Uncertainty-Aware Action Advising for Deep Reinforcement Learning Agents

Felipe Leno Da Silva,[1*] **Pablo Hernandez-Leal,**[2] **Bilal Kartal,**[2] **Matthew E. Taylor**[2]
[1]University of São Paulo, Brazil
[2]Borealis AI, Canada
f.leno@usp.br, {pablo.hernandez, bilal.kartal, matthew.taylor}@borealisai.com

Abstract

Although Reinforcement Learning (RL) has been one of the most successful approaches for learning in sequential decision making problems, the sample-complexity of RL techniques still represents a major challenge for practical applications. To combat this challenge, whenever a competent policy (e.g., either a legacy system or a human demonstrator) is available, the agent could leverage samples from this policy (advice) to improve sample-efficiency. However, advice is normally limited, hence it should ideally be directed to states where the agent is uncertain on the best action to execute. In this work, we propose *Requesting Confidence-Moderated Policy advice* (RCMP), an action-advising framework where the agent asks for advice when its epistemic uncertainty is high for a certain state. RCMP takes into account that the advice is limited and might be suboptimal. We also describe a technique to estimate the agent uncertainty by performing minor modifications in standard value-function-based RL methods. Our empirical evaluations show that RCMP performs better than *Importance Advising*, not receiving advice, and receiving it at random states in *Gridworld* and *Atari Pong* scenarios.

Introduction

Reinforcement Learning (RL) (Sutton and Barto 1998) techniques have been applied to solve increasingly complex problems, such as video game playing (Mnih and others 2015) and robotics tasks (Singh et al. 2019). Despite being one of the most effective approaches to autonomously learn in sequential decision making problems, RL requires exploring the environment for gathering samples, which makes the learning process potentially costly or dangerous for many real-world applications. For this reason, many of the recent investigations in the area aim at reducing the amount of required samples during learning.

When a competent policy is available, the learning agent might leverage samples of it (hereafter referred as *advice*) to reduce the need of (potentially harmful) random exploration, especially at the beginning of learning when the agent is acting randomly. This policy might be available either because a human can demonstrate how to perform the task (Argall et al. 2009) or because other automated agents are available to give suggestions based on their policies (Silva, Glatt, and Costa 2017).

Many works from the *Learning from Demonstration* (Argall et al. 2009) and *Action Advising* (Torrey and Taylor 2013) subareas have successfully leveraged advice to reduce the sample-complexity of challenging tasks.

However, the use of advice requires coping with some open problems. For example, the amount of advice is often limited by human availability or communication costs, hence the learning agent must make effective use of the available samples, while taking into account the demonstrator availability. In the literature, advice is usually given at a prefixed frequency or based on heuristics unrelated to the agent epistemic uncertainty[1], such as based on the demonstrator's knowledge (Taylor et al. 2014). This means that the advice is not aimed to disambiguate situations in which the agent has high uncertainty and the demonstrator is likely to give advice when the agent does not need it.

We here propose *Requesting Confidence-Moderated Policy advice* (RCMP), an algorithm to selectively give advice to a learning agent in situations where its epistemic uncertainty is high. In contrast, when the agent has a low uncertainty (assuming that the uncertainty estimation is accurate), this indicates that the value estimate for the current state is close to convergence, and advice might be saved for more useful situations. For picturing the situation in which this would be useful, imagine a robot receiving a couple of minutes of advice in an episodic task from a human. Instead of simply sequentially demonstrating the solution of the task multiple times, the human might provide initial demonstrations and let the robot try to solve the task itself. The robot then can ask for advice in states where it has not learned what to do yet, or in new states encountered during the execution. This strategy can result in a better coverage of the advised state space than repeating the demonstration of the

[*]This work was completed while an intern at Borealis AI.

[1]The uncertainty stemmed from lack of information about the environment the agent is trying to model. On the other hand, *aleatoric* uncertainty comes from the environment stochasticity. The former is possible to reduce by collecting more samples, while the latter is not.

solution over and over again.

Our second contribution is describing a method for estimating epistemic uncertainty for model-free RL algorithms. This measure is used by RCMP to define if advice should be given in the current state. Our approach consists in learning simultaneously multiple estimates of the value function from a single network, similarly as done in Bootstrapped DQN (Osband et al. 2016). The variance between those estimates is then used as a metric of the epistemic uncertainty, used to define when advice is expected to be useful. Our approach presents the advantage of being flexible and applicable to many value-function-based RL algorithms. We exemplify this by describing adaptions for two popular algorithms in our method description.

We show in our empirical evaluation that our method is able to make efficient use of the provided advice, performing better than the baselines in *Gridworld* and *Pong* domains.

Background

In this section we lay the background for our proposal. First, we describe Reinforcement Learning and representative techniques, then we describe Action Advice and Learning from Demonstration.

Reinforcement Learning

Reinforcement Learning enables the solution of *Markov Decision Processes* (MDP) (Puterman 2005). An MDP is described by a tuple $\langle S, A, T, R, \gamma \rangle$. S is the set of states in the system, A is the set of actions available to the agent, $T : S \times A \times S \rightarrow [0, 1]$ is the state transition function, $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, and γ is a discount factor. The goal of the agent is learning a policy $\pi : S \rightarrow A$ that dictates the action to be applied in each possible state, where the optimal policy π^* maximizes the expected reward achieved. However, in learning problems the functions T and R are unavailable to the agent, that can only observe samples of them by actuating in the environment. Therefore, RL consists in gathering samples of $\langle s, a, s', r \rangle$, where $s' = T(s, a)$ and $r = R(s, a, s')$. Those samples are the only feedback the agent has for solving the task.

Commonly, RL algorithms aim at learning a state-action value function (generally known as Q-function) that approximates the expected return of applying each action in a particular state $Q : S \times A \rightarrow \mathbb{R}$. The optimal Q-function is $Q^*(s, a) = E\left[\sum_{i=0}^{\infty} \gamma^i r_i\right]$, where r_i is the reward received after i steps from using action a on state s and following the optimal policy on all subsequent steps, and γ is a discount factor. Q can be used to extract a policy $\pi(s) = \arg\max_{a \in A} Q(s, a)$, where using Q^* results in π^*.

Although classical RL algorithms such as Q-Learning (Watkins and Dayan 1992) and SARSA (Sutton 1996) learn Q^* under restrictive conditions, directly applying those algorithms in problems with huge state spaces is usually infeasible. For those problems, function approximators might be able to learn a Q function from which a good policy can be extracted. Deep Q-Network (DQN) (Mnih and others 2015) leverages *Deep Neural Networks* (Schmidhuber 2015) to learn Q-functions. The training process of DQNs typically consists of storing the observed samples of interactions with the environment and updating the function approximator with a portion of them, called minibatch $\mathbf{D}$, periodically. The network is optimized by minimizing the following loss function:

$$\mathcal{L}^{DQN} = \mathbb{E}_{\boldsymbol{D}}[(r + \gamma \max_{a'} Q^t(s', a') - Q(s, a))^2], \quad (1)$$

where Q^t is a target network that is periodically updated to have the same weights as Q: $Q^t \leftarrow Q$.

The *Asynchronous Advantage Actor-critic* (A3C) (Mnih and others 2016) leverages multiple simultaneous executions of the learning process to learn in a more efficient way. Assuming the task can be executed multiple times in parallel (e.g., it is a simulated environment), multiple instances of the learning agent will simultaneously update a locally shared actor-critic Deep Neural Network. The same network will learn the critic (estimate of the value of each state) and the actor (policy). The loss function for the critic is:

$$\mathcal{L}^{A3C\_critic} = \mathbb{E}_{\boldsymbol{t}}[(R_i - V(s_i))^2], \quad (2)$$

where $\boldsymbol{t}$ is the trajectory of states and rewards observed since the beginning of the episode until the end, $R_i = \sum_{k=i}^{k=|\boldsymbol{t}|} \gamma^{k-i} r_k$ is the observed discounted return for this episode, and $V(s)$ is the critic estimate of the network for state s. The actor is then updated according to the estimated advantage function A_d, as:

$$\mathcal{L}^{A3C\_actor} = \mathbb{E}_{\boldsymbol{t}}[-log\pi(a_i|s_i)A_d(s_i)], \quad (3)$$

where $A_d(s_i) = \sum_{k=0}^{|\boldsymbol{t}|-1} \gamma^k r_{i+k} + \gamma^{|\boldsymbol{t}|} V(s_{|\boldsymbol{t}|}) - V(s_i)$. Although effective, both DQN and A3C suffer from high-sample complexity. The next section describes approaches to reduce sample-complexity through action advising.

Action Advice and Learning from Demonstration

Despite the advances on RL described on the last section, the sample complexity of those techniques is still high. However, for many tasks, a good policy might be available before starting the training process, more commonly because an older system for solving the task is available or because a human can provide some examples of how to solve the task.

The literature can be roughly divided on two interpretations of how to leverage those available policies. The first one, *Learning from Demonstrations* (LfD) (Argall et al. 2009), typically has a human providing demonstrations to a learning agent. This paradigm has had successes in many challenging robotics applications, where a human teleoperates (Ross et al. 2013) or applies external forces to the learner so as to generate a kinesthetic demonstration (Hersch et al. 2008). The advice usually covers entire episodes, and in general the learning agents try to model the demonstrated policy in a supervised learning fashion (Torabi, Warnell, and Stone 2018), being unable to improve the demonstrator's policy via exploration.

On the other hand, *Action Advising* (Silva and Costa 2019), the second interpretation, consists of receiving action advice for a single state where it is expected to be useful (ideally states that the agent has not explored before and that have a high difference in expected returns for different actions) (Torrey and Taylor 2013). The decision on

when the advice should be provided can be taken by either the learning agent (Amir et al. 2016) or the demonstrator (Taylor et al. 2014). This paradigm is also appropriate to multiagent systems where the agents are learning together (Silva, Glatt, and Costa 2017; Omidshafiei et al. 2019), as they can provide advice as a way of sharing the knowledge gained through their exploration. This paradigm is closer to what we intend to achieve in our work, because the action advising tries to estimate what is the best timing for giving advice instead of giving it arbitrarily. One of most used metrics for defining when to give advice is the *importance advising* (Torrey and Taylor 2013; Amir et al. 2016):

$$I(s) = \max_{a \in A} Q_{\mathcal{D}(s,a)} - \min_{a \in A} Q_{\mathcal{D}(s,a)}, \tag{4}$$

where $Q_{\mathcal{D}}$ is the Q-table of the demonstrator (a RL demonstrator is usually assumed). The intuition of this metric is to give advice when there is a huge difference between the best and worst actions. Although effective in some scenarios, notice that the decisions are taken through the point of view of the demonstrator. This has two undesired consequences: (i) the advising-function does not consider the learning agent policy, which means that advice is possibly given in states where the agent has had sufficient experiences, while neglecting new states for the learning agent; (ii) the demonstrator has to observe the learning agent during all time steps. Our proposal intends to address both of those key limitations.

As opposed to LfD, action advising applies normal exploration when not receiving advice, which means that the learning agents can converge to better policies than the demonstrator's.

The main challenge for both paradigms is that the amount of available advice is usually limited. Either the human is available for a short period of time or communication costs constrain advice from other automated agents. For this reason, ideally the advice should be directed to states which the agent is less confident. In the next section we describe our approach for using the agent epistemic uncertainty to decide when advice should be given.

Uncertainty-Aware Advice

We are interested in leveraging action advice to accelerate the learning process, while allowing the agent to explore the environment and improve upon the demonstrator's policy. For that purpose, we propose the *Requesting Confidence-Moderated Policy advice* (RCMP) algorithm.

We assume that a demonstrator $\pi_\Delta : S \times A \rightarrow [0, 1]$ is available to the agent and can be queried to give action suggestions (we use $\pi_\Delta(s)$ for getting an action sample from π_Δ for state s). While π_Δ might follow any algorithm, the learning agent has no knowledge about the internal representation of π_Δ and can only get samples of $\pi_\Delta(s)$. Furthermore, we assume that π_Δ has a policy that performs significantly better than a random policy. The demonstrator might be unavailable at some times, e.g. if the human will be participating in the learning process only for a short period of time, hence the learning agent is equipped with an availability function $\mathcal{A}_t$ to check if the demonstrator is available at a given step t. In our evaluation, we assume that the learning agent has a budget of advice to be used. Once the budget is spent, the demonstrator is unavailable for the rest of training. The availability can also be evaluated in a domain-specific way, e.g. considering the demonstrator as unavailable when the physically distance between the agents is too high, and this obstructs their communication. In most cases, the demonstrator can neither be assumed to be available at all times nor to have an optimal policy.

Algorithm 1 fully describes RCMP. The learning agent might use any value-function-based algorithm as long as it is able to estimate an epistemic uncertainty measure μ from its model of the value function (we propose a way of adapting DQN-like algorithms in the next section). Firstly, the agent initializes the Q-function $\hat{Q}$ (or value function, e.g., for A3C) and the policy π (line 1). Then, for every learning step, the agent will check its epistemic uncertainty in the current state (line 4) and, in case it is high[2] and the demonstrator is available (line 5), the agent will ask for an advice and follow the suggested action (line 6). Otherwise, the usual exploration will be applied (line 8). $\hat{Q}$ and π are updated normally according to the chosen learning algorithm.

Algorithm 1 RCMP

Require: Value function approximator $\hat{Q}$, agent policy π, uncertainty estimator μ, demonstrator π_Δ, availability function $\mathcal{A}_t$

1: Initialize $\hat{Q}$ and π
2: **for** $\forall$ learning step t **do**
3: Observe s
4: $u \leftarrow \mu(s)$ ▷ Calculate uncertainty (Eq. (5))
5: **if** u is high and $\mathcal{A}_t(t)$ **then**
6: $a \leftarrow \pi_\Delta(s)$ ▷ Ask for advice
7: **else**
8: $a \leftarrow \pi(s)$ ▷ Normal policy
9: **end if**
10: Apply action a and observe s', r
11: Update $\hat{Q}$ and π with $\langle s, a, s', r \rangle$
12: **end for**

Calculating the Uncertainty

Value-based algorithms estimate the expected value of applying each action in a given state. However, vanilla algorithms cannot estimate the uncertainty on their predictions, which means that we can compare the expected values of each action but there is no direct way of estimating the uncertainty of the predictions. For that purpose, we propose a way to measure the uncertainty with a small enhancement in standard algorithms. Consider the illustration of a DQN network in Figure 1. The first layer consists of the state features, whereas the last layer outputs an estimate of the expected value for each action. We propose to add as a last layer multiple *heads* estimating separately expected values for each action, as done in Bootstrapped DQN (Osband et

[2]We decide if the uncertainty is high through a predefined thresholds of 0.11 (Gridworld) and 0.1 (Pong) in our evaluations.

al. 2016). Due to the aleatoric nature of the exploration and network initialization, each head will output a different estimate of the action values. As the learning algorithm updates the network weights, their predictions will get progressively closer to the real function, and consequently one close to the others as the variance of the predictions is reduced. Therefore, we use the variance of the predictions across the heads as an estimate of uncertainty for a given state:

$$\mu(s) = \frac{\sum_{\forall a \in A} var(\boldsymbol{Q}(s,a))}{|A|}, \quad (5)$$

where $\boldsymbol{Q}(s,a) = \begin{bmatrix} Q_1(s,a) \\ \vdots \\ Q_h(s,a) \end{bmatrix}$, $Q_i(s,a)$ is the Q-value given by head i for state s and action a, var is the variance, and h is the chosen number of heads. The final value prediction (used, for example, for extracting a policy from the value function) is the average of the predictions given by each head:

$$\hat{Q}(s,a) = \frac{\sum_{i=1}^{h} Q_i(s,a)}{h} \quad (6)$$

Each head will have their own loss function to minimize. The DQN algorithm might be adapted by calculating a loss for each head as:

$$L_i^{DQN} = \mathbb{E}_{\boldsymbol{D}}[(r + \gamma \max_{a'} Q_i(s',a') - Q_i(s,a))^2], \quad (7)$$

where L_i^{DQN} is the loss function for head i and $\boldsymbol{D}$ is the minibatch for the update.

Similarly, the A3C algorithm is adapted by adding multiple heads for the critic. The loss function for the critic will then be:

$$L_i^{A3C\_critic} = \mathbb{E}_{\boldsymbol{t}}[(R_i - V_i(s))^2], \quad (8)$$

where R_i has the same definition as in Equation (2) and $V_i(s)$ is the value estimate given by the i-th head. The actor will be updated normally using $\hat{V}$ (as in Equation (6)). The variance in Equation (5) is then computed over the value estimates $\boldsymbol{V}(s)$.

Implementation-friendly Description

Although the previous section fully describes our proposal, we show here further implementation details. Our description is a viable and efficient implementation of the training of DQN with multiple heads.

As illustrated in Figure 1b, we assume the network Q is implemented giving as output a prediction for each action $a \in A$ in each head $i \in \{1, \ldots, h\}$, given a batch of states $\boldsymbol{s}$. Therefore, the output of a forward pass in Q is of dimension $h \times |\boldsymbol{s}| \times |A|$. We assume that a *target* network Q^t is used for stability[3].

[3]In cases where no target network is used, simply consider $Q = Q^t$ in this section.

In practice, updating each head with different samples might be desired to help to reduce the bias that might artificially reduce the variance on the predictions. For this purpose, we make use of a *sample selecting function* $d : \boldsymbol{D} \times h \rightarrow \{0,1\}^{h \times |\boldsymbol{D}|}$, where $\boldsymbol{D}$ is the mini-batch for the current update. This function will sample either 0 (not use) or 1 (use) for each sample and each head. The simplest way of implementing d is by sampling $|\boldsymbol{D}|h$ numbers from $\{0,1\}$ with a fixed probability, but any other strategy might be used. Alternatively, different mini-batches could be sorted for each head, but using d is simpler to implement efficiently. Any network architecture might be used for the hidden layers according to the desired domain of application, as long as the input and output layers are defined as specified.

Algorithm 2 describes the implementation of the loss function for DQN, where vectors and matrices are in bold and the comments on the right side depict the dimensionality of the result of each calculation. For a particular minibatch $\boldsymbol{D}$, we first convert the applied actions to the one-hot representation (line 2). Then, we predict the values of the next states (line 3) and for the observed state-action tuples (line 4). Finally, we calculate a loss for each head, using the sorted samples (line 5) to calculate the predicted and target values (lines 7 and 8). Here, $\odot$ represents the element-wise multiplication. Those loss functions can be easily used for network training in any contemporary machine learning framework.

Algorithm 2 Implementation of DQN with heads

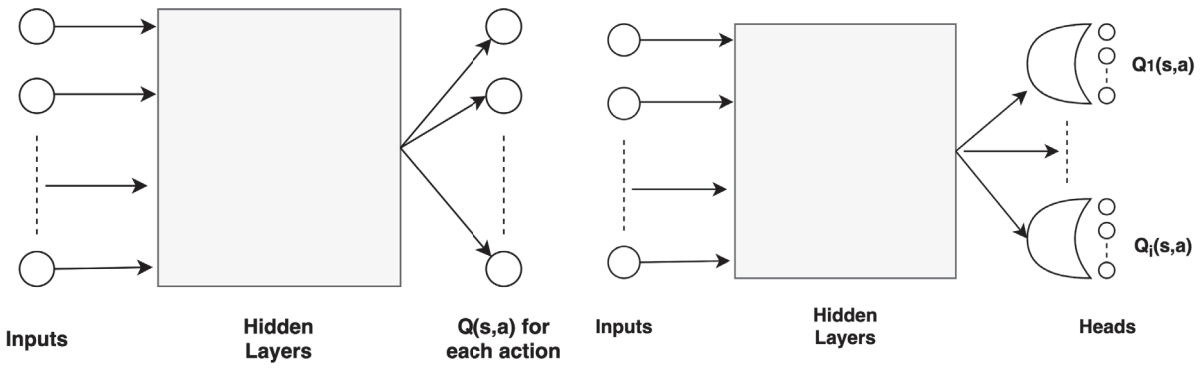
Require: minibatch $\boldsymbol{D}$, Q-network Q, target network Q^t, sample selecting function d, number of heads h

1: $\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}, \boldsymbol{r} = \boldsymbol{D}$
2: $\boldsymbol{act} \leftarrow one\_hot(\boldsymbol{a})$ ▷ $|A| \times |\boldsymbol{D}|$
3: $\boldsymbol{q}^t \leftarrow \max_a Q^t(\boldsymbol{s'})$ ▷ $h \times |\boldsymbol{D}|$
4: $\boldsymbol{q} \leftarrow Q(\boldsymbol{s})\boldsymbol{act}$ ▷ $h \times |\boldsymbol{D}|$
5: $\boldsymbol{p} \sim d(\boldsymbol{D}, h)$ ▷ $h \times |\boldsymbol{D}|$
6: **for** $\forall i \in \{1, \ldots, h\}$ **do**
7: $\quad \boldsymbol{target} \leftarrow \boldsymbol{r} + \gamma \boldsymbol{q}^t[i] \odot \boldsymbol{p}[i]$ ▷ $|\boldsymbol{D}|$
8: $\quad \boldsymbol{pred} \leftarrow \boldsymbol{q}[i] \odot \boldsymbol{p}[i]$ ▷ $|\boldsymbol{D}|$
9: $\quad loss[i] \leftarrow \frac{1}{|\boldsymbol{D}|} \sum (\boldsymbol{target} - \boldsymbol{pred})^2$
10: **end for**

Empirical Evaluation

We evaluated RCMP in two domains varying (i) learning algorithms; (ii) competency level of the demonstrators; (iii) domain complexity. The first one is a relatively simple gridworld-like domain where we can define the optimal policy and use it as a demonstrator for a DQN agent. The second one is the *Pong* Atari game, a much more complex domain where we use as demonstrator a previously-trained A3C agent. With both evaluation domains we hope to show that RCMP is useful across different scenarios.

- **RCMP**: Our proposal as described in the last section.
- **No Advice**: A baseline learning with no advice.
- **Random**: The agent receives uniformly random advice with no regard to its uncertainty.



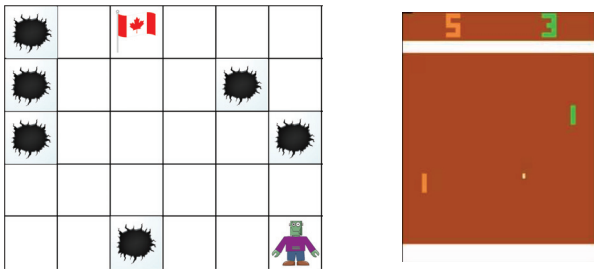
(a) (b)

Figure 1: (a) Illustration of a regular DQN network and (b) a network with *heads*. Each head estimates a value for each action.

- **Importance**: Advice is given according to the importance advising metric calculated from the Q-function of a trained agent (Eq. (4)), as in previous action advising literature (Taylor et al. 2014; Amir et al. 2016). Notice that this algorithm is more restrictive than RCMP because: (i) the demonstrator needs a Q-function; (ii) the learning agents needs to be observed at all time steps, while for RCMP the own agent monitors its uncertainty and queries the demonstrator only when needed.

Gridworld

Our *Gridworld* domain is illustrated in Figure 2a. The agent aims at reaching a goal as quickly as possible, while avoiding falling in one of the holes spread in the environment. The agent has 4 actions $A = \{up, down, left, right\}$ that most of the times have the intended effect, unless the agent is in the 4-neighborhood of a hole. In that case, the agent has a 25% probability of falling into the hole regardless of the applied action. An episode ends when the agent has either reached the goal or fallen into a hole. In the former case, a reward of $+1$ is awarded, whereas in the latter the reward is -1.



(a) (b)

Figure 2: (a) Initial state of the *Gridworld* domain and (b) illustration of the *Pong* domain.

Before evaluating the learning performance of the algorithms, we use this domain for analyzing the effect of the number of heads h on the uncertainty estimate. Figure 3 shows the average uncertainty observed in each learning episode over time for different configurations of the parameters. Regardless of the chosen parameter value the estimate works as expected. At first the uncertainty is high, then it gets progressively lower as the agent trains for longer. However, if the number of heads is very low ($h = 2$), sudden spikes in the uncertainty might be observed when the agent encounters new situations (e.g., around after 120 and 200 learning episodes). The uncertainty curve tends to become smoother for higher number of heads, as shown in the smooth curve of $h = 100$. However, adding more heads means adding parameters to be trained for each head, hence a trade-off is desired.

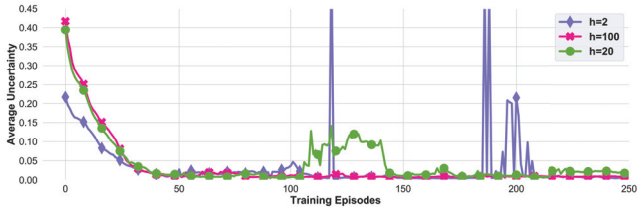
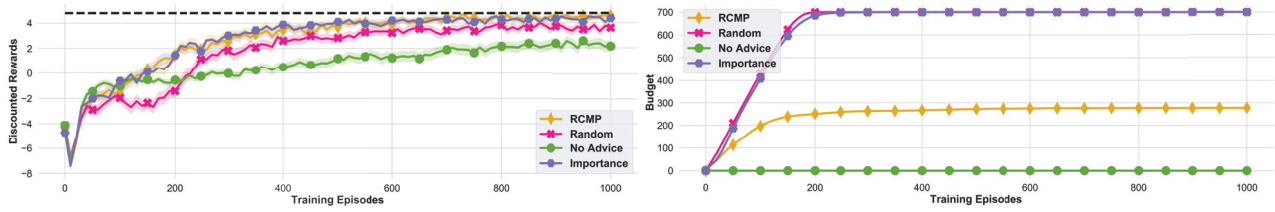


Figure 3: Average uncertainty of the learning episodes over time. Averaged over 200 repetitions.

For evaluating the learning performance in this domain, we use DQN as the base learning algorithm and the optimal policy as the demonstrator. All algorithms are trained for 1000 episodes in total, where the agents are evaluated (exploration and updates turned off) for 10 episodes at every 10 learning episodes. The maximum number of demonstrated steps is set to 700 for the algorithms that can receive advice.For all algorithms, $\alpha = 0.01, h = 5$, and $\gamma = 0.9$. The network architecture is composed of 2 fully-connected hidden layers of 25 neurons each before the layer with the heads. When the agent uncertainty is high, advice is given until the end of the episode.

Figure 4a shows the performance in observed discounted reward for each algorithm, while Figure 4b shows the amount of advice used. RCMP asks for advice until around 200 learning steps, after which the algorithm already has



(a) (b)

Figure 4: (a) Discounted rewards and (b) amount of advice used in 200 repetitions of the *Gridworld* experiment. The shaded area corresponds to the 90% confidence interval. The dashed line corresponds to the optimal performance.

high confidence on its predictions and stop asking for advice. Both *Random* and *Importance*, on their turn, keep asking for advice until the maximum budget is used. RCMP achieves better performance than both *Random* and *No Advice* and the ties with *Importance* for the best performance, while using less advice among all the advice-based algorithms. Notice that RCMP does not use the maximum budget, stopping to ask for advice when it is not expected to be useful anymore, while *Importance* and *Random* spend all the available advice regardless of how fast the learning agent converges. In all cases, the use of advice helped converging faster towards the optimal policy than *No Advice*. After 1000 episodes, *No Advice* still has not converged to the same performance as the algorithms making use of advice.

Figure 5 shows the accumulated reward achieved by each algorithm throughout the entire evaluation. In this experiment, RCMP performed better than both *No Advice* and *Random* and tied for the best performance with *Importance*, while using less advice than all other advice-based algorithms.

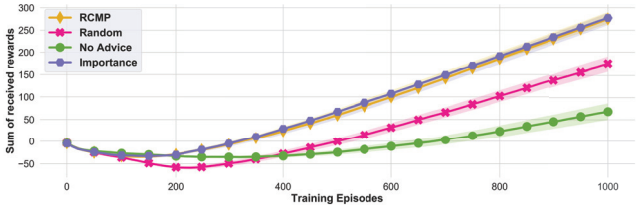


Figure 5: Sum of discounted rewards observed in 200 repetitions of the *Gridworld* experiment. The shaded area corresponds to the 90% confidence interval.

Pong

Pong is an extensively-used evaluation domain, illustrated in Figure 2b.This two-dimensional game consists of controlling an in-game paddle by moving it across the screen to hit a ball towards the opposing side. The learning agent competes against a fixed-strategy opponent. An episode lasts 21 goals, in which a reward of $+1$ is awarded to the player that scores the goal and -1 is given to the other player. Pong is a much harder problem to solve, as the game input consists simply of the game screenshot and winning the game requires a sequence of carefully chosen actions.

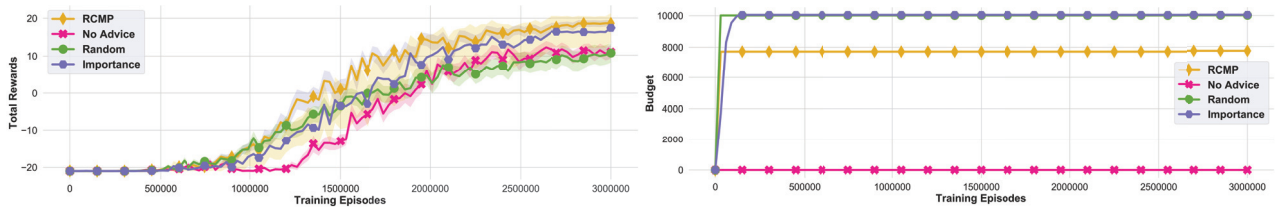
For this domain, we use A3C as the base learning algorithm. We train an A3C agent until it is able to achieve a score of $+21$ in an episode and use it as the demonstrator. All algorithms are trained for 3 million steps, where an evaluation phase of 1 episode is carried out after each $30,000$ learning steps. For all algorithms, $\alpha = 0.0001, h = 5$, and $\gamma = 0.99$. The network architecture is composed of 4 sequences of Convolutional layers followed by max pooling layers, connected to the critic head and actor layers that are fully-connected. Following those layers, we add a Long Short-Term Memory (LSTM) layer which is connected to the critic heads and actor outputs.

Figure 6a and 6b show, respectively, the undiscounted reward achieved by each algorithm and the amount of received advice. RCMP starts to show performance improvements over *No Advice* roughly around after $1,000,000$ learning steps. The apparent disconnect between when the agents receive advice and when the improvement happens is because a sequence of actions must be learned before an improvement in score is seen. Although all advice-based algorithms are getting closer to a winning behavior as they receive advice, seeing an improvement in score takes longer. While *Random* and *Importance* quickly spends all the available advice, RCMP asks for advice only for a short period of time, after which the uncertainty is not high enough to ask for it anymore. Although the pattern in advice use and improvement over *No Advice* is the same as for the *Gridworld* domain for all algorithms, here RCMP presents clear improvements over all the other algorithms while receiving less advice (more visible in Figure 7).

Summary of Empirical Evaluation

We have evaluated RCMP in the *Gridworld* and *Pong* domains. The main conclusions drawn from our evaluation are:

- RCMP performs better than regular learning, randomly receiving advice, and importance advising across domains of different complexity levels.
- Receiving advice based on epistemic uncertainty is advantageous both when the demonstrator is optimal (*Gridworld*) or a trained agent with no optimality guaranteed (*Pong*).
- Our procedure to estimate epistemic uncertainty is effective and easily adaptable across different value-function-based learning algorithms (DQN and A3C were evaluated in this paper).



(a) (b)

Figure 6: (a) Undiscounted rewards and (b) amount of advice used in 20 repetitions of the *Pong* experiment. The shaded area corresponds to the 60% confidence interval.

Figure 7: Sum of undiscounted rewards observed in 20 repetitions of the *Pong* experiment.

Related Works

Our work is related to an extensive literature on *Learning from Demonstrations* (Argall et al. 2009) and on *Action Advice* (Silva and Costa 2019). The former is usually associated to humans providing a sequence of advice, while the latter consists in agents providing action suggestions for a single step. Hester et al. (2018) combines a supervised learning loss imitating a dataset of demonstrated data with the regular DQN loss, which was shown to accelerate learning. Kartal et al. (2019) uses a planning-based suboptimal demonstrator that continuously provide advice. However, none of those previous works provide advice according to the agent uncertainty as we do. Although RL has a long history involving uncertainty estimation and learning from an expert that predates Deep RL (Garcıa and Fernández 2015), most of those works are either inapplicable or inefficient in domains for which Deep RL is the state-of-the-art solution.

Given the many applications of estimating the agent uncertainty, several works studied how to define epistemic uncertainty measures. Most of them are either based on dropout schemes (Chen et al. 2017) or ensemble of networks (Clements et al. 2019; Osband et al. 2016) as our method. However, those works have other purposes than of using this estimate for defining when to receive advice.

Silva and Costa (2017) propose *Ad Hoc Advising* where agents have a confidence function to define when to ask for or when to give advice, similarly as we do. However, their confidence function is based on the number of visits in each state, which not necessarily corresponds to the uncertainty of the agent model and might lead to unnecessary advice. Ilhan *et al.* (2019) propose a Deep RL version of *Ad Hoc Advising*, estimating visit counts through a Deep Neural Network, which maintains the same characteristics of Silva's work. Thakur *et al.* (2019) use Bayesian Neural Networks to estimate the epistemic uncertainty of the agent and ask for demonstrations based on that, similarly as we do. However, their method imitates the demonstrator in a *Supervised Learning* fashion, whereas our method uses the advice only for exploration and is able to learn a better policy than the demonstrator's. Finally, Lee and Lee (2019) combine demonstrations with RL exploration as we do to find good policies faster. However, they do not estimate agent uncertainty to make better use of advice.

Conclusion and Further Work

We proposed an action-advising framework, named *Requesting Confidence-Moderated Policy advice* (RCMP), to accelerate Reinforcement Learning (RL). Our method estimates the agent epistemic uncertainty for each decision-making step and asks for advice from a demonstrator when the uncertainty is high, avoiding to take decisions where the value function is not expected to have converged yet. RCMP takes into account that the demonstrator might not be available during the whole process and might provide suboptimal advice, hence we use the provided advice to help with exploration, instead of simply trying to copy the demonstrator policy. We also describe a way to estimate the agent epistemic uncertainty through small changes in standard value-function-based RL algorithms. We show that RCMP performs better than learning without advice, with importance advising, and with randomly-timed advice in two domains with different scenarios and levels of complexity.

RCMP opens up several avenues for further work. The first one would be leveraging more complex (principled) ways of estimating uncertainty in deep neural networks (Lakshminarayanan, Pritzel, and Blundell 2017). In addition to existing methods that estimate the uncertainty to purposes other than defining when to receive advice, the recent literature has highlighted the potential of Distributional RL (Bellemare, Dabney, and Munos 2017) to separate aleatoric and epistemic uncertainties (Clements et al. 2019). RCMP could also be improved to make better used of the provided advice, such as devising a cost function moving towards the demonstrated behavior more quickly than just using it for exploration as we do here (Lee and Lee 2019; Hester and others 2018). Finally, the use of epistemic uncertainty could be extended to other purposes, such as devising experience replay (Schaul et al. 2016) algorithms that are more likely to select samples with higher uncertainty.

References

Amir, O.; Kamar, E.; Kolobov, A.; and Grosz, B. 2016. Interactive Teaching Strategies for Agent Training. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 804–811.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A Survey of Robot Learning from Demonstration. *Robotics and Autonomous Systems* 57(5):469 – 483.

Bellemare, M. G.; Dabney, W.; and Munos, R. 2017. A Distributional Perspective on Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*.

Chen, L.; Zhou, X.; Chang, C.; Yang, R.; and Yu, K. 2017. Agent-Aware Dropout DQN for Safe and Efficient On-line Dialogue Policy Learning. In *Conf. on Empirical Methods in Natural Language Processing (EMLNP)*, 2454–2464.

Clements, W. R.; Robaglia, B.; Delft, B. V.; Slaoui, R. B.; and Toth, S. 2019. Estimating Risk and Uncertainty in Deep Reinforcement Learning. *arXiv preprint* arXiv:1905.09638.

Garcıa, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research (JMLR)* 16(1):1437–1480.

Hersch, M.; Guenter, F.; Calinon, S.; and Billard, A. 2008. Dynamical System Modulation for Robot Learning via Kinesthetic Demonstrations. *IEEE Transactions on Robotics* 24(6):1463–1467.

Hester, T., et al. 2018. Deep Q-learning from Demonstrations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence*, 3223–3230.

Ilhan, E.; Gow, J.; and Liebana, D. P. 2019. Teaching on a Budget in Multi-Agent Deep Reinforcement Learning. *arXiv preprint* arXiv:1905.01357.

Kartal, B.; Hernandez-Leal, P.; and Taylor, M. E. 2019. Action Guidance with MCTS for Deep Reinforcement Learning. In *Proceedings of the 15th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 153–159.

Lakshminarayanan, B.; Pritzel, A.; and Blundell, C. 2017. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles.

Lee, H.-R., and Lee, T. 2019. Improved Cooperative Multi-agent Reinforcement Learning Algorithm Augmented by Mixing Demonstrations from Centralized Policy. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 1089–1098.

Mnih, V., et al. 2015. Human-level Control through Deep Reinforcement Learning. *Nature* 518(7540):529–533.

Mnih, V., et al. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 1928–1937.

Omidshafiei, S.; Kim, D.; Liu, M.; Tesauro, G.; Riemer, M.; Amato, C.; Campbell, M.; and How, J. P. 2019. Learning to Teach in Cooperative Multiagent Reinforcement Learning. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI)*.

Osband, I.; Blundell, C.; Pritzel, A.; and Van Roy, B. 2016. Deep Exploration via Bootstrapped DQN. In *Advances in Neural Information Processing Systems (NIPS)*, 4026–4034.

Puterman, M. L. 2005. *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. Hoboken (N. J.): J. Wiley & Sons.

Ross, S.; Melik-Barkhudarov, N.; Shankar, K. S.; Wendel, A.; Dey, D.; Bagnell, J. A.; and Hebert, M. 2013. Learning Monocular Reactive UAV Control in Cluttered Natural Environments. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized Experience Replay. In *International Conference on Learning Representations (ICLR)*.

Schmidhuber, J. 2015. Deep learning in neural networks: An overview. *Neural Networks* 61:85–117.

Silva, F. L. D., and Costa, A. H. R. 2019. A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems. *Journal of Artificial Intelligence Research (JAIR)* 69:645–703.

Silva, F. L. D.; Glatt, R.; and Costa, A. H. R. 2017. Simultaneously Learning and Advising in Multiagent Reinforcement Learning. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1100–1108.

Singh, A.; Yang, L.; Hartikainen, K.; Finn, C.; and Levine, S. 2019. End-to-End Robotic Reinforcement Learning without Reward Engineering. *ArXiv preprint* arXiv:1904.07854.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1st edition.

Sutton, R. S. 1996. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *Advances in Neural Information Processing Systems (NIPS)* 1038–1044.

Taylor, M. E.; Carboni, N.; Fachantidis, A.; Vlahavas, I. P.; and Torrey, L. 2014. Reinforcement Learning Agents Providing Advice in Complex Video Games. *Connection Science* 26(1):45–63.

Thakur, S.; van Hoof, H.; Higuera, J. C. G.; Precup, D.; and Meger, D. 2019. Uncertainty Aware Learning from Demonstrations in Multiple Contexts using Bayesian Neural Networks. *arXiv preprint* arXiv:1903.05697.

Torabi, F.; Warnell, G.; and Stone, P. 2018. Behavioral Cloning from Observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*, 4950–4957.

Torrey, L., and Taylor, M. E. 2013. Teaching on a Budget: Agents Advising Agents in Reinforcement Learning. In *Proceedings of 12th the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1053–1060.

Watkins, C. J., and Dayan, P. 1992. Q-Learning. *Machine Learning* 8(3):279–292.