Agent-Agnostic Human-in-the-Loop Reinforcement Learning

David Abel
Brown University
david_abel@brown.edu

John Salvatier
AI Impacts
jsalvatier@gmail.com

Andreas Stuhlmüller
Stanford University
andreas@stuhlmueller.org

Owain Evans
University of Oxford
owaine@gmail.com

Abstract

Providing Reinforcement Learning agents with expert advice can dramatically improve various aspects of learning. To this end, prior work has developed teaching protocols that enable agents to learn efficiently in complex environments. In many of these methods, the teacher's guidance is tailored to agents with a particular representation or underlying learning scheme, offering effective but highly specialized teaching procedures. In this work, we introduce *protocol programs*, an agent-agnostic schema for Human-in-the-Loop Reinforcement Learning. Our goal is to incorporate the beneficial properties of a human teacher into Reinforcement Learning without making strong assumptions about the inner workings of the agent. We show how to represent existing approaches such as action pruning, reward shaping, and training in simulation as special cases of our schema, and conduct preliminary experiments evaluating the effectiveness of protocols in simple domains.

1 Introduction

A central goal of Reinforcement Learning (RL) is to design agents that learn in a fully autonomous way. An engineer designs a reward function, input/output channels, and a learning algorithm. Then, apart from debugging, the engineer need not intervene during the actual learning process. Yet fully autonomous learning is often infeasible due to the complexity of real-world learning problems, the difficulty of specifying reward functions that lead to good performance, and the presence of potentially dangerous outcomes.

Consider a robot learning to perform household chores. Human engineers design and supervise a curriculum, moving the agent between simulation, practice environments, and real house environments. Over time, they might tweak reward functions, heuristics, and state representations. They may intervene directly in real-world training to prevent the robot damaging itself, destroying valuable goods, or harming people it interacts with.

In this example, humans do not just design the learning agent: they are also "in the loop" of the agent's learning process. Many learning systems have a human in the loop. Self-driving cars learn with humans ready to intervene in dangerous situations. Facebook's algorithm for recommending "trending" news stories has humans filtering out inappropriate content [1]. In both examples, the agent's environment is complex, non-stationary, and there are a wide range of damaging outcomes (like a traffic accident). As RL is applied to increasingly complex real-world problems, such interactive guidance will become more important.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

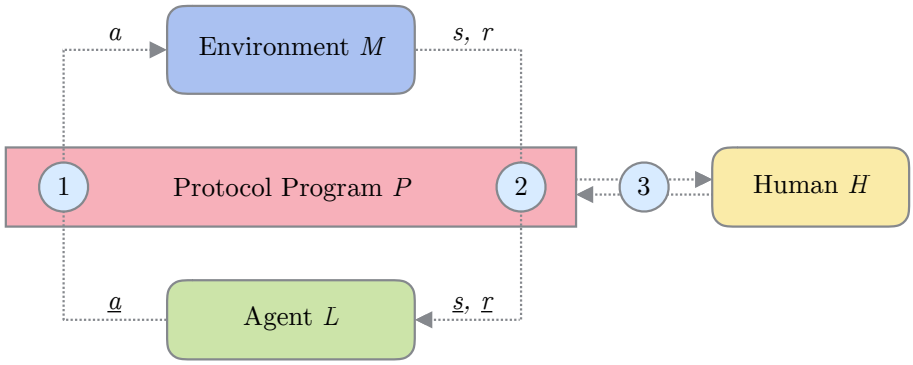

Figure 1: A general setup for RL with a human in the loop. By instantiating P with different protocol programs, we can implement different mechanisms for human guidance of RL agents.

1.1 Our contribution: agent-agnostic guidance of RL algorithms

Our goal is to develop a framework for human-agent interaction that is (a) agent-agnostic and (b) can capture a wide range of ways a human can help an RL agent. This paper targets environments where the state is fully observed; that is, the learning agent interacts with a Markov Decision Process (MDP) [33, 20, 36].

Prior literature has investigated how people can help RL agents learn more efficiently through different methods of interaction [29, 22, 25, 32, 37, 43, 19, 16, 23, 38, 21, 44, 40, 39, 27, 12]. Often, the human's role is to pass along knowledge about relevant quantities of the RL problem, like Q-values, action optimality, or the true reward for a particular state-action pair. This way, the person can bias exploration, prevent catastrophic outcomes, and accelerate learning.

Most existing work develops *agent-specific* protocols for human interaction. That is, protocols for human interaction or advice that are designed for a specific RL algorithm (such as Q-learning). For instance, Griffith et al. [16] investigate the power of policy advice for a Bayesian Q-Learner. Other works assume that the states of the MDP take a particular representation, or that the action space is discrete or finite. Making explicit assumptions about the agent's learning process can enable more powerful teaching protocols that leverage insights about the learning algorithm or representation.

Agent-specific protocols for human interaction can be contrasted with *agent-agnostic* protocols. An agent-agnostic protocol is a single protocol that can be plugged into *any* RL agent, such as an agent based on Q-learning, policy gradient, MCTS, Deep Q-learning, and so on, and that can provide advice/guidance to any such agent. There are obvious disadvantages to agent-agnostic protocols. The agent is not specialized to the protocol, so it is unable to use Active Learning (asking the human informative questions as in [4]) and will not automatically have an observation model that faithfully represents the process the human uses to generate advice (as in [16, 19]). Likewise, the human cannot provide optimally informative advice to the agent as they don't know the agent's prior knowledge, exploration technique, representation, or learning method.

Conversely, agent-specific protocols may perform well for one type of algorithm or environment, but poorly on others. When researchers tackle challenging RL problems, they tend to explore a large space of algorithms with important structural differences: some are model-based vs. model-free, some approximate the optimal policy, others a value function, and so on. It is substantial effort to adapt a human-advice protocol to each such algorithm. Moreover, as advice protocols and learning algorithms become more complex, greater modularity will help limit design complexity.

In our framework, the interaction between the person guiding the learning process, the agent, and the environment is formalized as a *protocol program*. This program controls the *channels* between the agent and the environment based on human input (see Figure 1). This allows the human extensive control over the agent: in an extreme case, the agent can be prevented from interacting with the real environment entirely and only interact with a simulation. At the same time, we require that the human *only* interact with the agent during learning through the protocol program—both agent and environment are a black box to the human.

2 Framework

Any system for RL with a human in the loop has to coordinate three components:

The *environment* is an MDP and is specified by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$, denotes the transition function, a probability distribution on states given a state and action, $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function, and γ is the discount factor.

The *agent* is a (stateful, potentially stochastic) function $L\colon \mathcal{S} \times \mathcal{R} \to \mathcal{A}$.

The *human* can receive and send advice information of flexible type, say X_{in} and X_{out}, so, we will treat the human as a (stateful, potentially stochastic) function $H\colon X_{\text{in}} \to X_{\text{out}}$. For example, X_{in} might contain the history of actions, states, and rewards so far, and a new proposed action a', and X_{out} might be an action as well, either equivalent to a' (if accepted) or different (if rejected). We assume that the human knows in general terms how their responses will be used and is making a good-faith effort to be helpful.

The interaction between the environment, the agent, and a human advisor sets up a mechanism design problem: how can we design an interface that orchestrates the interaction between these components such that the combined system maximizes the expected sum of γ-discounted rewards from the environment? In other words, how can we write a protocol program $P\colon \mathcal{S} \times \mathcal{R} \to \mathcal{A}$ that can take the place of a given agent L, but that achieves higher rewards by making efficient use of information gained through sub-calls to L and H?

By formalizing existing and new techniques as explicit programs, we facilitate understanding and comparison of these techniques within a common framework. By abstracting from particular agents and environments, we may be able to design mechanisms that scale to more advanced agents [7].

Algorithm 1 Agent in control (standard)

procedure AGENTCONTROL(s, r)
  return $L(s, r)$
end procedure

Algorithm 2 Human in control

procedure HUMANCONTROL(s, r)
  return $H(s, r)$
end procedure

Algorithm 3 Training in simulation

$M^* = (\mathcal{S}, \mathcal{A}, \mathcal{T}^*, \mathcal{R}^*, \gamma)$ ▷ Simulation
$\eta = []$ ▷ History: array of $(\mathcal{S} \times \mathcal{R} \times \mathcal{A})$
procedure TRAININSIMULATION(s, r)
  $\underline{s} = s$
  $\underline{r} = r$
  while $H(\eta) \neq$ "agent is ready" **do**
    $\underline{a} = L(\underline{s}, \underline{r})$
    append $(\underline{s}, \underline{r}, \underline{a})$ to η
    $\underline{r} \sim \mathcal{R}^*(\underline{s}, \underline{a})$
    $\underline{s} \sim \mathcal{T}^*(\underline{s}, \underline{a})$
  end while
  return $L(s, r)$
end procedure

Algorithm 4 Action pruning

$\Delta = \{\}$ ▷ Pruned: $\mathcal{S} \times \mathcal{A} \to \mathcal{A} \times \mathcal{R}$
$r^* = \varnothing$ ▷ Reward if previous action pruned
procedure PRUNEACTIONS(s, r)
  $\underline{r} = r$ **if** $r^* = \varnothing$ **else** r^*
  $\underline{a} = L(s, \underline{r})$
  if $(s, \underline{a}) \in \Delta$ **then**
    $(a, r^*) = \Delta[(s, \underline{a})]$
  else
    $(a, r^*) = H(s, r, \underline{a})$
    if $a = \varnothing$ **then**
      $a = \underline{a}$
      $r^* = \varnothing$
    else
      $\Delta[(s, \underline{a})] = (a, r^*)$
    end if
  end if
  return a
end procedure

Algorithm 5 Reward manipulation

procedure MANIPULATEREWARD(s, r)
  $\underline{r} = H(s, r)$
  return $L(s, \underline{r})$
end procedure

Figure 2: Many schemes for human guidance of RL algorithms can be expressed as protocol programs. These programs have the same interface as the agent L, but can be safer or more efficient learners by making use of human advice H.

3 Capturing Existing Advice Schemes

Protocol programs cannot capture *all* advice protocols. Any protocol that depends on prior knowledge of the agent's learning algorithm, priors or hyper-parameters is ruled out. Despite this constraint, the framework can capture a range of existing protocols where a human-in-the-loop guides an agent. Figure 1 shows that the human can manipulate the *actions* sent to the environment and the agent's observed *states* and *rewards*. This points to a combinatorial set of protocols, including *reward shaping*, *action pruning*, *providing hand-coded features of state*, and *training in simulation*.

3.1 Reward shaping

Section 2 defined the reward function $\mathcal{R}$ as part of the MDP M. However, while humans don't design real-world transition functions, we do design reward functions. Usually the reward function is hand-coded prior to learning and must accurately assign reward values to any state the agent might reach. An alternative is to have a human generate the rewards "interactively". The human observes the state and action and returns a scalar to the agent. This setup has been explored in work on TAMER [21]. A similar setup (with an agent-specific protocol) was applied to robotics by Daniel et al. [8]. It is straightforward to represent rewards that are generated interactively (or "online") using protocol programs.

We now turn to other protocols in which the human manipulates rewards. These protocols assume a fixed reward function $\mathcal{R}$ that is part of the MDP M.

3.1.1 Reward shaping and Q-value initialization

In Reward Shaping protocols, the human engineer changes the rewards given by some fixed reward function in order to influence an agent's learning. Ng et al. [29] introduced "potential-based" shaping, which shapes rewards without changing an MDP's optimal policy. In particular, each reward received by the environment is augmented by a shaping function:

$$F(s, a, s') = \gamma\phi(s') - \phi(s), \tag{1}$$

so the agent actually receives $\underline{r} = F(s, a, s') + \mathcal{R}(s, a)$. Wiewiora et al. [43] showed potential shaping to be equivalent (for Q-learners) to a subset Q-value initialization under some assumptions. Further, Devlin and Kudenko [10] propose *dynamic* potential shaping functions that change over time. That is, the shaping function F also takes as two time parameters, t and t', such that:

$$F(s, t, s', t') = \gamma\phi(s', t') - \phi(s, t) \tag{2}$$

Where $t' > t$. Their main result is that dynamic shaping functions of this form also guarantee optimal policy invariance. Similarly, Wiewiora et al. [43] extend potential shaping to *potential-based advice* functions, which identifies a similar class of shaping functions on (s, a) pairs.

In Section 4, we show that our Framework captures reward shaping, and consequently, some notion of Q-value initialization.

3.2 Training in Simulation

It is common practice to train an agent in simulation and transfer it to the real world once it performs well enough. Algorithm 3 (Figure 2) shows how to represent this as a protocol program. The idea is as follows: Let M be the real-world'. The protocol program includes a simulator M^*. The agent L interacts with M^* while the human observes performance and at some point transfers L to M.[1]

3.3 Action Pruning

Action pruning is a technique for dynamically removing actions from the MDP to reduce the branching factor of the search space. Such techniques have been shown to accelerate learning and planning time [35, 17, 34, 2]. In Section 5, we apply action-pruning to the problem of preventing catastrophic actions ("Safe RL").

[1]The human may need to wait for the start of an episode to do the transfer. Otherwise the agent may experience an impossible state-transition.

Protocol programs allow action pruning to be carried out *interactively*. Instead of having to decide which actions to prune prior to learning, the human can wait to observe the states that are actually encountered by the agent, which may be valuable in cases where the human has limited knowledge of the environment or the agent's learning ability. In Section 4, we exhibit an agent-agnostic protocol for interactively pruning actions that preserves the optimal policy.

Our pruning protocol is illustrated in a gridworld with lava pits (Figure 3). The agent is represented by a gray circle, "G" is a goal state that provides reward +1, and the red cells are lava pits with reward -200. All white cells provide reward 0.

At each time step, the human checks whether the agent moves into a lava pit. If it does not (as in moving "DOWN" from state **34**), the agent continues as normal. If it does (as in moving "RIGHT" from state **33**), the human sends a "STAY" action to the MDP (preventing movement right) and a "next state" of **33** to the agent. The agent doesn't actually fall in the lava but the human sends them a reward $r \leq -200$. After this negative reward, the agent is unlikely to try the action again. For the protocol program, see Algorithm 4 in Figure 2.

Note that the agent receives no explicit signal that their attempted catastrophic action was blocked by the human. They observe a big negative reward and a "self-loop" state transition but no information about whether the human or environment generated their observation.

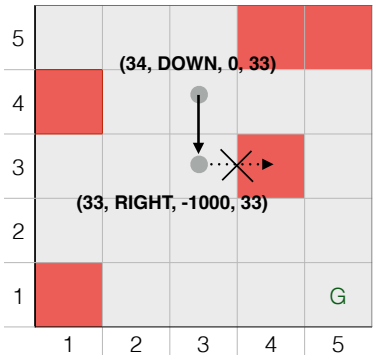

Figure 3: The human allows movement from state **34** to **33** but blocks agent from falling in lava (at **43**).

3.4 Manipulating state representation

The agent's state representation can have a significant influence on its learning. Suppose the MDP M has a "raw" state vector s. The human engineer can specify a mapping ϕ such that the agent always receives $\phi(s) = \bar{s}$ in place of s. Such mappings are used to specify high-level features of state that are important for learning, or to dynamically ignore confusing features from the agent. Other methods have focused on state abstraction functions of this form that can decrease learning time and preserve the quality of learned behavior, as in [24, 30, 13, 18, 9, 3, 14].

4 Theory

Here we illustrate some simple ways in which our proposed agent-agnostic interaction scheme captures other existing agent-agnostic protocols. The following results all concern Tabular MDPs, but are intended to offer intuition for high-dimensional or continuous environments as well.

4.1 Reward Shaping

First we observe that protocol programs can precisely capture methods for shaping reward functions.

Remark 1: *For any reward shaping function F, including potential-based shaping, potential-based advice, and dynamic potential-based advice, there is a protocol that produces the same rewards.*

To construct such a protocol for a given F, simply let the reward output by the protocol, $\underline{r}$, take on the value $F + r$ at each time step. That is, in Algorithm 5, simply define $H(s, \underline{r}) = F(s) + r$.

4.2 Action Pruning

We now show that there is a simple class of protocol programs that carry out action pruning of a certain form.

Remark 2: *There is a protocol for pruning actions in the following sense: for any set of state action pairs $\boldsymbol{sa} \subset S \times A$, the protocol ensures that, for each pair $(s_i, a_j) \in \boldsymbol{sa}$, action a_j is never executed*

in the MDP in state s_i.

The protocol is as described in Section 3.3 and shown in Algorithm 4. The premise is this: in all cases where the agent executes an action that should be pruned, the protocol gives the agent low reward and forces the agent to self-loop.

Knowing which actions to prune is itself a challenging problem. Often, it is natural to assume that the human guiding the learner knows something about the environment of interest (such as where high rewards or catastrophes lie), but may not know every detail of the problem. Thus, we consider a case in which the human has partial (but useful) knowledge about the problem of interest, represented as an approximate Q-function. The next remark shows there is a protocol based on approximate knowledge with two properties: (1) it never prunes an optimal action, (2) it limits the magnitude of the agent's worst mistake:

Remark 3: *Assuming the protocol designer has a β-optimal Q function:*

$$||Q^*(s,a) - Q_H(s,a)||_\infty \leq \beta \tag{3}$$

there exists a protocol that never prunes an optimal action, but prunes all actions so that the agent's mistakes are never more than 4β below optimal. That is, for all times t:

$$V^{L_t}(s_t) \geq V^*(s_t) - 4\beta, \tag{4}$$

where L_t is the agent's policy after t timesteps.

Proof of Remark 3. The protocol designer has a β-approximate Q function, denoted Q_H, defined as above. Consider the state-specific action pruning function $H(s)$:

$$H(s) = \left\{ a \in \mathcal{A} \mid Q_H(s,a) \geq \max_{a'} Q_H(s,a') - 2\beta \right\} \tag{5}$$

The protocol prunes all actions not in $H(s)$ according to the self-loop method described above. This protocol induces a pruned Bellman Equation over available actions, $H(s)$, in each state:

$$V_H(s) = \max_{a \in H(s)} \left(\mathcal{R}(s,a) + \gamma \sum_{s'} \mathcal{T}(s,a,s') V_H(s') \right) \tag{6}$$

Let a^* denote the true optimal action: $a^* = \arg\max_{a'} Q^*(s,a')$. To preserve the optimal policy, we need $a^* \in H(s)$, for each state. Note that $a^* \notin H(s)$ when:

$$Q_H(s,a^*) < \max_{a'} Q_H(s,a') - 2\beta \tag{7}$$

But by definition of $Q_H(s,a)$:

$$|Q_H(s,a^*) - \max_{a} Q_H(s,a)| \leq 2\beta \tag{8}$$

Thus, $a^* \in H(s)$ can never occur. Furthermore, observe that $H(s)$ retains all actions a for which:

$$Q_H(s,a) \geq \max_{a'} Q_H(s,a') - 2\beta, \tag{9}$$

holds. Thus, in the worst case, the following two hold:

1. The optimal action estimate is β too low: $Q_H(s,a^*) = Q^*(s,a^*) - \beta$
2. The action with the lowest value, a_{bad}, is β too high: $Q_H(s,a_{bad}) = Q^*(s,a_{bad}) + \beta$

From Equation 9, observe that the minimal $Q^*(s,a_{bad})$ such that $a_{bad} \in H(s)$ is:

$$Q^*(s,a_{bad}) + \beta \geq Q^*(s,a^*) - \beta - 2\beta$$
$$\therefore Q^*(s,a_{bad}) \geq Q^*(s,a^*) - 4\beta$$

Thus, this pruning protocol never prunes an optimal action, but prunes all actions worse then 4β below a^* in value. We conclude that the agent may never execute an action 4β below optimal. □

5 Experiments

This section applies our action pruning protocols (Section 3.3 and Remarks 2 and 3 above) to concrete RL problems. In Experiment 1, action pruning is used to prevent the agent from trying catastrophic actions ("safe exploration"). In Experiment 2, action pruning is used to accelerating learning.

5.1 Protocol for Preventing Catastrophes

As noted in the Introduction, human-in-the-loop RL can help prevent disastrous outcomes that result from ignorance of the environment's dynamics or of the reward function. Our goal for this experiment is to prevent the agent from taking *catastrophic actions*. These are real-world actions so costly that we want the agent to *never* take the action[2]. This notion of catastrophic action is closely related to ideas in "Safe RL" [15, 28] and to work on "significant rare events" [31].

Section 3.3 describes our protocol program for preventing catastrophes in finite MDPs using action pruning. There are two important elements of this program:

1. When the agent tries a catastrophic action (s, a), a is blocked and the new state and reward is (s, r_{bad}), where r_{bad} is an extreme negative reward.
2. This (s, a) is stored so that the protocol program can automate the human's intervention, which could allow the human to stop monitoring after all catastrophes have been stored.

This protocol prevents catastrophic actions while preserving the optimal policy and having only minimal side-effects on the agent's learning. We can extend this protocol to environments with high-dimensional state spaces. Element (1) above remains the same. But (2) must be modified: preventing future catastrophes requires *generalization* across catastrophic actions (as there will be infinitely many such actions). We briefly discuss this setting in Appendix A.

5.2 Experiment 1: Preventing Catastrophes in a Pong-like Game

Our protocol for preventing catastrophes is intended for use in a real-world environment. Here we provide a preliminary test of our protocol in a simple video game.

Our protocol treats the RL agent as a black box. So we applied our protocol to an open-source implementation of the state-of-the-art RL algorithm "Trust Region Policy Optimization" [**?**]. The environment was *Catcher*, a simplified version of Pong with non-visual state representation. Since there are no "catastrophic" actions in Catcher, we modified the game to give a large negative reward when the paddle's speed exceeds a speed limit.[3] We compare the performance of an agent who has assistance by the protocol ("Pruned") and so is blocked from the catastrophic actions to the performance of a normal RL agent ("Not Pruned").

Figure 4 shows the agent's mean performance over the course of learning. We see that the agent with protocol support ("Pruned") performed much better overall. This is unsurprising, as it was blocked from ever doing a catastrophic action. The gap in mean performance is large early on but diminishes as the "Not Pruned" agent learns to avoid high speeds. By the end (i.e. after 400,000 actions), "Not Pruned" is close to "Pruned" in mean performance but its total returns over the whole period are around 20 times worse. Note that the "Pruned" agent observes "odd" state transitions due to being blocked by our protocol. We saw no evidence of these observations having negative side effects on learning.

5.3 Protocol for Accelerating Learning

We also conducted a simple experiment in the Taxi domain, introduced by Dietterich [11]. The Taxi problem is a more complex version of grid world: the agent directs a taxi to each passenger, picks them up, and brings them to their destination. We use Taxi to evaluate the effect of our action pruning protocol for discrete MDPs. There is a natural procedure for pruning suboptimal actions that

[2] We allow an RL agent to take sub-optimal actions while learning. Catastrophic actions are not allowed because their cost is orders of magnitude worse than non-catastrophic actions.

[3] In this toy environment, a human observer could easily recognize catastrophic actions: they just need to watch the speed of the paddle.

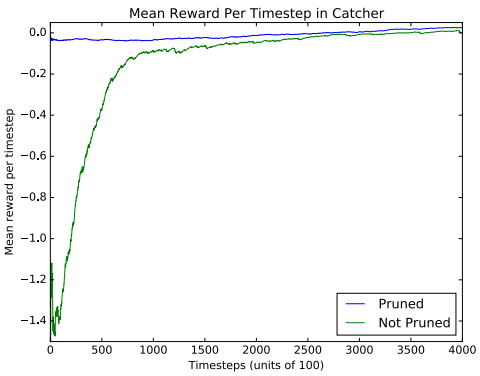

Figure 4: Preventing Catastrophic Speeds

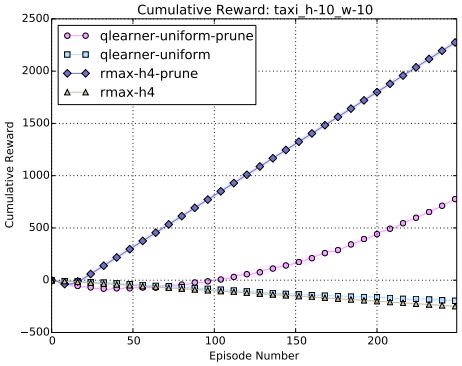

Figure 5: Pruning in Taxi.

dramatically reduces the size of the reachable state space: if the taxi is carrying a passenger but is not at the passenger's destination, we prune the `dropoff` action by returning the agent back to its current state with -0.01 reward. This prevents the agent from exploring a large portion of the state space, thus accelerating learning.

5.4 Experiment 2: Accelerated Learning in Taxi

We evaluated Q-learning [41] and R-MAX [5] with and without action pruning in a simple 10×10 instance with one passenger. The taxi starts at $(1, 1)$, the passenger at $(4, 3)$ with destination $(2, 2)$. We ran standard Q-Learning with ε-greedy exploration with $\varepsilon = 0.2$ and with R-MAX using a planning horizon of four. Results are displayed in Figure 5.

Our results suggest that the action pruning protocol simplifies the problem for a Q-Learner and dramatically so for R-Max. In the allotted number of episodes, we see that pruning substantially improves the overall cumulative reward achieved; in the case of R-MAX, the agent is able to effectively solve the problem after a small number of episodes. Further, the results suggests that the agent-agnostic method of pruning is effective without having any internal access to the agent's code.

6 Conclusion

We presented an agent-agnostic method for giving guidance to Reinforcement Learning agents. Protocol programs written in this framework apply to any possible RL agent ("agent-agnosticism") so sophisticated schemes for human-agent interaction can be designed in a modular fashion without the need for adaptation to different RL algorithms. We presented some simple theoretical results that relate our method to existing schemes for interactive RL and illustrated the power of action pruning using one such protocol on two toy domains.

In the future, we plan on exploring state manipulation protocols, which offer a powerful avenue for guiding an agent's learning process. For instance, one could imagine a protocol dynamically imposing abstractions that hide confusing features from the agent and encourage efficient learning. Additionally, we want to investigate whether certain types of value initialization protocols can be captured by protocol programs, such as the optimistic initialization for arbitrary domains developed by Machado et al. [26].

Acknowledgments

This work was supported by Future of Life Institute grant 2015-144846 and by the Future of Humanity Institute (Oxford). We thank Shimon Whiteson, James MacGlashan, and D. Ellis Herskowitz for helpful conversations.

References

[1] How does facebook determine what topics are trending? `https://www.facebook.com/help/737806312958641`. Accessed: 2016-10-12.

[2] David Abel, David Ellis Hershkowitz, Gabriel Barth-Maron, Stephen Brawner, Kevin O'Farrell, James MacGlashan, and Stefanie Tellex. Goal-based action priors. In *ICAPS*, pages 306–314, 2015.

[3] David Abel, D Ellis Hershkowitz, and Michael L. Littman. Near optimal behavior via approximate state abstraction. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

[4] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara Grosz. Interactive teaching strategies for agent training. IJCAI, 2016.

[5] Ronen I Brafman and Moshe Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

[6] Denny Britz. Learning reinforcement learning. `http://www.wildml.com/2016/10/learning-reinforcement-learning/`, 2016. Accessed: 2016-10-13.

[7] Paul Christiano. Scalable ai control. `https://medium.com/ai-control/scalable-ai-control-7db2436feee7`. Accessed: 2016-10-13.

[8] Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Proceedings of Robotics Science & Systems*, 2014.

[9] Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 124–131. Morgan Kaufmann Publishers Inc., 1997.

[10] Sam Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, (June):433–440, 2012.

[11] Thomas G Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[12] Kurt Driessens and Sašo Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, 2004.

[13] Eyal Even-Dar and Yishay Mansour. Approximate equivalence of Markov decision processes. In *Learning Theory and Kernel Machines*, pages 581–594. Springer, 2003.

[14] Norman Ferns, Pablo Samuel Castro, Doina Precup, and Prakash Panangaden. Methods for computing state similarity in markov decision processes. *Proceedings of the 22nd conference on Uncertainty in artificial intelligence*, 2006.

[15] Javier Garcia and Fernando Fernandez. A Comprehensive Survey on Safe Reinforcement Learning. *The Journal of Machine Learning Research*, 16:1437–1480, 2015.

[16] Shane Griffith, Kaushik Subramanian, and J Scholz. Policy Shaping: Integrating Human Feedback with Reinforcement Learning. *Advances in Neural Information Processing Systems (NIPS)*, pages 1–9, 2013.

[17] Eric A Hansen, Andrew G Barto, and Shlomo Zilberstein. Reinforcement learning for mixed open-loop and closed-loop control. In *NIPS*, pages 1026–1032. Citeseer, 1996.

[18] Nicholas K Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, pages 752–757, 2005.

[19] Kshitij Judah, Saikat Roy, Alan Fern, and Thomas G Dietterich. Reinforcement Learning Via Practice and Critique Advice. *AAAI*, pages 481–486, 2010.

[20] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, pages 237–285, 1996.

[21] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16. ACM, 2009.

[22] W Bradley Knox and Peter Stone. Augmenting reinforcement learning with human feedback. *Proceedings of the ICML Workshop on New Developments in Imitation Learning*, page 8, 2011.

[23] Pradyot K.V.N. Beyond Rewards : Learning from Richer Supervision. (August), 2012.

[24] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *ISAIM*, 2006.

[25] Robert Loftin, Bei Peng, James MacGlashan, Michael L Littman, Matthew E Taylor, Jie Huang, and David L Roberts. Learning something from nothing: Leveraging implicit human feedback strategies. In *Robot and Human Interactive Communication, 2014 RO-MAN: The 23rd IEEE International Symposium on*, pages 607–612. IEEE, 2014.

[26] Marlos C. Machado, Sriram Srinivasan, and Michael Bowling. Domain-Independent Optimistic Initialization for Reinforcement Learning. *AAAI Workshop on Learning for General Competency in Video Games*, 2014.

[27] Richard Maclin and Jude W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22:251–281, 1996.

[28] Teodor Mihai Moldovan and Pieter Abbeel. Safe Exploration in Markov Decision Processes. *Proceedings of the 29th International Conference on Machine Learning*, 2012.

[29] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[30] Ronald Ortner. Adaptive aggregation for reinforcement learning in average reward Markov decision processes. *Annals of Operations Research*, 208(1):321–336, 2013.

[31] Supratik Paul, Kamil Ciosek, Michael A Osborne, and Shimon Whiteson. Alternating optimisation and quadrature for robust reinforcement learning. *arXiv preprint arXiv:1605.07496*, 2016.

[32] Bei Peng, James MacGlashan, Robert Loftin, Michael L Littman, David L Roberts, and Matthew E Taylor. A need for speed: Adapting agent action speed to improve task learning from non-expert humans. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 957–965. International Foundation for Autonomous Agents and Multiagent Systems, 2016.

[33] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[34] Benjamin Rosman and Subramanian Ramamoorthy. What good are actions? accelerating learning using learned action priors. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–6. IEEE, 2012.

[35] A.A. Sherstov and P. Stone. Improving action selection in mdp's via knowledge transfer. In *Proceedings of the 20th national conference on Artificial Intelligence*, pages 1024–1029. AAAI Press, 2005.

[36] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[37] Andrea Lockerd Thomaz and Cynthia Breazeal. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. *Aaai*, 6:1000–1005, 2006.

[38] Lisa Torrey and Matthew 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*, pages 1053–1060. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[39] Lisa Torrey and Matthew E. Taylor. Help an agent out: Student/teacher learning in sequential decision tasks. *Proceedings of the Adaptive and Learning Agents Workshop 2012, ALA 2012 - Held in Conjunction with the 11th International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012*, pages 41–48, 2012.

[40] Thomas J. Walsh, Daniel Hewlett, and Clayton T Morrison. Blending Autonomous Exploration and Apprenticeship Learning. *Advances in Neural Information Processing Systems 24*, pages 2258–2266, 2011.

[41] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[42] Eric Wiewiora. Potential-Based Shaping and Q-Value Initialization are Equivalent. *Artificial Intelligence*, 19:205–208, 2003.

[43] Eric Wiewiora, Garrison Cottrell, and Charles Elkan. Principled methods for advising reinforcement learning agents. In *ICML*, pages 792–799, 2003.

[44] Yusen Zhan, Haitham Bou Ammar, and Matthew E. Taylor. Theoretically-Grounded Policy Advice from Multiple Teachers in Reinforcement Learning Settings with Applications to Negative Transfer. pages 1–10, 2016.

A Protocol program for preventing catastrophes in high-dimensional state spaces

We provide an informal overview of the protocol program for avoiding catastrophes. We focus on the differences between the high-dimensional case and the finite case described in Section 5.1. In the finite case, pruned actions are stored in a table. When the human is satisfied that all catastrophic actions are in the table, the human's monitoring of the agent can be fully automated by the protocol program. The human may need to be in the loop until the agent has attempted each catastrophic action once – after that the human can "retire".

In the infinite case, we replace this look-up table with a supervised classification algorithm. All visited state-actions are stored and labeled ("catastrophic" or "not catastrophic") based on whether the human decides to block them. Once this labeled set is large enough to serve as a training set, the human trains the classifier and tests performance on held-out instances. If the classifier passes the test, the human can be replaced by the classifier. Otherwise the data-gathering process continues until the training set is large enough for the classifier to pass the test.

If the class of catastrophic actions is learnable by the classifier, this protocol prevents all catastrophes and has minimal side-effects on the agent's learning. However, there are limitations of the protocol that will be the subject of future work:

- The human may need to monitor the agent for a very long time to provide sufficient training data. One response to this issue is for the human to augment the training set by adding synthetically generated states to it. For example, the human might add noise to genuine states in such a way that doesn't change the labels. Alternatively, the human might have an accurate generative model for states and augment with states drawn from the model.
- Some catastrophic outcomes have a "local" cause that is easy to block. If a car moves very slowly, then it can avoid hitting an obstacle by braking at the last second. But if a car has lots of momentum, it cannot be slowed down quickly enough. In such cases a human in-the-loop would have to recognize the danger some time before the actual catastrophe would take place.
- To prevent catastrophes from ever taking place, the classifier needs to correctly identify every catastrophic action. This requires strong guarantees about the generalization performance of the classifier. Yet the distribution on state-action instances is non-stationary (violating the usual i.i.d assumption).