) $Q(s,a)$ values, and $var(\boldsymbol{Q}(s,a))$ denotes the variance of these values. Their experiments against other baselines of Importance Advising and randomly advising in a simple GridWorld domain as well as in the more complex Atari game Pong demonstrated significant improvements both in learning acceleration and budget usage efficiency. An obvious drawback of these uncertainty-based approaches is that they require specific neural network architectures for accessing uncertainty estimations to be functional.

Chapter 3

Games in this Thesis

In this chapter, we describe the game environments we used as evaluation domains in the experiments of this thesis.

3.1 Cover the Landmarks

This is a multi-agent grid-based game with discrete space and time, in which the agents must spread and cover the landmark positions cooperatively to maximise their shared rewards. The grid is sized 10×10 and consists of 3 agents and 3 landmarks. The objective of the agents is to move as quickly as possible to cover all the landmarks of the level. While the landmarks remain stationary, the agents can navigate around the grid by using a discrete set of actions $\mathcal{A} = \{\ Do\,Nothing, Move\,Up, Move\,Down, Move\,Left, Move\,Right\ \}$. Movement actions executed by an agent change its position on the grid by 1 tile in the corresponding direction. At each timestep, the agents observe the relative positions of the other agents and landmarks in form of x-axis and y-axis values, plus the current timestep. All values are normalised to $[-1, 1]$ using the maximum distance and game duration, respectively. Agents also receive a common reward in $[0, 1]$, determined by how many of the landmarks have an agent covering

them. The game's reward is calculated as:

$$\frac{1}{|\mathcal{L}|}\sum_{l=1}^{|\mathcal{L}|}\mathbb{1}_{(\sum_{i=1}^{|\mathcal{N}|}\mathbb{1}_{\|d_{li}\|=0})} \geqslant 1, \tag{3.1}$$

where $\mathcal{N}$ is the set of agents; $\mathcal{L}$ is the set of landmarks; $\mathbb{1}$ is the indicator function which equals 1 if the value of its subscript satisfies the condition coming after the notation, and equals to 0 otherwise; d_{li} is the Manhattan distance between landmark l and agent i. In other words, in order to act optimally and maximise the total reward, the agents must determine the lowest cost (in terms of distance) of the agent-landmark pair set and move to the appropriate landmark following the shortest path. Since the agents have no access to a forward model, they are expected to learn these strategies through interactions.

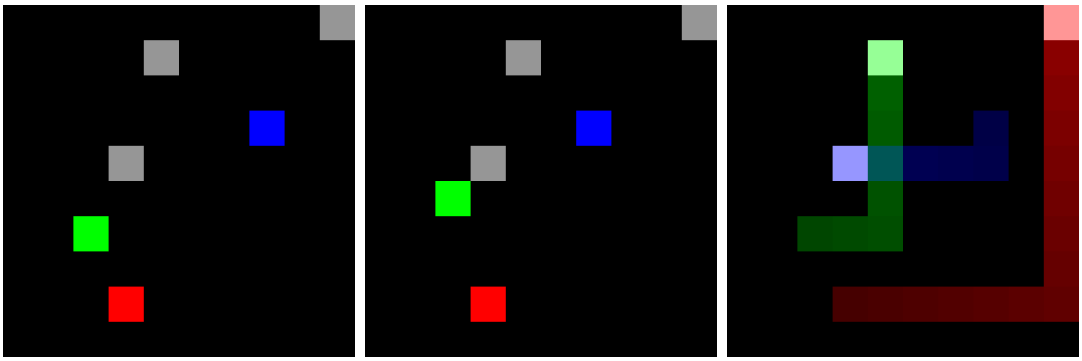


Figure 3.1: Three visualised frames from the Cover the Landmarks game. On the left is an initial game state at $t = 0$ with three landmarks (grey) and three agents (blue, green and red). In the middle is an advanced game state at $t = 1$ when the agents (blue, green and red) take actions On the right is the terminal state with all agents over the landmarks with their (suboptimal) trajectories are shown as a shaded colour.

A set of rendered visuals can be seen in Figure 3.1 with Blue (B), Green (G) and Red (R) agents Figure 3.1 (a) shows an example initial state s_0 at $t = 0$. Figure 3.1 (b) shows the successor state s_1 at $t = 1$ resulted by the agents B, G, R taking actions $Move\,Left$, $Move\,Up$, $Do\,Nothing$, respectively. Following are the numerical observations the

agents perceive as observations in these states s_0 and s_1:

$$
\begin{aligned}
\boldsymbol{o}_0^B &= [\ \ 0.3\ -0.5\ \ 0.5\ -0.4\ -0.2\ -0.3\ \ 0.1\ -0.4\ -0.3\ \ 0.2\ \ 1], \\
\boldsymbol{o}_0^G &= [-0.3\ \ 0.5\ \ 0.2\ \ 0.1\ -0.5\ \ 0.2\ -0.2\ \ 0.1\ -0.6\ \ 0.7\ \ 1], \\
\boldsymbol{o}_0^R &= [-0.5\ \ 0.4\ -0.2\ -0.1\ -0.7\ \ 0.1\ -0.4\ \ 0\ \ -0.8\ \ 0.6\ \ 1].
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{o}_1^B &= [\ \ 0.2\ -0.4\ \ 0.5\ -0.3\ -0.2\ -0.2\ \ 0.1\ -0.3\ -0.3\ \ 0.3\ \ 0.92], \\
\boldsymbol{o}_1^G &= [-0.2\ \ 0.4\ \ 0.3\ \ 0.1\ -0.4\ \ 0.2\ -0.1\ \ 0.1\ -0.5\ \ 0.7\ \ 0.92], \\
\boldsymbol{o}_1^R &= [-0.5\ \ 0.3\ -0.3\ -0.1\ -0.7\ \ 0.1\ -0.4\ \ 0\ \ -0.8\ \ 0.6\ \ 0.92].
\end{aligned}
$$

Here, the first 10 numbers denote the relative distances of entities in the y-axis and x-axis, where the origin is considered to be at the upper-left corner of the observation grid. In o^B, the order of pairs is green agent, red agent followed by the landmarks; in o^G, the order of pairs is blue agent, red agent followed by the landmarks; in o^R, the order of pairs is blue agent, green agent followed by the landmarks. All of these observation arrays are also appended with the remaining timesteps value for the episode (normalised to be in $[-1, 1]$). In this particular example, the order of landmarks in the observation array is the middle landmark, the left landmark and the right landmark (according to Figure 3.1); however, the landmarks are equal with no special meanings and their order is randomly determined every time a level is generated.

Despite this environment's representational and mechanical simplicity, it still is capable of presenting complex behavioural challenges for MARL. Therefore, this game was employed in the experiments in Chapter 5 to focus on behavioural learning challenges while keeping the computational expense at minimum. The game's complexity was tuned in terms of observation size and reward sparsity, in order to ensure that the environment is solvable at least sub-optimally by independent DQN learners employed in Chapter 5, and there remains to be some exploration challenge to keep the knowledge of the peers valuable to the agent.

3.2 Reach the Goal

Reach the Goal[1] is a grid structured environment with a size of 9×9, composed of ground (grey), pit (black) tiles, and goal (green) as visualised in Figure 3.2 where the agent is represented in white. The environment is perceived by the agent as binary tensors with a size of $9 \times 9 \times 3$. In this observation, every channel represents the presence of different tile types which are the agent, grounds (or pits) and the goal. The agent starts in a fixed position in the top-left corner of the top-left room (as shown in a white cell in Figure 3.2, top-left), and must navigate to the goal tile in the top-right corner of the top-right room, with 4 available actions of spatial movement in order to get a reward of +1. Every time the agent attempts to take an action, it has a chance to execute another action uniformly at random instead, with a probability of 0.1; however, if a random action happens to move the agent on a pit tile, it instead holds the agent in its previous position to prevent uncontrollable terminations. Reaching the goal, stepping on a pit tile or exceeding the maximum timesteps of 50 terminates the episode with 0 points of reward. Considering the possibility of random movements and the tight timesteps limit, this game presents a significant exploration challenge despite its simple rules.

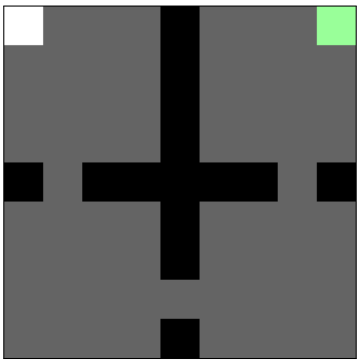


Figure 3.2: Rendered observation of Reach the Goal's initial state.

[1]Codes are available at `https://github.com/ercumentilhan/gridworld/tree/1df98160e8c009f61e4bac68aae025e54ed7aa87`

3.3 MinAtar

MinAtar (Young & Tian 2019) is an environment composed of minimal versions of five popular Atari games[2]. While the core game mechanics are kept the same, the observation and the action spaces are reduced to cut down the representational complexity. Every game has a common action space of 6, and the observations are binary tensors with sizes of $10 \times 10 \times n$, where n denotes the number of the object categories in the game. Different from the default version, we set any game episode to have a maximum timesteps of 1000. Figure 3.3 shows the screenshots taken from these games, and their brief descriptions are as follows:

- **Asterix:** The game has a constant stream of enemies and treasures that spawn and move in horizontal directions. The agent can move in 4 directions and must collect treasures to obtain +1 reward while avoiding enemies which kill the agent and terminates the game.

- **Breakout:** The agent controls a paddle in the bottom of the screen by moving it in horizontal directions. The objective is to keep hitting the ball to keep it within the screen to avoid losing the game. Hitting the bricks with the ball breaks them and yields +1 reward. New lines of bricks keep appearing after they are cleared up.

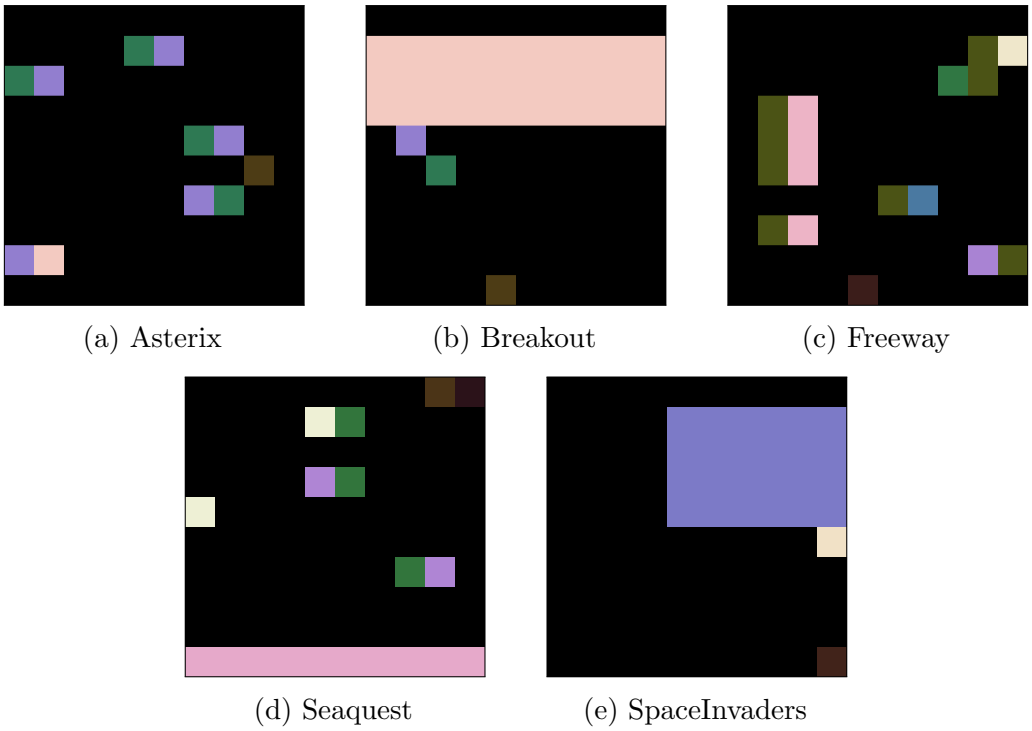
- **Freeway:** The agent's objective is to cross the way by avoiding cars moving at different speeds in horizontal directions. Upon reaching the goal, +1 reward is awarded and the agent is sent back to the starting point, and the cars' directions and speeds are randomised. Being hit by a car also sends the agent back to the starting point.

- **Seaquest:** This is the game with the most complex rules among MinAtar games. The agent controls a submarine in a sea filled with enemy submarines, fish, and

[2]Our version: `https://github.com/ercumentilhan/MinAtar/tree/original`

divers to be rescued by navigating around. Shooting the enemies yields +1 reward, as well as taking each diver to the surface. Moreover, the agent must keep an eye on the remaining oxygen level and have it replenished by going to the surface; which increases the difficulty each time it is done. If the agent goes up to the surface with no divers, is hit, or has no remaining oxygen, the game terminates.

- **Space Invaders:** The agent is in charge of controlling a spaceship that can shoot bullets at the upcoming group of aliens from the top of the screen. Each shot-down alien yields a reward of +1, and upon being cleared up, a new wave of aliens spawns with an increased movement speed. The aliens can also shoot back at the agent. If the agent is hit by a bullet or an alien, the game is terminated.



(a) Asterix (b) Breakout (c) Freeway

(d) Seaquest (e) SpaceInvaders

Figure 3.3: Rendered observations of random states from MinAtar games Asterix, Breakout, Freeway, Seaquest and Space Invaders.

Directions of the moving sprites are also encoded in the observations by having a separate category for their trails to ensure full observability. Asterix, Seaquest and

Space Invaders also involve a periodical difficulty ramping that occurs at every $100th$ timestep.

3.4 Arcade Learning Environment

In order to have an adequate amount of representational complexity in the experiments, e.g. continuous state space, that is relevant to the modern deep RL algorithms, we employ the widely experimented Arcade Learning Environment (ALE) (Bellemare et al. 2013) that contains more than 100 Atari 2600 games.

We selected a subset of 5 well-known games among these, namely Enduro, Freeway, Pong, Q*bert, Seaquest, which involve different mechanics and present various learning challenges to be used in the different stages of our studies.

- **Enduro:** The player controls a racing car on a long-distance track over multiple in-game days. On each day, if the player manages to pass a certain number of other cars (200 on the first day, 300 on the rest) in the race, it gets to advance to the next day. Progression during the days is visualised by different colour schemes that resemble the day-night cycle. Furthermore, there are other factors of seasonal events that affect the gameplay such as fogs and icy patches appearing on the road. Finally, as the days progress, the game increases in difficulty due to the other cars' behaviour becoming more aggressive.

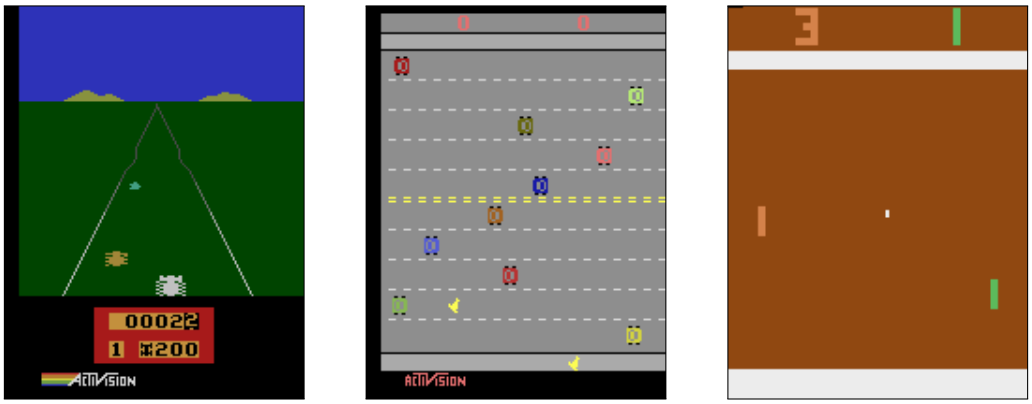
- **Freeway:** In this game, the objective is to cross a chicken across a highway comprised of ten lanes with vehicles traversing in different directions and speeds. If the player hits the cars along the way, it gets pushed back toward starting point. Every time the player manages to reach the goal, it acquires a reward and gets teleported back to the starting point.

- **Pong:** This game consists of two paddles on each side of the screen and a ball traversing around. The paddles are controlled by one player each. The players

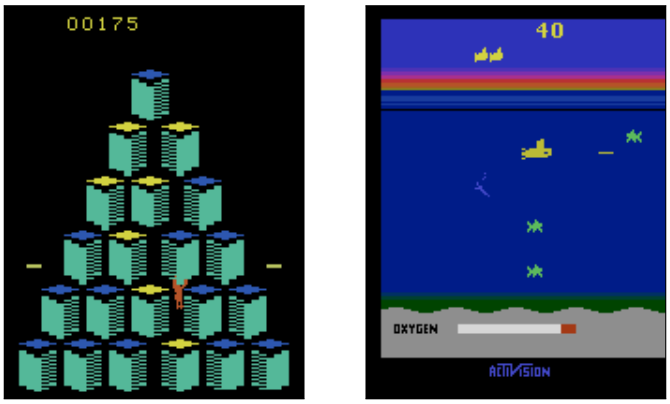
must hit the incoming balls to avoid them passing through their side as well as getting them thrown back at the opponent. If a player lets the ball pass through the gap behind its paddle, the opponent earns 1 point. In the single-agent variant of this game used in our study, the player controls the right side paddle while the other one is controlled by a built-in AI.

- **Q*bert:** The player is in control of an avatar that navigates over platforms in a pyramid-structured layout. In order to advance the game to the next stage, every platform must be jumped on once to have its colours changed. In the later stages, it takes multiple jumps to change the colour of a platform. Even though the player receives rewards for every successful platform interaction, it requires solving the puzzle elements in the game layout to beat the game levels.

- **Seaquest:** The player controls a submarine manoeuvring in the sea to destroy enemy sprites by shooting at them while also rescuing the divers to score. While doing so, the player must avoid the bullets sent by the enemies as well as not having its oxygen-depleted completely; else the 1 life is lost, and the game terminates after 3 losses. In order to refill the oxygen and return the rescued divers, the submarine must go to the surface. As time passes, the game becomes gradually difficult with more aggressive enemies with different behaviours.



(a) Enduro (b) Freeway (c) Pong

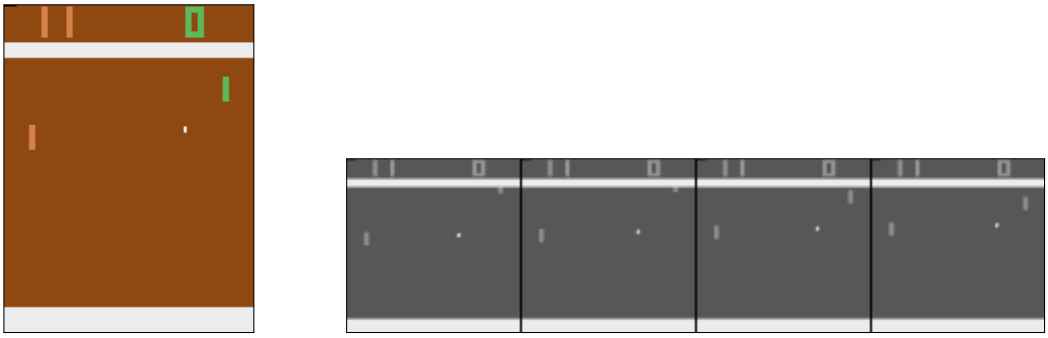


(d) Q*bert (e) Seaquest

Figure 3.4: Rendered observations of random states from ALE games Enduro, Freeway, Pong, Q*bert and Seaquest.

Each of these games has an observation size of $160 \times 210 \times 3$, representing RGB images of the game screen that are produced at 60 frames per second (FPS). To make experimenting in these games computationally tractable, we employ some preprocessing steps that are also followed commonly in other studies (Castro et al. 2018). First, each observation is made greyscale and resized down to the size of $80 \times 80 \times 1$. Since the games run at a high FPS, the frame that is shown to the player is set to be only every 4th one (which is composed of the maximum pixel values of previous 3 frames), and the player's actions are repeated for the skipped frames. Moreover, since these games contain a fair amount of partial observability, such as the direction of the ball in Pong, the final form of the observation to be perceived by the player is made to be a stack of

4 pre-processed frames with a size of $84 \times 84 \times 4$ (which contains the information of the most recent 16 actual game frames). An example of input and output observations generated with this preprocessing technique in the game of Pong is shown in Figure 3.5. The motion of the ball is obvious in the preprocessed version whereas this information is absent in the original input. This is a clear example of how beneficial frame stacking can be to deal with partial observability in ALE.



(a) Input observation (b) Preprocessed observation

Figure 3.5: A $160 \times 210 \times 3$ RGB input observation from the game of Pong (a) and its $84 \times 84 \times 4$ greyscale preprocessed version (unstacked) augmented with previous frames.

In order to deal with the varying range of reward scales and reward mechanisms within these games, every reward obtained in a single step in the game is clipped to be in $[-1, 1]$.

Every game episode is limited to last for maximum 108k frames by default, which corresponds to approximately 30 minutes of actual gameplay time in real life. In some of our experiments, the maximum episode length may be different than this default value; it is stated in that case.

Finally, another set of modifications also takes place to introduce more stochasticity within the games to turn them into more challenging RL tasks. At the beginning of the games, the player takes no-op actions a random number of times in $[0, 30]$, to simulate the effect of having different initial states. Additionally, with a probability of 0.25, the actions executed by the player are repeated for an additional step, which is referred to as *sticky actions*.

Chapter 4

Teaching on a Budget in Multi-Agent Deep Reinforcement Learning

This chapter covers our first study that is published with the title "Teaching on a Budget in Multi-Agent Deep Reinforcement Learning" (Ilhan et al. 2019). The research questions we addressed in this chapter are as follows:

- [RQ1] To what extent can we accelerate Deep RL via (budget-limited) action advising with one or more knowledgeable peer(s)?
- [RQ2] How can we scale the state-of-the-art action advising approaches from classical RL to Deep RL/MARL domains?

4.1 Introduction

There are many opportunities for knowledge reuse in MARL presented through the addition of other agents (da Silva et al. 2018, Da Silva & Costa 2019). The earliest works in this line of research involve learning the distributions of agents' roles to use as priors in new tasks with different roles (Wilson et al. 2008), reusing joint action policies

across tasks of varying difficulty in terms of agent number (Boutsioukis et al. 2011) and employing propositional rule-based methods across tasks decomposed in complexity (Vrancx et al. 2011). Later, in Taylor et al. (2013) a framework for simultaneous learning in source and target tasks through inter-agent experience transfer is proposed. Curriculum learning has also been a popular choice of making learning feasible within complex strategies (Bansal et al. 2017) and environments (Shao et al. 2018). Obviously, even though the existing challenges make it more problematic to employ the prominent single-agent knowledge reuse techniques, the range of opportunities also becomes much wider.

In MARL, an agent can potentially benefit from other agents' knowledge via observing and/or communicating with them. This makes it possible to reuse the knowledge obtained in the same task from different sources. Due to this, MARL is considered to be a very suitable and attractive domain for the action advising framework. Nonetheless, the presence of such a rich set of possibilities brings some difficulties together. The most significant of these MARL inherent knowledge reuse challenges that are relevant to action advising, in particular, can be listed as follows:

- The algorithms have to deal with not only the differences in the tasks themselves but also the number and type of agents present in them.

- Choosing the right agent(s) to learn from is difficult. The system may be heterogeneous, containing agents of different types with distinct abilities and goals. Furthermore, it may not be obvious which agents actually have useful information and are worth learning from.

- Determining the right moments to obtain and utilise knowledge from other agents is important. Communication between agents may have limited bandwidth. Additionally, performing such transfers too frequently either through communication or observation may be infeasible in terms of computational budget.

- Partial observability and limitations in communication may result in distorted

information exchange between the agents.

- Consistently fusing heterogeneous information acquired from different types of sources is not straightforward.

Due to these challenges, the research on the applications of action advising techniques in MARL has not seen a great progression. At the time of conducting this particular study, the most recent approaches were the ones that focus on having agents autonomously learning to perform knowledge transfer between each other without having fixed teacher-student roles (da Silva et al. 2017, Omidshafiei et al. 2018) in non-Deep RL domains.

In this study, we aim to showcase the significance of peer-to-peer knowledge exchange via action advising framework in a cooperative Deep MARL scenario for the first time in literature. For this purpose, we extended the idea of multi-agent ad hoc peer-to-peer action advising with no fixed roles (da Silva et al. 2017) to be compatible with Deep RL agents. The problem setting we deal with does not hold any assumptions of teacher-student roles for agents, is completely decentralised in training and execution stages and assumes that the communication between agents is limited. We chose these specifications to reflect real-world challenges as accurately as possible. The novel technical contributions we have made in this work can be listed as follows:

1. We proposed a novel heuristic-based jointly-initiated action advising strategy that is compatible with Deep RL agents and multi-agent setting of decentralised independent learners.

2. We demonstrated the action advising framework's efficiency for the first time in a Deep RL/MARL problem.

3. We use RND technique as a state confidence measurement to drive jointly-initiated advice exchanging decisions.

In Section 4.2 we describe the problem formulation and our approach in detail. Then, in Section 4.3, the experimental setup is explained. The results and the related

discussion are then presented in Section 4.4. Finally, Section 4.5 concludes the study with our final remarks.

4.2 The Approach

Our aim is to accelerate the learning of a cooperative team of agents in a multi-agent environment via action advising. The problem setting we take into consideration is fully observable, completely decentralised in every stage, and most importantly, assumes that the environment is too complex for tabular methods and requires non-linear models to be approximated/generalised across successfully, unlike the majority of previous work. In this setting, every agent operates with a local observation; however, they are also assumed to be able to observe each other and infer/communicate any other agent's local observations. Additionally, they can exchange action advice requests and responses. We take the budget constraints in inter-agent interactions into consideration as introduced in Torrey & Taylor (2013) and we want to achieve acceleration in learning while keeping the agent interactions at minimum since we believe that such behaviour will be the most useful in practical applications. Even though there are no such requirements in our approach, we set every agent to be identical in terms of their learning specifications for the sake of simplicity. The general structure of our proposal can be expressed in two parts: the agents' Deep RL policies and the teacher-student framework, which we explain in the following sections. We follow the Dec-POMDP formalisation (Chapter 2.1.2) to formulate the problem and our algorithm.

4.2.1 Agent Specifications

At task-level learning and decision-making, agents are designed to employ Deep RL algorithms. Specifically, the agents are DQN learners along with its well-established improvements of Double DQN, Prioritised Experience Replay, Dueling Networks and NoisyNets. Maybe the most important of these in this particular study here is Prioritised

Experience Replay as it helps with sampling the transitions produced by teacher advice since they are believed to yield larger temporal-difference errors more frequently by intuition.

In terms of the multi-agent learning strategy, we follow the independent learners approach in which each agent treats other agents as part of a non-stationary environment and behaves by taking only its own actions into account. This is the simplest approach in MARL without requiring the algorithm to have any specialisations to consider the presence of multiple agents. Therefore, we keep DQN as it is without any further modifications in this regard, which is also referred to as independent DQNs. Despite having no theoretical convergence guarantees in multi-agent environments, independent DQNs have been able to exhibit promising empirical results in previous studies and are often used as a baseline for further improvements (Foerster et al. 2016, Tampuu et al. 2015).

4.2.2 Teaching on a Budget

We adopt action advising in the form of the teacher-student framework to perform inter-agent knowledge transfer. This method requires only a common action set and minimal similarity between agents. In addition, considering communication costs, exchanging actions instead of episodes or policy parameters is preferable especially when task-level policy employs a complex model like DQN. In our approach, agents can broadcast requests for advice when they need it, and can also respond to requests for advice from other agents when they think they are qualified enough. They have separate budgets of asking b_{ask} and giving b_{give} advice, which determines the total number of times they can perform these interactions.

Adapting the action advising framework to the Deep MARL domain have some problematic aspects to deal with. In our problem setting, every agent learns simultaneously, and as result, they no longer have fixed policies. Consequently, they end up having different levels of knowledge about the task and have no information on each other's

expertise since they can not have any access to internals (decentralised). Therefore, they hold no assumptions about being a teacher or a student. In order to overcome this, we follow a jointly-initiated advising strategy (Amir et al. 2016). Every time an agent requests advice from another agent, the agent in the position of teacher-to-be has to confirm this interaction if it thinks that it will be appropriate to take the role. Furthermore, as the agents have no fixed roles or knowledge levels that are defined previously, they need to be capable of determining whether they need advice based on the state they are in as well as determining whether they are experienced enough to give advice. In da Silva et al. (2017) where they deal apply action advising in MARL, they utilise the number of visits to measure an agent's certainty in a given state. However, this is not applicable when state space is large and nonlinear function approximation is used to represent states. We propose to use the RND technique as a metric to measure agents' uncertainties as an alternative to tabular state counting. Every time an agent uses a sample tuple consisting of a state observation o[1] to train its Deep RL model, this sample is also used to update (or *distill*) its predictor network $\hat{G}$. Consequently, it will be able to measure how uncertain it is in a state with observation o by measuring the error $\|\hat{G}_\omega(o) - G(o)\|^2$ when determining if it needs advice or is capable of giving it. This method, which was originally proposed to serve as an exploration bonus through state novelty, can be treated as a proxy of visit counts in the non-linear function approximation domain.

At each timestep, each agent chooses its action in their state s with observation o according to teacher-student augmented action selection procedure defined in Algorithm 11. If it has any asking budget b_{ask} available, it measures its proxy uncertainty $U(o)$ by using its own RND models G and $\hat{G}$; if $U(o)$ is higher than a predefined asking threshold τ_{ask} the agent broadcasts its advice requests to other agents as described in Algorithm 12. As this hyperparameter goes smaller the students become more likely to ask for advice. Agents who have any remaining advice-giving budget b_{give} then attempt

[1]We refer to the state s as state observation o in this chapter to emphasise the presence of partial observability, as it is specified in Dec-POMDP formalisation.

to respond to this request. If the agent receives any advice, it determines which action to follow by using majority voting (ties broken at random) to then execute such action. Otherwise, if no advice is received, it continues exploring by following its own policy.

Responding to an advice request happens as described in Algorithm 13. Upon getting a request, the agent first checks if it has any remaining giving budget b_{give}. If so, it then determines its expertise by using its internal G and $\hat{G}$, computing $\|\hat{G}_\omega(o) - G(o)\|^2$. It then compares this value with threshold τ_{give} to decide whether to proceed or not. As this hyperparameter goes smaller the teachers limit their advice to the states they are more certain about, hence resulting in a smaller number of advice exchange interactions. Finally, if it is set to follow the Early Advising heuristic, it broadcasts action advice according to its policy; otherwise, the specified underlying advice collection heuristic is followed, e.g., Importance Advising (importance threshold is denoted by τ_{imp} in this case.)

The general learning flow of an agent with DQN policy and action advising mechanism is summarised in Algorithm 10[2]. This is a slightly modified version of the baseline DQN (Algorithm 8). The differences are highlighted in the lines enclosed with dots (lines 4, 7 and 14). As it can be seen, the underlying RL algorithm can easily be isolated from our enhancements except for its action selection policy[3]. Therefore, it can be changed for any other policy as long as it satisfies the assumption of computing state-action values for Importance Advising to be applicable.

4.3 Experimental Setup

The objective of the empirical evaluation is to understand if and how our novelty-based action advising framework can enhance the agents' learning performance compared to the previous baselines. For this purpose, we conducted experiments through multiple *learning sessions*, each one consisting of a set fixed number of different game episodes

[2]The code for our experiments can be found at `https://github.com/ercumentilhan/teaching-on-a-budget`

[3]This part corresponds to the line 1102 of `dqn-ps.py` file in the code repository.

Algorithm 10 Deep Q-Learning with Teacher-Student Framework (Single Agent Perspective)

Input: Number of learning steps t_{max}, learning rate α, target network update period T_{target}, training period T_{train}, replay memory capacity $N_{\mathcal{D}}$ replay memory size to start learning $M_{\mathcal{D}}$.

Initialise empty replay memory $\mathcal{D}$ with capacity $N_{\mathcal{D}}$ to store state transition tuples
Initialise online deep Q-network with weights θ randomly
Initialise target deep Q-network with weights $\overline{\theta}$ copied from θ
.....
Initialise the RND networks G and $\hat{G}$
.....
for training steps $t \in \{1, 2, \ldots t_{max}\}$ **do**
  Get observation o_t from *Environment* if *Environment* is reset
  .....
  $a_t \leftarrow \textsc{SelectAction}(o_t)$ ▷ Determine the action to take
  .....
  Execute a_t and obtain r_{t+1}, o_{t+1} from *Environment*
  Store transition $\langle o = o_t, a = a_t, r = r_{t+1}, o' = o_{t+1} \rangle$ in $\mathcal{D}$
  Remove $\langle o_{t-N_{\mathcal{D}}}, a_{t-N_{\mathcal{D}}}, r_{t+1-N_{\mathcal{D}}}, o_{t+1-N_{\mathcal{D}}} \rangle$ from $\mathcal{D}$ if $|\mathcal{D}| > N_{\mathcal{D}}$
  if $\mathcal{D} \geq M_{\mathcal{D}}$ **and** $t \mod T_{train} = 0$ **then**
    Draw a minibatch of transitions $\mathcal{B}$ from $\mathcal{D}$
    Perform a gradient descent step with $\mathcal{B}$ to minimise Equation 2.11 (at α rate)
    .....
    Update ω weights to minimise $\|\hat{G}_{\omega}(o) - G(o)\|^2$ for every o in $\mathcal{B}$
    .....
  end if
  $\overline{\theta} \leftarrow \theta$ if it is target network updating period ($t \mod T_{target} = 0$)
  $o_t \leftarrow o_{t+1}$
end for

Algorithm 11 Teacher-Student Augmented Action Selection (SELECTACTION)

Input: State observation o.
Locals: Agent's policy π, RND model components $\hat{G}_\omega$ and G, advice-asking budget b_{ask}, advice-asking threshold τ_{ask}.

$a \leftarrow None$ ▷ Set action as non-determined
if $b_{ask} > 0$ **then** ▷ Check if there is enough budget to ask advice
  $U(o) \leftarrow \|\hat{G}_\omega(o) - G(o)\|^2$ ▷ Measure state uncertainty via RND
  if $U(o) > \tau_{ask}$ **then** ▷ Check if state uncertainty high enough to seek advice for
    $a \leftarrow$ ASKFORADVICE(o) ▷ Request advice from other agents
  end if
end if
if a is $None$ **then** ▷ No valid advice is received
  Determine a via the RL algorithm π, i.e. DQN
else
  $b_{ask} \leftarrow b_{ask} - 1$ ▷ Decrease the available budget by 1
end if
return a

which are initialised with random, yet non-overlapping, agent and landmark positions.

The performance of the agents is assessed as a team through a learning session that consists of multiple episodes. After the agents come to the end of an episode (due to time limit, details are provided in Section 3.1), a new episode begins with a random level structure (agent and landmark positions); this is repeated until the maximum timesteps defined for the learning session is reached. The agents are evaluated every 100 episodes in a predefined set of 50 evaluation levels. During the evaluation, learning process and teaching procedures are disabled, and the levels used are fixed across all learning sessions. This way, evaluation periods have no effect in the actual learning progression and the agents can be evaluated in a way that results from different evaluation steps are comparable (by having fixed level structures). Evaluation score is calculated by normalising the average episode rewards obtained across 50 levels with the maximum possible total reward (determined by a set of hand-crafted expert agents), giving a score in $[0, 1]$, where 1 indicates the optimal performance.

The performance of the proposed methods can be assessed by looking at the

Algorithm 12 Request Advice as Student (AskForAdvice)

Input: State observation o.
$a \leftarrow None$ ▷ Set final action as $None$
$\mathcal{A}_r \leftarrow \emptyset$ ▷ Initialise empty set of received advice
for every other agent i **do** ▷ Advice is to be requested from every other agent
  $a_{advised} \leftarrow i$.Advise ▷ Attempt to receive advice from agent i
  if $a_{advised} \neq None$ **then** ▷ Check if agent i has generated a valid advice
    Add $a_{advised}$ to $\mathcal{A}_r$ ▷ Keep the advice
  end if
end for
if $\mathcal{A}_r \neq \emptyset$ **then** ▷ Check if there is at least 1 valid advice
  $a \leftarrow$ perform majority voting in $\mathcal{A}_r$ ▷ Decide which advice to follow by majority voting
end if
return a

evaluation scores across a learning session, according to the following two metrics:

- **Asymptotic performance:** This is measured directly by looking at the evaluation scores values and represents how good the agents are at solving the game.
- **Learning speed:** This is measured by looking at the area under the curve of evaluation scores against the number of training episodes graph.

For agents to be able to benefit from knowledge transfer, there must be some form of knowledge heterogeneity within the team. In MARL, such heterogeneity tends to arise when the agents explore different parts of the state space, use different task-level policies in terms of complexity and representation, or are in different stages of training. Since our environment is fully observable and the agents are identical, only the latter is applicable in our setting. One objective of this study is to determine how the proposed methods work in different types of knowledge heterogeneity. Therefore, we designed the following 2 scenarios:

- **Scenario I:** We train a team of agents in levels from a single distribution of levels; then, we take agents from different stages of pre-training to form a team to be

Algorithm 13 Respond to Advice Request as Teacher (ADVISE)

Locals: Agent's policy π, advice-giving budget b_{give}, advice-giving threshold τ_{give}, importance threshold τ_{imp}, RND model components $\hat{G}_\omega$ and G.

$a \leftarrow None$ ▷ Set the advice to be given as $None$ initially
if $b_{give} > 0$ **then** ▷ Check if there is enough budget provide advice
  $o \leftarrow$ observe/communicate advice requesting student's state.
  $U(o) \leftarrow \|\hat{G}_\omega(o) - G(o)\|^2$ ▷ Measure state uncertainty via RND
  if $U(o) < \tau_{give}$ **then** ▷ Check if state certainty is enough to advise in o
    switch teaching method
      case Early Advising:
        $a \leftarrow$ produce greedy action via $\pi(o)$ ▷ Generate action advice
        $b_{give} \leftarrow b_{give} - 1$ ▷ Decrease the available budget by 1
      case Importance Advising:
        $I(o) \leftarrow \max_a Q(o,a) - \min_a Q(o,a)$ ▷ Measure state importance of o
        if $I(o) > \tau_{imp}$ **then** ▷ Check if o is important enough to advise
          $a \leftarrow$ produce greedy action via $\pi(o)$ ▷ Generate action advice
          $b_{give} \leftarrow b_{give} - 1$ ▷ Decrease the available budget by 1
        end if
    end switch
  end if
end if
return a

evaluated.

- **Scenario II:** We train 3 sets of agents in 3 different level distributions, in which landmarks and initial agents locations are strictly limited to predefined regions (see Figures 4.1 and 4.2). Then, we take one agent with moderate performance arbitrarily from each level type to form a team to evaluate. Note that the learning sessions used to pre-train agents are generated with different seeds than the ones where we run the final evaluation.

The following is a list of the different types of agents used in this study. These agents are trained in the levels first to acquire some knowledge and then picked to form teams of 3 agents as explained above. These agents are:

- **A0:** Agent with no prior knowledge.

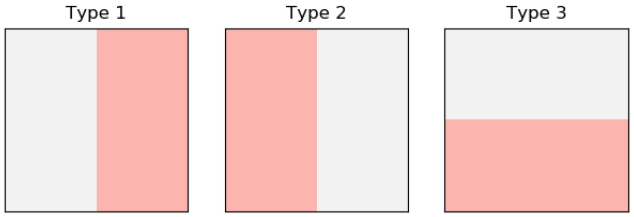


Figure 4.1: Level structure types in terms of possible regions for initial positions of agents (grey) and landmarks (red).

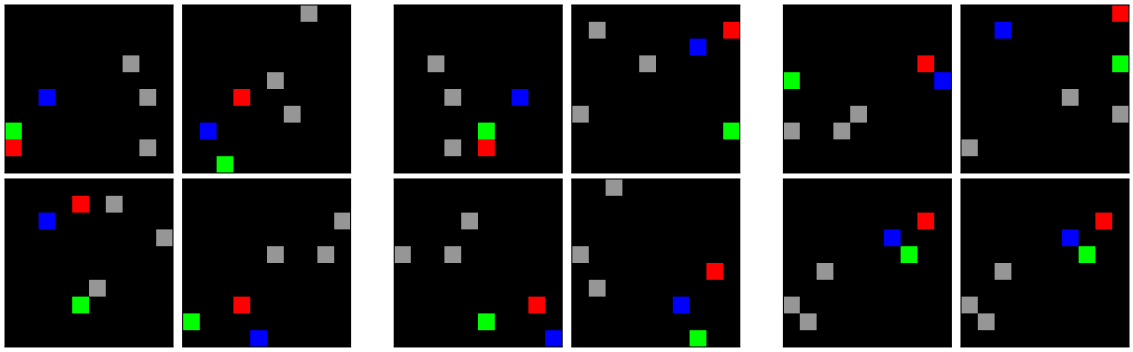


Figure 4.2: Some examples of initial states generated for different level structure types. The two leftmost columns are generated with Type 1; the two columns in the middle are generated with Type 2; the two rightmost columns are generated with Type 3.

- **A10:** Agent taken from a team of agents trained for 10k episodes. This agent obtained an evaluation score of 0.45 in a level distribution identical to the evaluation sessions.
- **A20:** Agent taken from a team of agents trained for 20k episodes. The evaluation score is 0.91 in a level distribution identical to the learning evaluation sessions.
- **A10-1:** Agent taken from a team of agents trained for 10k episodes, with 0.51 evaluation score in a restricted level distribution of type 1.
- **A10-2:** Agent taken from a team of agents trained for 10k episodes, with 0.41 evaluation score in a restricted level distribution of type 2.
- **A10-3:** Agent taken from a team of agents trained for 10k episodes, with 0.43 evaluation score in a restricted level distribution of type 3.

Due to the way the state observation vector is constructed (described in Section 3.1), it may be possible that agents learn to consider only a particular landmark in the

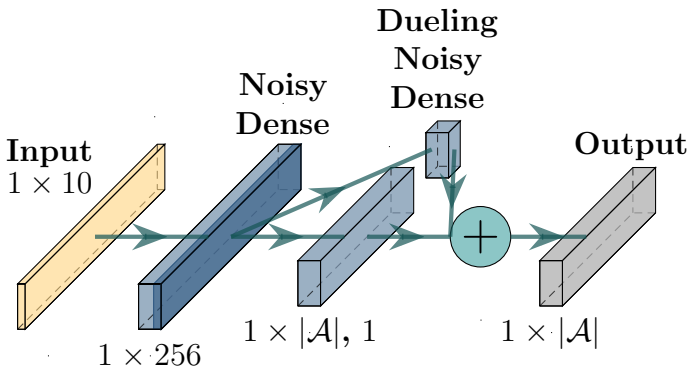


Figure 4.3: Neural network architecture of DQN where $|\mathcal{A}|$ is the number of actions. The dark-shaded slice denotes the presence of rectified linear unit activation function. Output is the Q-values. The numbers of filters and units in the layers are indicated below each layer.

observation. However, the training process involves many randomly generated levels which create enough variety in the level structure to incentivise the agents to learn to treat the landmarks equally at the end.

The methods we evaluate along with the agent teams used in them are as follows:

- **XP:** The team to be evaluated is formed by agents A0, A10, A20 and no advising is used. This method serves as a baseline for the first scenario.
- **XP-EA:** The team to be evaluated is formed by agents A0, A10, A20, and Early Advising is enabled.
- **XP-IA:** The team to be evaluated is formed by agents with different knowledge levels A0, A10, A20, and Importance Advising is enabled.
- **XP-L:** The team to be evaluated is formed by agents A10-1, A10-2, A10-3 and no advising is used. This method serves as a baseline for the second scenario.
- **XP-L-EA:** The team to be evaluated is formed by agents A10-1, A10-2, A10-3, using Early Advising.
- **XP-L-IA:** The team to be evaluated is formed by agents with different knowledge levels A10-1, A10-2, A10-3, using Importance Advising.

We run the tests for the methods with teaching enabled, namely XP-EA and XP-IA,

with different advice budgets of 1k, 2k, 5k, 10k, 20k and unlimited (∞). For Scenario I (XP, XP-EA, XP-IA), learning is run for 10k episodes, while 20k episodes are run for Scenario II (XP-L, XP-L-EA, XP-L-IA). The budgets are separate for each individual agent and are set the same for asking and giving advice initially. Moreover, due to the well-known learning inconsistency of Deep RL methods, especially in multi-agent settings, every experiment is repeated 10 times with different random seeds.

The agents are identical independent learners that employ DQN with the improvements of Double DQN, Dueling Networks, Noisy Networks and Prioritised Experience Replay. The network structure (Figure 4.3) is a multilayer perceptron which is the archetypal form of the deep learning models (Goodfellow et al. 2016) with a single hidden layer formed by 256 units. The addition of NoisyNets removes the need to follow an explicit exploration policy such as ϵ-greedy with the aid of perturbed weights providing a learnable and state-specific exploration (see Chapter 2.3.1). NoisyNets noise parameters are generated with factorised Gaussian noise. The hidden layer uses rectified linear units as the activation functions.

In the RND models, we employ a very similar network architecture with a single non-noisy hidden layer formed by 256 units and an output size of 1; the only difference with the architecture shown in Figure 4.3 is the absence of Dueling Networks and the output size being 1. The weights of these models are initialised via a normal distribution with means 0.0, 0.0 and standard deviations of 1.0, 3.0 for the predictor and the target networks, respectively. This is done to promote the difference in the outputs produced by the RND networks by making them dissimilar at the beginning of training.

All the relevant hyperparameters are determined empirically and are given in Table 4.1. They are kept the same across different scenarios and configurations.

4.4 Results and Discussion

Tables 4.2 and 4.3 show the results for the scenarios described in this study. These results show the asymptotic performance (score) and the area under the curve (AUC)

Table 4.1: Hyperparameters used in the experiments for the students' DQN (top section), RND (middle section) and the action advising module (bottom section). $\mathcal{U}$ denotes uniform distribution.

Parameter name	Value
Replay memory size to start learning $M_{\mathcal{D}}$	10k
Replay memory capacity $N_{\mathcal{D}}$	25k
Prioritisation type	Proportional
Prioritisation exponent	0.6
Prioritisation importance sampling	$0.5 \rightarrow 1$
Target network update period (steps) T_{target}	10k
Train period (steps) T_{train}	2
Minibatch size	64
Learning rate α	0.001
Discount factor γ	0.99
Adam epsilon	1.5×10^{-4}
Huber loss delta	1
Noisy layer Gaussian noise type	Factorised
Noisy layer initial μ distribution for p inputs	$\mathcal{U}[-1/\sqrt{p}, +1/\sqrt{p}]$
Noisy layer initial σ for p inputs	$0.4/\sqrt{p}$
RND weight initialisation	Random normal
RND predictor's weight init. mean and std. dev.	0.0 and 1.0
RND target's weight init. mean and std. dev.	0.0 and 3.0
RND learning rate	0.001
Adam epsilon	1.5×10^{-4}
Advice-asking budget b_{ask}	1k, 2k, 5k, 10k, 20k, ∞
Advice-giving budget b_{give}	1k, 2k, 5k, 10k, 20k, ∞
Advice-asking threshold τ_{ask}	10
Advice-giving threshold τ_{give}	3
Importance threshold τ_{imp}	1

values as indicators of the learning performance of the agents. Results are reported for different moments of the evaluated training: after 2.5k, 5k, 7.5k, 10k, 15k and 20k episodes, and they include the standard error of the measure (10 repetitions). Results that are significantly different than the first row (baseline) of each table, according to Welch's t-test (p-value < 0.05) are denoted in bold. Additionally, evaluation scores of the methods with the highest final AUC values from each of the scenarios are plotted against their respective baseline as shown in Figures 4.4 and 4.5.

Table 4.2: Asymptotic performance (score) and area under the curve (AUC) values of XP, XP-EA and XP-IA at learning episodes 2.5k, 5k, 7.5k and 10k with standard errors in parentheses. Results of XP-EA and XP-IA that are significantly different than XP algorithm according to Welch's t-test with p-value < 0.05 are denoted in bold. Each algorithm is run 10 times.

Algorithm	At 2500th		At 5000th	
	AUC	Score	AUC	Score
XP	13.94 (0.04)	0.58 (0.005)	28.72 (0.12)	0.62 (0.009)
XP-EA (1K)	13.9 (0.05)	0.58 (0.002)	28.81 (0.14)	0.63 (0.014)
XP-EA (2k)	13.85 (0.07)	0.57 (0.005)	28.58 (0.13)	0.61 (0.009)
XP-EA (5k)	13.88 (0.03)	0.58 (0.004)	28.75 (0.11)	0.63 (0.018)
XP-EA (10k)	13.85 (0.04)	0.57 (0.004)	28.57 (0.09)	0.61 (0.006)
XP-EA (20k)	13.76 (0.05)	0.57 (0.004)	28.56 (0.09)	0.64 (0.016)
XP-EA (∞)	13.76 (0.05)	0.57 (0.003)	28.53 (0.06)	0.64 (0.011)
XP-IA (1k)	13.94 (0.04)	0.58 (0.003)	29.0 (0.15)	0.66 (0.016)
XP-IA (2k)	13.91 (0.08)	0.58 (0.005)	28.86 (0.21)	0.64 (0.02)
XP-IA (5k)	13.92 (0.03)	0.57 (0.004)	28.84 (0.12)	**0.65 (0.008)**
XP-IA (10k)	13.97 (0.04)	0.58 (0.003)	29.22 (0.22)	**0.68 (0.024)**
XP-IA (20k)	13.96 (0.04)	0.58 (0.003)	29.19 (0.2)	**0.67 (0.022)**
XP-IA (∞)	13.96 (0.04)	0.58 (0.003)	29.19 (0.2)	**0.67 (0.022)**

Algorithm	At 7500th		At 10000th	
	AUC	Score	AUC	Score
XP	47.15 (0.44)	0.83 (0.013)	68.26 (0.6)	0.87 (0.006)
XP-EA (1k)	47.18 (0.54)	0.81 (0.013)	68.27 (0.73)	0.86 (0.01)
XP-EA (2k)	46.39 (0.59)	0.79 (0.024)	67.16 (0.87)	0.86 (0.009)
XP-EA (5k)	47.02 (0.56)	0.81 (0.024)	67.86 (0.86)	0.87 (0.007)
XP-EA (10k)	46.3 (0.47)	0.8 (0.021)	67.28 (0.68)	0.87 (0.008)
XP-EA (20k)	47.38 (0.37)	0.84 (0.005)	68.87 (0.45)	0.88 (0.003)
XP-EA (∞)	47.11 (0.24)	0.82 (0.006)	68.0 (0.31)	0.86 (0.003)
XP-IA (1k)	47.62 (0.59)	0.82 (0.012)	68.92 (0.71)	0.87 (0.006)
XP-IA (2k)	47.11 (0.61)	0.81 (0.015)	68.16 (0.84)	0.87 (0.009)
XP-IA (5k)	48.01 (0.38)	0.83 (0.006)	69.49 (0.41)	0.88 (0.004)
XP-IA (10k)	48.26 (0.72)	0.8 (0.019)	69.6 (0.98)	0.88 (0.006)
XP-IA (20k)	47.92 (0.58)	0.8 (0.012)	69.01 (0.75)	0.87 (0.004)
XP-IA (∞)	47.92 (0.58)	0.8 (0.012)	69.01 (0.75)	0.87 (0.004)

Table 4.3: Asymptotic performance (score) and area under the curve (AUC) values of XP-L, XP-L-EA and XP-L-IA at learning episodes 5k, 10k, 15k and 20k with standard errors in parentheses. Results of XP-L-EA and XP-L-IA that are significantly different than XP-L algorithm according to Welch's t-test with p-value < 0.05 are denoted in bold. Each algorithm is run 10 times.

Algorithm	At 5000th		At 10000th	
	AUC	Score	AUC	Score
XP-L	15.38 (0.12)	0.36 (0.005)	38.34 (0.43)	0.52 (0.006)
XP-L-EA (1k)	15.58 (0.1)	**0.39 (0.008)**	**39.51 (0.33)**	**0.54 (0.005)**
XP-L-EA (2k)	15.4 (0.12)	0.37 (0.009)	38.28 (0.66)	0.53 (0.012)
XP-L-EA (5k)	15.51 (0.14)	**0.39 (0.007)**	**39.98 (0.37)**	**0.54 (0.006)**
XP-L-EA (10k)	15.53 (0.1)	**0.38 (0.008)**	**39.53 (0.35)**	**0.55 (0.005)**
XP-L-EA (20k)	15.53 (0.1)	**0.38 (0.008)**	**39.53 (0.35)**	**0.55 (0.005)**
XP-L-EA (∞)	15.53 (0.1)	**0.38 (0.008)**	**39.53 (0.35)**	**0.55 (0.005)**
XP-L-IA (1k)	15.39 (0.13)	**0.38 (0.006)**	38.54 (0.5)	0.52 (0.008)
XP-L-IA (2k)	15.27 (0.14)	0.37 (0.008)	38.4 (0.43)	0.53 (0.007)
XP-L-IA (5k)	15.37 (0.09)	0.38 (0.009)	38.67 (0.32)	0.52 (0.005)
XP-L-IA (10k)	15.37 (0.1)	0.37 (0.009)	38.58 (0.34)	0.52 (0.006)
XP-L-IA (20k)	15.37 (0.09)	0.38 (0.009)	38.67 (0.32)	0.52 (0.005)
XP-L-IA (∞)	15.37 (0.09)	0.38 (0.009)	38.67 (0.32)	0.52 (0.005)

Algorithm	At 15000th		At 20000th	
	AUC	Score	AUC	Score
XP-L	66.97 (0.71)	0.66 (0.02)	105.46 (1.37)	0.86 (0.011)
XP-L-EA (1k)	**69.85 (0.85)**	0.73 (0.024)	**110.8 (1.55)**	0.88 (0.018)
XP-L-EA (2k)	68.55 (1.49)	0.7 (0.022)	108.4 (2.28)	0.86 (0.02)
XP-L-EA (5k)	**71.18 (0.68)**	**0.75 (0.022)**	**112.69 (1.27)**	**0.89 (0.005)**
XP-L-EA (10k)	**71.3 (1.02)**	**0.74 (0.026)**	**112.14 (1.77)**	0.87 (0.014)
XP-L-EA (20k)	**71.3 (1.02)**	**0.74 (0.026)**	**112.14 (1.77)**	0.87 (0.014)
XP-L-EA (∞)	**71.3 (1.02)**	**0.74 (0.026)**	**112.14 (1.77)**	0.87 (0.014)
XP-L-IA (1k)	67.95 (0.97)	0.68 (0.025)	107.6 (1.59)	0.87 (0.01)
XP-L-IA (2k)	67.73 (0.69)	0.67 (0.017)	106.47 (1.42)	0.85 (0.016)
XP-L-IA (5k)	68.13 (0.59)	0.69 (0.025)	107.32 (1.46)	0.86 (0.019)
XP-L-IA (10k)	68.06 (0.65)	0.68 (0.027)	107.04 (1.6)	0.85 (0.02)
XP-L-IA (20k)	68.13 (0.59)	0.69 (0.025)	107.32 (1.46)	0.86 (0.019)
XP-L-IA (∞)	68.13 (0.59)	0.69 (0.025)	107.32 (1.46)	0.86 (0.019)

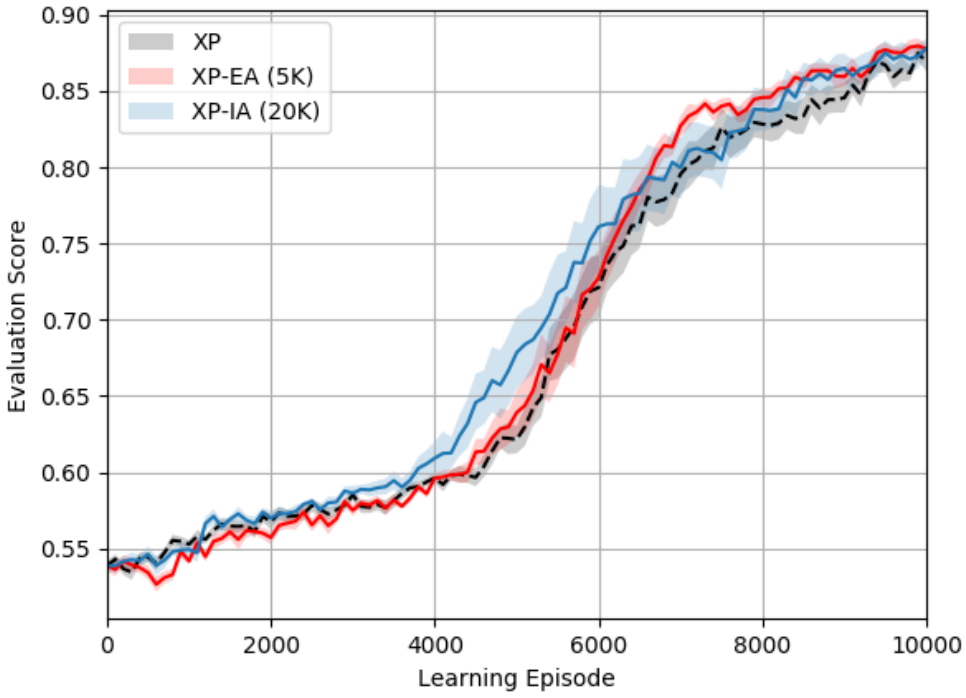


Figure 4.4: Evaluation scores versus the number of learning episodes of XP, XP-EA (5k) and XP-IA (20k). Shaded areas indicate 95% confidence intervals.

In the first scenario, both of the algorithms provided slight accelerations in learning and achieved very similar final performances with the baseline method at 10k*th* episode. The only significant difference from the baseline was seen in scores at 5k*th* episode, by XP-IA (5k). The overall best-performing agent in this scenario is XP-IA (10k). This can be seen as an indication that the importance metric was indeed a useful heuristic to distribute advice over more important states in this scenario. The budget seems to have more effect on XP-IA, achieving its best result at 10k, confirming the claim that too much advice may have negative effects on performance (Torrey & Taylor 2013).

In the second scenario, XP-L-IA failed to show any significant advantage over the baseline XP-L except for the score at 5k*th* episode with a budget of 1k. This can be a result of the importance metric not being accurate at reflecting the actual relevance of states in this kind of agent knowledge setting. On the other hand, XP-L-EA performed

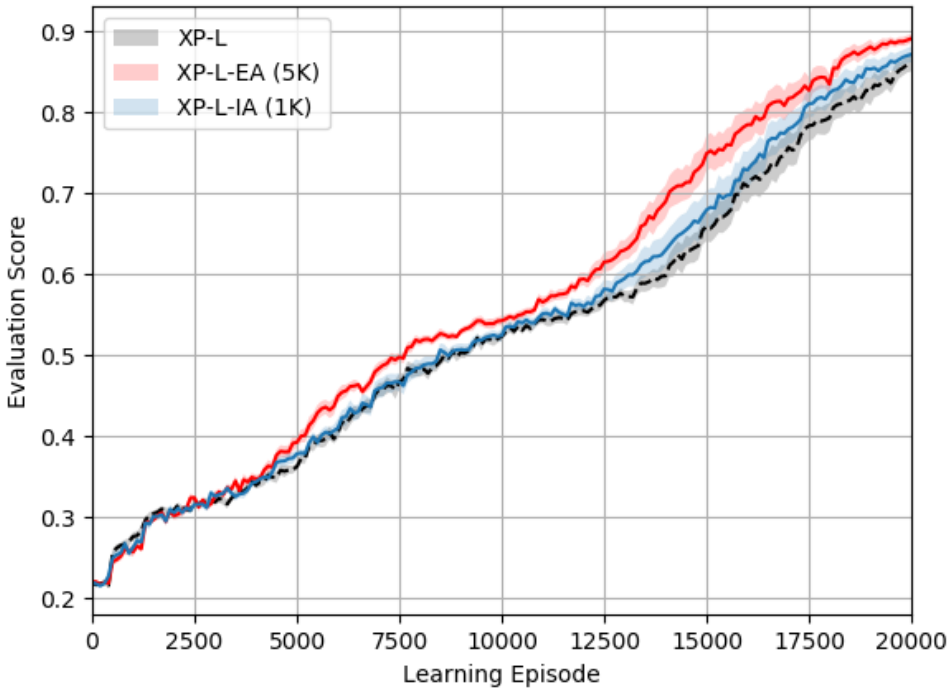


Figure 4.5: Evaluation scores versus the number of learning episodes of XP-L, XP-L-EA (5k) and XP-L-IA (1k). Shaded areas indicate 95% confidence intervals.

very well with significant improvements in terms of asymptotic performance and learning speed at multiple stages of learning. Moreover, it even managed to achieve a significantly better final performance. This may be caused by the agents starting with a similar (and moderate) amount of knowledge, so the early advice are likely to be useful for any of them without having a need for additional importance assessment. The identical results of XP-L with 10k or a higher budget are caused by not having the need to make use of it beyond some point, once the agent is certain of the decision it is making on its own (as controlled with the advice budgets). This can be considered as another benefit of using this uncertainty measurement technique if it is tuned well.

4.5 Conclusions

This study described the application, for the first time, of action advising via a heuristic-based teacher-student framework with Deep RL agents in a multi-agent setting. RND, which is originally a bonus-based exploration method, was used as a state novelty measurement as a replacement for tabular state counters to drive the advice exchange interactions.

Our proposals provided improvements both in the speed and final performance of the agents, particularly when the team is composed of agents with heterogeneous knowledge. Clearly, action advising can present invaluable benefits when the system consists of one or more knowledgeable peers to leverage. Moreover, RND was found to be a reliable metric to be utilised as a proxy of state visit counter for the domains with high state-space complexity. Another interesting finding is that the state importance metric may be inefficient in some cases of knowledge distribution amongst agents, for example, if they are all experts but on different state distributions. Additionally, it is worthwhile highlighting that, even if it is possible to determine the experience of the agents for their roles in knowledge exchange relationships, this experience is importantly biased by the other agents that were present at the time they built their knowledge. This is because the agents are considered to be independent learners which render other agents as a part of the environment, which forms a non-stationary component.

Further investigation on how to adapt the state importance metric for agent advising can be an interesting line of future work. Off-policy learning through replay memory may be a slowing down factor in action advising, as it takes given advice into consideration in a delayed way and reduces the rate at they influence the agent's current policy, especially for independent learners in a multi-agent system. Therefore, in addition to the enhancements like PER, it may be useful to implement more specific techniques like multi-step advice and continual monitoring of agents for fixed periods of time after advice exchange. This is similar to previous work in the field (Amir et al. 2016, Kim et al. 2020). Finally, another interesting line of future work would be to expand the

problem to have more than 3 agents, which brings interesting aspects to the discussion such as defining a more accurate peer selection and advice fusing beyond majority voting.

Chapter 5

Student-Initiated Action Advising via Advice Novelty

This chapter is based on our second publication titled "Student-Initiated Action Advising via Advice Novelty" (Ilhan et al. 2022). The targeted research questions in this piece of study are the following:

- [RQ1] To what extent can we accelerate Deep RL via (budget-limited) action advising with one or more knowledgeable peer(s)?
- [RQ3] What can be an efficient heuristic to perform student-initiated action advising that is also robust to teacher absence conditions in Deep RL?
- [RQ4] Can imprecise usage of action advising budget hamper Deep RL performance?

5.1 Introduction

In the previous chapter, we showed how action advising can be a promising option to be incorporated in a Deep MARL task to accelerate agents' learning. Yet, the underlying multi-agent learning dynamics make it more difficult to study the dynamics of action advising itself. Therefore, in further studies including this one, we change our scope to

be restricted to the single-agent Deep RL domains to better understand and develop more principled action advising strategies that can also be used in more complicated problems later.

The initial problem setup of the action advising framework (Torrey & Taylor 2013) used in the single-agent scenarios assumed the learning agent, namely, the student, to be monitored constantly by a teacher who is responsible to manage the distribution of these advice. However, this is considered to be impractical because of the possible limitations in communication and the teacher's attention span. Moreover, since the teacher has no access to the student's internal model, the advice decisions are determined solely by the teacher's initiative. Due to these, this framework has also been extended into different versions such as student-initiated and jointly-initiated, letting the student take an active role in initiating these interactions, as covered in Chapter 2. While the student-initiated version seems to be the potentially most advantageous variant in Deep RL, it has its own challenges to be dealt with to be successful as the limited number of studies indicate.

In Deep RL, it is a challenging task to credit a transition for its long-term contribution to learning and to determine the actual importance of a state to obtain a piece of advice for. Therefore, the student-initiated approaches in Deep RL has followed heuristics as proxies, such as state novelty (Ilhan et al. 2019) (Chapter 4) and state uncertainty (Chen et al. 2018, da Silva, Hernandez-Leal, Kartal & Taylor 2020) estimations. Even though these techniques demonstrate promising performance by simplifying the challenging characteristics of Deep RL, they assume the teacher(s) to be in the loop from the beginning and consider the convergence of the estimations (state novelty, state uncertainty) to be an indicator of student's task performance improvement. Moreover, if the student's learning algorithm incorporates mechanisms like experience replay as in off-policy learning, e.g., DQN, these estimations are likely to be subject to delays of varying severity depending on the model specifications.

In this study, we first highlight and demonstrate some drawbacks of the existing student-initiated action advising methods. Then, we propose an advice novelty method

based on RND, which, unlike our previous work, is exclusively updated for the states that are advised (hence the name *advice novelty* rather than state novelty). By doing so, it is ensured that the student always make use of the teacher's knowledge, regardless of how late the teacher becomes accessible or how converged the student's task-level model is then. Furthermore, RND updates are performed with single samples instead of batches to prevent the RND model from converging to a global optima, which gives the states a chance to be asked for advice again periodically. This is similar to the idea of keeping expert demonstrations in the replay memory over the course of learning to be revisited occasionally, as in Hester et al. (2018). Our technical contributions can be summaries as follows:

1. We propose a novel student-initiated advice collection strategy via RND, namely, advice novelty-based action advising.

2. We show how the existing action advising approaches can be ineffective if the teacher happens to be absent especially early on during the training.

3. We show evidence that excessive amounts of action advising can hamper the student's learning in Deep RL.

Following the algorithm details given in Section 5.2, we describe the experimental setup in Section 5.3. Afterwards, the results are discussed in Section 5.4. And finally, the study is concluded in Section 5.5.

5.2 The Approach

In the problem formulation, we follow the standard RL framework and the MDP formalisation given in Chapter 2. Our setup considers a situation where a student agent that employs an off-policy Deep RL algorithm, e.g., DQN, with policy π_S is learning to excel in a given task. There is also a peer who is competent in this task to be treated as a teacher with a fixed task-level policy π_T. The student can access this peer and sample

actions from π_T for a limited number of times determined by the action advising budget b. The objective of the agent is to utilise an action advising procedure to distribute the available advice requesting budget b in the best possible way to maximise its learning performance.

The samples obtained in Deep RL are not only used for discovering the environment as in classical RL, but are also responsible for driving the learning of the agent's approximator model. Additionally, every single step of environment interaction and model update influences the sample collection process right after. These make it very difficult for Deep RL agents to predict what kind of long-term effects a transition, or a single piece of advice in this case, will have on their learning progress. Therefore, student-initiated action advising techniques rely on simple heuristics and proxies of the expected usefulness of samples in terms of learning contribution. These, however, are prone to fail in several ways.

Early Advising is a strong baseline due to the fact that the samples obtained early on have more influence on Deep RL algorithms' learning progress. However, it lacks the ability to distribute the available advising budget in more critical states. This is likely to result in the budget being wasted and make the agent miss important advice opportunities especially when the budget is small. Additionally, having no stopping condition in making advice requests may cause the agent to get over-advised, which can deteriorate the learning performance as we show in Section 5.4 later. Employing another common heuristic, uniformly random advising, can alleviate this drawback. Nonetheless, not being able to follow the teacher's policy consistently causes this method to be unsuccessful in the tasks with sparse rewards that require deep exploration, as also shown in Section 5.4.

The more advanced techniques that rely on state importance surrogates such as state novelty (Ilhan et al. 2019) (Chapter 4) and model uncertainty (Chen et al. 2018, da Silva, Hernandez-Leal, Kartal & Taylor 2020) perform better in general. Though, they still have drawbacks that can be problematic in some cases.

First of all, updates of these estimations are driven by the student's task-level

learning progression, regardless of having teacher interactions. If the teacher is present from the beginning, the student's task-level convergence can be assumed to have sufficient amount of teacher contribution. Otherwise, if the teacher joins the session at a later time, the student would already have its state novelty and uncertainty below a level, and end up ignoring the teacher without benefiting from its knowledge at all. However, the student's convergence does not necessarily mean a competent task-level performance, and ensuring to learn from what the teacher has to offer at any time can be vital. As we show in Section 5.4, it is possible for the student to under-explore and converge into a poor task-level policy yet become certain in the states it acts.

Secondly, based on the student's deep RL model's properties, there may be some delays between the advice collection and its actual value to be taken effect in the model, e.g., off-policy updates with replay memory where the collected samples are held in a buffer and are employed periodically to update model weights. Even though we do not demonstrate this behaviour, we ensure in our approach that there can be no model-induced delays in estimations.

Finally, these approaches require several restrictions on the student's task-level algorithm. State novelty-based approach needs to have access to the batches of samples the task-level algorithm uses, preventing the student from being a blackbox. Uncertainty-based methods require the agent's model to be capable of providing a notion of uncertainty.

In order to address these shortcomings, we propose *action advising via advice novelty*, a method that employs state novelty measurements to time the advice requests. In our technique, the student agent employs an RND module that consists of two randomly initialised neural networks G and $\hat{G}$ with identical structures. At each step with available advice requesting budget, the agent measures novelty n_s of the state s it encounters as the RND loss $U(s) = \|\hat{G}_\omega(s) - G(s)\|^2$. This value is then converted into a linearly decreasing advice requesting probability as follows:

$$p_{ask}(s) = U(s)/\lambda, \tag{5.1}$$

where λ is a predefined threshold that is used as a proportion term to determine the probability. To ensure obtaining a valid probability, the result of this function is also clipped to be in $[0, 1]$. If the advice request takes place, then $\hat{G}_\omega$ is updated via changing its weights ω to minimise the loss term $\|\hat{G}_\omega(s) - G(s)\|^2$. Thus, $U(s)$ becomes smaller as the agent receives advice for s. This can then be seen as a novelty metric for a piece of advice to be obtained in a particular state, considering the assumption of teacher policy π_T being fixed, and ignoring the environment's stochasticity. By performing updates only when advised, it is ensured that the student always attempts to learn from the teacher, no matter how far into convergence its task-level model is. Furthermore, the RND updates occur right after the student is advised with only a single piece of observation rather than a batch of them. This prevents the RND module to minimise its loss globally and causes it to have relatively high novelty for the states it has not encountered in a while, giving these states a chance to be re-advised. Finally, since RND employs neural networks with non-linear function approximation, our method can function in complex domains and is capable of generalising between unseen states. A full description of our method can be seen in Algorithm 14[1]. It can be seen clearly that the proposed modification (lines 5-15 in Algorithm 14)[2] can easily be separated from the underlying RL algorithm and a DQN algorithm (Algorithm 8) can be integrated as the RL algorithm to this loop as we did in this particular study.

5.3 Experimental Setup

We are interested in investigating the shortcomings of the existing student-initiated action advising approaches, and evaluating how our approach compares with them in different scenarios. For this purpose, we choose Reach the Goal game from a GridWorld domain (Chapter 3.2) and 5 MinAtar (Chapter 3.3) games to conduct experiments in two stages involving different challenges. We compare the following modes of student

[1]Code for our experiments can be found at`https://github.com/ercumentilhan/advice-novelty`

[2]The implementation of this part can be found in lines 319-368 of `action_advising/executor.py` file in the code repository.

Algorithm 14 Action Advising via Advice Novelty

Input: Number of learning steps t_{max}, RL algorithm-related parameters, e.g. DQN, teacher policy π_T, advice-requesting budget b_{ask}, advice-requesting probability proportion term λ_{ANA}.
Initialise RL algorithm-related variables, e.g. DQN
for training steps $t \in \{1, 2, \dots t_{max}\}$ **do**
  Get observation s_t from *Environment* if *Environment* is reset
  .....
  $a_t \leftarrow None$
  if $b_{ask} > 0$ **then**
    $U(s_t) \leftarrow \|\hat{G}(s_t) - G(s_t)\|^2$ ▷ Compute novelty
    $p_{ask}(s_t) \leftarrow U(s_t)/\lambda_{ANA}$ ▷ Compute probability (clipped to be in $[0, 1]$)
    if $p_{ask}(s_t) > u \sim \mathcal{U}(0, 1)$ **then**
      $a_t \leftarrow \pi_T(s_t)$ ▷ Obtain advice from the teacher
      Update ω to minimise $\|\hat{G}_\omega(s_t) - G(s_t)\|^2$ ▷ Update RND
      $b_{ask} \leftarrow b_{ask} - 1$
    end if
  end if
  .....
  if a_t is $None$ **then**
    Determine a_t via the RL algorithm, e.g. DQN
  end if
  Execute a_t and obtain r_{t+1}, s_{t+1} from *Environment*
  Update the RL algorithm, e.g. DQN
  $s_t \leftarrow s_{t+1}$
end for

agents with different action advising approaches:

- **No Advising (None)**: The agent does not employ any form of action advising; it follows its own policy at all times.

- **Early Advising (EA)**: The agent follows early advising heuristic to distribute its action advising budget by requesting advice until it runs out of its action advising budget.

- **Random Advising (RA)**: The agent follows random advising heuristic and

determines whether to request advice uniformly at random at each step until it runs out of its action advising budget.

- **Uncertainty-Based Advising (UA)**: Advice requests are made according to the task-level model uncertainty, similarly to Chen et al. (2018), da Silva, Hernandez-Leal, Kartal & Taylor (2020). Specifically, at each step, the student's NoisyNets uncertainty is obtained for the current state and then is divided by a threshold λ_{UA} to determine advice requesting probability (as in Equation 5.1 where $U(s)$ is state uncertainty and λ is λ_{UA}).

- **State Novelty-Based Advising (SNA)**: Advice requests are driven by state novelty measurements, similarly to Ilhan et al. (2019) (Chapter 4). For each state the student encounters, its novelty is measured by a separate RND module. Then, this value is divided by a predefined threshold λ_{SNA} to obtain advice requesting probability (as in Equation 5.1 where $U(s)$ is state novelty and λ is λ_{SNA}). The RND module is updated simultaneously with the student's task-level model, by the same batches of samples.

- **Advice Novelty-Based Advising (ANA)**: The agent that follows our proposed approach.

The student agent's RL algorithm is set to be DQN, including the prominent extensions of Double DQN, Dueling Networks and NoisyNets exploration strategy. The neural network structure is comprised of a single convolutional layer consisting 16 3×3 filters with a stride of 1, followed by a fully connected noisy layer with 128 hidden units. This network architecture is visualised in Figure 5.1.

We use a similar structure for the RND components of predictor and target networks. Input is fed to a single convolutional layer that has 16 3×3 filters with a stride of 1 which is followed by a fully connected dense layer with 128 hidden units. It should be noted that we don't use Dueling Networks or NoisyNets in this component. Finally, an arbitrary sized (6 in this study) output is generated in the end. This network

Table 5.1: Hyperparameters used in the experiments for the student's DQN (top section), RND module of SNA and ANA (bottom section) for Reach the Goal (RtG) and MinAtar domains, on the left and right columns, respectively. $\mathcal{U}$ denotes uniform distribution.

	Value	
Hyperparameter name	RtG	MinAtar
Replay memory size to start learning $M_{\mathcal{D}}$	1k	10k
Replay memory capacity $N_{\mathcal{D}}$	10k	100k
Target network update period (steps) T_{target}	250	1000
Train period (steps) T_{train}	2	2
Minibatch size	32	32
Learning rate α	0.0001	0.0001
Discount factor γ	0.99	0.99
Adam epsilon	1.5×10^{-4}	1.5×10^{-4}
Huber loss delta	1	1
Noisy layer Gaussian noise type	Factorised	Factorised
Noisy layer initial μ distribution for p inputs	$\mathcal{U}[-1/\sqrt{p}, +1/\sqrt{p}]$	$\mathcal{U}[-1/\sqrt{p}, +1/\sqrt{p}]$
Noisy layer initial σ for p inputs	$0.4/\sqrt{p}$	$0.4/\sqrt{p}$
RND weight initialisation	Random normal	Random normal
RND predictor's weight init. mean and std. dev.	0.0 and 1.0	0.0 and 1.0
RND target's weight init. mean and std. dev.	0.0 and 3.0	0.0 and 3.0
RND learning rate	0.001	0.001
Advice-asking budget b_{ask}	5k, 50k	50k, 250k
Advice-requesting probability scaling term of UA λ_{UA}	0.001	Varies
Advice-requesting probability scaling term of SNA λ_{SNA}	0.0001	Varies
Advice-requesting probability scaling term of ANA λ_{ANA}	0.001	Varies

visualisation can be seen in Figure 5.2.

As described in the problem setup, there needs to be a teacher for the student agent to be able to get advice from. In Reach the Goal, we set our teacher policy as following the shortest path from the current position to the goal tile. In MinAtar, we generated competent teachers by training separate DQN agents for each of the games for 3M steps, who achieve final evaluation scores (as defined later on) of 29.28, 81.15, 5.77, 146.64, 146.06 in Asterix, Breakout, Freeway, Seaquest, Space Invaders, respectively. The teachers are made to employ ϵ-greedy exploration instead of NoisyNets, to have them as dissimilar as possible from the student in order to eliminate any possible advantages that may arise in the knowledge exchange process due to them being identical.

Every student variant (e.g., None, EA, etc.) is trained for a fixed number of steps which we define as a learning session. During a learning session, the agent is evaluated

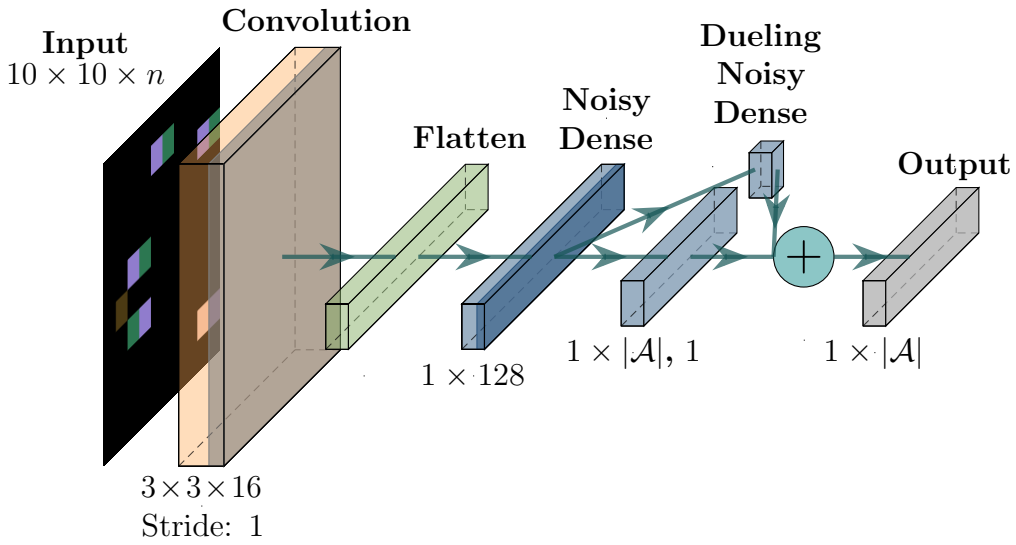


Figure 5.1: Neural network architecture of DQN where $|\mathcal{A}|$ is the number of actions. The dark shaded slices denote the presence of rectified linear unit activation function. Output is the Q-values. The number of filters and units in the layers are indicated below each layer.

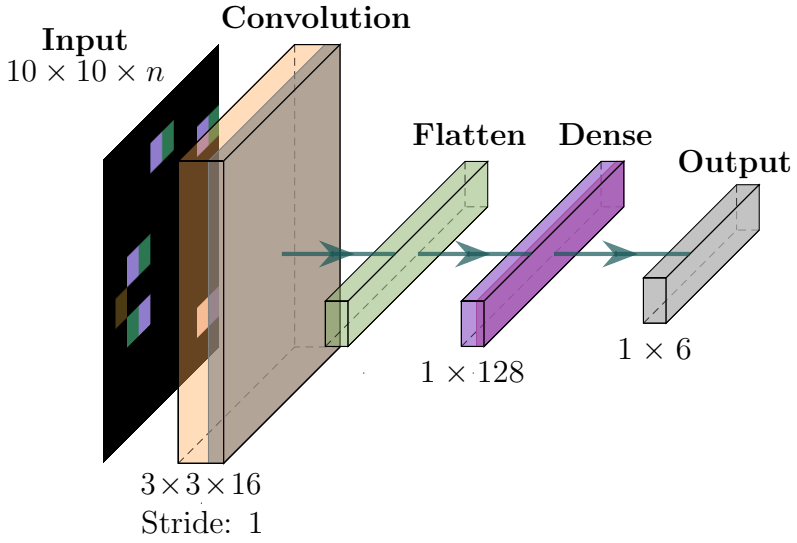


Figure 5.2: Neural network architecture used in the RND component (identical for the predictor and the target) in SNA and ANA. The dark shaded slices denote the presence of rectified linear unit activation function. Output is an arbitrary output with a fixed size of 6. The number of filters and units in the layers are indicated below each layer.

periodically in a separate sequence of episodes with any form of exploration and teaching disabled; then, the scores obtained in these episodes are averaged to determine the

evaluation score for this evaluation step. These scores reflect the agent's actual expertise in the corresponding step in the learning session. In Reach the Goal, since the actual cumulative reward at the end of an episode is either 0 or 1, a more informative evaluation score is defined to be in $[0, 1]$ by taking the number of remaining timesteps and distance to the goal tile into calculations. In MinAtar, the original game scores in the framework are used as evaluation scores. In addition to the evaluation scores, we also plot the number of advice taken in every 100 steps as well as cumulatively, to observe the trends in budget spending.

In the first stage of experiments, we use Reach the Goal as a simple and interpretable task to highlight the aforementioned drawbacks, as well as perform a preliminary benchmark on the methods to validate their suitability for tasks with more complex mechanics. The learning session lengths are set as 100k steps, and evaluations are performed at every 100^{th} step in a sequence of 5 episodes. We set two different scenarios, namely, Scenario I and Scenario II, with small and large budget options of 5k and 50k each, resulting in 4 cases in total. In Scenario I, the teacher is present from the beginning of learning sessions, which is the common experimental setting used in the previous action advising studies. In Scenario II, the teacher joins the loop at the 25k^{th} step. By having such a scenario, we test the ability of the student in dealing with the belated teacher. UA, SNA, ANA hyperparameters λ_{UA}, λ_{SNA}, λ_{ANA} are determined empirically in the 5k budget setting to be 0.001, 0.0001, 0.001, and are kept the same for all 4 cases.

In the second stage, we evaluate the approaches in a set of tasks with more complex dynamics presented via 5 different games in MinAtar environment. Learning sessions are set to have a length of 1.5M steps and evaluations are performed at every 1000^{th} step in a sequence of 5 episodes. Since the Scenario II experiments in Reach the Goal are sufficient to demonstrate the weakness regarding the extensive unavailability of the teacher, experiments in MinAtar are only conducted for Scenario I to evaluate the general performance of the methods, again with two different budget options of 50k and 250k. UA, SNA, ANA hyperparameters λ_{UA}, λ_{SNA}, λ_{ANA} are determined empirically

on game by game basis for 50k budget, to be 0.0001, 0.0025, 0.001 for Asterix; 0.001, 0.025, 0.025 for Breakout; 0.0001, 0.1, 0.05 for Freeway; 0.001, 0.05, 0.05 for Seaquest; 0.0025, 0.01, 0.0025 for Space Invaders, respectively.

Experiment results are aggregated over 9 different seeds for Reach the Goal, and over 5 different seeds for MinAtar games. Training and evaluation episode sequences (random game events) are fixed via random seeds and kept the same across different experiment seeds. Therefore, these experiment seeds only affect the agents' internal computations. Finally, we perform RND observation normalisation as in Burda et al. (2018) by using the mean and standard deviation calculated over the first 1000 and 5000 observations, in Reach the Goal and MinAtar, respectively. All the hyperparameters are tuned separately for each environment prior to the experiments which can be seen in Table 5.1 for the student's DQN and RND components.

5.4 Results and Discussion

The results of our experiments are presented in Figures 5.3 (Reach the Goal); and in Figures 5.4, 5.5, 5.6 and 5.7; Tables 5.2 and 5.3 (MinAtar). In Figure 5.3, the leftmost column contains two plots for the number of advice taken in total and in every 100 steps, in Scenario I with the budget amount of 50k; the middle column displays the evaluation scores in Scenario I, and the rightmost column displays the evaluation scores in Scenario II (with the budgets of 5k on top and 50k on bottom). Figures 5.4 and 5.5 include the plots of the number of advice taken in total and in every 100 steps, in Scenario I with the budget amount of 250k for all five MinAtar games. In Figures 5.6 and 5.7, plots of evaluation scores obtained in MinAtar games with two different budget settings of 50k (top row) and 250k (bottom row) are displayed. These results are presented numerically in Tables 5.2 and 5.3 with the area under the curve and final values. The table also shows whether the results are significantly different from ANA's results according to Welch's t-test with p-value < 0.05. $(+)$ sign on the left-hand side of a result indicates that ANA is significantly better than the corresponding method. Similarly, $(-)$ indicates

that ANA is significantly worse than it. We also denoted the best results in their own brackets in bold. The plots in Figure 5.3 and Figures 5.4 and 5.5 that display the number of advice taken are only generated for Scenario I with the maximum budget settings since the budget distribution is identical in the cases with smaller budgets with only the difference of being cut off early. The curves are plotted with appropriate moving average smoothing for the sake of comprehensibility, and the standard deviation across the runs is shown with the shaded areas.

5.4.1 Reach the Goal

In Scenario I with a small budget of 5k, all of the action advising methods performed reasonably well, with the exception of RA which fails to be any better than None as seen in Figure 5.3 (b) on the left. The poor performance of None indicates how challenging it can be to conduct exploration successfully even with an advanced method like NoisyNets when the time constraints are tight as in this Reach the Goal game. Despite taking plenty of expert advice, RA also fails due to its inability to consistently follow the teacher, which is especially essential in tasks requiring deep exploration like this one. This makes RA an unreliable action advising heuristic. Another noticeable trend here is how EA saturates below other approaches, which is caused by the inconsistent and highly variable learning dynamics of deep RL which became apparent mostly for EA in this case, as it can also be seen from the width of the error band. This indicates that EA was stuck with suboptimal policies in some of the runs.

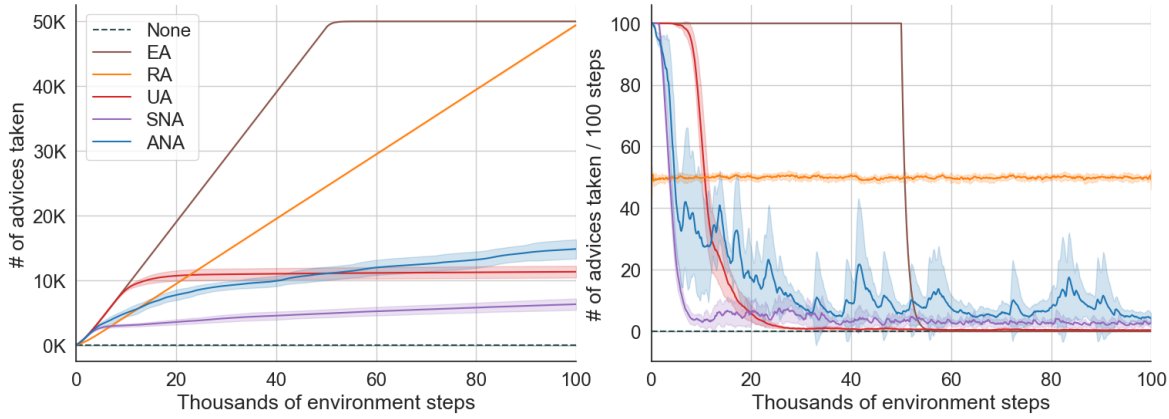
When the budget is increased to 50k, we see how the performances collapse due to taking too much advice hence not executing their own policies adequately to collect integral samples. This is most obvious in EA since it employs the most greedy way of spending the budget among all methods, which makes it dangerously susceptible to such high-budget settings. With the addition of more budget, RA finally manages to benefit from expert advice, however still very inadequately. The advanced methods of UA, SNA, and our ANA do a good job of not overusing all the budget given to

them even though they are not tuned to handle 50k. While UA performs slightly worse, ANA does better with the addition of an extra budget. As it can be seen, ANA does this while using more budget in the end than UA; this shows that it is not just about cutting off the advice requests but is also about distributing them in the appropriate states and across the learning session. In terms of budget efficiency, SNA seems to be doing the best in this case; however, it also has worse task performance. Finally, differently than UA and SNA, ANA is observed to follow a trend with occasional peaks in the number of advice taken per 100 steps. This is very likely to be caused by its RND update rule that uses single samples rather than batches that are also non-i.i.d. when the advice requests are made consecutively. As a result, RND model does not achieve global optimum and remains to yield significantly higher loss for the sample(s) that are not encountered recently. This is a unique characteristic of ANA which can be advantageous as it makes the teacher advice to be re-acquired occasionally.

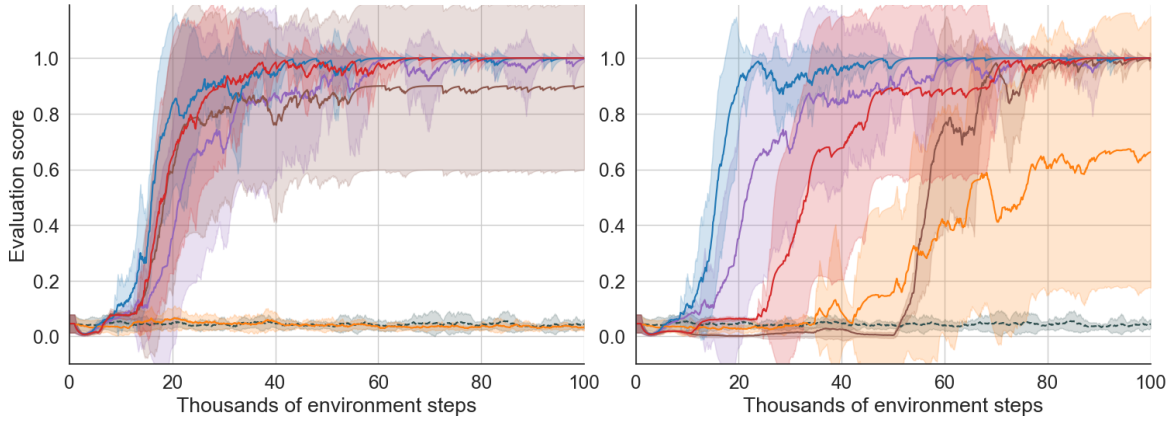
In Scenario II (right column of Figure 5.3 (b) and (c)), where the teacher joins the learning session at the 25k^{th} step, both UA and SNA fail to learn from the teacher as expected which aligns with poor performance caused by the exploration challenge as it is highlighted in Scenario I. This is due to their teacher-independent convergence in estimations of uncertainty and novelty, respectively. Our method ANA, however, manages to leverage the teacher's knowledge in both budget options, despite of the student being converged to suboptimal Q-value targets due to under-exploration. Clearly, if there is such a possibility as depicted in this scenario, methods like UA and SNA are not going to be suitable action advising methods to be employed.

5.4.2 MinAtar

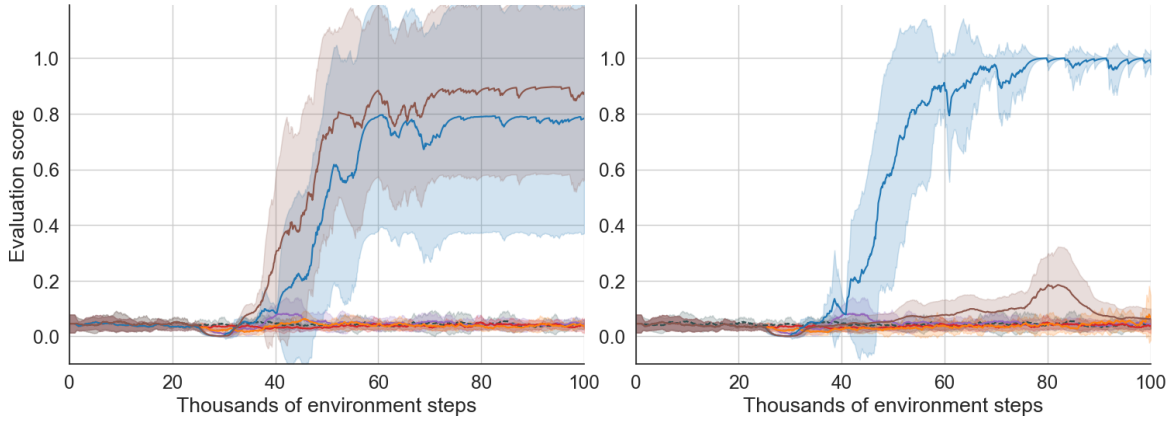
Results in MinAtar games present us a more general performance evaluation of the action advising techniques in a variety of tasks. The final evaluation scores are what we mainly consider when comparing the algorithms. Yet, we also analyse the AUC values to assess their performance in terms of learning speed.



(a) Distribution of 50k budget in Scenario I



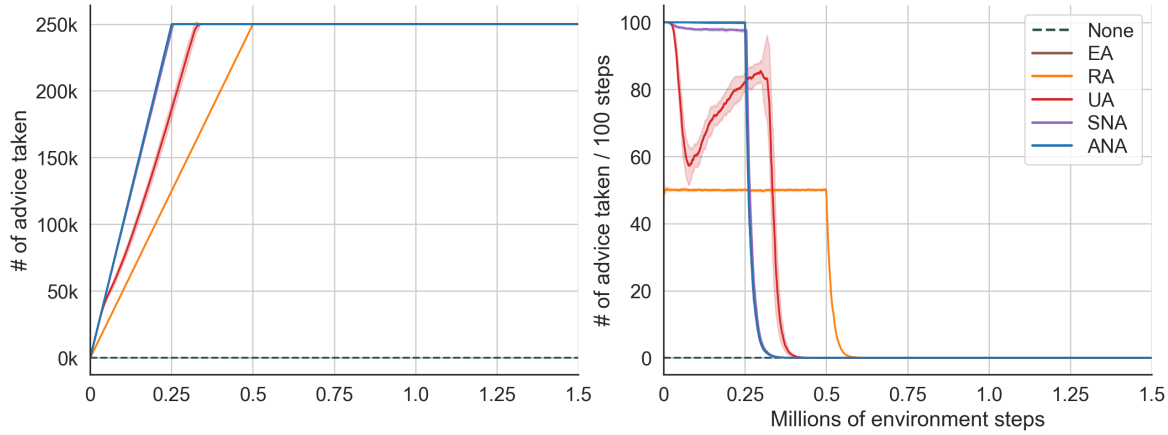
(b) Evaluation performance in Scenario I with 5k (left) and 50k (right) budgets



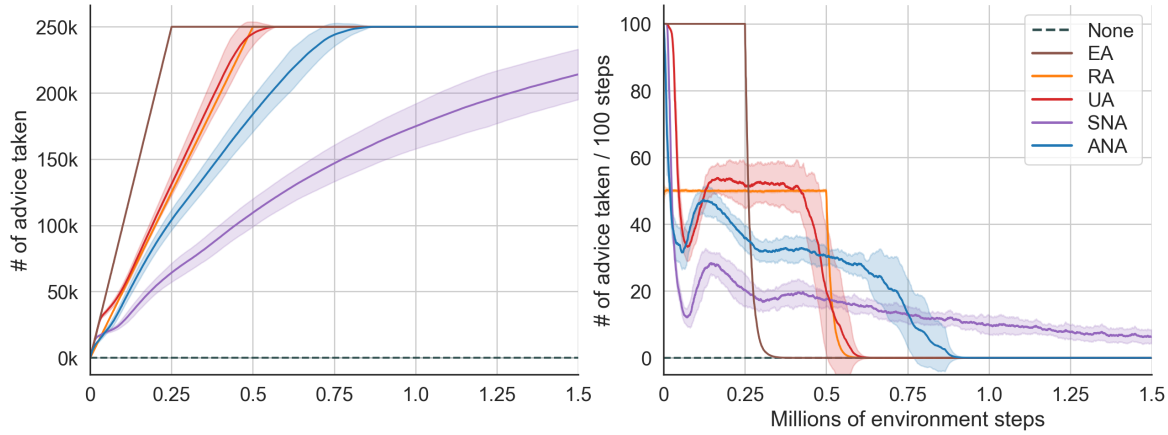
(c) Evaluation performance in Scenario II with 5k (left) and 50k (right) budgets

Figure 5.3: Number of advice taken cumulatively and in every 100 steps period in Scenario I with 50k budget (a); evaluation scores in Scenario I (b) and Scenario II (c) with 5k (left column) and 50k (right column) budgets, obtained in Reach the Goal game with no action advising (None) and action advising methods EA, RA, UA, SNA, ANA.

We first discuss the results obtained in the 50k budget setting as the primary comparison case, since the algorithms are tuned particularly for this setup. In Asterix, ANA is the best performing method with SNA and EA being very close to it, and they are followed by UA and RA. Considering the inefficacy of RA and the successful budget spending patterns of EA, SNA and ANA; it can be speculated that it is critical in this particular game to follow the teacher over many consecutive steps, similarly to Reach the Goal. In Breakout, ANA outperforms other advanced methods which already do well compared to None and EA. This in comparison to the case in Asterix shows how EA and RA can be unreliable choices as a trade-off for being very simple heuristics. Unlike Reach the Goal and Asterix, the successful methods in Breakout are the ones that ask for advice less frequently; this may be due to Breakout not requiring expert advice over many steps since the game events unfold on their own once the ball is hit with the paddle, and the different random moves taken in the meantime may be much more valuable sources to reduce RL model error, rather than taking the same expert advised actions such as just waiting stationarily until the ball traverses back down. Seaquest has very interesting results where every method achieves different standings in different stages of the learning session. In terms of the final performance, however, ANA manages to come on top again with SNA and RA following it after. UA performs rather poorly here, despite its rapid progression earlier in the learning. As a result of the Seaquest's in-game dynamics being the most complicated amongst all, it is not very clear what causes the learning fluctuations in these plots. In Freeway and Space Invaders, despite all the different advice requesting patterns followed by different methods, they are very similar when it comes to the evaluation scores. Even though it may be surprising at first considering that Freeway is a game with sparse rewards, its action space and the possible positions the agent can traverse in the game grid spatially are rather small. Therefore, the need for the expert advice to be distributed strategically across the game episode is rather negligible.



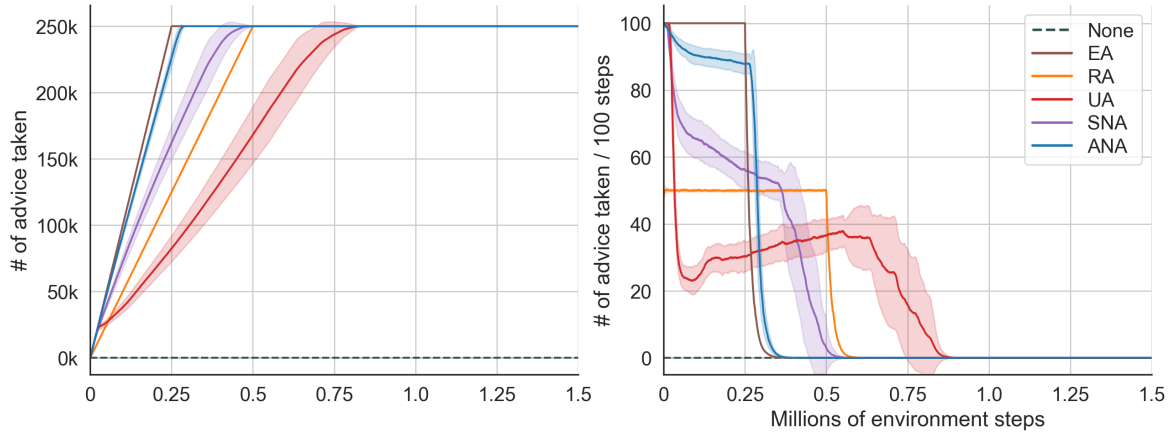
(a) Asterix



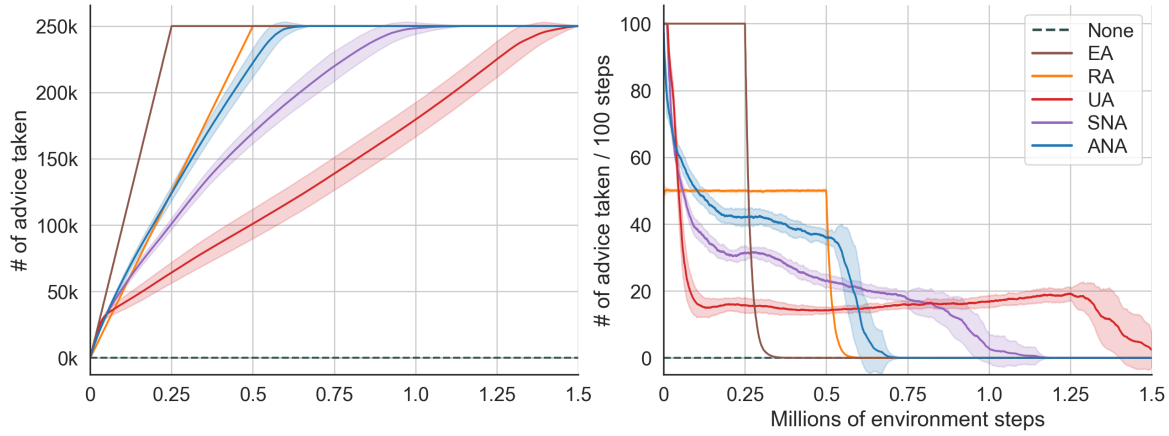
(b) Breakout

Figure 5.4: Number of advice taken cumulatively and in every 100 steps period in Scenario I with 250k budget, obtained in MinAtar games (Asterix, Breakout) with no action advising (None) and the action advising methods EA, RA, UA, SNA, ANA.

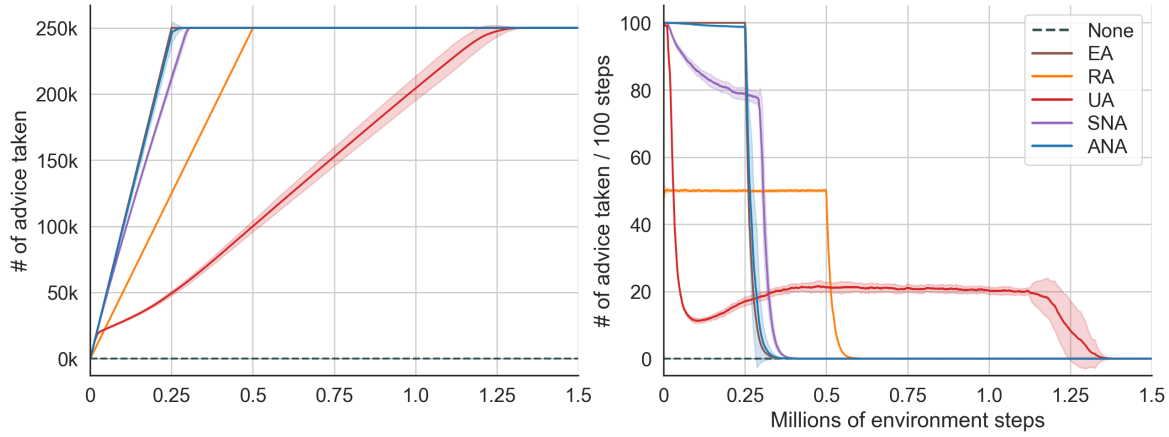
When the budget is increased to 250k, we observe a fair amount of performance deterioration in almost every advising mode, especially in the learning speeds. In Asterix, Freeway, and Space Invaders, even though the final performance is not affected greatly, there is a sharp drop in the learning progression caused by the over-advising induced delay in the collection of useful samples. This is closely linked to them behaving more similarly to EA in these cases as it can be seen in the budget plots. These results emphasise the importance of handling the redundant advice budgets. Currently, none of these action advising methods have an awareness of how many times they will get to



(a) Freeway



(b) Seaquest

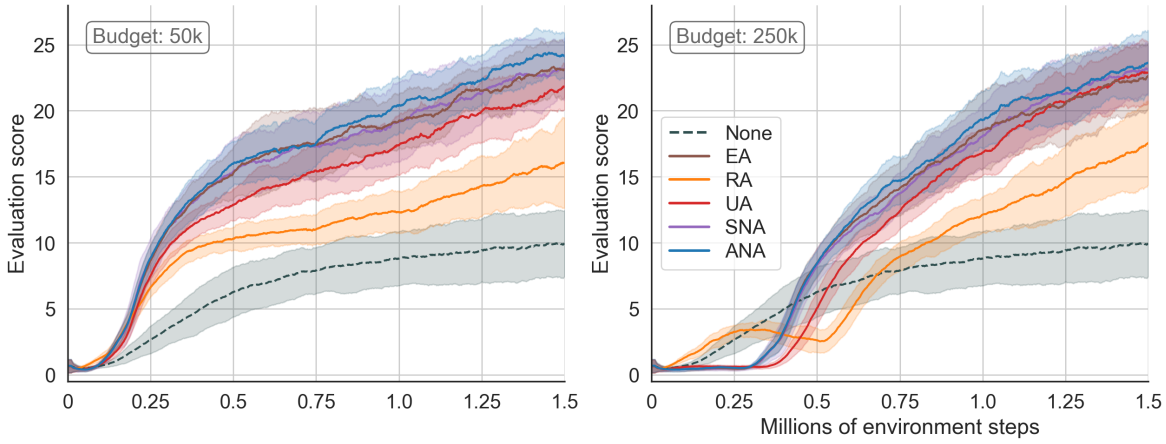


(c) Space Invaders

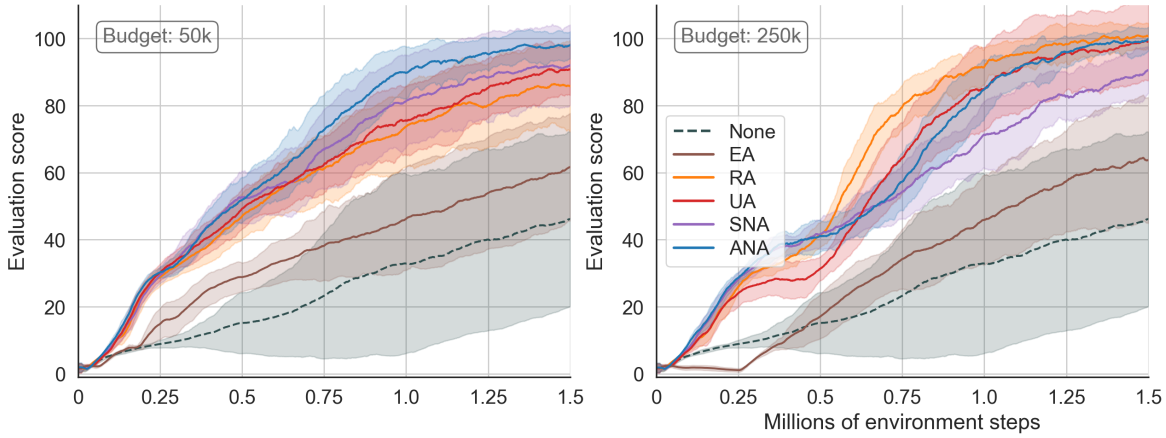
Figure 5.5: Number of advice taken cumulatively and in every 100 steps period in Scenario I with 250k budget, obtained in MinAtar games (Freeway, Seaquest, Space Invaders) with no action advising (None) and the action advising methods EA, RA, UA, SNA, ANA.

ask for advice. Instead, they are designed to make the most out of some supposedly small budget they are given, without any notion of long-term planning of its utilisation.

Overall, as the performance superiority of the advising methods over None suggests, action advising provides substantial advantages to accelerate learning. By looking at the standings of the algorithms in every game, we can see that our ANA is the winner in terms of the final performance both in the 50k and 250k budget options; it either achieves the top scores or remain very close to them. This is also visible in the overall percentages of final score improvements over EA, in which ANA achieved 18.7% and 19.8% while its closest followers got 13.8% (SNA) and 15.3% (RA), respectively in 50k and 250k budget settings. The significance analyses also support ANA's superiority by showing that it significantly outperformed its competitors in 23 out of 60 cases. Depending on the budget, ANA's performance is followed by the other methods in a different order. For instance, while SNA is far ahead of others and is behind ANA in 50k budget, RA takes its place in the 250k budget scenario. The decline in performance with higher budgets and RA's robustness to this by spacing out the advice requests to allow the student to execute its self policy more often points out the importance of collecting on-policy samples adequately even when the agent itself employs an off-policy RL algorithm, as highlighted in Fujimoto et al. (2019) and Kumar et al. (2019) as well. Clearly, different games require different strategies to distribute the action advising budget, and there is no straightforward way to determine a way that applies to all cases successfully. Furthermore, the ways these methods behave are also dependent on the underlying task, since the uncertainty and novelty estimations may be affected differently even with the identical streams of observations. For instance, while UA tends to spend its budget more slowly than others in most cases, in Breakout it behaves differently to output its best possible performance. Finally, it is worth it to mention that ANA is also observed to stand out in terms of computational efficiency compared to the runner-up SNA; while ANA updates its RND model with a single sample only when it successfully receives advice, SNA updates it for a batch of samples every time it performs learning until its budget reaches zero.

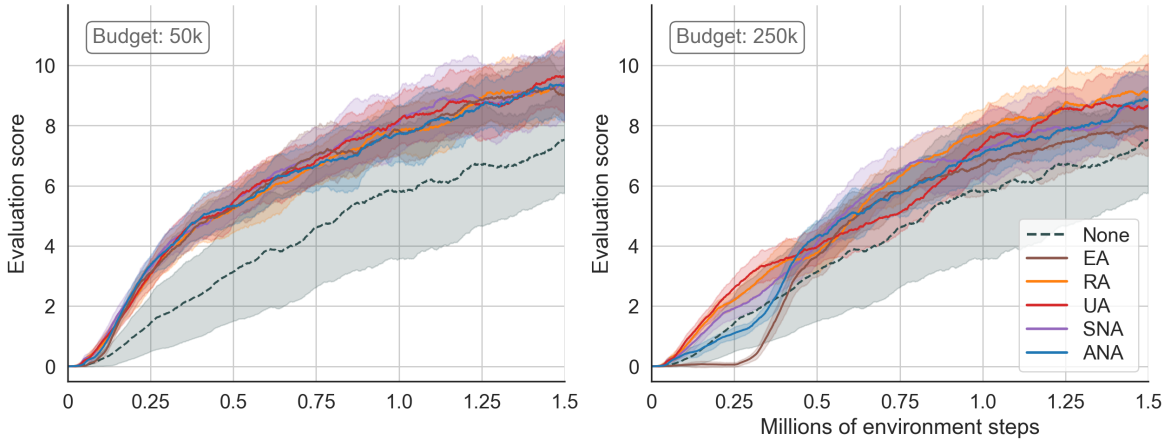


(a) Asterix

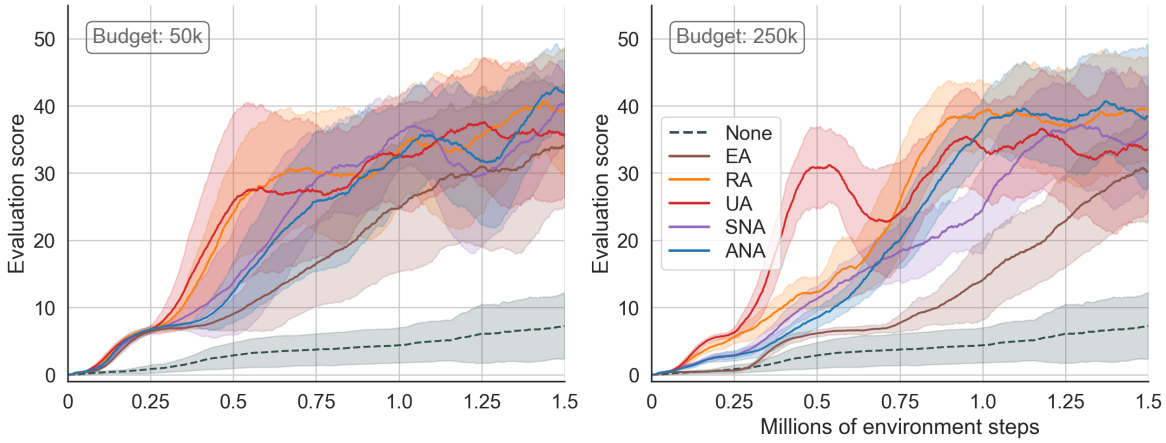


(b) Breakout

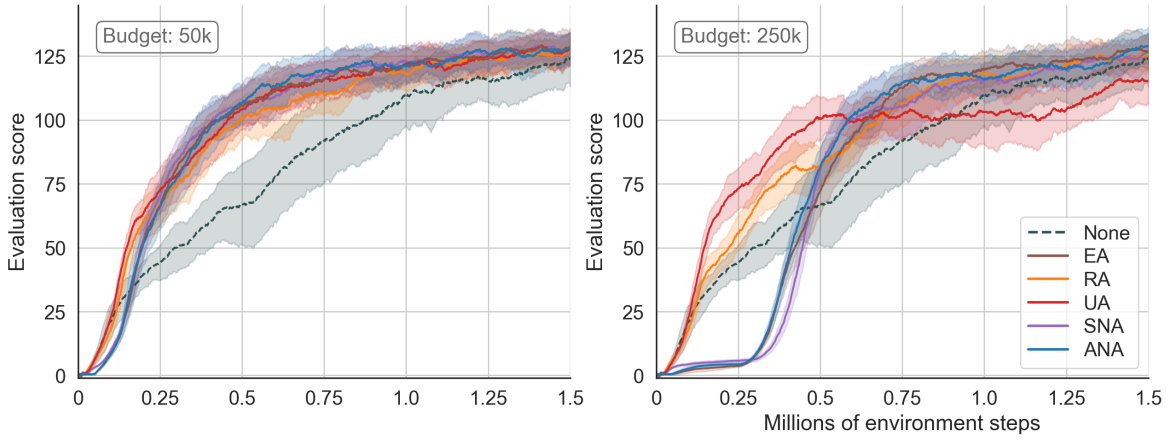
Figure 5.6: Evaluation scores of Scenario I with 50k (left) and 250k (right) budgets, obtained in MinAtar games (Asterix, Breakout) with no action advising (None) and action advising methods EA, RA, UA, SNA, ANA.



(a) Freeway



(b) Seaquest



(c) Space Invaders

Figure 5.7: Evaluation scores of Scenario I with 50k (left) and 250k (right) budgets, obtained in MinAtar games (Freeway, Seaquest, Space Invaders) with no action advising (None) and action advising methods EA, RA, UA, SNA, ANA.

5.5 Conclusions

In this work, we evaluated the prominent student-initiated action advising methods that are compatible with off-policy Deep RL agents. We highlighted their shortcomings such as not being able to handle the belated availability of the teacher, or requiring very specific Deep RL models to function. To address these, we proposed an alternative student-initiated action advising algorithm that utilises state novelty computed via RND to determine when to request a piece of advice. Differently from our previous work, RND is updated only with the states that are involved in the advice exchanges. Thus, it is ensured that the student will take advantage of the teacher as soon as it becomes available regardless of its own Deep RL model's convergence.

Empirical results in Reach the Goal and MinAtar games validate our speculations of the aforementioned drawbacks and show that the state-of-the-art methods that utilise state novelty or model uncertainty can be ineffective if the teacher is not present from the beginning. Furthermore, our advice novelty approach manages to be the strongest among its competitors by yielding the best overall standings in the majority of the experiments, as well as being able to handle belated teachers, not requiring Deep RL model uncertainty estimations, and also not interfering with the student's RL algorithm. It is also seen that there is no trivial way to define a general action advising strategy to distribute the budget efficiently across many different cases. Finally, it is found to be challenging for even the most complicated methods to handle excessive budgets without encountering a significant performance deterioration. Accordingly, the hyperparameters that are responsible to manage budget distribution require careful tuning considering both the task characteristics and the total available budget.

An interesting extension of this study would be further investigating the components of our approach to precisely determine how they behave and how their variants affect the performance. For instance, training RND incrementally with batches drawn from the complete set of collected advice instead of only the latest samples can provide valuable insights for comparison. Another direction for future work could involve devising a form

of threshold adaptation to make the action advising techniques more robust against the changes in task and budget specifications. Additionally, further analyses in the dynamics of different RL algorithms operating with action advising would be imperative to invent more general action advising methods. Our study employs a student agent with DQN and NoisyNets exploration; it will be worthwhile to investigate the performance of the action advising algorithms with different RL algorithms and exploration methods to see how their standings vary. Finally, there is still a significant research gap in action advising with multiple teachers which requires further attention, and we believe that our approach is likely to be useful in such a problem setting.

Table 5.2: The mean of last 50 values (Final Score) of the evaluation score plots of None, EA, RA, UA, SNA, ANA agent modes obtained in five MinAtar games averaged over 5 runs. The numbers denoted by ± represent standard deviation. The percentage values in parentheses indicate the relative difference to the values obtained by EA. *Overall* section in bottom presents these percentages averaged over these 5 games. (+) and (−) on the left-hand side of the results indicate whether ANA's results are significantly better or worse than them, respectively (according to Welch's t-test with p-value < 0.05). The best results in their own brackets are denoted in bold.

		Final Score			
Game	Mode	50k		250k	
Asterix	None	(+)	9.91 ± 2.4 (−57.2%)	(+)	9.91 ± 2.4 (−55.7%)
	EA		23.16 ± 2.0		22.37 ± 2.4
	RA	(+)	15.83 ± 2.9 (−31.7%)	(+)	17.16 ± 3.0 (−23.3%)
	UA	(+)	21.57 ± 1.4 (−6.9%)		22.79 ± 2.3 (+1.9%)
	SNA		23.00 ± 2.2 (−0.7%)		23.04 ± 1.9 (+3.0%)
	ANA		**24.24 ± 1.6(+4.7%)**		**23.36 ± 2.3(+4.4%)**
Breakout	None	(+)	45.30 ± 25.5 (−25.1%)	(+)	45.30 ± 25.5 (−28.9%)
	EA	(+)	60.48 ± 15.7	(+)	63.73 ± 18.1
	RA	(+)	86.06 ± 12.2 (+42.3%)		**100.65 ± 3.6(+57.9%)**
	UA	(+)	90.45 ± 7.1 (+49.6%)		98.84 ± 11.5 (+55.1%)
	SNA	(+)	91.50 ± 11.6 (+51.3%)	(+)	89.32 ± 7.4 (+40.1%)
	ANA		**97.78 ± 4.0(+61.7%)**		99.12 ± 3.7 (+55.5%)
Freeway	None	(+)	7.36 ± 1.6 (−19.3%)	(+)	7.36 ± 1.6 (−7.3%)
	EA		9.12 ± 0.8	(+)	7.94 ± 0.9
	RA		9.30 ± 0.9 (+2.0%)		**9.09 ± 1.1(+14.4%)**
	UA		**9.60 ± 1.0(+5.2%)**		8.59 ± 1.2 (+8.2%)
	SNA		9.29 ± 1.2 (+1.9%)		8.61 ± 1.0 (+8.4%)
	ANA		9.33 ± 1.0 (+2.2%)		8.82 ± 0.9 (+11.1%)
Seaquest	None	(+)	7.06 ± 4.6 (−79.1%)	(+)	7.06 ± 4.6 (−76.7%)
	EA	(+)	33.76 ± 8.8	(+)	30.36 ± 7.3
	RA		39.38 ± 8.2 (+16.6%)		**39.37 ± 7.4(+29.7%)**
	UA	(+)	35.98 ± 9.6 (+6.6%)		33.83 ± 9.8 (+11.4%)
	SNA		39.29 ± 6.3 (+16.4%)		35.19 ± 8.0 (+15.9%)
	ANA		**42.21 ± 6.0(+25.0%)**		38.57 ± 9.6 (+27.1%)
Space Invaders	None		122.62 ± 9.4 (−3.8%)	(+)	122.62 ± 9.4 (−3.8%)
	EA		**127.45 ± 6.8**		127.40 ± 6.2
	RA		125.95 ± 3.9 (−1.2%)		124.91 ± 5.2 (−2.0%)
	UA		126.56 ± 6.0 (−0.7%)	(+)	115.38 ± 8.7 (−9.4%)
	SNA		127.40 ± 5.4 (−0.04%)		124.95 ± 6.6 (−1.9%)
	ANA		127.23 ± 5.5 (−0.2%)		**128.26 ± 6.6(+0.7%)**
Overall	None		−36.9%		−34.5%
	RA		+5.6%		+15.3%
	UA		+10.8%		+13.4%
	SNA		+13.8%		+13.1%
	ANA		**+18.7%**		**+19.8%**

Table 5.3: Area under the curve (AUC) of the evaluation score plots of None, EA, RA, UA, SNA, ANA agent modes obtained in five MinAtar games averaged over 5 runs. The numbers denoted by ± represent standard deviation. The percentage values in parentheses indicate the relative difference to the values obtained by EA. *Overall* section in bottom presents these percentages averaged over these 5 games. (+) and (−) on the left-hand side of the results indicate whether ANA's results are significantly better or worse than them, respectively (according to Welch's t-test with p-value < 0.05). The best results in their own brackets are denoted in bold.

		AUC ($\times 10^3$)			
Game	Mode	50k		250k	
Asterix	None	(+)	10.01 ± 2.2 (−57.1%)	(+)	10.01 ± 2.2 (−45.2%)
	EA		23.33 ± 2.1		18.28 ± 2.1
	RA	(+)	15.67 ± 1.5 (−32.9%)	(+)	12.62 ± 1.4 (−31.0%)
	UA	(+)	20.77 ± 1.7 (−11.0%)	(+)	16.83 ± 1.4 (−7.9%)
	SNA		23.25 ± 2.6 (−0.3%)		18.29 ± 1.9 (+0.02%)
	ANA		**24.19 ± 1.6(+3.7%)**		**19.02 ± 1.6(+4.0%)**
Breakout	None	(+)	35.94 ± 22.2 (−32.2%)	(+)	35.94 ± 22.2 (−24.1%)
	EA	(+)	52.98 ± 12.0	(+)	47.34 ± 10.6
	RA	(+)	83.73 ± 9.7 (+58.1%)	(−)	**97.32 ± 3.4(+105.6%)**
	UA	(+)	86.81 ± 7.0 (+63.9%)		88.02 ± 9.8 (+86.0%)
	SNA	(+)	90.67 ± 10.3 (+71.2%)	(+)	80.58 ± 6.2 (+70.2%)
	ANA		**97.38 ± 5.9(+83.8%)**		89.83 ± 4.3 (+89.8%)
Freeway	None	(+)	6.25 ± 2.2 (−31.1%)	(+)	6.25 ± 2.2 (−10.0%)
	EA		9.08 ± 0.6	(+)	6.95 ± 0.6
	RA		9.03 ± 0.7 (−0.5%)	(−)	**8.37 ± 0.7(+20.5%)**
	UA		9.28 ± 0.7 (+2.2%)		7.97 ± 0.7 (+14.7%)
	SNA		**9.38 ± 0.8(+3.4%)**		7.98 ± 0.7 (+14.8%)
	ANA		9.06 ± 0.6 (−0.2%)		7.58 ± 0.6 (+9.1%)
Seaquest	None	(+)	5.33 ± 3.1 (−79.4%)	(+)	5.33 ± 3.1 (−68.4%)
	EA	(+)	25.84 ± 6.2	(+)	16.85 ± 3.6
	RA	(−)	**37.10 ± 5.4(+43.6%)**	(−)	35.33 ± 3.6 (+109.6%)
	UA	(−)	36.60 ± 5.2 (+41.6%)	(−)	**36.82 ± 4.0(+118.5%)**
	SNA		33.46 ± 4.0 (+29.5%)	(+)	28.07 ± 2.7 (+66.6%)
	ANA		32.77 ± 4.6 (+26.8%)		31.72 ± 3.0 (+88.2%)
Space Invaders	None	(+)	123.41 ± 13.2 (−17.3%)		123.41 ± 13.2 (−0.4%)
	EA		149.19 ± 5.2		123.86 ± 4.7
	RA		146.72 ± 5.9 (−1.7%)	(−)	**137.96 ± 4.7(+11.4%)**
	UA		150.67 ± 5.8 (+1.0%)	(−)	135.39 ± 7.4 (+9.3%)
	SNA		**150.82 ± 5.2(+1.1%)**	(+)	120.99 ± 6.3 (−2.3%)
	ANA		150.14 ± 6.4 (+0.6%)		125.05 ± 4.8 (+1.0%)
Overall	None		−43.4%		−29.6%
	RA		+13.3%		+43.2%
	UA		+19.5%		**+44.1%**
	SNA		+21.0%		+29.9%
	ANA		**+22.9%**		+38.4%

Chapter 6

Action Advising with Advice Imitation in Deep Reinforcement Learning

The contents of this chapter are based on our third publication titled "Action Advising with Advice Imitation in Deep Reinforcement Learning" (Ilhan et al. 2021*a*) which tackles the following research questions:

- [RQ1] To what extent can we accelerate Deep RL via (budget-limited) action advising with one or more knowledgeable peer(s)?
- [RQ5] How can we further utilise the collected advice by memorising and reusing them in Deep RL domains?

6.1 Introduction

The scope of the action advising problem is generally limited to answering "*when* to ask for advice?". In the previous chapters, we attempted to answer this same question with two alternative approaches. It is commonly not of any interest how the collected advice is utilised by the student agent's RL algorithm, e.g., how it is stored, replayed,

or discarded; especially since these are dealt with by the studies that focus on off-policy experience replay dynamics in general (Schaul et al. 2016, De Bruin et al. 2015) or the specific case of having a demonstration dataset as in LfD. However, even without interfering with the student's RL mechanism, it is still possible to make more of the collected advice by storing and reusing them.

The present action advising algorithms in Deep RL at the time this study has taken place have no way of telling if they have asked for advice in a very similar or even identical state already in the learning session. Thus, they do not record these in any way and usually end up requesting redundant advice from the teacher. In order to address this, we incorporate a separate neural network to do Behavioural Cloning (BC) (Pomerleau 1991) on the samples (state-action pairs which are equal to the state-advice pairs in the context of action advising) collected from the teacher. This network then will be able to serve as a state-conditional generative model that will let us sample advice for any given observation. However, since this model should also have a notion of distinguishing the recorded states from the unrecorded ones to avoid producing false advice for unfamiliar states, we also propose incorporating a well-known regularisation mechanism called Dropout (Srivastava et al. 2014) within this network to serve as an epistemic uncertainty estimator (Gal & Ghahramani 2016) which will allow the student to determine whether the state is recorded by comparing this estimation with a threshold. A key difference in this study from our work presented in the previous chapters is that we also scale up the complexity of the target Deep RL domain. Specifically, instead of experimenting in discrete state space (e.g., Reach the Goal, MinAtar), this time we target domains with more complex, continuous state space. This way, state-advice memorisation and advice reusing becomes an actual challenge, and the methods we develop can be applied to the majority of Deep RL domains that holds the assumption that a state is never encountered more than once. In parallel to this change, since we deal with a different aspect of action advising now, we decided to keep the baseline algorithm to be Early Advising. Our contributions in this study can be listed as follows:

1. We show that it is possible to generalise teacher advice across similar states in Deep RL with high accuracy.
2. We present an RL algorithm-agnostic approach to memorise and imitate the collected advice that is suitable for the Deep RL settings.
3. We demonstrate that advice reuse via imitation provides significant boosts in the learning performance in Deep RL even when it is paired with a simple baseline like Early Advising.

In Section 6.2, we present our approach in detail. Then, in Section 6.3, we describe our experiment procedure along with the algorithm specifications. Afterwards, we discuss the results in Section 6.4 and finalise this chapter of the study with concluding remarks in Section 6.5

6.2 The Approach

In this setting, a student agent that employs an off-policy Deep RL algorithm performs learning in an episodic single-agent environment through trial-and-error interactions. It receives an observation s_t and then executes an action a_t generated by its policy π_S to receive a reward r_{t+1} at each timestep t, in order to maximise its cumulative discounted rewards in any episode. According to the teacher-student paradigm (Chapter 2.4.3) we adopt, there is also an isolated peer that is competent in this same task and is referred to as the teacher. For a limited number of times defined by the action advising budget b, the student is allowed to acquire action advice from the teacher for the particular state s it is in. While the teacher can have its own teaching strategies to generate actions to advise, in our setting, we determine the action to be advised greedily from the teacher's behaviour policy as $\pi_T(s)$. This is a commonly followed approach with the assumption of the teacher and the student's optimal task-level strategies are equivalent. The student considers this advice as a part of a high-reward strategy and follows them upon collection. In this final form of the problem, the student's objective is to spend

its budget at the most appropriate times to maximise its learning performance.

We aim to devise a method that will enable the student to memorise the collected advice to be able to re-execute them in similar states; therefore, avoiding wasting its budget in redundant states and potentially being able to follow the teacher's advice many more times than its budget. In tabular RL, this is trivial to achieve simply by storing the advised actions paired with the states in a look-up table. When it comes to deep RL where any particular observation is not expected to be encountered more than once, however, there needs to be a generalisable approach. For this purpose, we propose the student agent to employ a separate behavioural cloning module, which consists of a neural network as the state-conditional generative model $H_\eta \colon \mathcal{S} \times \mathcal{A} \to [0, 1]$. By training H_η in a supervised fashion with the obtained state-advice pairs (stored in a buffer $\mathcal{C}$) to minimise the negative log-likelihood loss $\mathcal{L}(\eta) = \sum_{\langle s,a \rangle \in \mathcal{C}} - \log H_\eta(a \mid s)$, the student can imitate the teacher's advice to reuse them accordingly. However, this method does not have any mechanisms to prevent the student from generating incorrect advice from the states it has not collected. Therefore, we also employ Dropout regularisation in H_η in order to grant this behavioural cloning module a notion of epistemic uncertainty through measuring the variance in the outputs obtained from multiple forward passes for a particular input state. We denote this uncertainty estimation by $H^u_\eta(s)$. The states H_η is trained on will be less susceptible to the variance caused by the dropout and yield smaller uncertainty values. By this means, the student can determine how likely a state is to be already recorded as advised when it comes to reusing them, and can make a decision according to a threshold.

An obvious question regarding the feasibility of reusing advice in deep RL arises here: can the teacher's advice be generalised over similar states accurately? As we investigate in the experiments in Section 6.4, actions generated by the teacher policy usually span over similar states. Clearly, the uncertainty threshold to consider a state as recorded is responsible for the trade-off between the reusing amount and the accuracy of the self-generated teacher advice. A small threshold value makes the student reuse its budget in fewer states with higher accuracy, whereas a larger value results in more

frequent reusing with lower accuracy.

The detailed breakdown of our approach is summarised with an emphasis on the proposed modifications as follows (shown in detail in Algorithm 15[1]): The student starts with a randomly initialised H_η and empty $\mathcal{C}$. At each timestep t with the (observed) state s_t and an undecided action a_t, the student first checks if $\mathcal{C}$ has any new samples. As soon as $\mathcal{C}$ reaches the size defined by $N_\mathcal{C}$, H_η is trained with mini-batch gradient descent over the samples in $\mathcal{C}$ for K_{BC} iterations[2]. Afterwards, if the environment was reset (a new episode started), the student determines whether to enable advice reuse via imitation for this particular episode with a probability of ϵ_{reuse}, which is combined with other conditions too later on in the algorithm. The idea behind employing this condition is to ensure that the student can also execute its own exploration policy in order to increase the data diversity in its replay memory, which is crucial to improve the quality of learning. Furthermore, determining this variable on an episodic basis lets the agent follow consistent policies in the exploration steps, rather than dithering between two policies. In the next phase, the student deals with the advice collection. We adopt the simple yet strong baseline of early advising here. According to this, the agent just collects advice without any conditions until its budget runs out[3]. In the next phase, the student decides whether to reuse advice generated by its H_η. There are several conditions to be satisfied for this to occur in addition to the advice reuse being allowed for this particular episode. Firstly, a_t must be non-determined, which implies the agent has not collected any advice from the teacher already. Secondly, H_η must be already trained so that it can generate meaningful actions. Then, the student also checks if its own action a_t determined via π_S is explorative. This condition limits the action advising actions to the exploration steps only in order to prevent overriding the student's actual policy which may result in a lack of Q-value corrections and cause

[1]Code for our experiments can be found at `https://github.com/ercumentilhan/naive-advice-imitation`

[2]The implementation of this part can be found in lines 188-198 of `code/executor.py` file in the code repository.

[3]The implementation of this part can be found in lines 218-225 of `code/executor.py` file in the code repository.

Algorithm 15 Action Advising with Advice Imitation

Input: Number of training iterations t_{max}, RL algorithm-related parameters, e.g. DQN, teacher policy π_T, advice-requesting budget b_{ask}, advice reuse uncertainty threshold τ_{reuse}, advice reuse probability (episodic) ϵ_{reuse}, number of samples needed in $\mathcal{C}$ to trigger initial BC training $N_{\mathcal{C}}$, number of BC training iterations K_{BC}.
Initialise RL algorithm-related variables, e.g. DQN
Initialise BC network H_η
Initialise empty buffer $\mathcal{C}$ to store state-action tuples
$reuse\_enabled \leftarrow False$ ▷ Disable advice reuse by default
for training steps $t \in \{1, 2, \dots t_{max}\}$ **do**
  Get state observation s_t from *Environment* if *Environment* is reset
  .....
  if $|\mathcal{C}| = N_{\mathcal{C}}$ **then**
    Train H_η for K_{BC} iterations ▷ Behavioural cloning training
  end if
  $a_t \leftarrow None$ ▷ Set action as non-determined
  Set $reuse\_enabled$ $True$ with ϵ_{reuse} probability
  if $b_{ask} > 0$ **then**
    $a_t \leftarrow \pi_T(s_t)$ ▷ Obtain advice from the teacher
    $\mathcal{C} \leftarrow \mathcal{C} \cup \langle s_t, a_t \rangle$ ▷ Add the state-advice pair to the BC dataset
    $b_{ask} \leftarrow b_{ask} - 1$ ▷ Decrease budget by 1
  end if
  if a_t is *None* **and** $a_t \leftarrow \pi_S(s_t)$ is explorative **and**
  H_η is trained **and** $H_\eta^u(s_t) < \tau_{reuse}$ **and**
  $reuse\_enabled$ **then**
    $a_t \leftarrow \arg\max_a H_\eta(a \mid s_t)$ ▷ Generate imitated advice to reuse
  end if
  .....
  if a_t is *None* **then**
    Determine a_t via the RL algorithm, e.g. DQN (denoted as π_S)
  end if
  Execute a_t and obtain r_{t+1}, s_{t+1} from *Environment*
  Update the RL algorithm, e.g. DQN
  $s_t \leftarrow s_{t+1}$
end for

deteriorative effects when too much advising occurs. Finally, it is checked whether $H_\eta^u(s_t)$ is smaller than the reuse threshold τ_{reuse}[4]. Incorporating such a threshold is important to limit the imitated advice to the states that have low uncertainty according to H_η to achieve higher accuracy of generating correct teacher actions. On one hand, having this threshold too high would make the student consistently follow H_η which would result in a dataset with lower diversity. On the other hand, if τ_{reuse} is set too small, then H_η would be ignored in most of the cases and the student would be following its own exploration policy. After all these steps, if a_t is still non-determined, the student follows its own policy and decide a_t via π_S. Clearly, our proposal can be isolated from the student's underlying RL algorithm (which is DQN in this particular study) as it can be seen in Algorithm 15 which encloses our approach by dots in lines 8-28.

6.3 Experimental Setup

The goal of our experiments is to demonstrate that it is possible to generalise the teacher advice to the unseen yet similar states with our method, and that it is an effective way of improving the performance of action advising, in complex domains especially. Therefore we choose the ALE games Enduro, Freeway and Pong, described in Section 3.4 as our test-beds. The set of the student agent variants we compare are listed as follows:

- **No Advising (None)**: No action advising procedure is followed; the student learns as normal.
- **Early Advising (EA)**: The student follows the early advising heuristic to distribute its advising budget. Specifically, the teacher is queried for a piece of advice at every step until the budget runs out.
- **Early Advising with Advice Reuse via Imitation (AR)**: The student fol-

[4]The implementation of this part can be found in lines 228-241 of `code/executor.py` file in the code repository.

lows our proposed strategy (Section 6.2) combined with the early advising heuristic. It starts off by greedily asking for advice until its budget runs out; then, it activates its behavioural cloning module to imitate and reuse the previously collected advice in the remaining exploration steps.

All student agent variants employ the identical Deep RL algorithm which is DQN with Double DQN and Dueling Networks enhancements and ϵ-greedy policy as the exploration strategy. The convolutional neural network structure within the DQN in input-to-output order is as follows: 32 8×8 filters with a stride of 4, 64 4×4 filters with a stride of 2, 64 3×3 filters with a stride of 1, followed by a fully-connected layer with 512 hidden units and multiple streams that add up in the end (dueling). Additionally, the student agent variant AR also incorporates a behaviour cloning module, which is a neural network with an identical structure minus the dueling stream. All the layer activations are set to be rectified linear units. This network architecture can seen in Figure 6.1.

AR student variant employs an extra neural network for BC. This is a very similar convolutional network to the DQN one. It consists of 3 convolutional layers with 32 8×8 filters with a stride of 4, 64 4×4 filters with a stride of 2, 64 3×3 filters with a stride of 1, followed by a fully-connected layer with 512 hidden units with a final Softmax output layer that has $|\mathcal{A}|$ values. Differently from DQN, this network also employs Dropout in its fully-connected layers. The network architecture can be seen in Figure 6.2.

Both of these networks use rectified linear units (ReLU) in every layer but the final ones. Linked to this, the weight initialised techniques are set to be He initialisation (He et al. 2015) due to its stronger empirical results with ReLU than the other initialised methods at hand.

In this teacher-student setup, we also need a teacher from which the student can get good quality action advice. For this purpose, we trained a DQN agent for each of these games for 10M steps (40M actual game frames) to achieve a competent level of

performance in each.

The experiments are conducted by executing every student variant through a learning session 3M steps (12M actual game frames) for every game. The learning steps are kept relatively small compared to the teacher training since it is expected for the students to achieve high performance much quicker with the aid of advice. Through the learning sessions, the agents are also evaluated at every 25k^{th} step in a separate instance of the environment for 10 episodes. During the evaluation, any form of exploration and teaching is disabled in order to assess the actual proficiency of the students.

In terms of action advising setup, we set the action advising budget as 10k steps which correspond to only approximately 0.3% of the interactions in a learning session and also to almost one-third of a full game episode (27k steps). Besides the budget, our proposed method AR also uses some additional hyperparameters which were tuned prior to the full-length experiments and are kept the same across every game. The dataset size $N_{\mathcal{C}}$ to train H_η is set as 10k which is the action advising budget as we employ early advising prior to behavioural cloning training. The number iterations to train H_η is set as 50k. Episodic advice reuse probability ϵ_{reuse} is set as 0.5 meaning that the student will follow H_η in half the episodes (in the appropriate states). Finally, the advice reuse uncertainty threshold τ_{reuse} is set as 0.01 (determined empirically) and kept the same across all games. In the experiments with AR, we also record the actual advice actions generated by the teacher at every step (not seen by the student) to have access to the ground-truth values to measure the accuracy of the behavioural cloning module. Every particular experiment case is repeated and aggregated over 3 different random seeds. The hyperparameters are tuned prior to experiments and kept the same across all experiments can be seen in Table 6.1.

6.4 Results and Discussion

The results of our experiments are presented in Figures 6.3, 6.4 and Tables 6.2, 6.3. The left-hand side of the Figures 6.3 and 6.4 contains the plots for the evaluation scores

Table 6.1: Hyperparameters used in the student's DQN (top section), imitation and action advising modules (bottom sections).

Hyperparameter name	Value
Replay memory size to start learning $M_{\mathcal{D}}$	50k
Replay memory capacity $N_{\mathcal{D}}$	500k
Target network update period (steps) T_{target}	7500
Train period (steps) T_{train}	4
Minibatch size	32
Learning rate α	625×10^{-7}
Discount factor γ	0.99
Adam epsilon	1.5×10^{-4}
Huber loss delta	1
ϵ initial	1.0
ϵ final	0.01
ϵ decay steps	500k
Buffer size to trigger BC training N_C	10k
Number of BC training iterations K_{BC}	50k
Minibatch size	32
Learning rate	0.0001
Adam epsilon	1.5×10^{-4}
Dropout rate χ	0.2
Number of forward passes to assess uncertainty	100
Advice-asking budget b_{ask}	10k
Advice-reusing threshold τ_{reuse}	0.01
Advice-reusing probability ϵ_{reuse}	0.5

obtained by None, EA and AR modes of the student in the games Enduro, Freeway, Pong. On the right-hand side of the Figures 6.3 and 6.4, the plots of the advice reuse trends of AR in this set of games are displayed cumulatively (top row) and in every 100 steps windows (bottom row). These plots are limited to the first 500k steps to only consider the exploration stage determined by the agent's ϵ-greedy schedule. Purple lines here represent all advice reuses combined, while the green lines indicate only the correctly imitated (in terms of being equal to the ground-truth teacher advice) advice pieces. These results are also reported in Tables 6.2 and 6.3 in the numerical form where the evaluation scores are broken down into two parts of the final value and area-under-the-curve, which represent the final agent performance and the learning

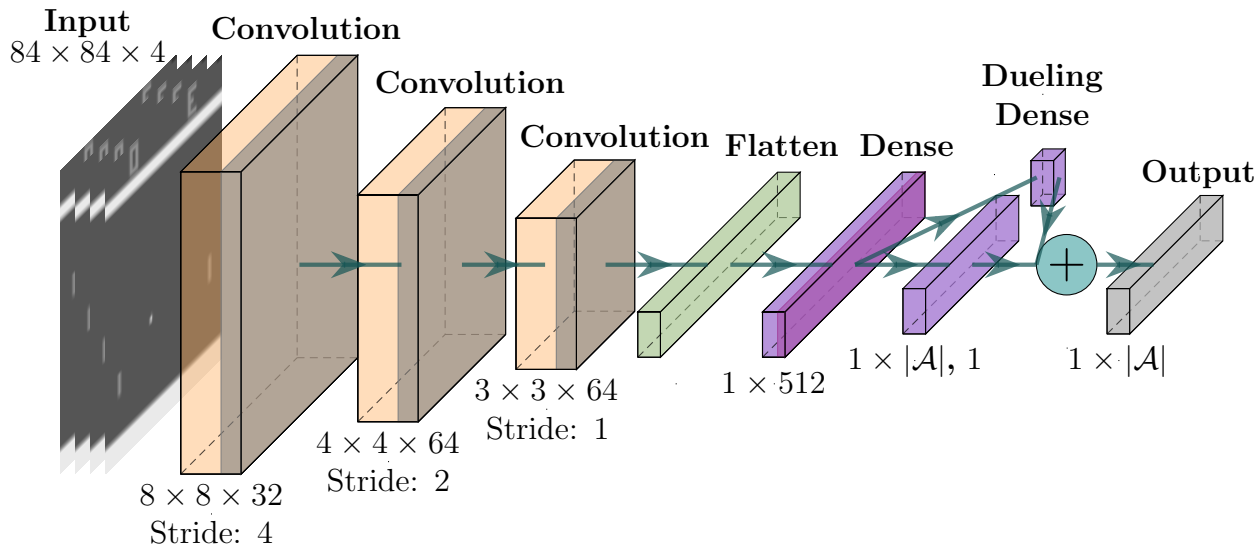


Figure 6.1: Neural network architecture of DQN where $|\mathcal{A}|$ is the number of actions. The dark shaded slices denote the presence of rectified linear unit activation function. Output is the Q-values. The number of filters and units in the layers are indicated below each layer.

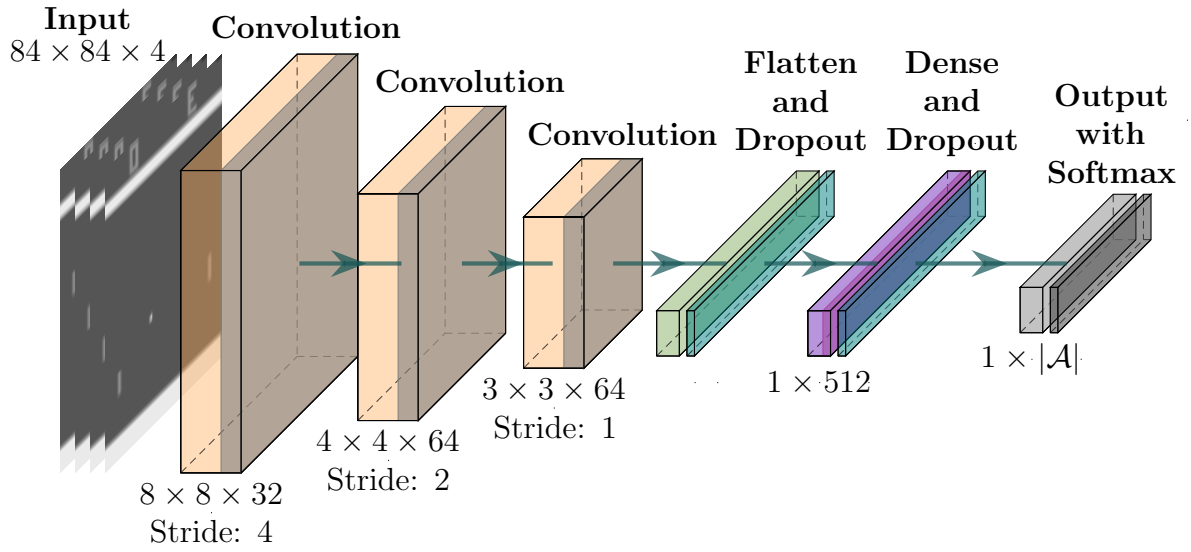
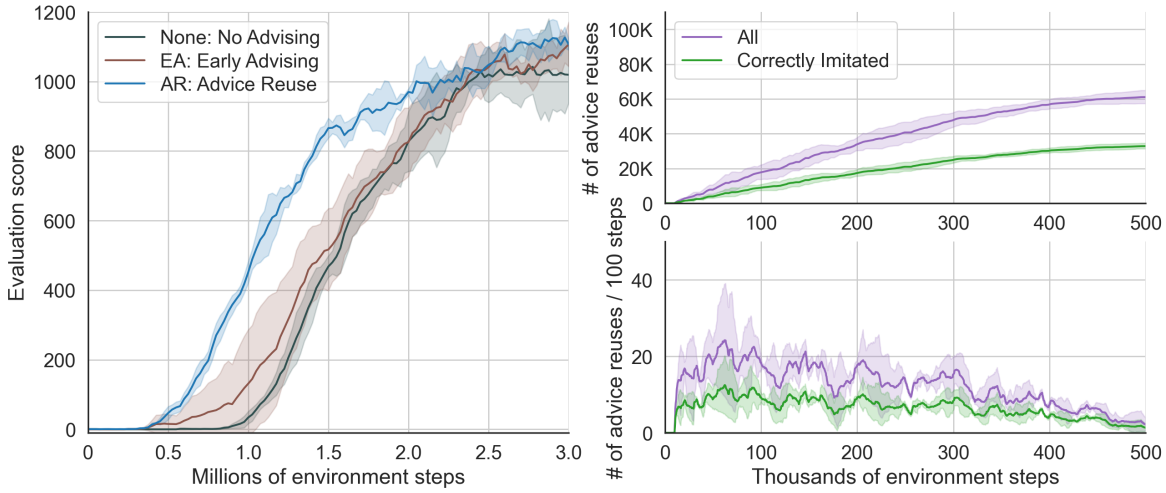
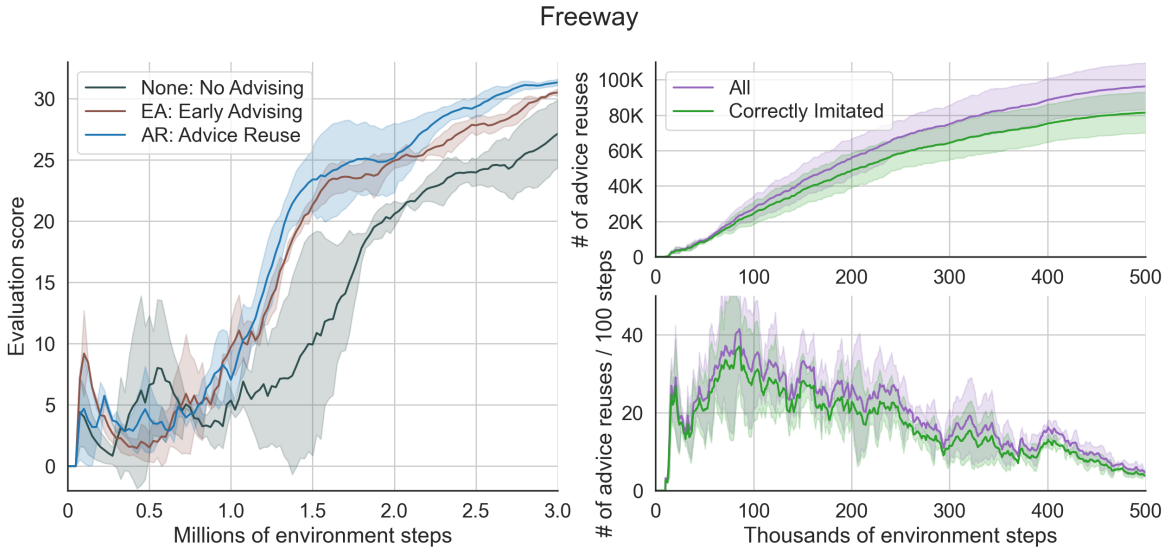


Figure 6.2: The architecture of the Behavioural Cloning neural network where $|\mathcal{A}|$ is the number of actions. The dark shaded slices denote the presence of rectified linear unit activation function, and the thin teal-coloured layers indicate Dropout. Output is the action probabilities. The number of filters and units in the layers are indicated below each layer.



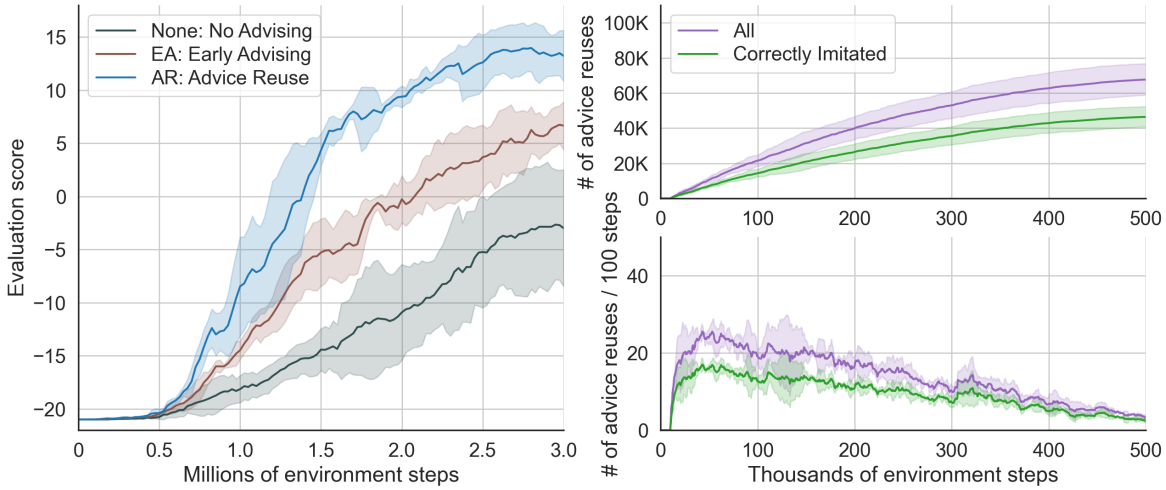
(a) Enduro



(b) Freeway

Figure 6.3: Evaluation scores of the student variants None, EA, AR (left) and number of advice reuses performed by the student with AR mode plotted cumulatively (upper right) and in every 100 steps (lower right), obtained in the Atari games of Enduro (a) and Freeway (b), aggregated over 3 runs. Purple lines represent the number of all advice reuses while the green lines represent the number of correctly imitated ones among these. Shaded areas show the standard deviation across the runs.

speed, respectively. Furthermore, the table also contains the total number of exploration steps taken, as well as the percentage of the number of reused advice in the exploration steps and the percentage of correctly imitated advice in the total number of reused



(a) Pong

Figure 6.4: Evaluation scores of the student variants None, EA, AR (left) and the number of advice reuses performed by the student with AR mode plotted cumulatively (upper right) and in every 100 steps (lower right), obtained in the Atari game of Pong (a), aggregated over 3 runs. Purple lines represent the number of all advice reuses while the green lines represent the number of correctly imitated ones among these. Shaded areas show the standard deviation across the runs.

advice (denoted in parentheses).

In the evaluation scores, we see different outcomes in each of these games. In Enduro, we see that AR provides a significant amount of jump start and performs the best in terms of learning speed while being far ahead of EA and None which are quite similar. When it comes to the final performance, however, while EA and AR both outperform None, they do not differ much from each other. In Freeway, EA and AR perform very similarly in terms of learning speed and final performance with AR being slightly ahead of EA. However, they outperform None significantly. This shows that it matters to be advised initially, though their repetitions may not always yield much acceleration in learning. Finally, in Pong, we see a great difference between the performances in every aspect. Our AR comes out far ahead of its closest follower EA both in terms of the final score and learning speed. This is an example of how getting very little advice in the beginning as well as repeating them across further explorative actions can cause a great impact on learning. Overall, AR manages to be the best in every game and suffers no

Table 6.2: Final and area-under-the-curve (AUC) values of evaluation score plots (Figures 6.3 and 6.4) of None, EA, AR student modes obtained in the Atari games of Enduro, Freeway, Pong aggregated over 3 runs. The numbers denoted by ± indicate standard deviation.

		Evaluation Score	
Game	Mode	Final	AUC ($\times 10^2$)
Enduro	None	1021.54 ± 79.5	570.61 ± 38.4
	EA	1095.55 ± 45.9	616.29 ± 58.1
	AR	$\mathbf{1112.79 \pm 16.6}$	$\mathbf{782.98 \pm 8.4}$
Freeway	None	26.87 ± 2.3	15.73 ± 1.7
	EA	30.44 ± 0.2	20.31 ± 0.4
	AR	$\mathbf{31.28 \pm 0.2}$	$\mathbf{21.52 \pm 1.0}$
Pong	None	-2.78 ± 4.3	-16.24 ± 2.6
	EA	6.66 ± 1.6	-8.83 ± 0.4
	AR	$\mathbf{13.35 \pm 1.7}$	$\mathbf{-1.36 \pm 1.0}$

performance loss even with high advice utilisation (as high as 104k in Freeway) which was shown to be harmful to learning in previous studies. Even though its performance boost over None seems to be not huge in every scenario, it should be noted that this is the case of it being combined with EA baseline. With more complicated methods, AR can be capable of training its imitation learning module with a more diverse set of experience and therefore, have a larger coverage which can potentially yield superior performance.

The task-level performance of our approach is affected primarily by two factors: the accuracy of advice imitation and its coverage/usage in the remainder of the exploration steps (the process of reusing). Therefore, we also analyse the advice reuse statistics of AR to form links between these outcomes. First of all, it should be noted that the decreasing trend in these plots is caused by the ϵ-greedy annealing. Enduro is the game with the smallest advice reuse rate as well as the lowest imitation accuracy. This is possibly because of the game episodes lasting long regardless of the agent's performance, which is likely to reduce the proportion of the familiar states according to the behavioural cloner. In Freeway, we observe a fairly high advice reuse rate

Table 6.3: The number of exploration steps and the number of advice reuses (all and correctly imitated) of None, EA, AR student modes obtained in the Atari games of Enduro, Freeway, Pong aggregated over 3 runs. The numbers denoted by ± indicate standard deviation. The numbers in the parentheses show the percentage of reused advice in the exploration steps (in the column titled "All") and the percentage of correctly imitated advice in total number of reused advice (in the column titled "Correctly Imitated").

Game	Mode	# of Exp. Steps	# of Advice Reuses	
			All	Correctly Imitated
Enduro	None	326939 ± 92.1	—	—
	EA	326753 ± 220.9	—	—
	AR	326889 ± 230.5	67198 ± 3061.0 (20.55%)	36534 ± 1210.9 (54.44%)
Freeway	None	326872 ± 199.9	—	—
	EA	327158 ± 6.2	—	—
	AR	326778 ± 494.4	104770 ± 12522.2 (32.05%)	88829 ± 10950.5 (84.74%)
Pong	None	326744 ± 25.2	—	—
	EA	326872 ± 199.9	—	—
	AR	326933 ± 371.2	72581 ± 7615.7 (22.20%)	49538 ± 4853.8 (68.32%)

with high accuracy of imitation. However, this is not reflected in the performance difference obtained versus EA, unlike in Enduro and Pong. Finally, in Pong, where the performance improvement is the most significant, the advice reuse ratio seems to be similar to Enduro, but with far higher imitation accuracy.

Clearly, as we see from all these results combined, we can say that it is definitely a viable idea to extend the teacher advice over future states through imitation since this can be achieved with relatively high accuracy. However, even when we have access to these imitated competent policies, it is still non-trivial to construct a *good* exploration policy. While a higher advice reuse rate produces a more consistent exploration policy with less random dithering, it also has the risk of limiting the sample diversity in the replay memory, which can be problematic especially if the imitation quality is also poor. As long as the reuse amount does not get excessively high, it is safe to have imitation learning accuracy around these reported levels, which makes tuning the uncertainty threshold straightforward. This is especially important for realistic applications where it is not possible to access the tasks to tune such hyperparameters beforehand.

Finally, we also analyse our approach's computational burden, which may be the primary concern when adopting it. Specifically, it involves two extra operations: behavioural cloning network training and uncertainty estimations. The former happens only once in the beginning and therefore is negligible. The uncertainty estimations that require multiple forward passes (which is 100 in our experiments) happen in every exploration step and were found to cause a maximum of 2× slowdown in our experiments. Considering that the exploration steps only span approximately 10% of a learning session, we can expect the runs to be taking at most 10% longer in total when AR is employed in a similar setting to ours; and, this becomes even smaller when the learning sessions last longer in terms of the total number of environment steps. Clearly, this is a small setback considering the sample efficiency benefits our method brings.

6.5 Conclusions

In this study, we developed an approach for the student to imitate and reuse advice previously collected from the teacher. This is the first time such an approach has been proposed in Deep RL. In order to do so, we followed an idea similar to behavioural cloning, employing a separate neural network that is trained with the advised state-action pairs via supervised learning. Thus, this module can imitate the teacher's policy in a generalisable way that lets us apply it to the unseen states. We also incorporated a notion of epistemic uncertainty via dropout in this neural network to be able to limit the imitations to the states that are similar to the advice collected states.

The results of the experiments in 3 Atari games have shown that it is a feasible idea to accurately generalise a small set of teacher advice over unseen yet similar states in future. Furthermore, our approach of employing behavioural cloning was found to be a successful way of achieving this, as it yielded a considerably high accuracy of imitation in multiple games. Additionally, reusing these self-generated advice across the exploration steps provided significant improvements in the learning speeds and the final performances without any over-advising-induced performance deterioration.

Therefore, our method can be considered as a promising enhancement to the existing action advising methods, especially since it is also very straightforward to implement and tune, with only a small computational burden. Finally, it was also seen that utilisation of such imitated advice policies to construct good quality exploration is non-trivial and requires further investigation.

This study lies at the intersection of action advising and exploration in RL and can be extended in various interesting ways. It is unclear how far the different qualities of imitation and reuse rates can affect performance in one particular game; it will be a worthwhile study to analyse these. Furthermore, evaluating the advice in terms of its contribution to learning progress is a promising direction to take as well.

Chapter 7

Learning on a Budget via Teacher Imitation

The contents of this final technical chapter are based on our publication titled "Learning on a Budget via Teacher Imitation" (Ilhan et al. 2021*b*). In this chapter, we provide answers to the following research questions:

- [RQ1] To what extent can we accelerate Deep RL via (budget-limited) action advising with one or more knowledgeable peer(s)?
- [RQ6] How can we use the advice memorisation techniques to build more efficient advice collection strategies?

7.1 Introduction

The previous studies we covered in earlier chapters demonstrated the remarkable ability of student-initiated action advising algorithms in speeding up Deep RL. While the majority of these focus on addressing *when to ask for advice* question, we have also investigated the ways to further utilise the collected advice by imitating and reusing the teacher policy in Chapter 6. Despite these developments, there are still several significant shortcomings present. These techniques often employ some threshold hyperparameters

to control the decisions to initiate advice exchange interactions which play a key role in their efficiencies. However, these parameters are sensitive to the learning state of the models as well as the domain properties. Therefore, they need to be tuned very carefully prior to execution, which involves unrealistically accessing the target tasks for trial runs. Furthermore, the studies to further leverage the teacher advice beyond collection are currently in their early stages and do not provide a complete solution to the problem besides addressing the advice reusing aspect.

In this work, we present an all-in-one student-initiated approach that is capable of collecting and reusing advice in a budget-efficient manner, by extending Ilhan et al. (2021*a*) (Chapter 6) in multiple ways. First, we propose a method for automatically determining the threshold parameters responsible for the decisions to request and reuse advice. This greatly alleviates the burden of task-specific hyperparameter tuning procedures. Secondly, we follow a decaying advice reuse schedule that is not tied to the student's exploration strategy. Finally, instead of using the imitated policy only for reusing advice as in Ilhan et al. (2021*a*) (Chapter 6), we incorporate this policy to determine and collect more diverse advice to construct a more universal imitation policy. The technical contributions we made in this work are as follows:

1. We propose an all-in-one student-initiated advice collection and utilisation algorithm for Deep RL.

2. We present a method to automatically tune the relevant hyperparameters of our algorithm, i.e. advice collection and advice reuse thresholds, on the fly.

3. We use an imitated teacher model to drive the completely student-initiated advice collection process.

Section 7.2 describes the problem formulation and our approach in detail. In Section 7.3, we explain the experimental setup. Then, the results and the related discussion are presented in Section 7.4. Finally, the study is concluded in Section 7.5 with some final remarks.

7.2 The Approach

We adopt the MDP formalisation presented in Chapter 2 in our problem definition. The setup in this study includes an off-policy Deep RL agent (student) with policy π_S learning to perform some task in an environment with continuous state space and discrete actions. There is also another agent with policy π_T (teacher) that is knowledgeable in this particular task. The teacher is isolated from the environment itself but is reachable by the student via a communication channel for a limited number of times defined by the advising budget b. By using this mechanism, the student can request action advice $a = \pi_T(s)$ for its current state s. The objective of the student in this problem is to maximise its learning performance in this task by timing these interactions to make the most efficient use of π_T.

Our approach provides a unified solution for addressing *when to ask for advice* and *how to leverage the advice* questions. In addition to the RL algorithm, the student is equipped with a neural network H_η with weights η that are not shared with the RL model in any way. The student also has a transitions buffer $\mathcal{C}$ with no capacity limit that holds the collected state-advice pairs. By using the samples in $\mathcal{C}$, H_η is trained periodically to provide the student an up-to-date imitation model of π_T to make it possible to reuse the previously provided advice. Moreover, H_η is also used to determine what advice to collect by being regarded as a representation of $\mathcal{C}$'s contents. Obviously, making these decisions require H_η to have a form of awareness of what it is trained on (in terms of samples). Therefore, H_η employs Dropout regularisation in the fully-connected layers to have an estimation of epistemic uncertainty denoted by $H^u_\eta(s)$ for any state s as it is done in Gal & Ghahramani (2016) and Ilhan et al. (2021*a*) (Chapter 6). None of these components shares anything with or requires access to the student's RL algorithm. This is especially advantageous when it comes to pairing up our approach with different RL methods.

At the beginning of the student's learning process, η is initialised randomly and $\mathcal{C} = \emptyset$. Then, at every timestep t in state s_t, the student goes through 3 stages of

our algorithm: Collection, Imitation, Reuse. The remainder of this section describes these stages with the line number references to the complete flow of our algorithm summarised in Algorithm 16[1].

The collection stage (lines 13-19 in Algorithm 16)[2] remains active from the beginning until the student runs out of its advising budget b_{ask}. At this step, the student attempts to collect advice if its current state has not been advised before. This is determined by the value of $H^u_\eta(s_t)$. If it is higher than the uncertainty threshold τ_{reuse} (which is set automatically in the imitation stage), it is decided that s_t has not been advised before; thus, the student proceeds with requesting advice. However, if τ_{reuse} is undetermined, this request is carried out without performing any uncertainty check.

The imitation module is responsible for training H_η and tuning τ_{reuse} accordingly. This stage (lines 20-25 in Algorithm 16)[3] is always active, but it is only triggered when these conditions that are checked at every timestep t are met: the student has collected $N_\mathcal{C}$ new samples in $\mathcal{C}$ (since the last imitation) or the student has taken $t_{imitation}$ steps (since the last imitation) with at least $N_\mathcal{C}/2$ new samples in $\mathcal{C}$. Here, $N_\mathcal{C}$ and $T_{imitation}$ are hyperparameters. These are set in order to keep the number of imitation processes within a reasonable number while also ensuring H_η remains up-to-date with the collected advice. On one hand, if H_η was updated for every new state-advice pair, it would be a very accurate model of $\mathcal{C}$'s contents, but the total training times would be a significant computational burden. On the other hand, if H_η was updated infrequently, it would not cause any computational setbacks; however, it would not be a good representation of the collected advice either.

Once the imitation is triggered, H_η is trained for K_{BC} iterations (if it is the first-ever training; else, for K_{BCP} iterations) with the minibatches of samples drawn randomly from $\mathcal{C}$. This process resembles the simplest form of behavioural cloning where the supervised negative log-likelihood loss $\mathcal{L}(\eta) = \sum_{\langle s,a\rangle\in\mathcal{C}} -\log H_\eta(a \mid s)$ is minimised.

[1]The code for our experiments can be found at`https://github.com/ercumentilhan/advice-imitation-reuse`
[2]The implementation can be found in lines 228-259 of `code/executor.py` file in the code repository.
[3]The implementation can be found in lines 264-303 of `code/executor.py` file in the code repository.

Algorithm 16 Learning on a Budget via Teacher Imitation

Input: Number of training iterations t_{max}, RL algorithm-related parameters, e.g. DQN, teacher policy π_T, action advising budget b_{ask}, initial reuse probability $\epsilon_{reuse-init}$, final reuse probability $\epsilon_{reuse-final}$, number of reuse probability decaying steps T_ϵ, number of imitation training iterations K_{BC} (initial) and K_{BCP} (periodic), number of new samples and steps to trigger imitation $N_{\mathcal{C}}$ and $T_{imitation}$.
Initialise RL algorithm-related variables, e.g. DQN
Initialise BC network H_η
Initialise empty BC buffer $\mathcal{C}$ to store state-action tuples
reuse_enabled $\leftarrow False$ ▷ Disable advice reuse by default
$\tau_{reuse} \leftarrow None$ ▷ Start with no valid reuse threshold
$\epsilon_{reuse} \leftarrow \epsilon_{reuse-init}$ ▷ Set reuse probability with the initial value
$n_{last} \leftarrow 0, t_{last} \leftarrow 0$ ▷ Set last imitation sample count and timestep as 0
for training steps $t \in \{1, 2, \dots t_{max}\}$ **do**
  Get state observation s_t from *Environment* if it is reset
  Set *reuse_enabled* $True$ with ϵ_{reuse} probability
  $a_t \leftarrow None$ ▷ Set action as non-determined
  ▷ Collection
  if *reuse_enabled* is $True$ **and** $b_{ask} > 0$ **then**
    if H_η is not trained **or** $H^u_\eta(s_t) > \tau_{reuse}$ **then**
      $a_t \leftarrow \pi_T(s_t)$ ▷ Obtain advice from the teacher
      $\mathcal{C} \leftarrow \mathcal{C} \cup \langle s_t, a_t \rangle$ ▷ Add the state-advice pair to the BC dataset
      $b_{ask} \leftarrow b_{ask} - 1$ ▷ Decrease the budget
    end if
  end if
  ▷ Imitation
  if $|\mathcal{C}| - n_{last} \geqslant N_{\mathcal{C}}$ **or**
  $(|\mathcal{C}| - n_{last} \geqslant N_{\mathcal{C}}/2$ **and** $t - t_{last} \geqslant T_{imitation})$ **then**
    Train H_η with $\mathcal{C}$ for K_{BC} (if its the first training) or K_{BCP} (otherwise) iterations
    $n_{last} \leftarrow |\mathcal{C}|, t_{last} \leftarrow t$
    Determine τ_{reuse} as described in Section 7.2 (Algorithm 17)
  end if
  ▷ Reuse
  if *reuse_enabled* is $True$ **and** a_t is $None$ **and**
  H_η is trained **and** $H^u_\eta(s_t) < \tau$ **then**
    $a_t \leftarrow \arg\max_a H_\eta(a \mid s_t)$ ▷ Generate imitated advice to reuse
  end if
  Decay ϵ_{reuse} w.r.t. pre-defined schedule over T_ϵ if $\epsilon_{reuse} > \epsilon_{reuse-final}$
  if a_t is $None$ **then**
    Determine a_t via the RL algorithm, e.g. DQN ▷ This is denoted as π_S
  end if
  Execute a_t and obtain r_{t+1}, s_{t+1} from *Environment*
  Update the RL algorithm, e.g. DQN
  $s_t \leftarrow s_{t+1}$
end for

Algorithm 17 Automatic Threshold Tuning

Input: BC buffer $\mathcal{C}$, BC model H_η, percentile cutoff proportion $\rho_{\mathcal{H}}$.
    Initialise empty set $\mathcal{H}$
for every $\langle s, a \rangle \in \mathcal{C}$ **do**
  if $a = \text{argmax}_{a'} H_\eta(a' \mid s)$ **then**
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{H^u_\eta(s)\}$ ▷ Add uncertainty value of state s to $\mathcal{H}$
  end if
end for
Sort elements of $\mathcal{H}$ in ascending order
$i \leftarrow \text{round}(\rho_{\mathcal{H}}|\mathcal{H}|)$ ▷ Find index that corresponds to $\rho_{\mathcal{H}}$th proportion
return $\mathcal{H}_i$

Afterwards, τ_{reuse} is updated automatically to be compatible with the new state of this imitation network. This is done by measuring $H^u_\eta(s)$ and storing them in a set $\mathcal{H}$ for each $\forall \langle s, a \rangle \in \mathcal{C}$ that satisfies $a = \text{argmax}_{a'} H_\eta(a' \mid s)$, in other words, considering the uncertainty value only for the correctly learned state-action pairs. Then, the uncertainty value that corresponds to the $\rho_{\mathcal{H}}$ proportion (hyperparameter) in the ascending-order sorted $\mathcal{H}$ is assigned to τ_{reuse}. We do this to pick a threshold τ_{reuse} such that H_η can consider these samples it classifies correctly as "known" while leaving a small portion that is likely to be outliers out when H^u_η is compared with τ_{reuse}. The process is shown in Algorithm 17. We believe that determining the $\rho_{\mathcal{H}}$ value to set does not hold much importance given that it is a reasonable number, e.g. 0.9, just to leave the outliers (the samples H_η has learnt correctly but still holds a high uncertainty value for). This approach could be further developed by also considering the true-positive and false-positive rates, however, we opted for a simpler approach in this study.

Finally, the reuse stage (lines 26-30 in Algorithm 16)[4] handles the execution of the imitated advice whenever appropriate, to aid the student in efficient exploration. It becomes active as soon as the imitation model H_η is trained for the first time. Then, whenever $H^u_\eta(s_t) < \tau_{reuse}$ (i.e. H_η is familiar with s_t), no advice collection is occurred at t and reusing is enabled for this particular episode, the student executes the imitated advice $\text{argmax}_a H_\eta(a \mid s_t)$. Unlike Chapter 6, we do not limit advice reusing to the exploration stage of learning, e.g. the period ϵ is annealed to its final value in ϵ-greedy.

[4]The implementation can be found in lines 307-333 of `code/executor.py` file in the code repository.

Instead, we define a reuse schedule that is independent of the underlying RL algorithm's exploration strategy. At the beginning of each episode, the agent either enables reuse module with a probability of ϵ_{reuse} (set as $\epsilon_{reuse-init}$ initially). This value is decayed until it reaches its final value $\epsilon_{reuse-final}$ over T_ϵ steps, similarly to ϵ-greedy annealing. This approach further eliminates the dependency of our algorithm on the RL algorithm's exploration strategy. Setting $\epsilon_{reuse-init}$ and $\epsilon_{reuse-final}$ is non-trivial and can be done in a similar way it is done for ϵ-greedy. We ideally want the agent to perform enough reusing to boost its exploration in an informed way to collect useful transitions at the beginning, but we also do not want the agent to rely on reusing in the later stages of learning to have its underlying RL model to be independent of it eventually. The annealing time-span T_ϵ needs to be set in a way that it gives the agent enough time to incorporate the knowledge the teacher imitation model has to offer. While this is a domain-dependent hyperparameter, setting it as half of the maximum learning timesteps can be a reasonable starting point.

7.3 Experimental Setup

We designed our experiments to answer the following questions about our proposal:

- How does our automatic threshold tuning perform against the manually-set ones in terms of reuse accuracy and learning performance?
- How much does using the advice imitation model to drive the advice collection process help with collecting a more diverse state-advice dataset?
- Does collecting a dataset with more diverse samples make any significant impact on the learning performance?
- How much does every particular modification contribute to the final performance?

In the remainder of this section, we first describe our evaluation domain. Then, we provide the details of our experimental process along with the substantial implementation

details.

We experiment with an extensive set of agents to be able to determine the most beneficial enhancements included in our algorithm. The student agent variants we compare in our experiments are as follows:

- **No Advising (NA):** No form of action advising is employed, the agent relies on its RL algorithm only.

- **Early Advising (EA):** The student asks for advice greedily until its budget runs out. There is no further utilisation of advice beyond their execution at the time of collection. This is a simple yet well-performing heuristic.

- **Random Advising (RA):** The student asks for advice randomly with 0.5 probability. This heuristic uses the intuition that spacing out requests may yield more diverse and information-rich advice.

- **EA+Advice Reuse (AR):** The agent employs our previously proposed advice reuse approach (Ilhan et al. 2021*a*) (Chapter 6). Advice is collected with early advising strategy, and the teacher is imitated via these advice. Then, advice are reused in place of the random exploration actions in approximately 0.5 of the episodes.

- **AR+Automatic Threshold Tuning (AR+A):** AR is combined with our automatic threshold tuning technique.

- **AR+A+Extended Reuse (AR+A+E):** AR is combined with both our automatic threshold tuning technique and the extended reusing scheme.

- **Advice Imitation & Reuse (AIR):** This agent mode incorporates all of our proposed enhancements (as detailed in Section 7.2). On top of AR+A+E, this mode also uses the imitation module's uncertainty to drive the advice collection process instead of relying on early advising.

We test the agents in learning sessions with a length of 5M steps (equals to 20M game frames due to frame skipping) with an advising budget of 25k that corresponds to only 0.5% of the total number of steps in a session. At every $50k^{th}$ step, the agents are evaluated in a separate set of 10 episodes by having their action advising and exploration mechanisms disabled. The cumulative rewards obtained in these episodes are averaged and recorded as evaluation scores for that corresponding learning session step. This lets us measure the actual learning progress of the agents as the main performance metric.

The Deep RL algorithm of the student agent is identical to the one from Chapter 6: Double DQN with a neural network structure comprised of 3 convolutional layers (32 8×8 filters with a stride of 4 followed by 64 4×4 filters with a stride of 2 followed by 64 3×3 filters with a stride of 1) and fully-connected layers with a single hidden layer (512 units) and dueling stream output which can be seen in Figure 6.1. We use the rectified linear unit (ReLU) as the activation function in every layer except for the final one. Weights are initialised with He initialisation (He et al. 2015) which was found to be working better with ReLU.

The imitation module consists of a convolution neural network that is similar to the DQN one and is identical to the one used in Chapter 6. There are 3 convolutional layers with 32 8×8 filters with a stride of 4, 64 4×4 filters with a stride of 2, 64 3×3 filters with a stride of 1, which are followed by a fully-connected layer with 512 hidden units and a final $|\mathcal{A}|$ sized Softmax output layer. The choice of layer activations is also ReLU accompanied with He initialisation for the weight initialisation. There are also Dropout layers in the fully-connected layers to enable uncertainty estimations. This network architecture is visualised in Figure 6.2.

For exploration, ϵ-greedy strategy with linearly decaying ϵ is adopted. The teacher agents are generated separately for each of the games prior to the experiments, by using the identical DQN algorithm and structure with the student. Even though the resulting agents are not necessarily at super-human levels achievable by DQN, they have competent policies that can achieve the evaluation scores of 1556, 28.8, 12, 3705, 8178 for Enduro, Freeway, Pong, Q*bert, Seaquest, respectively.

As we described in Section 7.2, our approach requires the student to be equipped with an additional behavioural cloning module that includes a neural network. We used the identical neural network structure to the student's DQN model except for the dueling streams. Fully-connected layers of this network are enhanced with Dropout regularisation with a dropout rate of 0.35 and the number of forward passes to measure the epistemic uncertainty via variance is set at 100.

The uncertainty threshold for AR is set as 0.01 for every game. Determining a reasonable value for this parameter requires accessing the tasks briefly, which we have performed prior to the experiments; even though this will not be reflected in the numerical results, it should be noted that this is a critical disadvantage of AR. The automatic threshold tuning proportion used in AR+A, AR+A+E, AIR is set as 0.9. This is a very straightforward hyperparameter to adjust compared to the (manual) uncertainty threshold itself and can potentially be valid in a wide variety of tasks. For the extensive reuse scheme in AR+A+E and AIR, we set $\epsilon_{reuse-init}$ and $\epsilon_{reuse-final}$ as 0.5 and 0.1, respectively. We defined the annealing schedule to begin at 500kth step and last until 2Mth step. For the imitation triggering conditions in AIR, $N_{\mathcal{C}}$ is set as 2.5k (samples) and $T_{imitation}$ is set as 50k (timesteps). Finally, the number of imitation network training iterations is set as 200k for the initial one (applies to all modes but NA, EA and RA) and 50k for the periodic ones (only applies to AIR).

Every experiment is repeated 3 times with different random number generation seeds in order to produce more meaningful aggregated results. All of the hyperparameters reported in this section are set empirically prior to the experiments and are kept the same across every game. The most significant ones among the unmentioned hyperparameters of the student's DQN and imitation components are presented in Table 7.1.

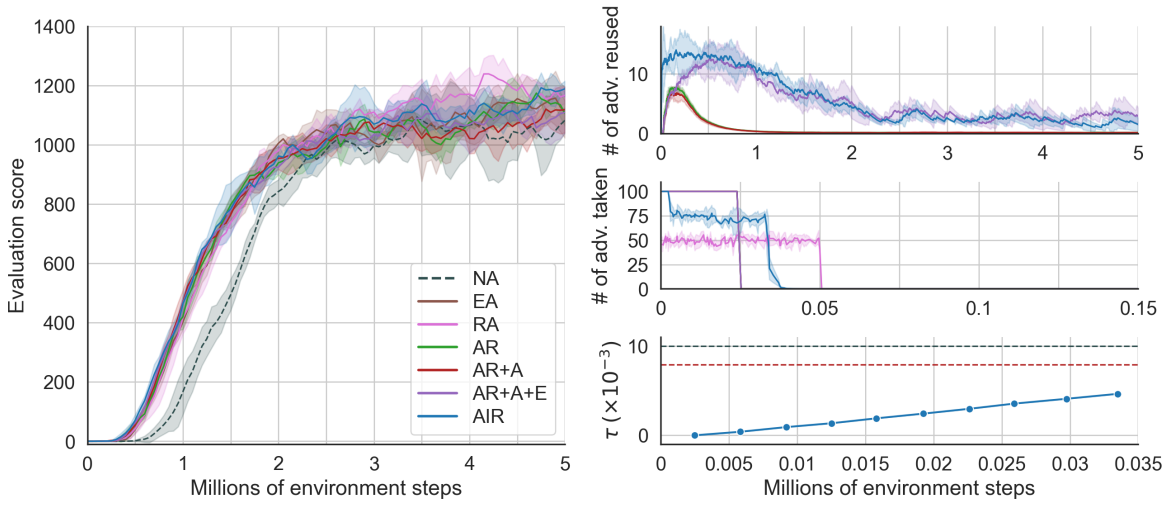
7.4 Results and Discussion

The results of our experiments in Enduro, Freeway, Pong, Q*bert and Seaquest are presented in several plots to let us analyse the performance of the student modes

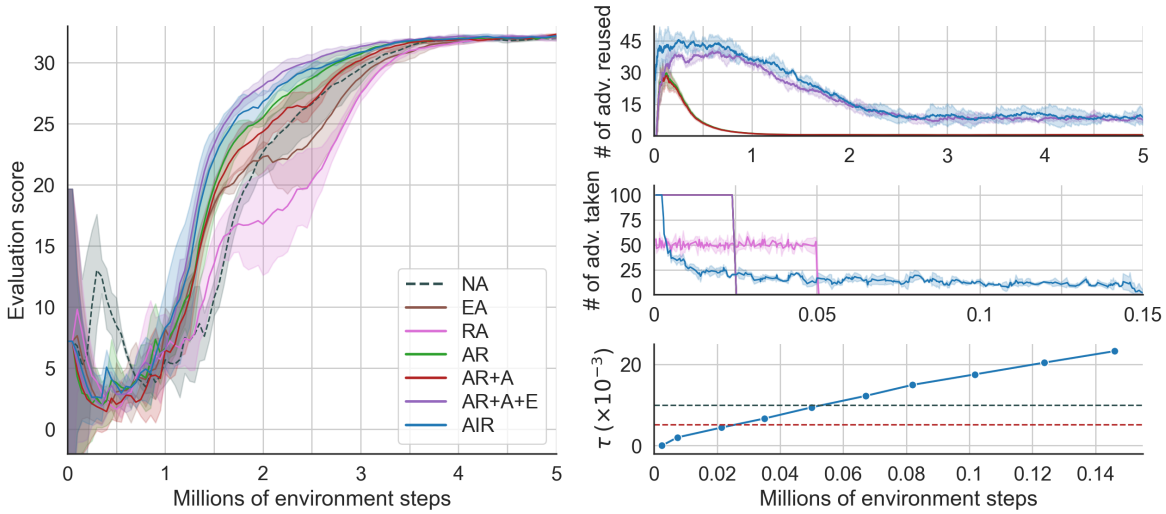
Table 7.1: Hyperparameters of the student's DQN (top section) and its imitation module for AR, AR+A, AR+A+E, AIR as well as the general action advising process (bottom section).

Hyperparameter name	Value
Replay memory size to start learning $M_{\mathcal{D}}$	50k
Replay memory capacity $N_{\mathcal{D}}$	500k
Target network update period (steps) T_{target}	7500
Train period (steps) T_{train}	4
Minibatch size	32
Learning rate α	625×10^{-7}
Discount factor γ	0.99
Adam epsilon	1.5×10^{-4}
Huber delta	1
ϵ initial	1.0
ϵ final	0.01
ϵ decay steps	500k
Buffer size to trigger BC training N_C	2.5k
Timesteps needed to trigger periodic BC training $T_{imitation}$	50k
Number of initial BC training iterations K_{BC}	200k
Number of initial BC training iterations K_{BCP}	50k
Minibatch size	32
Learning rate	0.0001
Dropout rate χ	0.35
Number of forward passes to assess uncertainty	100
Action-advising budget b_{ask}	25k
Proportion cutoff point $\rho_{\mathcal{H}}$	0.9
Initial advice-reusing probability $\epsilon_{reuse-init}$	0.5
Final advice-reusing probability $\epsilon_{reuse-final}$	0.1
Timesteps to anneal advice-reusing probability T_ϵ	1.5M (500k $\rightarrow$ 2M)

NA, EA, RA, AR, AR+A, AR+A+E, AIR in different aspects. Left-hand side of the Figures 7.1, 7.2 and 7.3 contains the evaluation scores plots. On the right-hand side of the Figures 7.1, 7.2 and 7.3, plots for the number of advice reuses per 100 steps (top row) performed by AR, AR+A, AR+A+E, AIR; plots for the number of advice collections per 100 steps performed by NA, RA, AR, AR+A, AR+A+E, AIR (middle row); plots for the values of the τ_{reuse} hyperparameter (uncertainty threshold) used by AR, AR+A, AR+A+E, AIR (bottom row) in a single run are shown. The shaded areas



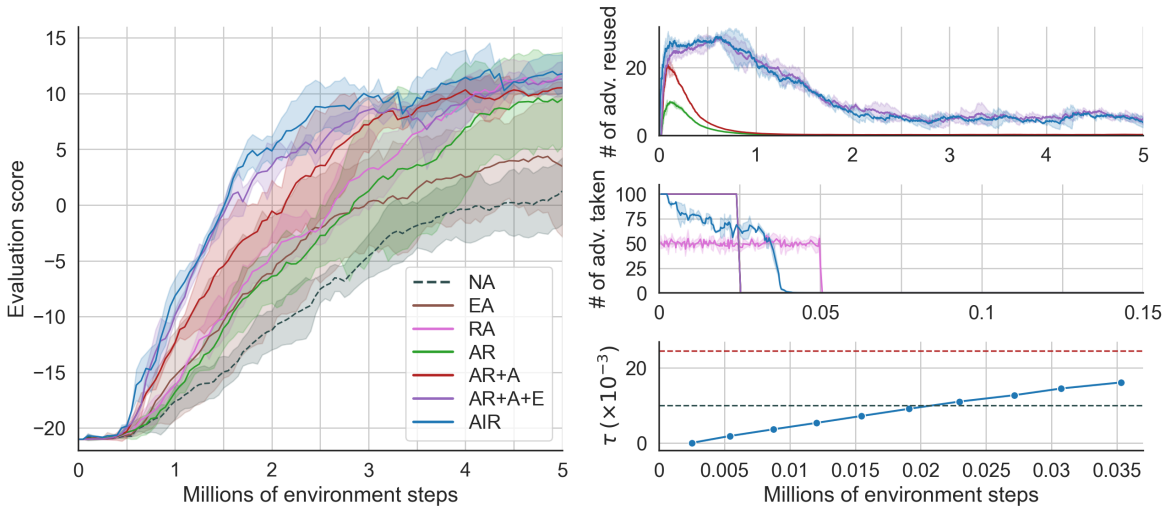
(a) Enduro



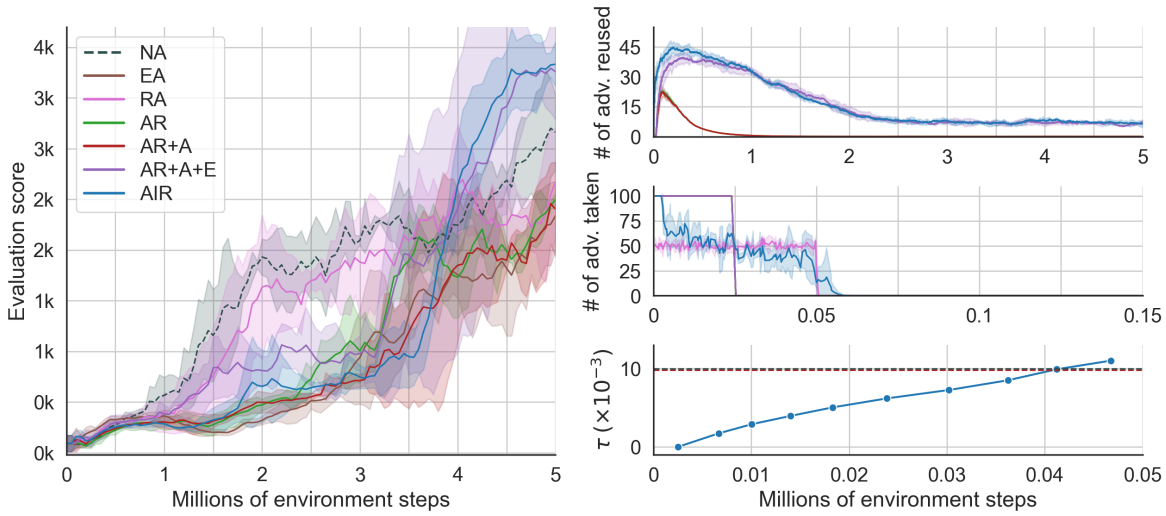
(b) Freeway

Figure 7.1: Evaluation scores of the student variants None, EA, AR (left) and the number of advice reuses performed by the student with AR mode plotted cumulatively (upper right) and in every 100 steps (lower right), obtained in the Atari games of Enduro (a) and Freeway (b), aggregated over 3 runs. Purple lines represent the number of all advice reuses while the green lines represent the number of correctly imitated ones among these. Shaded areas show the standard deviation across the runs.

in these plots show the standard deviation across 3 runs. The final evaluation scores, percentage of advice reuses in the total number of environment steps as well as their accuracies are presented in Table 7.2. Finally, in order to highlight the differences in



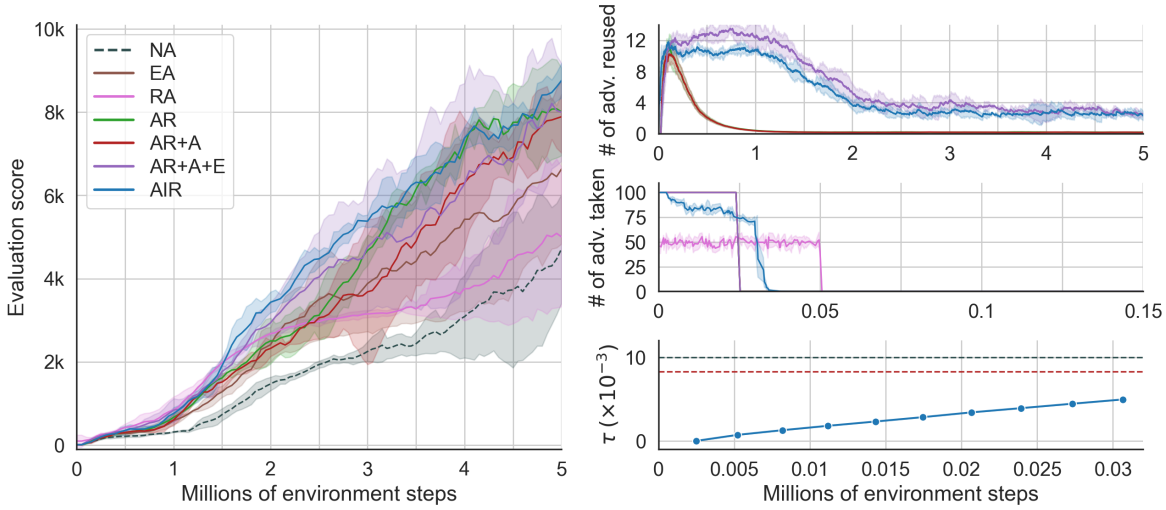
(a) Pong



(b) Q*bert

Figure 7.2: Evaluation scores of the student variants None, EA, AR (left) and the number of advice reuses performed by the student with AR mode plotted cumulatively (upper right) and in every 100 steps (lower right), obtained in the Atari games of Pong (a) and Q*bert (b), aggregated over 3 runs. Purple lines represent the number of all advice reuses while the green lines represent the number of correctly imitated ones among these. Shaded areas show the standard deviation across the runs.

the advice-collected state diversity of EA (identical collection strategy to AR, AR+A, AR+A+E), RA and AIR, these states from a single Pong run are visualised with the aid of Uniform Manifold Approximation and Projection (UMAP) (McInnes & Healy

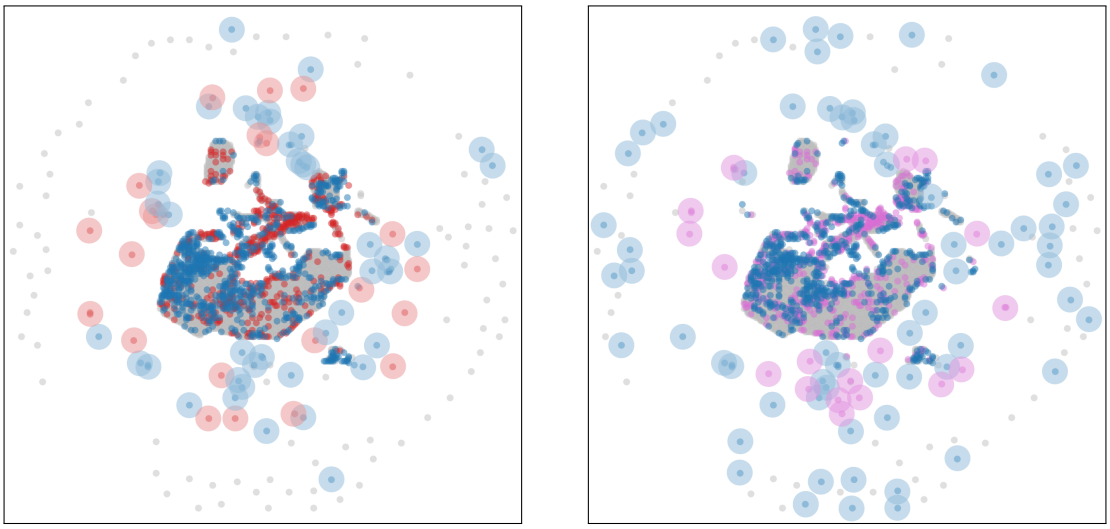


(a) Seaquest

Figure 7.3: Evaluation scores of the student variants None, EA, AR (left) and the number of advice reuses performed by the student with AR mode plotted cumulatively (upper right) and in every 100 steps (lower right), obtained in the Atari game of Seaquest (a), aggregated over 3 runs. Purple lines represent the number of all advice reuses while the green lines represent the number of correctly imitated ones among these. Shaded areas show the standard deviation across the runs.

2018) dimensionality reduction technique in Figure 7.4. Here, the scatter plot on the left compares AIR (blue) vs. EA (red) and the one on the right compares AIR (blue) vs. RA (pink), and the samples collected in common are shown in grey.

We first analyse the learning performances via evaluation scores. In Enduro, all methods but NA have a very similar learning speed and final scores, with AIR being slightly ahead of the rest. In Freeway, they all achieve nearly the same final scores, but they are distinguishable with small differences in learning speed where AR+A+E is on the top followed by other advanced student modes AIR, AR, AR+A. The initial peak followed by a drop that is apparent for every method is caused by the mechanics specific to this game. Specifically, repeatedly executing the action of moving upward always gets the agent to the reward position at the top. As the DQN policy executed during the evaluation stage is deterministic, it exhibits a consistent behaviour defined by the way it is initialised until it gets any model updates. In this particular case, it



(a) AIR vs. EA (b) AIR vs. RA

Figure 7.4: UMAP embeddings of the advice collected states in Seaquest by AIR (blue) vs. EA (red) in (a) and AIR (blue) vs. RA (pink) in (b), where the samples in common are shown in grey. Areas denoted with larger circles are the outliers covered only by either AIR (blue), EA (red) or RA (pink).

is seen to be executing the goal-leading action for every observation it gets until the model updates begin which causes the dip; then, as the learning progresses, a much better strategy is devised than this to yield higher results. The size of action space and similarity of state observations of this game make this situation very likely to be encountered. When we move to Pong, Q*bert and Seaquest, we finally see the student mode performances to be more distinctive. Even though the basic heuristics (RA and EA) show that a little number of advice from a competent policy can make a substantial boost in learning, these modes fall behind those that employ advice reuse and fail to be a reliable choice, i.e. performing worse than NA in Freeway and Q*bert. Another interesting result here is seen in the plots of Q*bert where NA performs significantly better than other approaches and even beats them all but AR+A+E and AIR at the final score. One reason behind this may be the fact that regardless of the complex visuals of ALE, Q*bert in reality has a very small number of states due to the limited number of positions the agent can navigate to. Linked to this, the number of advice required to learn the game is also much smaller than the given budget. As result, greedy

advice collection strategies like NA suffer from the negative effects of receiving too much advice, e.g. learning with transitions coming from a different fixed policy that results in bootstrapped error terms due to the unseen actions (one of the main problems tackled in Offline RL) concentrated on a limited set of states, unlike RA. Overall, the best mode AIR and the runner-up AR+A+E are clearly ahead of all, with AR and AR+A following them.

Among the advice reuse approaches, we see that the most beneficial modification is the extended reuse schedule (+E) as it is highlighted by the difference between AR+A and AR+A+E. Defining such a schedule independently from the student's RL exploration strategy involves using some extra hyperparameters, nevertheless, they are rather trivial to set arbitrarily. The trends in the advice reuse plots (Figures 7.1, 7.2, 7.3, right-hand side, top rows) show how these schedules differ. The versions with +E (AIR and AR+A+E) yield around 10× more reusing, which apparently plays an important role in performance improvement. However, it is still not clear how to define the optimal reuse schedules.

Automatic threshold tuning (+A) also performs comparably, if not better, with the manual tuning approach (AR) as we can observe in the evaluation scores. Additionally, AR+A managed to achieve very similar reuse accuracies to AR; this also supports its success. When we examine the τ_{reuse} values, we see AR+A determined values that are close to the hand-tuned ones, except for the case in Pong where the difference is more significant. This is reflected in reuse trend and evaluation performance, giving AR+A a very advantageous head start. These results support the idea that +A is a far more preferable approach considering how problematic it can be to tune the sensitive τ_{reuse} threshold manually. For instance, if they were to be deployed in some significantly different domains as they are, then we could potentially see AR+A coming far ahead of AR with a poorly tuned τ_{reuse}. Furthermore, the periodic imitation model updates incorporated in AIR make +A an essential component. In the right-hand side bottom rows of Figures 7.1, 7.2, 7.3, we also show how AIR changes its τ_{reuse} values over time as it collects more advice samples and updates its imitation model accordingly. Clearly,

it is very difficult to manage these changes manually.

Finally, we also see that collecting advice by utilising the imitation model's uncertainty (as it is done by AIR) contributes to the agent's learning. When we look at the advice collection plots, we see 3 different types of behaviours: early collection (EA, AR, AR+A, AR+A+E), random collection (RA), and AIR. Even though they seem to be occurring mostly in the same time windows, AIR does this in an uncertainty-aware fashion; hence the decreasing collection rate over time. Freeway is the case in which AIR is very selective. This is possibly due to the fact that in Freeway, the agent can traverse only a limited space which consequently reduces the diversity of the acquired observations. Nonetheless, this is not reflected in the evaluation scores as dramatically due to this game being rather trivial to solve. Another interesting observation is made by analysing the advice collected states in Seaquest by AIR, EA (which is identical to AR, AR+A, AR+A+E in terms of collection strategy), RA in a reduced dimensionality as seen in Figure 7.4. We chose Seaquest since it is the game where AIR is significantly ahead of AR+A+E, which can be credited to AIR's only difference from it (collection strategy). We also include RA here mainly because it can potentially do better in acquiring different samples than EA. Here, the large circles denote the outliers (diverse samples) that are only covered only by either AIR, EA or RA. These are the important bits to pay attention to and compare. As it can be seen, AIR yields larger coverage, i.e. more diverse dataset of advice, in both cases against EA and RA collection strategies.

The sample (advice) efficiency our modifications provide comes with a computational cost trade-off. Therefore, in addition to the empirical agent performance analyses, we also measured the wall time it took different modifications to complete an experiment on identical machines. We have seen that the first modification AR takes 1.01 to 1.15 times (depending on the game) longer on average than NA (which is a default DQN agent). This is very reasonable and is also aligned with the computational cost analysis we did for this particular modification in our previous study (Section 6.3). Our second modification AR+A also achieved very similar times as expected since the only extra step it has on top of AR is the automatic threshold tuning. When we extend the advice

reuse beyond exploration steps (AR+A+E) the experiments took 1.18 to 1.35 times longer on average compared to NA. This significant slow-down comes from executing the uncertainty estimator over whole training sessions. AIR similarly took 1.2 to 1.4 times longer for the experiments, which is due to the periodic model training burden it comes with. Overall, the computational costs of adopting our techniques seem to be very reasonable and acceptable considering the sample efficiency and agent performance boost they bring.

7.5 Conclusions

In this study, we proposed an automatic threshold tuning technique, an extended advice reusing schedule and an imitation model uncertainty-based advice collection procedure by extending the previously proposed advice reusing algorithm. We also developed a combined approach by incorporating these components, which is able to collect a diverse set of advice to build a more widely applicable advice imitation model for advice reuse.

The experiments in 5 different Atari games from the ALE domain have shown that our enhancements provide significant improvements over the baseline advice reuse method as well as the basic action advising heuristics. First, being able to tune the uncertainty thresholds on-the-fly was observed to yield the learning performance of the carefully tuned threshold, which require unrealistic access to the tasks and extra effort to be adjusted. Secondly, we found that having the advice reusing process span across a larger portion of the learning session rather than just the steps that involve random exploration can yield superior performance. However, defining the best schedule for the maximum advice utilisation efficiency remains to be an open question. Thirdly, the uncertainty-driven advice collection method was found to be a successful way to improve the imitation module's dataset diversity. Nevertheless, the periodic training process can be improved with better incremental learning techniques to make better use of this simultaneous collection-imitation idea. Finally, our unified algorithm demonstrated

top-level performance across 5 Atari games by performing either on par or better than its closest competitors.

The future extensions of this work can involve experimenting with more principled approaches in the literature to further improve the advice reuse strategy beyond this randomly executed version. Furthermore, it will be a worthwhile study to make the teacher imitation better at learning online from the new samples it acquires. Finally, even though it is in the core motivations of our approach not to access and modify the agent's RL components, it will be beneficial to investigate LfD techniques and their potential contributions to our framework.

Table 7.2: Final evaluation scores, percentages of advice reuses in the total environment steps, percentages of advice reuse accuracies achieved by NA, EA, RA, AR, AR+A, AR+A+E, AIR (+ signs are omitted) in 5 ALE games aggregated over 3 runs. The standard deviations across runs are indicated with ±. The best scores/accuracies are denoted in bold.

		Evaluation Score	Advice Reuse	
Game	Mode	Final	Ratio (%)	Accuracy (%)
Enduro	NA	1066.34 ± 37.3	—	—
	EA	1131.13 ± 69.0	—	—
	RA	1170.60 ± 19.0	—	—
	AR	1127.72 ± 26.9	0.49 ± 0.03	70.06 ± 0.6
	ARA	1117.93 ± 59.0	0.45 ± 0.04	72.36 ± 0.3
	ARAE	1102.44 ± 62.4	4.97 ± 0.44	$\mathbf{75.24 \pm 0.5}$
	AIR	$\mathbf{1184.02 \pm 19.6}$	4.79 ± 0.44	73.63 ± 0.2
Freeway	NA	31.98 ± 0.1	—	—
	EA	32.09 ± 0.1	—	—
	RA	32.02 ± 0.2	—	—
	AR	32.06 ± 0.1	1.60 ± 0.11	93.02 ± 0.3
	ARA	$\mathbf{32.26 \pm 0.1}$	1.53 ± 0.10	93.48 ± 0.1
	ARAE	32.14 ± 0.0	15.76 ± 0.08	$\mathbf{95.43 \pm 0.1}$
	AIR	32.14 ± 0.1	17.89 ± 0.47	95.39 ± 0.2
Pong	NA	0.95 ± 2.4	—	—
	EA	3.73 ± 4.9	—	—
	RA	11.48 ± 0.2	—	—
	AR	9.41 ± 3.5	0.52 ± 0.02	79.80 ± 0.5
	ARA	10.48 ± 0.4	0.93 ± 0.01	75.07 ± 0.7
	ARAE	11.21 ± 1.2	10.4 ± 0.59	79.00 ± 0.1
	AIR	$\mathbf{11.81 \pm 1.2}$	9.98 ± 0.36	$\mathbf{80.46 \pm 0.8}$
Q*bert	NA	3154.91 ± 408.9	—	—
	EA	2277.98 ± 300.5	—	—
	RA	2528.70 ± 505.4	—	—
	AR	2434.70 ± 54.8	0.93 ± 0.04	80.37 ± 0.6
	ARA	2359.39 ± 371.9	0.93 ± 0.02	78.86 ± 0.7
	ARAE	3763.72 ± 340.3	14.57 ± 0.25	$\mathbf{92.87 \pm 0.7}$
	AIR	$\mathbf{3814.34 \pm 134.6}$	15.16 ± 0.21	92.83 ± 0.2
Seaquest	NA	4496.41 ± 1101.0	—	—
	EA	6538.18 ± 1445.1	—	—
	RA	5033.04 ± 1413.3	—	—
	AR	8053.93 ± 935.9	0.61 ± 0.03	72.96 ± 0.8
	ARA	7851.03 ± 556.7	0.56 ± 0.03	$\mathbf{76.29 \pm 0.7}$
	ARAE	8082.69 ± 1105.2	5.93 ± 0.49	74.18 ± 1.5
	AIR	$\mathbf{8614.04 \pm 268.7}$	4.79 ± 0.20	74.87 ± 0.5

Chapter 8

Conclusions and Future Work

This thesis presented a set of action advising approaches to accelerate Deep RL through leveraging the knowledge available within the peers accessible by the learning agent. Our methods tackled various Deep RL problem variants and different shortcomings of the action advising framework itself by addressing the following questions:

[RQ1] To what extent can we accelerate Deep RL via (budget-limited) action advising with one or more knowledgeable peer(s)?

[RQ2] How can we scale the state-of-the-art action advising approaches from classical RL to Deep RL/MARL domains?

[RQ3] What can be an efficient heuristic to perform student-initiated action advising that is also robust to teacher absence conditions in Deep RL?

[RQ4] Can imprecise usage of action advising budget hamper Deep RL performance?

[RQ5] How can we further utilise the collected advice by memorising and reusing them in Deep RL domains?

[RQ6] How can we use the advice memorisation techniques to build more efficient advice collection strategies?

In Chapter 4, we dealt with [RQ1] and [RQ2]. We demonstrated the first-ever application of action advising in Deep RL/MARL domain. Specifically, we made multiple simultaneously learning agents perform jointly-initiated peer-to-peer knowledge

exchange in the form of actions with no predefined teacher-student roles in the process. To do so, the Deep RL agents employ an additional RND module to be able to assess the novelty of the states they encounter. This way, they can decide when it is appropriate to take the teacher role to give advice or when they are not knowledgeable about a state and need advice for it by taking the student role. RND technique helps with generalisation between similar and unseen states which is mandatory for Deep RL where typical state counting techniques are rendered useless. The results we obtained in this study showed that action advising can indeed be a promising technique to accelerate Deep RL and motivated us to study them further. We have also seen that RND can be an efficient proxy for the tabular state counters and can be used as an action advising heuristic.

In Chapter 5, we addressed the questions [RQ1], [RQ3] and [RQ4]. To do so, we shifted our attention to single-agent Deep RL setting to develop useful action advising strategies in less complicated learning dynamics that can later be extended for full-scale multi-agent domains. We proposed a completely student-initiated action advising algorithm that is robust to the situations of teacher absence and does not require specific Deep RL models that enable uncertainty estimations. Similarly to Chapter 4, we incorporate RND to measure the state novelty. But this time, we update RND only for the teacher-advised states and call it advice novelty to emphasise the distinction. This key difference ensures that the student will consider the teacher's advice regardless of the training state it is in. Our experiments in a grid-like environment and MinAtar games showed that our approach performs on par with the state-of-the-art and demonstrates significant advantages in scenarios where the existing methods are prone to fail.

In Chapter 6, we aimed to answer [RQ1] and [RQ5]. We investigated a different aspect in the action advising framework, which is further utilisation of the advice beyond their collecting. We propose memorising and reusing the collected advice in order to make more efficient use of the limited budget by spending it in dissimilar states to get advice. Furthermore, the student gets to repeat previous advice later in the training to

learn more efficiently. To accomplish these in Deep RL domains, we use BC to imitate the teacher's advice decision by training a model with the state-advice samples. We also employ dropout regularisation in the BC model to give it a notion of uncertainty, so the student is able to tell which of the states have actually received advice via the BC model's uncertainty. The experiments we conducted in the continuous state-space ALE domain made it evident that our approach is can boost the learning performance even when it is paired with the simplest advice collection method, e.g., Early Advising.

In Chapter 7, the main questions in target are [RQ1] and [RQ6]. We built upon our previous advice imitation study (Chapter 6) and proposed our final unified student-initiated action advising algorithm for advice collection and utilisation. Specifically, we make the student agent imitate the teacher's advice by using BC. Additionally, the student also uses this imitation model to drive its advice collection decisions differently from Chapter 6. Thus, the budget is spent on more diverse samples to construct a more efficient advice imitation model which contributes to much better learning performance overall. Moreover, we also alleviated another critical setback in the previous methods about having critical yet difficult-to-tune hyperparameters. By using the state-advice sample buffer we make the agent self-adjust its relevant hyperparameters on the fly. This grants a huge advantage when it comes to applying these techniques in real-world tasks where it is usually not realistic to pre-tune the algorithm for the task. The experiment results we obtained in ALE verified that our modifications bring significant improvements to the learning performance and our final approach achieves top-level performance in student-initiated action advising for Deep RL.

Overall, we have developed multiple action advising techniques suitable for the Deep RL domains either as standalone solutions or as add-ons to enhance the existing action advising algorithms. We have seen that action advising can be a promising framework to accelerate Deep RL through leveraging knowledge present within different peers. Our findings made it evident that state novelty and advice novelty-based advice collection strategies can be strong alternatives to the uncertainty-based ones, especially when the student agent's model has no such capability of estimating uncertainty. That being said,

further advice utilisation by memorisation and reuse was the key enhancement that made the actual performance breakthrough. Therefore, we built our final algorithm around this idea, which significantly outperformed the simple yet powerful baselines of EA and RA. Even though the previously investigated advice collection techniques (UA, SNA, ANA) appeared to be promising, they were not experimented with in the final chapters of the study since they are only about handling the advice collection which plays a little role in the learning performance; not to mention that they also require to be tuned carefully prior to training due to lacking the automatic threshold tuning functionality present in our final algorithm. Nevertheless, it may still be insightful to analyse them in the more complex domains like ALE as well as have them modified with advice imitation and reuse. Besides progressively creating better action advising methods, we observed that it is critical to not overuse an external policy, e.g., the teacher, even when the Deep RL method is off-policy; this aligns with the findings and the core motivation of Offline RL that claims the agent must ensure to be some state-action combinations itself to avoid extrapolation of errors. Finally, it is far from trivial to devise a principled idea to time the advice execution moments (at the time of collection or by reusing) to maximise the learning performance without taking advantage of multiple learning sessions, e.g., Meta RL. Therefore, linking this work with the exploration in RL literature is pivotal to discovering useful approaches. We believe that the contributions made in this Ph.D. study will enable more applications of action advising to be made in Deep RL, which will eventually be extended to solve practical problems in real-world scenarios.

8.1 Future Work

This section describes some of the future work we have identified to be significant extensions to our study.

8.1.1 Action Advising for Safe Exploration

We have shown that exchanging advice can significantly accelerate Deep RL. Additionally, action advising can be even more valuable in some scenarios where other knowledge reuse paradigms, e.g. LfD, Imitation Learning, are rendered inapplicable. A good example of this is the RL exploration processes with safety requirements, which are tackled in Safe RL line of research (Garcia & Fernández 2015). In these cases, having a teacher to be accessible at the time of learning (as in action advising) is the only truly safe option. Thus, applying the action advising techniques in the Safe RL problem domain can make a good opportunity to showcase their practical relevance.

When we apply Deep RL in real-world scenarios, the setback of needing a vast number of samples to learn competent policies worsens due to the fact that most of these cases also involve costly and catastrophic interactions. Even though the agents can learn to avoid these situations via trial-and-error learning, they do so only after actually experiencing these many times, which is obviously not affordable. Therefore, achieving safer exploration is considered to be of the utmost importance in these cases. A promising way to minimise the number of unsafe transitions throughout the learning can be to employ a safety-aware teacher, be it a human or another agent, to supervise the agent's exploration process. This teacher can distinguish the critical moments to be able to interfere with the learning agent's exploration whenever it is necessary. Despite ensuring safety, such approaches require the teacher's presence for very long periods of time (Saunders et al. 2018, Goecks et al. 2019), which may be infeasible in some domains where the teacher's attention span or communication budget is limited to cover the exhaustive exploration process.

We believe that an action advising algorithm to minimise the number of unsafe interactions throughout the learning by leveraging a safety-aware yet incompetent teacher with limited availability can be a promising idea to study. Specifically, imitating the teacher's decisions with multiple models to assess the safety of the states and to execute safe actions whenever it is appropriate could be the goal. This way, the need for

the teacher itself can be substantially decreased at the expense of only a small number of costly interactions. Having a way to control this trade-off can be especially valuable in the domains where the cost of accessing a teacher is also one of the concerns.

8.1.2 Learning from Incompetent Peers

The potentially best teachers are obviously those with the most amount of relevant knowledge due to the fact that this lets the student benefit in a multitude of situations. Nonetheless, peers with only a little expertise in the tasks may also possess some useful knowledge to be leveraged. For instance, the teacher may have knowledge only about a little portion of the domain or be incompetent yet have a better policy than a random policy that can be useful to augment the student's exploration process early on.

The significance of this idea has already been acknowledged by the research communities focusing on different learning from past knowledge paradigms such as Imitation Learning (Wu et al. 2019) and LfD (Gao et al. 2018, Brown et al. 2019, Zhu, Lin, Dai & Zhou 2020). Nevertheless, it is still to be studied in the area of action advising in the Deep RL domain to devise methods for the student agents to identify the states and time periods the teacher's policy is worth learning from. Due to the aforementioned motivations and gaps in the research, we believe that investigating the concept of learning from incompetent teachers in Deep RL domains via action advising will be a very valuable development.

8.1.3 Learning from Multiple Peers

Scaled-up decision-making systems with a network of multiple agents in them can offer the student agents a wide range of peers to learn from. These peers may happen to be learning different aspects of the domain and the tasks, which causes them to offer a rich set of knowledge. Due to this, the importance of being able to take advantage of these opportunities can not be ignored.

To this date, learning from multiple agents via action advising is investigated in

some other studies (da Silva et al. 2017, Omidshafiei et al. 2018, Kim et al. 2020) as well as in our approach presented in Chapter 4. However, these studies kept the number of agents very limited and incorporated a simple majority voting approach to fuse the collected advice. Furthermore, the concept of teacher peer selection by the student was not considered in any of them. Clearly, there is lots of room for improvement in this aspect of action advising. Such an ability not only would make the student to use its communication budget more efficiently, but also would help it construct a curriculum to learn more efficiently. A similar idea has already been studied for DQfD (Seita et al. 2019) and Policy Reuse (Kurenkov et al. 2019) with promising results. For these reasons, it will surely be a worthwhile direction to explore the scenarios with multiple teachers for action advising in Deep RL.

Bibliography

Allis, L. V. et al. (1994), *Searching for solutions in games and artificial intelligence*, Rijksuniversiteit Limburg.

Amir, O., Kamar, E., Kolobov, A. & Grosz, B. J. (2016), Interactive teaching strategies for agent training, *in* S. Kambhampati, ed., 'Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016', IJCAI/AAAI Press, pp. 804–811.

Arulkumaran, K., Deisenroth, M. P., Brundage, M. & Bharath, A. A. (2017), 'A brief survey of deep reinforcement learning', *CoRR* **abs/1708.05866**.

Bansal, T., Pachocki, J., Sidor, S., Sutskever, I. & Mordatch, I. (2017), 'Emergent complexity via multi-agent competition', *CoRR* **abs/1710.03748**.

Bellemare, M., Candido, S., Castro, P., Gong, J., Machado, M., Moitra, S., Ponda, S. & Wang, Z. (2020), 'Autonomous navigation of stratospheric balloons using reinforcement learning', *Nature* **588**, 77–82.

Bellemare, M. G., Naddaf, Y., Veness, J. & Bowling, M. (2013), 'The arcade learning environment: An evaluation platform for general agents', *J. Artif. Intell. Res.* **47**, 253–279.

Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J.,

Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F. & Zhang, S. (2019), 'Dota 2 with large scale deep reinforcement learning', *CoRR* **abs/1912.06680**.

Bloembergen, D., Tuyls, K., Hennes, D. & Kaisers, M. (2015), 'Evolutionary dynamics of multi-agent learning: A survey', *J. Artif. Intell. Res.* **53**, 659–697.

Boutsioukis, G., Partalas, I. & Vlahavas, I. P. (2011), Transfer learning in multi-agent reinforcement learning domains, *in* 'Recent Advances in Reinforcement Learning - 9th European Workshop, EWRL 2011, Athens, Greece, September 9-11, 2011, Revised Selected Papers', pp. 249–260.

Brown, D. S., Goo, W., Nagarajan, P. & Niekum, S. (2019), Extrapolating beyond suboptimal demonstrations via inverse reinforcement learning from observations, *in* K. Chaudhuri & R. Salakhutdinov, eds, 'Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA', Vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 783–792.

Burda, Y., Edwards, H., Storkey, A. J. & Klimov, O. (2018), 'Exploration by random network distillation', *CoRR* **abs/1810.12894**.

Busoniu, L., Babuska, R. & Schutter, B. D. (2008), 'A comprehensive survey of multiagent reinforcement learning', *IEEE Trans. Systems, Man, and Cybernetics, Part C* **38**(2), 156–172.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S. & Bellemare, M. G. (2018), 'Dopamine: A research framework for deep reinforcement learning', *CoRR* **abs/1812.06110**.

Chen, S., Tangkaratt, V., Lin, H. & Sugiyama, M. (2018), 'Active deep q-learning with demonstration', *CoRR* **abs/1812.02632**.

Clouse, J. A. (1996), *On Integrating Apprentice Learning and Reinforcement Learning*, University of Massachusetts Amherst.

Da Silva, F. L. & Costa, A. H. R. (2019), 'A survey on transfer learning for multiagent reinforcement learning systems', *Journal of Artificial Intelligence Research* **64**, 645–703.

da Silva, F. L., Glatt, R. & Costa, A. H. R. (2017), Simultaneously learning and advising in multiagent reinforcement learning, *in* 'Proceedings of the 16th Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017', ACM, pp. 1100–1108.

da Silva, F. L., Hernandez-Leal, P., Kartal, B. & Taylor, M. E. (2020), Uncertainty-aware action advising for deep reinforcement learning agents, *in* 'The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020', AAAI Press, pp. 5792–5799.

da Silva, F. L., Taylor, M. E. & Costa, A. H. R. (2018), Autonomously reusing knowledge in multiagent reinforcement learning, *in* 'Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden.', pp. 5487–5493.

da Silva, F. L., Warnell, G., Costa, A. H. R. & Stone, P. (2020), 'Agents teaching agents: a survey on inter-agent transfer learning', *Auton. Agents Multi Agent Syst.* **34**(1), 9.

De Bruin, T., Kober, J., Tuyls, K. & Babuška, R. (2015), The importance of experience replay database composition in deep reinforcement learning, *in* 'Deep reinforcement learning workshop, NIPS'.

Dubey, R., Agrawal, P., Pathak, D., Griffiths, T. & Efros, A. A. (2018), Investigating human priors for playing video games, *in* 'Proceedings of the 35th International

Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018', pp. 1348–1356.

Fachantidis, A., Taylor, M. E. & Vlahavas, I. P. (2019), 'Learning to teach reinforcement learning agents', *Machine Learning and Knowledge Extraction* **1**(1), 21–42.

Finn, C., Abbeel, P. & Levine, S. (2017), Model-agnostic meta-learning for fast adaptation of deep networks, *in* D. Precup & Y. W. Teh, eds, 'Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017', Vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 1126–1135.

Foerster, J., Assael, I. A., de Freitas, N. & Whiteson, S. (2016), Learning to communicate with deep multi-agent reinforcement learning, *in* 'Neural Information Processing Systems', pp. 2137–2145.

Foerster, J. N., Farquhar, G., Afouras, T., Nardelli, N. & Whiteson, S. (2017), 'Counterfactual multi-agent policy gradients', *CoRR* **abs/1705.08926**.

Foerster, J. N., Nardelli, N., Farquhar, G., Torr, P. H. S., Kohli, P. & Whiteson, S. (2017), 'Stabilising experience replay for deep multi-agent reinforcement learning', *CoRR* **abs/1702.08887**.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Hessel, M., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C. & Legg, S. (2018), Noisy networks for exploration, *in* '6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings', OpenReview.net.

Fujimoto, S., Meger, D. & Precup, D. (2019), Off-policy deep reinforcement learning without exploration, *in* K. Chaudhuri & R. Salakhutdinov, eds, 'Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019,

Long Beach, California, USA', Vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 2052–2062.

Fulda, N. & Ventura, D. (2007), Predicting and preventing coordination problems in cooperative q-learning systems., *in* 'IJCAI', Vol. 2007, pp. 780–785.

Gal, Y. & Ghahramani, Z. (2016), Dropout as a bayesian approximation: Representing model uncertainty in deep learning, *in* M. Balcan & K. Q. Weinberger, eds, 'Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016', Vol. 48 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 1050–1059.

Gao, Y., Xu, H., Lin, J., Yu, F., Levine, S. & Darrell, T. (2018), Reinforcement learning from imperfect demonstrations, *in* '6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings', OpenReview.net.

Garcia, J. & Fernández, F. (2015), 'A comprehensive survey on safe reinforcement learning', *J. Mach. Learn. Res.* **16**, 1437–1480.

Goecks, V. G., Gremillion, G. M., Lawhern, V. J., Valasek, J. & Waytowich, N. R. (2019), Efficiently combining human demonstrations and interventions for safe training of autonomous systems in real-time, *in* 'The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019', AAAI Press, pp. 2462–2470.

Goodfellow, I. J., Bengio, Y. & Courville, A. C. (2016), *Deep Learning*, Adaptive Computation and Machine Learning, MIT Press.

Hafner, D., Lillicrap, T. P., Ba, J. & Norouzi, M. (2020), Dream to control: Learn-

ing behaviors by latent imagination, *in* '8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020'.

Hausknecht, M., Mupparaju, P., Subramanian, S., Kalyanakrishnan, S. & Stone, P. (2016), Half field offense: An environment for multiagent learning and ad hoc teamwork, *in* 'AAMAS Adaptive Learning Agents (ALA) Workshop'.

He, K., Zhang, X., Ren, S. & Sun, J. (2015), Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *in* '2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015', IEEE Computer Society, pp. 1026–1034.

Hernandez-Leal, P., Kaisers, M., Baarslag, T. & de Cote, E. M. (2017), 'A survey of learning in multiagent environments: Dealing with non-stationarity', *CoRR* **abs/1707.09183**.

Hernandez-Leal, P., Kartal, B. & Taylor, M. E. (2018), 'Is multiagent deep reinforcement learning the answer or the question? A brief survey', *CoRR* **abs/1810.05587**.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. G. & Silver, D. (2018), Rainbow: Combining improvements in deep reinforcement learning, *in* S. A. McIlraith & K. Q. Weinberger, eds, 'Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018', AAAI Press, pp. 3215–3222.

Hester, T., Vecerík, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., Dulac-Arnold, G., Agapiou, J. P., Leibo, J. Z. & Gruslys, A. (2018), Deep q-learning from demonstrations, *in* S. A. McIlraith & K. Q. Weinberger, eds, 'Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence

(IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018', AAAI Press, pp. 3223–3230.

Ho, J. & Ermon, S. (2016), Generative adversarial imitation learning, *in* D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon & R. Garnett, eds, 'Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain', pp. 4565–4573.

Ilhan, E., Gow, J. & Perez, D. (2022), 'Student-initiated action advising via advice novelty', *IEEE Transactions on Games* **14**(3), 522–532.

Ilhan, E., Gow, J. & Pérez-Liébana, D. (2019), Teaching on a budget in multi-agent deep reinforcement learning, *in* 'IEEE Conference on Games, CoG 2019, London, United Kingdom, August 20-23, 2019', IEEE, pp. 1–8.

Ilhan, E., Gow, J. & Perez-Liebana, D. (2021*a*), Action advising with advice imitation in deep reinforcement learning, *in* 'Proceedings of the 20th Conference on Autonomous Agents and Multi-Agent Systems, AAMAS 2021, May 3-7, 2021', IFAAMAS, pp. 1100–1108.

Ilhan, E., Gow, J. & Pérez-Liébana, D. (2021*b*), Learning on a budget via teacher imitation, *in* '2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, August 17-20, 2021', IEEE, pp. 1–8.

Kim, D., Liu, M., Omidshafiei, S., Lopez-Cot, S., Riemer, M., Habibi, G., Tesauro, G., Mourad, S., Campbell, M. & How, J. P. (2020), Learning hierarchical teaching policies for cooperative agents, *in* 'Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020', International Foundation for Autonomous Agents and Multiagent Systems, pp. 620–628.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2012), Imagenet classification with deep convolutional neural networks, *in* P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou & K. Q. Weinberger, eds, 'Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States', pp. 1106–1114.

Kumar, A., Fu, J., Soh, M., Tucker, G. & Levine, S. (2019), Stabilizing off-policy q-learning via bootstrapping error reduction, *in* H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox & R. Garnett, eds, 'Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada', pp. 11761–11771.

Kurenkov, A., Mandlekar, A., Martin, R. M., Savarese, S. & Garg, A. (2019), AC-Teach: A bayesian actor-critic method for policy learning with an ensemble of suboptimal teachers, *in* L. P. Kaelbling, D. Kragic & K. Sugiura, eds, '3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings', Vol. 100 of *Proceedings of Machine Learning Research*, PMLR, pp. 717–734.

Leibo, J. Z., Zambaldi, V. F., Lanctot, M., Marecki, J. & Graepel, T. (2017), 'Multi-agent reinforcement learning in sequential social dilemmas', *CoRR* **abs/1702.03037**.

Levine, S., Pastor, P., Krizhevsky, A., Ibarz, J. & Quillen, D. (2018), 'Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection', *Int. J. Robotics Res.* **37**(4-5), 421–436.

Lipton, Z. C., Li, X., Gao, J., Li, L., Ahmed, F. & Deng, L. (2018), BBQ-Networks: Efficient exploration in deep reinforcement learning for task-oriented dialogue systems, *in* S. A. McIlraith & K. Q. Weinberger, eds, 'Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications

of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018', AAAI Press, pp. 5237–5244.

Littman, M. L. (1994), Markov games as a framework for multi-agent reinforcement learning, *in* 'Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994', pp. 157–163.

Liu, Z., Li, X., Kang, B. & Darrell, T. (2021), Regularization matters in policy optimization - an empirical study on continuous control, *in* '9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021'.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P. & Mordatch, I. (2017), Multi-agent actor-critic for mixed cooperative-competitive environments, *in* 'Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA', pp. 6382–6393.

McInnes, L. & Healy, J. (2018), 'UMAP: uniform manifold approximation and projection for dimension reduction', *CoRR* **abs/1802.03426**.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. A. (2013), 'Playing atari with deep reinforcement learning', *CoRR* **abs/1312.5602**.

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

Nachum, O., Gu, S., Lee, H. & Levine, S. (2018), Data-efficient hierarchical reinforcement learning, *in* 'Advances in Neural Information Processing Systems 31: Annual

Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.', pp. 3307–3317.

Oliehoek, F. A. & Amato, C. (2016), *A Concise Introduction to Decentralized POMDPs*, Springer Briefs in Intelligent Systems, Springer.

Omidshafiei, S., Kim, D., Liu, M., Tesauro, G., Riemer, M., Amato, C., Campbell, M. & How, J. P. (2018), 'Learning to teach in cooperative multiagent reinforcement learning', *CoRR* **abs/1805.07830**.

Omidshafiei, S., Pazis, J., Amato, C., How, J. P. & Vian, J. (2017), 'Deep decentralized multi-task multi-agent reinforcement learning under partial observability', *CoRR* **abs/1703.06182**.

Piot, B., Geist, M. & Pietquin, O. (2014), Boosted bellman residual minimization handling expert demonstrations, *in* T. Calders, F. Esposito, E. Hüllermeier & R. Meo, eds, 'Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part II', Vol. 8725 of *Lecture Notes in Computer Science*, Springer, pp. 549–564.

Pomerleau, D. (1991), 'Efficient training of artificial neural networks for autonomous navigation', *Neural Comput.* **3**(1), 88–97.

Popova, M., Isayev, O. & Tropsha, A. (2018), 'Deep reinforcement learning for de novo drug design', *Science Advances* **4**(7), eaap7885.

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N. & Whiteson, S. (2018), 'QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning', *CoRR* **abs/1803.11485**.

Ross, S., Gordon, G. J. & Bagnell, D. (2011), A reduction of imitation learning and structured prediction to no-regret online learning, *in* G. J. Gordon, D. B. Dunson & M. Dudík, eds, 'Proceedings of the Fourteenth International Conference on Artificial

Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011', Vol. 15 of *JMLR Proceedings*, JMLR.org, pp. 627–635.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R. & Hadsell, R. (2016), 'Progressive neural networks', *CoRR* **abs/1606.04671**.

Saunders, W., Sastry, G., Stuhlmüller, A. & Evans, O. (2018), Trial without error: Towards safe reinforcement learning via human intervention, *in* E. André, S. Koenig, M. Dastani & G. Sukthankar, eds, 'Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018', International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, pp. 2067–2069.

Schaal, S. (1996), Learning from demonstration, *in* M. Mozer, M. I. Jordan & T. Petsche, eds, 'Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996', MIT Press, pp. 1040–1046.

Schaul, T., Quan, J., Antonoglou, I. & Silver, D. (2016), Prioritized experience replay, *in* '4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings'.

Seita, D., Chan, D. M., Rao, R., Tang, C., Zhao, M. & Canny, J. F. (2019), 'ZPD teaching strategies for deep reinforcement learning from demonstrations', *CoRR* **abs/1910.12154**.

Shao, K., Zhu, Y. & Zhao, D. (2018), 'StarCraft micromanagement with reinforcement learning and curriculum transfer learning', *CoRR* **abs/1804.00810**.

Shoham, Y., Powers, R. & Grenager, T. (2003), Multi-agent reinforcement learning: a critical survey, Technical report, Technical report, Stanford University.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S.,

Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T. & Hassabis, D. (2016), 'Mastering the game of go with deep neural networks and tree search', *Nat.* **529**(7587), 484–489.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: a simple way to prevent neural networks from overfitting', *J. Mach. Learn. Res.* **15**(1), 1929–1958.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K. et al. (2017), 'Value-decomposition networks for cooperative multi-agent learning', *arXiv preprint arXiv:1706.05296* .

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement learning: An introduction*, MIT press.

Taïga, A. A., Fedus, W., Machado, M. C., Courville, A. C. & Bellemare, M. G. (2019), 'Benchmarking bonus-based exploration methods on the arcade learning environment', *CoRR* **abs/1908.02388**.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J. & Vicente, R. (2015), 'Multiagent cooperation and competition with deep reinforcement learning', *CoRR* **abs/1511.08779**.

Taylor, A., Duparic, I., Galván-López, E., Clarke, S. & Cahill, V. (2013), 'Transfer learning in multi-agent systems through parallel transfer'.

Taylor, M. E., Carboni, N., Fachantidis, A., Vlahavas, I. P. & Torrey, L. (2014), 'Reinforcement learning agents providing advice in complex video games', *Connect. Sci.* **26**(1), 45–63.

Taylor, M. E. & Stone, P. (2009), 'Transfer learning for reinforcement learning domains: A survey', *Journal of Machine Learning Research* **10**, 1633–1685.

Teh, Y. W., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N. & Pascanu, R. (2017), 'Distral: Robust multitask reinforcement learning', *CoRR* **abs/1707.04175**.

Torrey, L. & Taylor, M. E. (2013), Teaching on a budget: agents advising agents in reinforcement learning, *in* 'International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013', pp. 1053–1060.

van Hasselt, H., Guez, A. & Silver, D. (2016), Deep reinforcement learning with double q-learning, *in* D. Schuurmans & M. P. Wellman, eds, 'Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA', AAAI Press, pp. 2094–2100.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gülçehre, Ç., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T. P., Kavukcuoglu, K., Hassabis, D., Apps, C. & Silver, D. (2019), 'Grandmaster level in StarCraft II using multi-agent reinforcement learning', *Nature* **575**(7782), 350–354.

Vrancx, P., Hauwere, Y. D. & Nowé, A. (2011), Transfer learning for multi-agent coordination, *in* 'ICAART 2011 - Proceedings of the 3rd International Conference on Agents and Artificial Intelligence, Volume 2 - Agents, Rome, Italy, January 28-30, 2011', pp. 263–272.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M. & de Freitas, N. (2016), Dueling network architectures for deep reinforcement learning, *in* M. Balcan & K. Q. Weinberger, eds, 'Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016', Vol. 48 of *JMLR Workshop and Conference Proceedings*, JMLR.org, pp. 1995–2003.

Watkins, C. J. C. H. (1989), 'Learning from delayed rewards'.

Wilson, A., Fern, A., Ray, S. & Tadepalli, P. (2008), Learning and transferring roles in multi-agent reinforcement, *in* 'Proc. AAAI-08 Workshop on Transfer Learning for Complex Tasks'.

Wu, Y., Charoenphakdee, N., Bao, H., Tangkaratt, V. & Sugiyama, M. (2019), Imitation learning from imperfect demonstration, *in* K. Chaudhuri & R. Salakhutdinov, eds, 'Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA', Vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 6818–6827.

Young, K. & Tian, T. (2019), 'Minatar: An atari-inspired testbed for more efficient reinforcement learning experiments', *CoRR* **abs/1903.03176**.

Zhan, Y., Bou-Ammar, H. & Taylor, M. E. (2016), 'Theoretically-grounded policy advice from multiple teachers in reinforcement learning settings with applications to negative transfer', *CoRR* **abs/1604.03986**.

Zhu, C., Cai, Y., Leung, H. & Hu, S. (2020), Learning by reusing previous advice in teacher-student paradigm, *in* A. E. F. Seghrouchni, G. Sukthankar, B. An & N. Yorke-Smith, eds, 'Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020', International Foundation for Autonomous Agents and Multiagent Systems, pp. 1674–1682.

Zhu, Z., Lin, K., Dai, B. & Zhou, J. (2020), 'Learning sparse rewarded tasks from sub-optimal demonstrations', *CoRR* **abs/2004.00530**.

Zhu, Z., Lin, K. & Zhou, J. (2020), 'Transfer learning in deep reinforcement learning: A survey', *CoRR* **abs/2009.07888**.

Zimmer, M., Viappiani, P. & Weng, P. (2014), Teacher-student framework: A reinforcement learning approach, *in* 'AAMAS Workshop Autonomous Robots and Multirobot Systems'.