Methodical Advice Collection and Reuse in Deep Reinforcement Learning

by

Sahir

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

Abstract

Reinforcement learning (RL) has shown great success in solving many challenging tasks via the use of deep neural networks. Although the use of deep learning for RL brings immense representational power to the arsenal, it also causes sample inefficiency. This means that the algorithms are data-hungry and require millions of training samples to converge to an adequate policy. One way to combat this issue is to use action advising in a teacher-student framework, where a knowledgeable teacher provides action advice to a student. Despite the promising results in the action advising literature, there are limitations, such as a limited advice budget, inflexibility of choices to conduct advice collection and reuse. This thesis proposes the use of single (student agent) or dual uncertainties (student and the model of teacher) to drive the advice collection and reuse process in order to provide more flexibility in our algorithms to more efficiently exploit a teacher's advice budget. Additionally, this thesis introduces a new method to compute uncertainty for a deep learning RL agent using a secondary neural network. The results show that using two uncertainties to drive advice collection and reuse improves learning performance across several Atari games.

Preface

This thesis is an original work by Sahir. No part of this thesis has been previously published.

Acknowledgements

First of all, I would like to express my deepest thanks to Professor Matthew Taylor, my supervisor, for his continuous and generous support. His invaluable insights, 360 support, and feedback have taught me lessons of a life-time in such short time. Without his continuous guidance, this thesis would not have been concluded.

I would also like to acknowledge the consistent efforts of my fellow researcher Ecrüment İllhan, from the Queen Mary University of London, in contributing to this work. Moreover, I would like to thank Dr. Srijita Das for introducing me to Ecrüment's work and supporting me along the way. I would also like to express my thanks to my writing coach Dr. Antonie Bodley for their consistent and timely feedback on this thesis.

I would like to recognize the efforts and teachings of Professor Richard Sutton in his course that made me fall in love with Reinforcement Learning. I would also like to thank Nikunj Gupta for expressing interest in this work and providing short-term support. I am also grateful to Khurram Javed and Zaheer Abbas for motivating me to pursue my thesis in Reinforcement Learning.

Lastly, I would like to thank my family in my home country and my dearest wife Gulraiz Sahir, here with me, for their consistent love and support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reinforcement learning (RL) is a sub-field of *machine learning* (ML) in which agents learn what actions to take depending on the environmental states they visit [1]. Recently, deep reinforcement learning (DRL), or reinforcement learning using deep neural networks, has shown great success in a diverse set of problems and real-world applications. For example, RL has provided solutions to many complicated problems, from Atari video games to Go, a board game with a number of legal positions exceeding the total number of atoms in the observable universe. Moreover, RL has played a pivotal role in solving many interdisciplinary problems such as optimization of molecules [2], drug discovery [3], and estimating optimal treatment regimes [4].

Despite this success, DRL algorithms still suffer from the sample inefficiency problem where millions of samples can be required to learn a decent policy. For example, OpenAI Five [5] defeated the Dota 2 world champions by obtaining experience equivalent to 45,000 years in 10 months of training. Similarly, AlphaStar [6] was trained with 16,000 matches to achieve the Grandmaster level in StarCraft. These are highly complicated applications, but it is evident that they require massive computing resources. This sample inefficiency becomes more critical in real-world applications where obtaining samples could incur any form of cost (e.g., robotics and healthcare).

To combat this issue, domain knowledge has been used in the past. Such methods include *reward shaping* [7] and *policy shaping* [8]. The first method works by modifying the reward function to accommodate feedback for good behavior and the other works by modifying the policy via provided feedback. *Learning from Demonstration* (LfD) [9] is another prominent technique that works by collecting demonstrations from the expert to model its behavior. *Human-Agent Transfer* (HAT) [10] and *Confidence-based Human-Agent Transfer* (CHAT) [11] are two methods that use learning from demonstration to alleviate the need for a significant amount of data. Following this approach, recent off-policy work leverages both perfect [12] and imperfect demonstrations [13] to speed up learning in a deep reinforcement learning setting.

Another important paradigm, introduced by Clouse [14], provided an integrated approach for reinforcement learning and *apprentice learning* to speed up learning. Apprentice learning [15], which is now known as learning from demonstration or *imitation learning*, is a supervised learning technique where an agent learns from an expert. Clouse showed that providing an expert's actions to the learning agent makes the agent learn faster than a traditional RL agent. He also introduced an uncertainty-based technique to ask for help by using the agent's Q-values. This notion of *action advising* was further refined into a *teacher-student framework* by Torrey and Taylor [16], in which the teacher acts as a source of knowledge for the naive student. They also introduced multiple heuristics to help in deciding when the teacher should provide advice.

Action advising uses a *teaching budget*, which is the number of times a teacher can provide action advice. This budget is limited in most scenarios as obtaining a knowledge source can be costly in a real-world setting and thus, should be used wisely by the agent. Whether it is the teacher providing advice or a student asking for advice, multiple techniques have been used in the past to account for the limited teaching budget [16–18].

Significant work is being done to investigate the different types of advising, but limited research exists on how to reuse collected advice effectively. Advice reuse is important as obtaining a teacher or expert could be costly. Moreover, as said earlier, there exists a limited teaching budget after which the teacher would not be available. In the past year, Illhan et al. [19, 20] proposed an approach where a supervised learning model is trained from the student-teacher advising. This model of the teacher allowed for effective advice reuse for the student RL agent by computing and using the model's uncertainty. Inspired by their work, this thesis provides a dual uncertainty-based advising framework that offers more flexibility for the student agent. Systematic use of uncertainties of both the student and the model of teacher offers an effective use of the advice budget. With the proposed framework, the student agent, based on its uncertainty, can decide between the 3 available choices at any time: asking the teacher for advice, reusing the model of teacher, or following its own policy. Advice reuse allows for prolonged use of teacher-like action advice to guide the student agent.

Leveraging the model of the teacher for advice reuse is a form of off-policy learning. Techniques such as a replay buffer for experience replay help achieve better convergence in DRL off-policy algorithms. However, they too have limitations. For example, using a replay buffer to replay experience from an old policy can be harmful if the respective actions are sub-optimal or random. Using the model of the teacher to reuse advice, in states where the student feels confused (or uncertain), should prevent the agent from suffering the same problems. This is possible since the model of the teacher intends to provide action advice similar to (or exactly) what the teacher would provide.

Lastly, this thesis proposes a new method for computing the uncertainty of the student agent or a typical deep RL agent. To compute the uncertainty without effecting the performance of the student RL agent itself, a secondary neural network is trained in supervised fashion. This neural network is equipped with *dropout* [21] to help compute the uncertainty from the data collected by the student agent. Dropout is a technique that is used in deep neural networks to reduce overfitting by randomly dropping the units or neurons in the network. In this way, dropout helps simulate exponentially different network architectures during training.

To summarize, this thesis proposes the following contributions in the deep reinforcement learning field:

1. A flexible action advising framework that gives a student agent the liberty to choose from requesting advice from the teacher, reusing advice from the model of the teacher, or following its own policy.

2. Improves on the previous methods by providing a methodical framework allowing the systematic use of uncertainties of both the student agent and the model of the teacher.

3. Introduces a new method of computing uncertainty for the student agent by using a secondary neural network.

Chapter 2

Background & Related Work

This section provides background and related work for reinforcement learning and its pertinent fields such as deep reinforcement learning, learning from demonstration, and action advising with student-teacher framework.

2.1 Reinforcement Learning

Reinforcement learning agents interact with the environment by taking actions and receiving rewards as feedback for their actions. The purpose of the agent is to find an optimal policy for solving the given task. We follow the notation as set by Sutton and Barto in their book [1]. RL problems are typically modelled as a Markov Decision Process (MDP) and represented by the tuple $\langle S, A, R, P, \gamma \rangle$, where S denotes the state space, A denotes the action space, R denotes the reward function, P denotes the state-transition probability, and $\gamma \in [0, 1)$ is the discount rate to account for future rewards. An RL agent at any time-step t and state $s_t \in S$ takes an action $a_t \in A$ to receive a reward $r_{t+1} \in R$ and the next state $s_{t+1} \in S$ by the environment. The agent's goal is to maximize the discounted sum of rewards G_t at any time step:

$$G_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}$$

In other words, an agent tries to find an optimal policy by maximizing the sum of discounted rewards by following the policy π until termination T (for continuous problems T $= \infty$). *Policy* π is a mapping from states to actions which can be denoted as the probability $\pi(a|s)$. The *value* of a state $v_\pi(s)$ is the expected sum of discounted rewards given that the agent starts in state s and follows policy π afterwards:

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid s_t = s] = \mathbb{E}_\pi\left[G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s\right].$$

Similarly, *action-value function* $q_\pi(s, a)$ is the value or expected return of taking action a starting at a state s and following policy π afterwards:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a] = \mathbb{E}_\pi\left[G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \,\middle|\, s_t = s, a_t = a\right].$$

These value functions are approximated by an agent's interaction with the environment, and their estimates get better over time. Due to this iterative improvement, agents come closer to achieving the optimal policy π_*:

$$\pi_*(s) = \operatorname*{argmin}_{a \in A} q_*(s, a)$$

where $q^*(s, a)$ is the optimal action-value function and defined as:

$$q_*(s, a) = \max_{\pi \in \Pi} q_\pi(s, a)$$

where Π denotes the space of all policies.

Q-learning [22] was one of the first successful off-policy reinforcement learning algorithms. An off-policy algorithm learns a greedy policy while following a behaviour policy. Usually, the behaviour policy is represented by an ϵ-greedy policy which enables random exploration by probability ϵ and greedy actions by probability $1 - \epsilon$. This *epsilon* is usually set to a small value (e.g., less than 10%) and can decay over time.

The optimal Q-function Q_* obeys the Bellman optimality equation:

$$Q_*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_*(s',a')\right]$$

which means that the value for a state s and action a is an expectation of the sum of reward obtained by taking action a and the discounted maximum Q-value over actions a' to reach the next possible states s'. The idea of Q-learning is to leverage this Bellman optimality to incrementally update the Q-value estimates to reach the optimal Q-function as $t \to \infty$:

$$Q_{t+1}(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} Q_t(s',a')\right]$$

2.2 Deep Reinforcement Learning

Deep reinforcement learning (DRL) allows us to use non-linear function approximators such as neural networks to apply reinforcement learning to highly complex tasks. DRL, for example, can be used to find a non-linear mapping of the states and actions to their action-values.

Deep Q-Network (DQN) [23] is a well-known off-policy algorithm to approximate the action-value function for high dimensional or non-tabular problems in the deep reinforcement learning setting. Since maintaining a tabular Q-function is not feasible for such complex problems, DQN uses neural networks called *convolutional networks* [24] to approximate the action-value function.

Convolutional layers in a neural network allow us to process images (or pixel data) for tasks such as object detection. Filters in a convolutional layer, typically square in shape, help process these images by sliding across the input to detect patterns. The application of a filter is controlled by a stride. Stride defines the offset by which the filters are moved and applied around the input features.

The notation of the q-function for DQN with parameters θ can be written as $q(s,a;\theta)$. The q-values are estimated by minimizing the loss function at every iteration i:

$$L_i(\theta_i) = \mathbb{E}_{s,a,r,s'}\left[(y_i - q(s,a;\theta_i))^2\right]$$

which is the squared error between the Temporal Difference (TD) target y_i and the predicted q-value from the parameterized neural network at the current iteration i. This TD target is given by:

$$y_i = r + \gamma \max_{a'} q(s', a'; \theta_{i-1})$$

where s' and a' are the next state and action, respectively, and the q-value is predicted over the network's previous iteration $i-1$. This copy of the previous iteration of the network is called the *target network*. The action a for any time step is then selected by computing:

$$a = \max_{a'} q(s, a, \theta)$$

Lastly, DQN maintains a *replay buffer* to store the transitions performed by the agent. This buffer is then used to sample the transitions for computing the loss. Thus, offering better stability for the off-policy algorithm and increasing the data efficiency at the same time. This technique is called Experience Replay. The success of DQN has inspired researchers to develop further enhancements, which are summarized in Rainbow DQN [25]. From these enhancements, we use Double Q-learning [26], and Dueling Networks [27] in this thesis.

2.3 Learning from Demonstration

Learning from Demonstration (LfD) [28] is a technique used to combat the sample inefficiency problem in deep reinforcement learning. It is an offline process that happens prior to the beginning of actual online training of the agent. Demonstrations (state-action pairs) from an expert are provided to the RL agent to boost performance and to make sure the agent does not start learning from scratch online. These expert demonstrations could span as long as the length of an entire episode.

Formally, a set of demonstrations D is provided to the agent. Each demonstration $d_i \in D$ is a pair of state $z \in Z$ and action $a \in A$: $d_i = (z, a)$. Significant work has been done to leverage LfD to battle the sample inefficiency of RL algorithms. Human-Agent Transfer (HAT) [10] used LfD to boost the performance of a Sarsa RL agent. Building upon HAT, Confidence-based Human Agent Transfer [11] used the confidence in demonstrations, using estimators such as a Neural Network, to exploit the knowledge for the RL agent. Deep Q-learning from Demonstrations (DQfD) [12] is another application that leveraged LfD to achieve state-of-the-art performance across 11 games. LfD has also been applied with imperfect demonstrations [13] to achieve superior performance (comparable or better than DQfD) using a unified objective function for both LfD and online reinforcement learning.

Recently, LfD has been used for a skill transmission application where a robot is trained for the path-following task using expert demonstrations [29]. Another recent example is the application of LfD for teaching a social robot about sign language [30].

2.4 Action Advising & Student-Teacher Framework

Action advising is an online technique where an RL agent, typically an expert, shares actions as a form of advice to teach another agent that is relatively naive. Action advising occurs for a limited number of times during training, characterized by a teaching or advising budget.

The idea of providing expert actions to an RL agent was originated by Clouse in his dissertation [14]. The main idea was to integrate apprentice learning, or learning from demonstration, and reinforcement learning to speed up the learning of RL agents. Clouse attempted to answer questions such as when the learner (student) and trainer (teacher) should interact during training and how the trainer's expertise can help a learner do well in a task? In doing so, Clouse showed that providing actions to a learner could help achieve better performance than the trainer itself. Moreover, he also studied uncertainty-based advising, and the results suggested that it is one of the best ways to ask for advice since it requires less interaction to obtain an adequate policy.

Since then, significant work has been done in the action advising literature. A predominant action advising framework is the teacher-student framework, proposed by Torrey and Taylor [16]. As the name suggests, the teacher in their work is an expert that intends to help a novice student agent. They explored multiple heuristics to decide when the teacher should provide advice to the student. Those *teacher-driven* heuristics are detailed below:

1. **Early Advising:** A well-known and simple heuristic for the teacher is to provide advice from the beginning until the teaching budget is exhausted. This thesis also uses early advising to serve as a baseline heuristic to compare against.

2. **Importance Advising:** Building upon Clouse's work, Importance Advising enables the teacher to provide advice when it is important to do so. To check whether it is important to provide advice, the difference between the Q-values of the teacher for the best and the worst action is computed; and if it is greater than some threshold, the teacher is bound to provide advice (given there is enough teaching budget left). Torrey and Taylor define importance I for a given state s as:

$$I(s) = \max_a Q(s, a) - \min_a Q(s, a)$$

3. **Mistake Correcting:** Building upon Importance Advising, Mistake Correcting further ensures advice is only provided when the teacher and the student's action does not match (assuming a two-way communication). Hence, the name *mistake correcting.*

4. **Predictive Advising:** Building upon Importance Advising and Mistake Correcting, the teacher in Predictive Advising tries to model student's actions using the data of student-environment interaction (state-action pair of the student). If the predicted action is then different from the teacher's action, and if it is important to provide advice, the teacher is bound to do so (given enough advice budget).

All heuristics except Early Advising are assumed to have access to the teacher's Q-value function or student's intended action beforehand. Thus, for the sake of simplicity and feasibility, we use Early Advising as one of the baseline heuristics to compare our proposed algorithms against.

Aside from the teacher-driven advising methods (mentioned above), *student-driven* advising is also investigated in the literature. To decide when to ask the teacher for advice, metrics such as epistemic uncertainty [18, 31] and advice-novelty [32] are used in previous work. Specifically, Requesting Confidence-Moderated Policy advice (RCMP) [18] is a recent method, closest to our work (discussed in Section 3.3), that computes the uncertainty by averaging the variances of multiple Q-values for each action and uses this uncertainty to decide when to ask for advice. Moreover, *jointly-initiated* action advising [33] is also explored in the literature, in which a student could query the teacher, and the teacher could provide action advice.

The student-teacher framework has been extended to include the support of multiple agents during training to leverage advising without the explicit need of fixing a student or teacher role [17]. Another extension includes a general framework for Learning to Coordinate and Teach Reinforcement (LeCTR) [34], where multiple objective functions are used to learn when and what to advise. Similar to [17], agents in the LeCTR framework act as peers. Each agent in the LeCTR framework learns two policies; a task-level policy; and an advising policy. The task-level policy enables the agent to solve the task at hand, and the advising policy helps the agent get better at advising other agents.

2.4.1 Transfer Learning versus Action Advising

Transfer learning in reinforcement learning [35] is categorized by multiple dimensions such as *task mappings* and *transfer knowledge*. This work focuses on the transfer knowledge type of transfer learning – specifically, the transfer of action as advice from the teacher agent to train a beginner student agent. Transfer of actions in this manner is called action advising. It is important to highlight that action advising is a specific type of transfer learning, as mentioned earlier, to define the context so that it is easy to categorize and follow.

2.4.2 Advice Imitation & Reuse

As discussed in Section 2.4, significant work has been done in the area of action advising. However, there exists limited work to capture and reuse advice for later stages of training. Advice reuse is useful because it allows for an extended form of action advising that is not possible with a limited teaching budget alone. Advice reuse was proposed by Zhu et al. [36] in a student-teacher framework. They used multiple heuristics to decide when to perform advice reuse. Those heuristics are detailed below:

1. **Q-Change per Step:** To store advice for later reuse, Q-Change per Step calculates the difference in Q-values before (Q_t) and after (Q_{t+1}) applying the advised action a_t to a given state s_t:

$$\beta = Q_{t+1}(s_t, a_t) - Q_t(s_t, a_t)$$

   if the difference β is within a predefined threshold, the state and action are stored for later reuse. If the agent visits the same state again, it will reuse this recorded action, and the same procedure will be applied to assess whether the action should be recorded again.

2. **Reusing Budget:** Another way to reuse advice is to have a budget allocated for each advised state. Reusing Budget allows advice reuse until the state-specific advising budget is exhausted. If the agent receives a different action advice for the same state in the future, the budget is reset and the agent will then reuse the new action.

3. **Decay Reusing Probability:** A more flexible approach is to allow the agent, at any state, to decide between reusing the advice, asking the teacher for advice, exploiting its own greedy action, or exploring randomly. To execute this, reusing probability P_{reuse} is defined, and advising probability P_{advice} is computed. The agent then takes an action a_t by following the function below:

$$a_t = \begin{cases} \text{Reuse Advice with probability:} & P_{reuse}, \\ \text{Request Advice with probability:} & P_{advice}, \\ \text{Greedy Action with probability:} & (1 - P_{reuse} \times P_{advice}) \times (1 - \epsilon), \\ \text{Random Action with probability:} & (1 - P_{reuse} \times P_{advice}) \times \epsilon \end{cases}$$

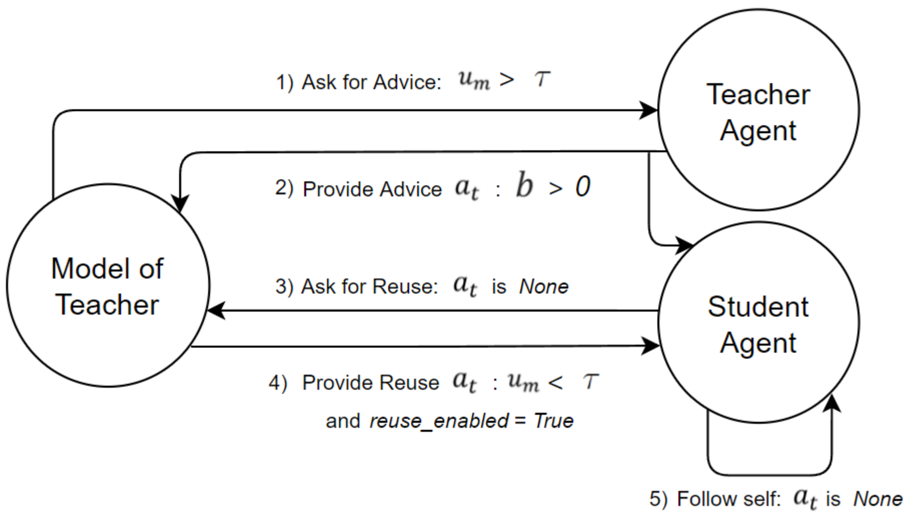
   where the reusing probability P_{reuse} is decayed over time.

Zhu et al.'s work was intended for tabular RL algorithms where it is easy to store advice and recognize states encountered. Taking inspiration from their work, Ilhan et al. [19, 20] proposed a new method to use advice reuse in the non-tabular DRL setting called Advice Imitation & Reuse (AIR). It comprises of two important steps:

1. **Advice Imitation:** They used Imitation Learning [37] (which is a supervised learning technique used for cloning the behavior of an expert) to train a model of the teacher using the advice interaction data between the student and the teacher. They used the model's uncertainty to drive the advice collection process. The model's uncertainty at any state is computed using dropout [21]

(which is a technique used for randomly turning off units in deep neural networks to increase generalization) with multiple forward passes.

2. **Advice Reuse:** Once this model of the teacher is trained, it would then predict the action of the teacher and provide it as reuse for the student. Moreover, the model would only provide reuse when it is certain or when its uncertainty is within some (adaptive or fixed) threshold, and with a probability of reuse ρ. The probability of reuse ρ is decayed over time.



Teacher Agent
Model of Teacher
Student Agent
1) Ask for Advice: $u_m > \tau$
2) Provide Advice a_t : $b > 0$
3) Ask for Reuse: a_t is *None*
4) Provide Reuse a_t : $u_m < \tau$ and *reuse_enabled* = *True*
5) Follow self: a_t is *None*

Figure 2.1: The flow of AIR.

The working flow of AIR is shown in Figure 2.1. In AIR, the model of teacher uses its uncertainty u_m to drive the advice collection and advice reuse processes. The model of teacher will ask the teacher for action advice if its uncertainty u_m is greater than some threshold τ. The teacher would then provide an action advice a_t if the teaching budget is not consumed $b > 0$. This action a_t is also provided to the student agent. However, if this advice is not received, the student agent can ask the model of the teacher for reuse. The model of the teacher would then provide advice reuse, according to reuse probability ρ (which decides if reuse is enabled), if its uncertainty is less than the threshold τ. If the action a_t is not determined at the end, the student agent would follow its own policy.

The limitation of their work includes the student's dependency on the model of the teacher for advice collection. As the student would continue to experience more states in the environment, its measure of uncertainty would account for a larger subset of the state space. This thesis is inspired by the work of Ilhan et al. and thus presents an effort to improve upon their work.

Frequently used abbreviations are listed in Appendix A.1.

Chapter 3

Uncertainty-Driven Advising with Advice Reuse

This section introduces the two algorithms proposed and tested in this thesis. The first section introduces and discusses the new method of computing the uncertainty using a secondary neural network. The second section introduces the problem and some notation to make the discussion easier to follow. The third section describes the algorithm that uses the newly computed uncertainty method. The fourth section introduces the amalgamation of the algorithm in Section 3.3 and the previous work on advice reuse. Finally, the last section proposes a new method that uses dual uncertainties to offer an improved and flexible action advising framework. The notation used in this section is similar to previous work [20] to maintain consistency.

3.1 Computing Uncertainty via a Secondary Neural Network

The methods proposed in this thesis, discussed in the upcoming sections, use a new way of computing uncertainty for the student agent. There are various approaches to computing uncertainties; one approach is to use dropout. This dropout approach was used in previous work [20] to calculate the uncertainty over probabilities of actions of the supervised model of the teacher. However, using a similar approach for the student RL agent is not straightforward. Adding another layer of dropout on a deep

RL agent means an increased training time or inferior performance for the same amount of training time. Thus, we propose a new method of computing uncertainty which is used by the student agent in Student's Uncertainty-driven Advising (SUA) (discussed in Section 3.3) and Student's Uncertainty-driven Advising with Advice Imitation & Reuse (AIR) (discussed in Section 3.3).

Here, we compute the uncertainty of the student RL agent by introducing a secondary neural network. Following previous work [19, 38], this network contains the dropout functionality to compute the epistemic uncertainty. Moreover, the secondary network is trained from the same interaction as that of the original student network. Since the secondary network is trained from the same data, it is safe to assume that it can serve as a decent proxy for the uncertainty estimations of the original student network. Moreover, it is easy to decouple the secondary neural network if need be. For example, early advising or no advising algorithms do not need an uncertainty measure to operate. For such cases, the secondary neural network can be easily turned off.

The uncertainty is computed by averaging the variances of the Q-values, obtained across multiple forward passes, for each possible action. These forward passes are different because the secondary neural network is equipped with dropout. Formally, the uncertainty is computed from the secondary neural network by conducting the following steps. First, N forward passes are performed for a given state s. This would give us the matrix $\mathbf{F} \in \mathbb{R}^{N \times A}$, which contains Q-values for each action $a \in [a_1, a_2, ..., a_A]$, for each forward pass $i \in [1, 2, ..., N]$:

$$\mathbf{F} = \begin{bmatrix} Q_1(s,a_1)\, Q_1(s,a_2) \ldots Q_1(s,a_A) \\ Q_2(s,a_1)\, Q_2(s,a_2) \ldots Q_2(s,a_A) \\ \vdots \\ Q_N(s,a_1)\, Q_N(s,a_2) \ldots Q_N(s,a_A) \end{bmatrix}$$

where A is the total number of possible actions. The matrix $\mathbf{F}$ is then used to compute the variance across each column or the Q-values $Q_i(s,a)$ for each action a.

This results in a row matrix $\mathbf{Q} \in \mathbb{R}^{: 1\times A}$:

$$\mathbf{Q} = \left[var(Q(s,a_1))\; var(Q(s,a_2)) \ldots var(Q(s,a_A)) \right]$$

The average of the values in matrix $\mathbf{Q}$ then gives us the uncertainty u_s:

$$u_s = \frac{\sum_{j=1}^{A} var(Q(s,a_j))}{A}$$

3.2 Problem Formulation

This thesis uses a methodical action advising framework that aims to leverage uncertainty as a metric to wisely reuse advice or to ask the teacher for advice. That being said, it is not necessary to use only uncertainty to drive decisions. This framework is intended to be used with a generic metric where instead of using uncertainty, metrics such as *value of information* [39] (or its approximation) can be used.

To formalize the problem, we train a student agent to learn its own policy π_S with the help of a trained teacher's policy π_E. There is a limited advice budget b for which the teacher agent E can provide action advice when requested by the student agent. Moreover, the student agent can ask for advice reuse from the model of teacher M_η that is built from prior advice interaction. Thus, the goal is to learn an optimal student policy π_S^*, where at any time step the student has to decide to either follow its own policy π_S, reuse the model of teacher M_η, or ask the teacher for advice π_E if the budget b is not consumed.

3.3 Student's Uncertainty-Driven Advising (SUA)

The first method we propose is Student's Uncertainty-driven Advising (SUA). This is not the first method to use the student agent's uncertainty to drive action advising (as discussed in Section 2.4). SUA is different from RCMP as it is not restricted to using a certain network architecture to estimate uncertainty. Moreover, RCMP uses a fixed uncertainty threshold that is tuned to be task-specific. However, both SUA and RCMP estimate the epistemic uncertainty, which is the uncertainty associated to the dearth of information about the environment (which can be reduced by interacting with the environment to collect more samples). Here, we use the newly proposed method of computing uncertainty, as described in Section 3.1, and refer to it as another baseline to compare against.

Specifically, SUA uses the student agent's uncertainty to decide when to ask the teacher for advice. Initially, it would use early advising to gather enough samples to perform an update for the DQN agent. Afterwards, it would shift to uncertainty-based advising using the secondary network. This method does not take advantage of the model of the teacher or advice reuse. This is done to capture the significance of advice reuse which is added in the next algorithm. Lastly, the uncertainty threshold could either be fixed initially or adaptively adjusted based on the previously seen uncertainty estimates. We use adaptive uncertainty thresholds in our experiments due to its dynamic behavior. Moreover, using adaptive thresholds eliminates the need for hyperparameter tuning for finding a fixed threshold.

The setup for SUA is as follows. A deep RL student agent is trained to learn a policy π_S for a maximum of t_{max} training iterations or steps. The teacher agent's policy π_E can be leveraged to obtain action advice $a = \pi_E(s)$ if the teaching budget b is not consumed. The student's secondary network H_ω with weights ω (initialized randomly) is used to compute the uncertainty u_s for a given state s. In the DQN case, this secondary network can be referred to as *twin DQN*. The uncertainty estimates u_s are stored in the uncertainty buffer D_u. The experience tuple of the current state s_t, action a_t, reward r_t, and next state s_{t+1} is stored in the replay buffer D_{dqn}.

The flow of decision-making involved in SUA is shown in Figure 3.1. At any state, the student agent can ask the teacher agent for advice if its uncertainty u_s is greater than the student's adaptive uncertainty threshold c_1. In other words, the student can ask the teacher for advice when it feels uncertain. The teacher agent can then provide

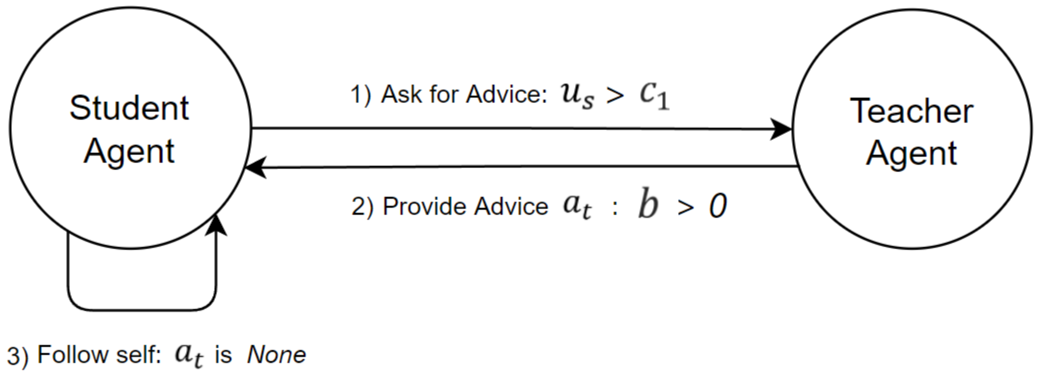

Student Agent
Teacher Agent
1) Ask for Advice: $u_s > c_1$
2) Provide Advice a_t : $b > 0$
3) Follow self: a_t is *None*

Figure 3.1: The flow of SUA.

action advice a_t if the teaching budget is not consumed ($b > 0$). Lastly, the student agent would continue to follow its own policy if no advice is received (a_t is None). This is possible if either the student agent is certain in the given state $u_s \leq c_1$, or if the teaching budget is consumed ($b = 0$).

The pseudocode of <u>**SUA**</u> is given in Algorithm 1. The algorithm is divided into two steps. The first step is Advice Collection (lines 9-28); which is responsible for deciding at any state whether the student agent can receive advice from the teacher agent. Initially, the student agent would receive direct advising from the teacher agent, also known as early advising (lines 24-27). Once the student agent is trained at least once, meaning if it has collected samples equal to the minimum replay memory size (line 14), and its uncertainty values are filled with minimum window size (line 15), the student agent can shift to uncertainty-based advising. To decide whether the student agent is uncertain, the adaptive uncertainty threshold c_1 is determined by obtaining the (hyperparameter) p_1^{th} percentile from the sorted uncertainty buffer D_u (line 16). The student agent can ask the teacher agent when its uncertainty u_s is greater than its adaptive uncertainty threshold c_1 (lines 20-23), given that the teaching budget b is still available (line 13). The second, and the last step is Self Policy (lines 29-31), in which the student agent would ultimately follow its own policy if action a_t is not determined. This will be the case when the teaching budget b is consumed or when the student is certain in the given state s_t, which means that the student's uncertainty u_s is less than or equal to the threshold c_1. The remaining code (lines 32-36) accounts for standard RL and network updates.

Algorithm 1 **S**tudent's **U**ncertainty-driven **A**dvising

```
1: Input: max training iterations $t_{max}$, teacher agent policy $\pi_E$, student DQN policy $\pi_S$, max
   advising budget $b$, secondary network $H_\omega$, minimum window size for uncertainty min-window-
   size
2: $D_{dqn} \leftarrow \emptyset$; $D_u \leftarrow \emptyset$          ▷ initialize replay and uncertainty buffers respectively
3: $c_1 \leftarrow None$;                                 ▷ initialize uncertainty threshold
4: for training steps $t \in \{1, 2, \cdots, t_{max}\}$ do
5:     if $Env$ is reset then
6:         Obtain start state $s_t$ from $Env$
7:     end if
8:     $a_t \leftarrow None$                                  ▷ action not determined yet
   ▷ Advice Collection
9:     if DQN-student-model is trained at least once then
10:        $u_s \leftarrow H^u_\omega(s_t)$                        ▷ measure student uncertainty
11:        $D_u$.append($u_s$)
12:    end if
13:    if $b > 0$ then
14:        if DQN-student-model is trained at least once then
15:            if $|D_u| \geq$ min-window-size then
16:                $c_1 \leftarrow$ determine threshold using $D_u$
17:            else
18:                $c_1 \leftarrow -\infty$                          ▷ activate early advising
19:            end if
20:            if $u_s > c_1$ then             ▷ uncertainty-based advising or early advising
21:                $a_t \leftarrow \pi_E(s_t)$
22:                $b \leftarrow b - 1$
23:            end if
24:        else
25:            $a_t \leftarrow \pi_E(s_t)$                               ▷ early advising
26:            $b \leftarrow b - 1$
27:        end if
28:    end if
   ▷ Self Policy (Ilhan et al., 2021)
29:    if $a_t$ is None then
30:        $a_t \leftarrow \pi_S(s_t)$                       ▷ follow student's current policy
31:    end if
32:    Take action $a_t$, obtain $s_{t+1}, r_t$ from $Env$
33:    $D_{dqn}$.append($\langle s_t, a_t, r_t, s_{t+1} \rangle$)                  ▷ Store experience
34:    Update DQN-student-model
35:    Update $H_\omega$ with DQN's mini-batch & target
36:    $s_t \leftarrow s_{t+1}$
37: end for
```

3.4 Student's Uncertainty-Driven Advising with Advice Imitation & Reuse (SUA-AIR)

Adding the functionality of advice reuse, as done in Advice Imitation & Reuse (AIR) [20], to SUA gives us a new method that we call Student's Uncertainty-based Advising with Advice Imitation & Reuse (SUA-AIR). AIR uses the advice interaction between the student and teacher agents to train a neural network to imitate the teacher. The uncertainty of this model is then used to drive advice collection and advice reuse processes. SUA-AIR, however, uses the student's uncertainty to drive advice collection and the model of teacher to drive advice reuse if the uncertainty of the model of teacher is less than the fixed or adaptive threshold. Moreover, as done in AIR, advice reuse is subject to a probability-based activation, where this probability is decayed over time. This is done to provide enough autonomy to the student in the later stages of training.

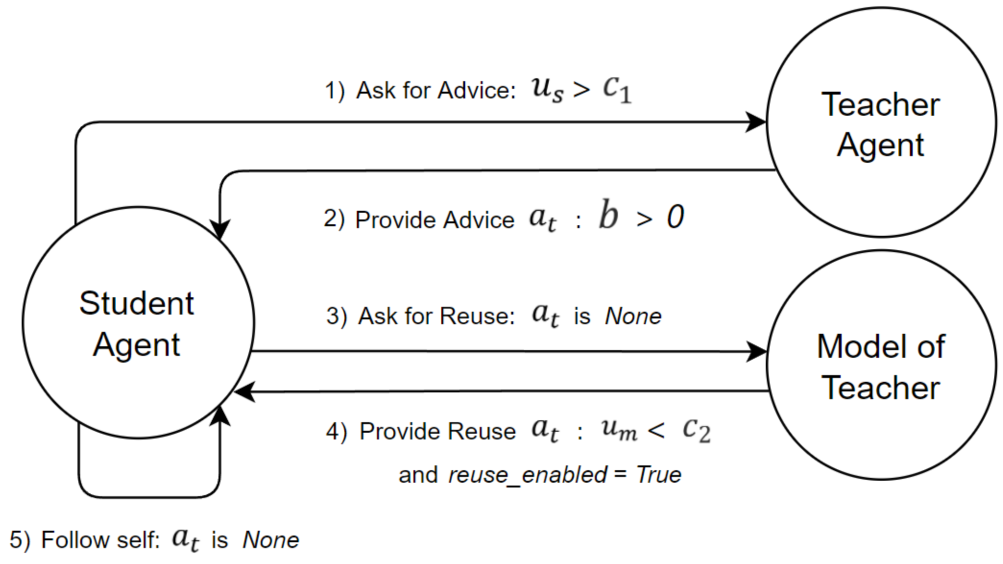

1) Ask for Advice: $u_s > c_1$
Teacher Agent
2) Provide Advice a_t : $b > 0$
Student Agent
3) Ask for Reuse: a_t is *None*
Model of Teacher
4) Provide Reuse a_t : $u_m < c_2$ and *reuse_enabled* = *True*
5) Follow self: a_t is *None*

Figure 3.2: The flow of SUA-AIR.

The extent of flexibility available in SUA-AIR and the conditions that trigger those choices are shown in Figure 3.2. At any state, the student agent can ask the teacher agent for advice if it feels certain or if its uncertainty u_s is greater than the student's adaptive uncertainty threshold c_1. The teacher agent will then provide action advice a_t if the teaching budget is not consumed ($b > 0$). The student agent can ask for advice reuse from the model of the teacher if action advice a_t is not received. The model of the teacher would provide reuse if its uncertainty u_m is less than its uncertainty threshold c_2, or simply if the model is certain, and if advice reuse is enabled ($reuse\_enabled = True$). At any step, advice reuse would be enabled if reuse probability ρ is greater than a random probability. Lastly, the student agent would follow its own policy if action a_t is not determined.

The setup for SUA-AIR is similar to SUA with the following additions. Here, as shown in Algorithm 2, a model of teacher or advice model M_η begins training after the student has taken a minimum of t_{min} steps and obtained $n_{min}/2$ samples. This is done to balance the computational complexity and timely training of the model of the teacher. The model of teacher is leveraged for advice reuse with a reuse probability ρ. This advice reuse probability is decayed from an initial probability of reuse ρ_{init} to the final probability of reuse ρ_{final} in t_ρ steps.

Algorithm 2 Advice Imitation Model

```
1: function BuildAdvModel($D, M_\eta, n_{last}, n_{min}, t_{min}, t, t_{last}, k_{init}, k_{periodic}$)
2:     if ($|D| - n_{last} >= n_{min}$ or
3:         ($|D| - n_{last} >= n_{min}/2$ and $t - t_{last} >= t_{min}$) then ▷ check if the model can be trained
4:         Train $M_\eta$ using $D$ for $k_{init}$ or $k_{periodic}$ iterations        ▷ train the imitation model
5:         Determine $c_2$ as explained in Section 3.4        ▷ compute the uncertainty threshold
6:         $n_{last} \leftarrow |D|$
7:         $t_{last} \leftarrow t$
8:     end if
9:     return $M_\eta, c_2, n_{last}, t_{last}$
10: end function
```

The pseudocode for **SUA-AIR** is given in Algorithm 3. The algorithm is divided into four steps. The first step is Advice Collection (lines 12-33), similar to SUA, with saving advice in advice buffer D (line 25 and line 30) being the only difference.

Algorithm 3 Student's Uncertainty-driven Advising with Advise Imitation and Reuse

```
1: Input: max. training iterations $t_{max}$, teacher agent policy $\pi_E$, student DQN policy $\pi_S$, advice
   model $M_\eta$, advising budget $b$, secondary network $H_\omega$, minimum window size for uncertainty
   min-window-size, initial reuse probability $\rho_{init}$, final reuse probability $\rho_{final}$, total $\rho$ decaying
   steps $t_\rho$, min. samples collected and steps taken to begin training $M_\eta$: $n_{min}$, $t_{min}$ respectively,
   initial imitation training iterations $k_{init}$, periodic imitation training iterations $k_{periodic}$
2: $D \leftarrow \emptyset$; $D_{dqn} \leftarrow \emptyset$; $D_u \leftarrow \emptyset$    ▷ initialize advice, replay, and uncertainty buffers respectively
3: $c_1 \leftarrow None$; $c_2 \leftarrow None$    ▷ initialize student and model uncertainty thresholds respectively
4: $n_{last} \leftarrow 0$; $t_{last} \leftarrow 0$
5: $\rho \leftarrow \rho_{init}$    ▷ set reuse probability with initial value
6: for training steps $t \in \{1, 2, \cdots, t_{max}\}$ do
7:     if $Env$ is reset then
8:         $reuse\_enabled \leftarrow True$ with probability $\rho$, $False$ otherwise
9:         Obtain start state $s_t$ from $Env$
10:    end if
11:    $a_t \leftarrow None$    ▷ action not determined yet
   ▷ Advice Collection
12:    if DQN-student-model is trained at least once then
13:        $u_s \leftarrow H^u_\omega(s_t)$    ▷ measure student uncertainty
14:        $D_u$.append($u_s$)
15:    end if
16:    if $b > 0$ then
17:        if DQN-student-model is trained at least once then
18:            if $|D_u| \geq$ min-window-size then
19:                $c_1 \leftarrow$ determine threshold using $D_u$
20:            else
21:                $c_1 \leftarrow -\infty$    ▷ activate early advising
22:            end if
23:            if $u_s > c_1$ then    ▷ uncertainty-based advising or early advising
24:                $a_t \leftarrow \pi_E(s_t)$
25:                $D$.append($\langle s_t, a_t \rangle$)    ▷ append to advice buffer
26:                $b \leftarrow b - 1$
27:            end if
28:        else
29:            $a_t \leftarrow \pi_E(s_t)$    ▷ early advising
30:            $D$.append($\langle s_t, a_t \rangle$)    ▷ append to advice buffer
31:            $b \leftarrow b - 1$
32:        end if
33:    end if
   ▷ Advice Imitation (Ilhan et al., 2021)
34:    $M_\eta, c_2, n_{last}, t_{last} \leftarrow BuildAdvModel(D, M_\eta, n_{last}, n_{min}, t_{min}, t, t_{last}, k_{init}, k_{periodic})$
   ▷ Advice Reuse (Ilhan et al., 2021)
35:    $u_m \leftarrow M^u_\eta(s_t)$    ▷ measure model uncertainty
36:    if $reuse\_enabled$ is $True$ and $a_t$ is None and $M_\eta$ is trained and $u_m < c_2$ then
37:        $a_t \leftarrow \arg\max_a M_\eta(a|s_t)$    ▷ Reuse action
38:    end if
39:    Decay $\rho$ w.r.t. pre-defined schedule if $\rho > \rho_{final}$
   ▷ Self Policy (Ilhan et al., 2021)
40:    if $a_t$ is None then
41:        $a_t \leftarrow \pi_S(s_t)$    ▷ follow student's current policy
42:    end if
```

```
43:     Take action $a_t$, obtain $s_{t+1}, r_t$ from $Env$
44:     $D_{dqn}$.append($\langle s_t, a_t, r_t, s_{t+1} \rangle$)          ▷ store experience
45:     Update DQN-student-model
46:     Update twin DQN $H_\omega$ with DQN's mini-batch & target
47:     $s_t \leftarrow s_{t+1}$
48: end for
```

The second, and new step here, is Advice Imitation (line 34), where a model of the teacher is trained using the data from the advice buffer. The $BuildAdvModel$ function (line 34), as described in algorithm 2, is responsible for training the model of teacher M_η from prior advice and adaptively determining its uncertainty threshold c_2. The model of teacher or imitation model is initially trained for k_{init} iterations and then trained periodically for $k_{periodic}$ iterations if all conditions (line 2 of algorithm 2) are met. Similar to AIR [20], the model of teacher is trained to minimize the loss of negative log-likelihood: $\mathcal{L}(\eta) = \sum_{(s,a)\in D} -logM_\eta(a \mid s)$. The uncertainty threshold c_2 is then determined by using the updated model of teacher. For this, the model's uncertainty M^u_η is computed for each state s_i in the advice buffer D and stored in a list U such that the action a_i suggested by the teacher (and followed by the student) is the action predicted by the model. In other words, each uncertainty in U must satisfy $a_i = \text{argmax}_a M_\eta(a \mid s)$. Then similar to c_1, c_2 is set to the p_2^{th} percentile of the sorted list U. lhan et al. followed this approach to ensure that the model can correctly classify the states it knows (within c_2) while marking the remaining states (above c_2) as potential outliers.

The third step in SUA-AIR is Advice Reuse (lines 35-39), which is in charge of the action-advice reuse mechanism. If all conditions are met (line 36), the student can leverage the model of teacher for advice reuse (line 37). To successfully reuse advice, reuse must be enabled ($reuse\_enabled = True$) with the probability of reuse ϵ_{reuse}, the current action a_t must not be determined or None, the model of teacher M_η must be trained and, deemed certain. The model would be certain when, for a given state s_t, its uncertainty $M^u_\eta(s_t)$ or u_m is less than its uncertainty threshold c_2. The probability of reuse is also set to decay here if not fully decayed (line 39).

The fourth and final step is Self Policy, which is the same as the previous method, where the student agent would follow its own policy if it is considered certain (i.e., $u_s \leq c_1$) or the teaching budget b is consumed. The remaining code (lines 43-47) accounts for standard RL and network updates.

Chapter 4

Experiments

This section details the experiments conducted as well as the research questions answered in this thesis. Below are the research questions explored through the experiments:

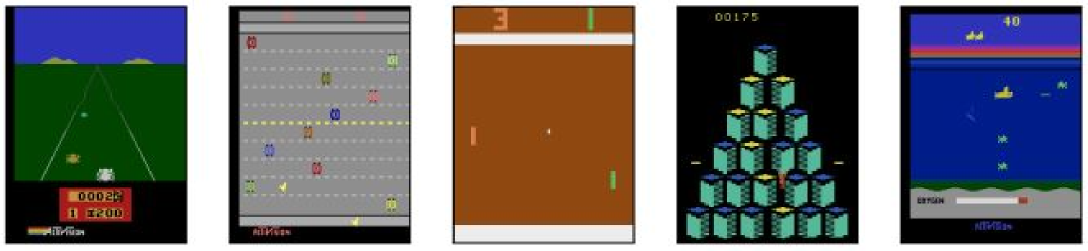
1. Does using the uncertainty estimates of the student agent, for advice collection, and model of the teacher, for advice reuse, help achieve adequate performance?
2. How well do the proposed algorithms perform against the baselines in the literature?
3. What change(s) do the proposed algorithms bring to the advice collection or reuse process?

The following section discusses the environments used to test the algorithms. The last section outlines the experiment details, including all the algorithms tested in this thesis. To maintain consistency, the pre-processing for domains, network architectures, and many experiment details (e.g., hyperparameter values) are kept the same as İlhan et al.'s work [20], which are fully specified here for replicability.

4.1 Testing Environments

The proposed algorithms and the baseline heuristics are evaluated here in five games of the Arcade Learning Environment (ALE) [40]. The five selected games are Enduro, Freeway, Pong, Q*bert and Seaquest. ALE offers a suitable challenge for deep reinforcement learning algorithms with its vast set of environments. Thus, ALE was selected as a means to test all algorithms.

Each game or environment works in a Red Green Blue (RGB) image space, where each observation takes the specific dimension of $160 \times 120 \times 3$. The first two dimensions correspond to the x and y dimensions of the image, whereby the last dimension (3) denotes that the image is in the RGB space. Pre-processing is applied to reduce the complexity of these observations by converting them to a squared-grayscale image of the dimension $80 \times 80 \times 1$. Furthermore, frames are repeated or skipped for 4 consecutive frames by applying the same action decided by the agent to accommodate for computational cost and high frame rate. The resulting four frames (16 in total) are stacked on top of each other to account for partial observability effects (e.g., capture information like movement), making the final dimensions of the observation to $80 \times 80 \times 4$. Furthermore, the steps for each episode across all games are limited to 27,000 steps which accounts for 108,000 frames (27,000 steps times 4 frames). Lastly, the rewards for each game are clipped to $[-1, 1]$ for maintaining learning stability. These design choices correspond with the literature [20, 23].



(a) Enduro (b) Freeway (c) Pong (d) Q*bert (e) Seaquest

Figure 4.1: A snapshot of the observations for the selected Atari games.

4.2 Experimental Setup

The student agents tested in this work are trained for 5 million steps (or 20 million total frames) and evaluated at an offset of every 50,000 steps across 10 episodes. A student agent can request advice from the teacher for a maximum of 25,000 steps which corresponds to the teaching budget. In evaluation episodes, the student agent is tested without any aid from the teacher or model of the teacher. This is done to measure the performance of the student agent alone. Statistics pertaining to evaluation performance are also recorded, such as the average of the total rewards obtained across 10 evaluation episodes for the corresponding training step. Furthermore, all student agents were tested for 10 independent runs across each domain. All student agents involved in the experiments are listed below:

1. **No Advising (NA):** An unadvised student agent similar to a usual RL agent learning from scratch.

2. **Early Advising (EA):** An advised student agent with action advice provided by the teacher at early training until it has consumed the teaching budget.

3. **Random Advising (RA):** An advised student agent with a 50% chance of teacher providing advice until the teaching budget exhausts.

4. **Advice Imitation & Reuse (AIR):** A previous benchmark [20] that uses an advised student agent and a model of teacher (trained with advising data) to drive advice collection and reuse via its uncertainty.

5. **Student's Uncertainty-driven Advising (SUA):** An advised student agent that uses its uncertainty estimates from the secondary neural network to drive advice collection. The model of teacher and advice reuse mechanism are absent in SUA.

6. **Student's Uncertainty-driven Advising with Advice Imitation & Reuse (SUA-AIR):** To build upon AIR, this agent uses student agent's uncertainty from the secondary neural network to drive advice collection and the model of teacher's uncertainty to drive advice reuse.

All student agents above use the same architecture which is a Double DQN with 3 convolutional layers followed by a fully-connected hidden layer, and a dueling output. The first convolutional layer, in the student agent architecture, is comprised of 32 filters of shape 8×8 with a stride of 4. The second convolutional layer is comprised of 64 filters of shape 4×4 with a stride of 2. The last convolutional layer is comprised of 64 filters of shape 3×3 with a stride of 1. The fully-connected layer is comprised of a single hidden layer with 512 units. All agents use the ϵ-greedy strategy for exploration where ϵ is decayed over time. The secondary network of the student is equipped with a network structure similar to student DQN with two added differences. The secondary network is a supervised learning model, and is equipped with a dropout layer. Following the literature [41], the dropout rate is set to 0.2 so that it does not hamper the learning progress. Dropout rate controls the percentage of units that are dropped at every training step. Moreover, the number of forward passes performed to compute the epistemic uncertainty is set to 100.

The supervised learning model of teacher (or imitation model), trained with the student-teacher interaction data, is equipped with the network structure identical to the student agent's secondary network. Here, the model of teacher predicts action probabilities (instead of Q-values in the secondary network). A dropout layer is also added to the model. Following AIR [20], the dropout rate here is set to 0.35. Furthermore, the number of forward passes to compute the epistemic uncertainty is set to 100.

For each game, a teacher agent is trained beforehand. The teacher agents have the same network structure and algorithm (DQN) as that of the student agent. The

Table 4.1: Hyperparameters for all DQN student agents and model of the teacher.

Hyperparameter	Student Agent	Imitation Model
Learning rate	6.25×10^{-5}	1×10^{-4}
Minibatch size	32	32
Discount factor γ	0.99	-
Replay memory min. and max. size	10k, 500k	-
Target network update frequency	7,500	-
$\epsilon_{initial}$, ϵ_{final}, total ϵ decaying steps	1.0, 0.01, 250k	-

teacher agents can be considered competent, as compared to results in the literature, and obtain the evaluation scores of 1556 for Enduro, 28.8 for Freeway, 12 for Pong, 3705 for Q*bert, and 8178 for Seaquest.

Since SUA, SUA-AIR, and AIR use adaptive uncertainty thresholds, a hyperparameter for fixing the uncertainty threshold is not required. However, all of the mentioned algorithms require one or two percentile values to determine the uncertainty thresholds automatically. The percentile p_1 for the student's uncertainty in SUA and SUA-AIR is set to 70 to maintain a balance for the student agent in asking the teacher for advice. Whereas percentile p_2 for the uncertainty of model of teacher in SUA-AIR and AIR is set to 90. For SUA-AIR and AIR, the initial probability of reuse ρ_{init} is set to 0.5, and the final probability of reuse ρ_{final} is set to 0.1. The decay from ρ_{init} to ρ_{final} happens in a total of 1.5 million steps, starting from 500,00 steps and ending at 2.5 million steps. For training the model of teacher or the imitation model (in SUA-AIR and AIR), the minimum steps t_{min} is set to 50,000, and the minimum samples n_{min} is set to 2,500. Moreover, the initial training iterations k_{init} and periodic training iterations $k_{periodic}$ for the model of teacher (in SUA-AIR and AIR) are set to 50,000 and 20,000, respectively. Lastly, the *min-window-size* for the uncertainty buffer D_u is set to 200, and the maximum size is set to 10,000. The values for minimum and maximum window size for the uncertainty buffer were selected based on general performance across all domains. The values for other hyperparameters pertaining to the student agent and the imitation model are listed in table 4.1. The values for all the mentioned hyperparameters are kept the same across experiments in all domains.

All other hyperparameter values except replay memory minimum size and ϵ decaying steps are kept the same as in previous work [20]. Minimum replay memory size is decreased to 10k (from 50k) to enable early training of the student agent (from the replay buffer) to produce accurate uncertainty estimations and ϵ decaying steps are decreased to 250k (from 500k) to limit exploration in later stages of training.

The list of all the hyperparameters for the student agents is shown in Appendix B.1, and the list of all the hyperparameters for the model of the teacher (or imitation model) is shown in Appendix B.2.

The next section will present and discuss the results of the experiments conducted.

Chapter 5

Results & Discussion

This section presents and discusses the results of the experiments conducted with the proposed algorithms (mentioned in Chapter 4) across all domains.

5.1 Evaluating Performance of Agents

The evaluation performance of all student agents, NA, EA, SUA, AIR, and SUA-AIR, across Enduro, Freeway, Pong, Q*bert, and Seaquest is reported in Figures 5.1, 5.2, 5.3, 5.4, and 5.5. The x-axis reports the environment steps (not frames) taken by the agent in millions. The y-axis reports the average of cumulative rewards across evaluation episodes for the corresponding training step (as mentioned in Section 4.2). The shaded bars indicate the standard deviation across 10 independent runs. To serve as a quick reminder, SUA-AIR and AIR are advised agents with advice reuse, whereas SUA, EA, and RA are advised agents without advice reuse. SUA-AIR and SUA are the methods proposed in this work. All advised agents without advice reuse serve as baselines for our study.

The evaluation scores of all student agents in different training phases across all domains are reported in Table 5.1. The reporting of evaluation scores is divided into three training phases. The first phase, called the initial performance, refers to the evaluation performance of the agent across one-third ($\frac{1}{3}$) of the training. The second phase, called the intermediate performance, refers to the evaluation performance of the agent across two-third ($\frac{2}{3}$) of the training. The last phase, called the last performance, refers to the evaluation performance of the agent across three-third ($\frac{3}{3}$) of the training. Moreover, the table reports the average cumulative rewards for the last three evaluations as the final performance. Lastly, the table also reports the total cumulative evaluation rewards as the total performance.

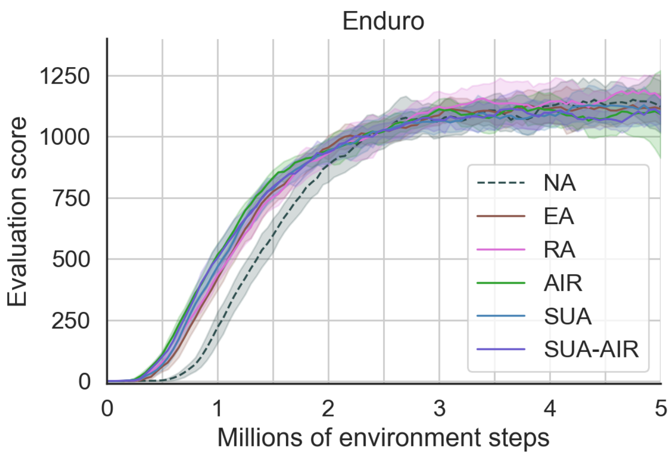


Figure 5.1: Evaluation rewards of each algorithm for Enduro.

In Enduro, the performance of all advised-agents was similar, with RA being ahead of all algorithms with a final performance of 1169.99 ± 25.13. However, the differences in final performance of RA and all other agents are not statistically significant as they are not more than twice the standard error in the respective means (for 95% confidence). Thus, it can not be said that RA performed better than all agents, with respect to final performance, in Enduro. In early training, the NA agent performed poorly with an initial performance of 196.63 ± 9.47, where the differences in initial performance of NA and all other agents are statistically significant (as they are more than twice the standard error in the respective means). However, NA managed to catch up to the performance of the advised-agents in the end (with respect to later and final performance). Moreover, NA agent also performed worse than all other agents with a total performance of 769.67 ± 12.02 in Enduro, where the differences in total performance of NA and all other agents are statistically significant.

		Evaluation Scores				
Domain	Student	Initial (1/3)	Inter. (2/3)	Later (3/3)	Final	Total
Enduro	NA	196.63 ± 9.47	975.27 ± 18.70	1120.26 ± 17.31	1133.44 ± 33.07	769.67 ± 12.02
	EA	316.59 ± 9.29	1011.35 ± 8.15	1113.29 ± 15.44	1116.52 ± 17.57	818.67 ± 6.99
	RA	320.86 ± 5.56	**1015.65 ± 13.25**	**1148.66 ± 17.64**	**1169.99 ± 25.13**	**833.41 ± 11.05**
	AIR	**374.08 ± 5.07**	1014.47 ± 7.82	1094.33 ± 16.12	1097.87 ± 47.19	832.12 ± 8.11
	SUA	341.69 ± 6.71	1000.59 ± 11.22	1102.97 ± 10.42	1106.68 ± 17.35	819.77 ± 7.66
	SUA-AIR	359.34 ± 7.33	1003.02 ± 6.80	1089.54 ± 11.23	1105.11 ± 19.09	821.84 ± 6.01
Freeway	NA	5.51 ± 0.28	23.68 ± 0.61	31.57 ± 0.13	32.04 ± 0.04	20.40 ± 0.28
	EA	6.82 ± 0.40	26.58 ± 0.35	31.87 ± 0.03	32.11 ± 0.04	21.91 ± 0.22
	RA	5.01 ± 0.59	21.55 ± 1.03	31.58 ± 0.07	32.09 ± 0.07	19.52 ± 0.53
	AIR	**8.95 ± 0.38**	28.84 ± 0.16	31.89 ± 0.03	32.12 ± 0.05	23.37 ± 0.13
	SUA	6.72 ± 0.24	25.29 ± 0.60	31.75 ± 0.05	32.10 ± 0.04	21.40 ± 0.25
	SUA-AIR	8.57 ± 0.43	**29.36 ± 0.23**	**31.98 ± 0.02**	**32.15 ± 0.07**	**23.45 ± 0.18**
Pong	NA	-18.37 ± 0.29	-6.70 ± 1.36	3.51 ± 1.59	6.12 ± 1.39	-7.08 ± 1.01
	EA	-16.58 ± 0.37	-1.02 ± 1.09	6.43 ± 1.73	7.97 ± 1.87	-3.59 ± 0.91
	RA	-17.17 ± 0.36	-2.08 ± 1.92	6.40 ± 2.08	9.11 ± 1.76	-4.16 ± 1.39
	AIR	-12.52 ± 0.24	**6.46 ± 0.30**	10.62 ± 0.45	11.36 ± 0.51	1.66 ± 0.23
	SUA	-16.89 ± 0.34	-2.50 ± 1.13	6.60 ± 1.30	9.35 ± 1.12	-4.14 ± 0.86
	SUA-AIR	**-11.80 ± 0.28**	6.39 ± 0.38	**11.19 ± 0.20**	**12.32 ± 0.27**	**2.06 ± 0.19**
Q*bert	NA	**523.17 ± 22.72**	**1709.45 ± 39.80**	2127.62 ± 182.23	2678.62 ± 225.15	**1462.62 ± 67.15**
	EA	239.70 ± 12.52	539.89 ± 49.23	1544.32 ± 157.99	2054.10 ± 225.66	779.93 ± 64.51
	RA	308.00 ± 23.54	1332.33 ± 150.13	2027.82 ± 115.96	2427.65 ± 165.35	1231.77 ± 78.08
	AIR	263.38 ± 20.06	610.25 ± 25.74	2588.68 ± 148.04	3508.49 ± 97.30	1162.92 ± 56.82
	SUA	233.19 ± 7.53	571.69 ± 51.65	1797.56 ± 116.56	2307.20 ± 191.48	873.76 ± 54.32
	SUA-AIR	270.13 ± 17.40	660.25 ± 60.88	**2641.93 ± 185.13**	**3653.54 ± 151.78**	1199.88 ± 78.84
Seaquest	NA	407.18 ± 21.80	2177.36 ± 40.02	4503.52 ± 378.01	5747.69 ± 606.93	2382.05 ± 127.50
	EA	703.16 ± 27.12	3628.69 ± 200.65	6799.10 ± 452.63	8131.73 ± 487.51	3740.09 ± 205.78
	RA	**900.46 ± 11.64**	2973.80 ± 94.90	4357.73 ± 426.44	5126.56 ± 628.49	2762.25 ± 165.89
	AIR	814.67 ± 27.28	4009.80 ± 202.07	7099.48 ± 258.61	**8569.04 ± 231.59**	4005.94 ± 146.88
	SUA	806.88 ± 19.11	3571.92 ± 266.72	6751.58 ± 483.42	8080.32 ± 581.71	3738.87 ± 240.52
	SUA-AIR	819.10 ± 31.64	**4091.41 ± 179.81**	**7135.82 ± 449.44**	8000.41 ± 541.18	**4047.09 ± 206.70**

Table 5.1: Evaluation scores with respect to initial, intermediate, later, final, and total performance of all agents in 5 domains averaged over 10 independent runs. The standard errors are reported with ±. The best scores are reported in bold.

The benefits of advice reuse become slightly more apparent when we look at the performance of agents in Freeway. Students with advice reuse, SUA-AIR and AIR, showed a boost in performance during early-to-mid training. AIR achieved an initial performance of 8.95 ± 0.38, and SUA-AIR achieved an initial performance of 8.57 ± 0.43, where the differences are not statistically significant. However, the differences in

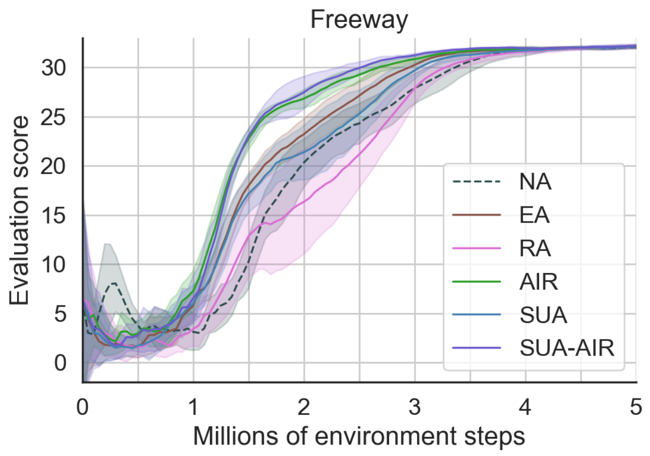


Figure 5.2: Evaluation rewards of each algorithm for Freeway.

the initial performance of SUA-AIR and all other agents, except AIR, are statistically significant. Similarly, the differences in the initial performance of AIR and all other agents, except SUA-AIR, are statistically significant. The same trend is observed for intermediate and total performance where SUA-AIR and AIR are better than all other agents, with the differences being statistically significant. In terms of later performance, SUA-AIR performed better than all other agents, except AIR, with statistically significant differences. AIR, on the other hand, only performed better than RA in later performance with statistically significant differences.

Advising methods such as EA, RA, and SUA took more time to catch up to the policies of students using advice reuse in Freeway (as shown in Figure 5.2). Moreover, SUA performed better than NA and RA in initial performance, with statistically significant differences. EA, on the other hand, only performed better than NA with statistical differences.

The difference of performance between students with and without advice reuse becomes more evident in Pong. Students with advice reuse (SUA-AIR and AIR) showed a boost in performance throughout the entire training. SUA-AIR performed better than all agents, except AIR, in initial, intermediate, later, and total performance with statistically significant differences. In terms of final performance, SUA-AIR per-

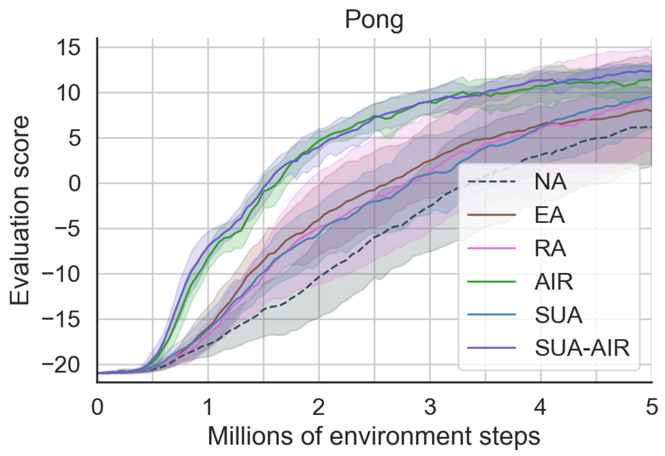


Figure 5.3: Evaluation rewards of each algorithm for Pong.

formed better than other agents except AIR and RA, with an evaluation score of 12.32 ± 0.27 and differences being statistically significant. AIR, on the other hand, performed better than other agents, except for SUA-AIR, in initial, intermediate, and total performance, with statistically significant differences. In terms of later performance, AIR is only better than NA and SUA with an evaluation score of 11.19 ± 0.2 and statistically significant differences. However, AIR performed better than other agents, except SUA-AIR, in total performance with an evaluation score of 2.06 ± 0.19 and differences being statistically significant. Advised agents without advice reuse (EA, SUA, and RA) failed to keep up with the performance of advice reuse students in Pong (as shown in Figure 5.3).

In Q*bert, the advice reuse agents performed better towards the end of training. SUA-AIR and AIR performed better than all other agents in terms of later performance with evaluation scores 2641.93 ± 185.13 and 2588.68 ± 148.04, respectively, and differences being statistically significant. Moreover, SUA-AIR and AIR performed better than all other agents in terms of final performance with evaluation scores of 3653.54 ± 151.78 and 3508.49 ± 97.30, respectively, with statistically significant differences. On the other hand, NA agent performed better than all agents in initial performance with an evaluation score of 523.17 ± 22.72 and statistically

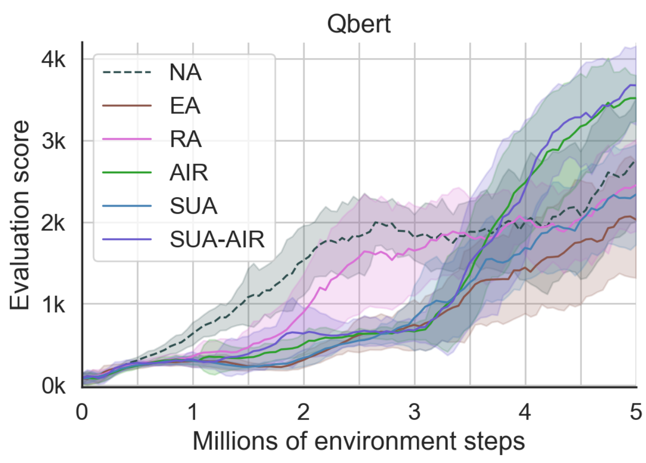


Figure 5.4: Evaluation rewards of each algorithm for Q*bert.

significant differences. Moreover, NA agent performed better than other agents, except RA, in intermediate performance with an evaluation score of 1709.45 ± 39.80 and statistically significant differences. NA agent also performed better than other agents, except SUA-AIR and RA, in total performance with an evaluation score of 1462.62 ± 67.15 and statistically significant differences. Furthermore, advised agents such as EA and SUA had difficulty catching up to the performance of the NA agent (as shown in Figure 5.4). This reinforces the need to have the model of the teacher available for advice reuse at any time during training.

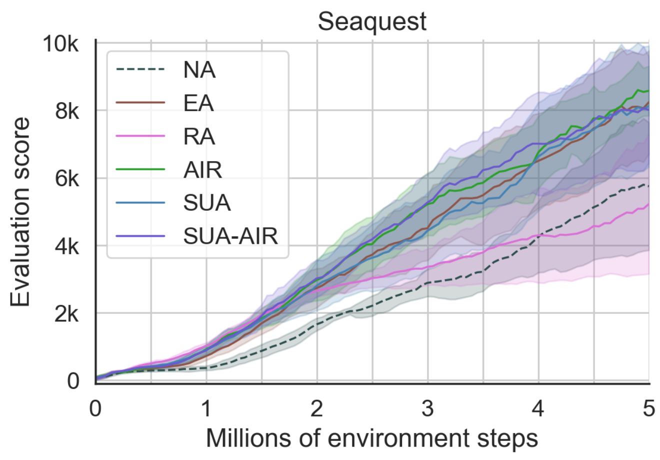


Figure 5.5: Evaluation rewards of each algorithm for Seaquest.

In Seaquest, most of the advised agents took the lead over the NA agent throughout the entire training (as shown in Figure 5.5). RA agent performed better than all agents, except SUA-AIR, in initial performance with an evaluation score of 900.46 $\pm$ 11.64 with statistically significant differences. In terms of intermediate performance, SUA-AIR performed better than NA and RA agents with an evaluation score of 4091.41 $\pm$ 179.81 and differences being statistically significant. Moreover, SUA-AIR also performed better than NA and RA agents in terms of last performance with an evaluation score of 7135.82 $\pm$ 449.44 and statistically significant differences. AIR, on the other hand, performed better than NA and RA agents in final performance with an evaluation score of 8569.04 $\pm$ 231.59 and statistically significant differences. In terms of total performance, SUA-AIR performed better than NA and RA agents with an evaluation score of 4047.09 $\pm$ 206.70 and statistically significant differences.

5.2 Evaluating Advice Taken & Reuse Schedule

The advice taken and reuse schedule are reported in Figures 5.7, 5.6, 5.8, 5.9, and 5.10. The left plots report the number of times advice is reused (y-axis) across million environment steps (x-axis) for each game tested. The plots on the right report the number of times the advice is taken (y-axis) across million environment steps (x-axis) for each game tested. In general, SUA-AIR (purple) required less reuse from the model of the teacher than AIR (green). This could be due to the student agent in SUA-AIR being more certain in states encountered than the model of the teacher in AIR. Although SUA-AIR reused less number of advice from the model of teacher, it still managed to perform on par or slightly better in some games than AIR in terms of evaluation scores. This is mainly due to the difference in the advice schedule. SUA and SUA-AIR showed a similar advice schedule since they use the secondary network to estimate the epistemic uncertainty.

AIR, on the other hand, uses the model of the teacher for advice collection. The advice taken schedule for SUA and SUA-AIR showed a more erratic pattern than AIR's advice taken schedule. This erratic pattern of SUA and SUA-AIR suggests

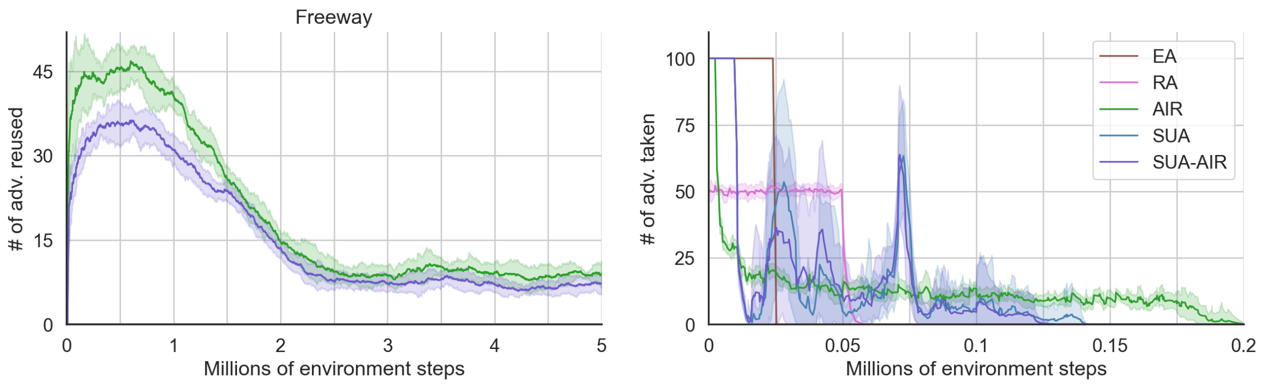


Figure 5.6: Advice reuse (actions taken from the model of the teacher) and advice taken (actions taken directly from the teacher agent) in every 100 steps taken by the student in Freeway.

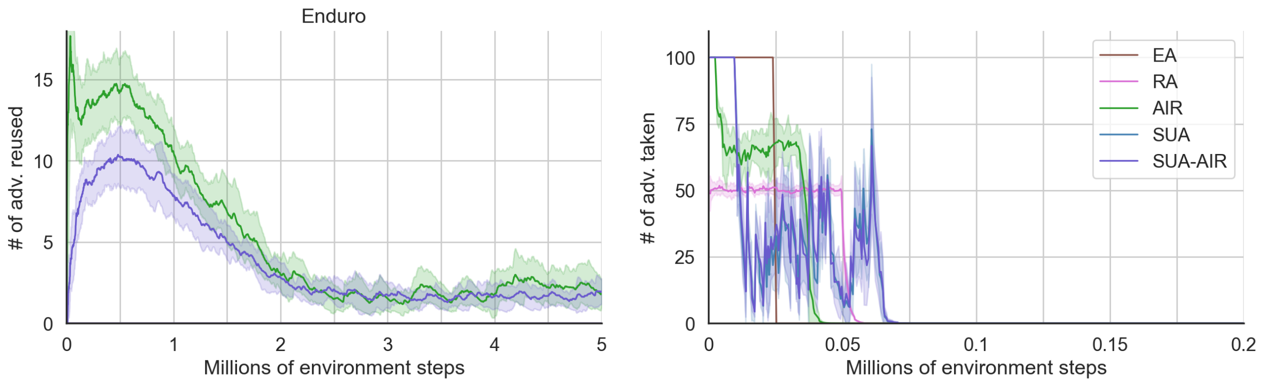


Figure 5.7: Advice reuse (actions taken from the model of the teacher) and advice taken (actions taken directly from the teacher agent) in every 100 steps taken by the student in Enduro.

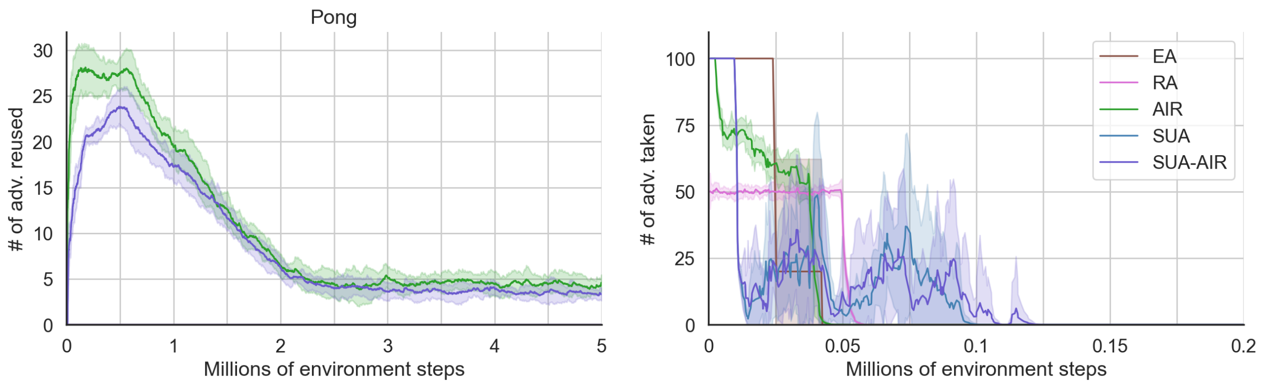


Figure 5.8: Advice reuse (actions taken from the model of the teacher) and advice taken (actions taken directly from the teacher agent) in every 100 steps taken by the student in Pong.

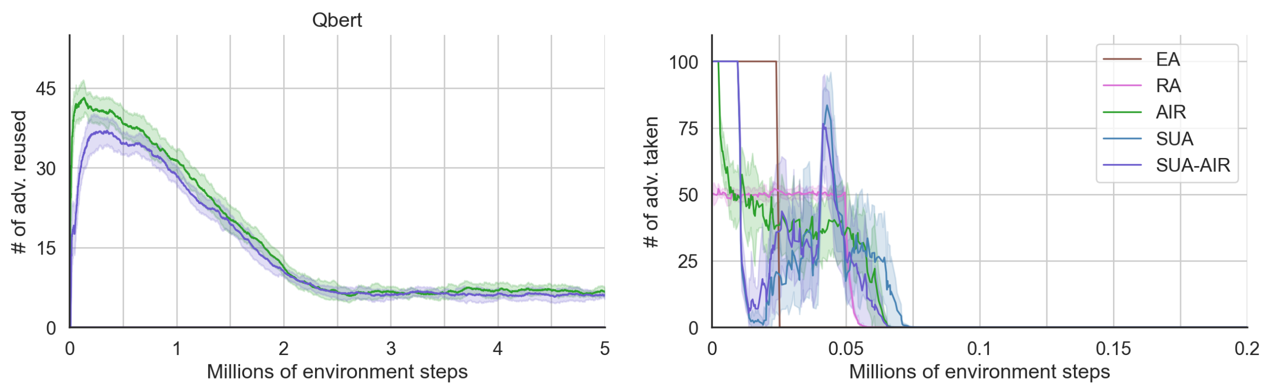


Figure 5.9: Advice reuse (actions taken from the model of the teacher) and advice taken (actions taken directly from the teacher agent) in every 100 steps taken by the student in Q*bert.

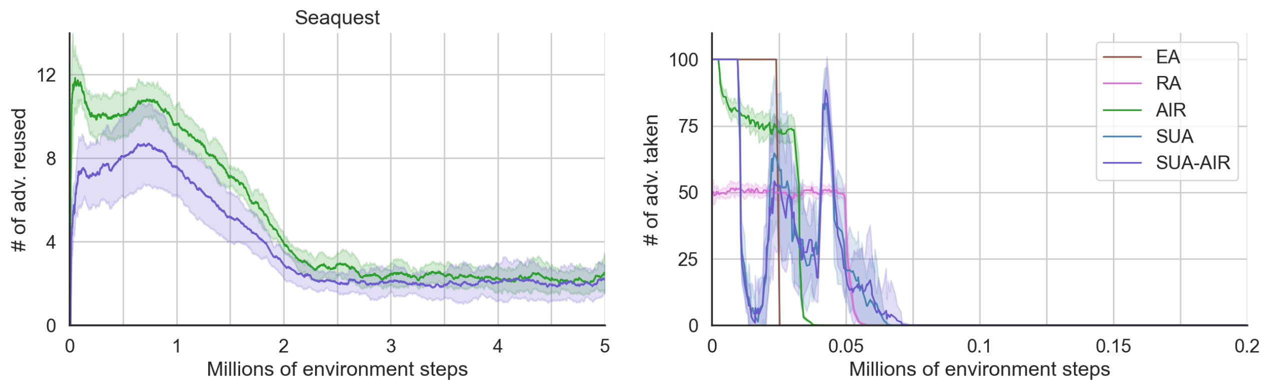


Figure 5.10: Advice reuse (actions taken from the model of the teacher) and advice taken (actions taken directly from the teacher agent) in every 100 steps taken by the student in Seaquest.

that they are asking the teacher for advice in matters of uncertainty. This is perhaps possible due to the low percentile value set for SUA and SUA-AIR. Lastly, the advice taken schedule for AIR is similar to EA, where most of the advice is taken in early training. This is due to the high percentile value set for AIR by default.

5.3 Evaluating Model Performance

To further investigate the similar evaluation performance of SUA-AIR and AIR, we evaluate the accuracy of the model of teacher by comparing the actions of the teacher and the model for the states that the student visits. This evaluation is shown in

Figure 5.11.

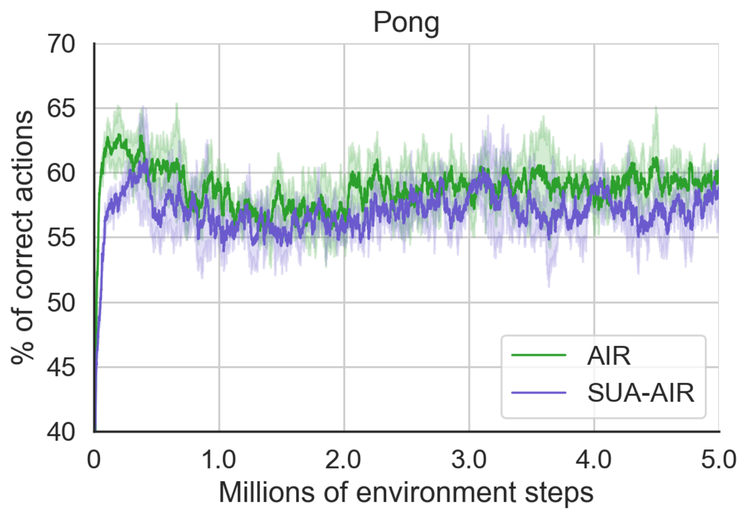


Figure 5.11: Percentage of correct actions taken by the model of the teacher for SUA-AIR and AIR across training steps in Pong averaged over 3 independent runs.

The percentage of correct actions taken by the model (y-axis), correct actions are actions that are the same as teacher's, over the course of millions of environment steps (x-axis) taken by the student agent in Pong are reported. It is evident from Figure 5.11 that the model of the teacher for SUA-AIR and AIR show similar accuracies across training. Since AIR uses the model of the teacher to collect advice, it is natural to see it slightly more accurate than the model of SUA-AIR. However, this does not impact the evaluation performance of SUA-AIR, as we have seen it perform slightly better than AIR (though not statistically significant) in the majority of the games tested. Lastly, due to the similar performance of these models of the teacher, SUA and SUA-AIR showed similar evaluation performance across all games.

Chapter 6

Conclusion

In this thesis, the use of two uncertainties is investigated to drive the advice collection and reuse process in the action advising paradigm of deep reinforcement learning. This work focused on the teacher-student framework in action advising literature.

We first presented a new way of computing the epistemic uncertainty for the student agent. We then proposed two new methods, Student's Uncertainty-driven Advising (SUA), and Student's Uncertainty-driven Advising with Advice Imitation & Reuse (SUA-AIR). Both SUA and SUA-AIR use our method for computing uncertainty to drive the advice collection process. This uncertainty of the student agent and the uncertainty of the model of teacher were then used for the advice collection (in SUA and SUA-AIR) and advice reuse processes (in SUA-AIR), respectively. The model of teacher was trained using advice interaction data between the student and teacher agents. Using this framework, the student agent can decide when to ask the teacher agent for direct advice, or the model of teacher for advice reuse, or when to follow the student's own policy.

We really hoped that SUA-AIR would outperform AIR but we found that the differences were not statistically significant. Moreover, the results show that using advice reuse, in action advising RL agents, provides a significant boost in performance in different stages of training.

There are multiple avenues for future work. Currently, the student agent in SUA-

AIR considerably leverages the model of the teacher for advice reuse after the teaching budget is consumed. This can be further extended to add more flexibility where the student agent could start asking the model of the teacher for reuse during the consumption of the teaching budget, to use the teaching budget efficiently. Moreover, the decision to reuse advice in SUA-AIR precedes the decision of the student agent. This can be extended to account for the student agent's uncertainty before reusing advice from the model of the teacher. Furthermore, a more thorough study could be conducted to test different advice reuse schedules by modifying the initial reuse probability, final reuse probability, and the total decaying steps. For example, the final reuse probability could be set to 0 to allow the student agent to become independent from the model of the teacher towards the later stages of training. Lastly, we currently use fixed percentile values to compute the uncertainties for the student agent and the model of teacher. This could be better extended to follow a dynamic schedule where the percentile values start closer to 50 in early training and then restricted to values closer to 100 in the later stages of training. This change would be better suited for advice reuse since a lower percentile value for the student agent's uncertainty would not capture the states for which the student is genuinely uncertain.

This work can be applied to real-world scenarios in which continuously deploying new RL models is critical. These new models can then leverage the previously-built models through action advising and reuse to speed up the training process significantly.

Bibliography

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

[2] Z. Zhou, S. Kearnes, L. Li, R. N. Zare, and P. Riley, "Optimization of molecules via deep reinforcement learning," *Scientific reports*, vol. 9, no. 1, pp. 1–10, 2019.

[3] S. K. Gottipati, Y. Pathak, B. Sattarov, Sahir, R. Nuttall, M. Amini, M. E.T̃aylor, and S. Chandar, "Towered actor critic for handling multiple action types in reinforcement learning for drug discovery," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 142–150.

[4] N. Liu, Y. Liu, B. Logan, Z. Xu, J. Tang, and Y. Wang, "Learning the dynamic treatment regimes from medical registry data through deep q-network," *Scientific reports*, vol. 9, no. 1, pp. 1–10, 2019.

[5] OpenAI. (2019). Openai five defeats dota 2 world champions, [Online]. Available: https://openai.com/blog/openai-five-defeats-dota-2-world-champions/.

[6] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[7] A Ng, D Harada, and S Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, 1999.

[8] S. Griffith, K. Subramanian, J. Scholz, C. Isbell, and A. L. Thomaz, "Policy shaping: Integrating human feedback with reinforcement learning," in *NIPS*, 2013.

[9] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, 1999.

[10] M. E. Taylor, H. B. Suay, and S. Chernova, "Integrating reinforcement learning with human demonstrations of varying ability," in *AAMAS*, 2011.

[11] Z. Wang and M. E. Taylor, "Improving reinforcement learning with confidence-based demonstrations.," in *IJCAI*, 2017.

[12] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, *et al.*, "Deep q-learning from demonstrations," in *AAAI*, 2018.

[13] Y. Gao, H. Xu, J. Lin, F. Yu, S. Levine, and T. Darrell, "Reinforcement learning from imperfect demonstrations," *ICLR Workshop Track Proceedings*, 2018.

[14] J. A. Clouse, *On integrating apprentice learning and reinforcement learning.* University of Massachusetts Amherst, 1996.

[15] A. L. Samuel, "Some studies in machine learning using the game of checkers. ii—recent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[16] L. Torrey and M. Taylor, "Teaching on a budget: Agents advising agents in reinforcement learning," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, 2013, pp. 1053–1060.

[17] F. L. Da Silva, R. Glatt, and A. H. R. Costa, "Simultaneously learning and advising in multiagent reinforcement learning," in *Proceedings of the 16th conference on autonomous agents and multiagent systems*, 2017, pp. 1100–1108.

[18] F. L. Da Silva, P. Hernandez-Leal, B. Kartal, and M. E. Taylor, "Uncertainty-aware action advising for deep reinforcement learning agents," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 5792–5799.

[19] E. Ilhan, J. Gow, and D. P. Liebana, "Action advising with advice imitation in deep reinforcement learning," in *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, F. Dignum, A. Lomuscio, U. Endriss, and A. Nowé, Eds., ACM, 2021, pp. 629–637. [Online]. Available: https://dl.acm.org/doi/10.5555/3463952.3464029.

[20] E. Ilhan, J. Gow, and D. P. Liebana, "Learning on a budget via teacher imitation," *CoRR*, vol. abs/2104.08440, 2021. arXiv: 2104.08440. [Online]. Available: https://arxiv.org/abs/2104.08440.

[21] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[22] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[24] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, contour and grouping in computer vision*, Springer, 1999, pp. 319–345.

[25] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Thirty-second AAAI conference on artificial intelligence*, 2018.

[26] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[27] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*, PMLR, 2016, pp. 1995–2003.

[28] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[29] J. Li, J. Wang, S. Wang, and C. Yang, "Human–robot skill transmission for mobile robot via learning by demonstration," *Neural Computing and Applications*, pp. 1–11, 2021.

[30] S. R. Hosseini, A. Taheri, M. Alemi, and A. Meghdari, "One-shot learning from demonstration approach toward a reciprocal sign language-based hri," *International Journal of Social Robotics*, pp. 1–13, 2021.

[31] P. Odom and S. Natarajan, "Active advice seeking for inverse reinforcement learning," in *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, 2016, pp. 512–520.

[32] E. Ilhan and D. P. Liebana, "Student-initiated action advising via advice novelty," *CoRR*, vol. abs/2010.00381, 2020. arXiv: 2010.00381. [Online]. Available: https://arxiv.org/abs/2010.00381.

[33] O. Amir, E. Kamar, A. Kolobov, and B. Grosz, "Interactive teaching strategies for agent training," in *In Proceedings of IJCAI 2016*, 2016.

[34] S. Omidshafiei, D.-K. Kim, M. Liu, G. Tesauro, M. Riemer, C. Amato, M. Campbell, and J. P. How, "Learning to teach in cooperative multiagent reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 6128–6136.

[35] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey.," *Journal of Machine Learning Research*, vol. 10, no. 7, 2009.

[36] C. Zhu, Y. Cai, H.-f. Leung, and S. Hu, "Learning by reusing previous advice in teacher-student paradigm," in *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, 2020, pp. 1674–1682.

[37] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural computation*, vol. 3, no. 1, pp. 88–97, 1991.

[38] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, PMLR, 2016, pp. 1050–1059.

[39] G. Chalkiadakis and C. Boutilier, “Coordination in multiagent reinforcement learning: A bayesian approach,” in *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, ser. AAMAS ’03, Melbourne, Australia: Association for Computing Machinery, 2003, 709–716, ISBN: 1581136838. DOI: 10.1145/860575.860689. [Online]. Available: https://doi.org/10.1145/860575.860689.

[40] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[41] L. Chen, X. Zhou, C. Chang, R. Yang, and K. Yu, “Agent-aware dropout dqn for safe and efficient on-line dialogue policy learning,” in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2454–2464.

Appendix A: List of Abbreviations

Abbreviation	**Full Form**
AIR	Advice Imitation & Reuse
ALE	Arcade Learning Environment
CHAT	Confidence-based Human Agent Transfer
DQN	Deep Q-Network
DQfD	Deep Q-learning from Demonstrations
DRL	Deep Reinforcement Learning
EA	Early Advising
HAT	Human-Agent Transfer
LeCTR	Learning to Coordinate and Teach Reinforcement
LfD	Learning from Demonstration
MDP	Markov Decision Process
NA	No Advising
RA	Random Advising
RCMP	Requesting Confidence-Moderated Policy advice
SUA	Student's Uncertainty-driven Advising
SUA-AIR	Student's Uncertainty-driven Advising with Advice Imitation & Reuse

Table A.1: List of all abbreviations with their full forms.

Appendix B: List of Student Agent Hyperparameters

Hyperparameter	**Value**	**Source or Selected from**
Dropout rate	0.2	[41]
No. of forward passes N	100	[20]
Learning rate	6.25×10^{-5}	[20]
Minibatch size	32	[20]
Discount factor γ	0.99	[20]
Replay memory min. size	10k	(10k, 50k)
Replay memory max. size	500k	[20]
Target network update frequency	7500	[20]
$\epsilon_{initial}$, ϵ_{final}	1.0, 0.01	[20]
total ϵ decaying steps	250k	(250k, 500k)
Percentile p_1	70	(70, 80, 90)
min_window_size for D_u	200	see text below
Maximum window size for D_u	10k	(5k, 10k)
Teaching budget b	25k	(12.5k, 25k, 100k)

Table B.1: List of all hyperparameters for the student agents. Hyperparameters such as drop out rate, no. of forward passes, percentile, minimum, and maximum uncertainty buffer window size are pertinent to SUA and SUA-AIR. Similarly, teaching budget is applicable to all agents (except NA).

min_window_size for D_u is set to 200 to initiate the computation of the dynamic threshold c_1 and to ensure that there is enough data to compute accurate estimates.

This will have a very little impact as we will reach this value within the first episode.

B.0.1 List of Imitation Model Hyperparameters

Hyperparameter	**Value**
Dropout rate	0.35
No. of forward passes	100
Learning rate	1×10^{-4}
Minibatch size	32
ρ_{init}, ρ_{final}, total ρ decaying steps	0.1, 0.5 1.5M
Percentile p_2	90
t_{min}, n_{min}	50k, 2.5k
k_{init}, $k_{periodic}$	50k, 20k

Table B.2: List of all hyperparameters for the model of the teacher for AIR and SUA-AIR taken from AIR [20].