Reinforcement Learning As End-User Trigger-Action Programming

Chace Hayhurst, Hyojae Park, Atrey Desai, Suheidy De Los Santos, Michael Littman

Brown University, Computer Science Department
{chace_hayhurst, hyojae_park, atrey_sanjeev_desai, suheidy_de_los_santos, michael_littman}@brown.edu

Abstract

We contend that the power of reinforcement learning comes from its fundamental declarative nature, allowing a system designer to consider *what* an agent's objective is instead of the details of *how* this objective can ultimately be achieved. This abstract provides some early design ideas for creating an end-user-oriented reinforcement-learning system based on the trigger-action programming model. We propose a study designed to highlight the similarities and differences of this end-user-based reinforcement-learning language to more established end-user trigger-action programming.

Introduction

Historically, much of research within the reinforcement-learning community has been directed at applying and designing algorithms to create intelligent agents that solve specific problems [Sutton and Barto 1998]. In recent years, this approach to reinforcement learning (RL) has produced exemplary results, with engineers being able to create agents that rival or even surpass the best human-level performances on some problems [Mnih et al. 2015, Silver et al. 2016]. In contrast, little effort has been put into studying how non-experts can interact with these systems.

An RL "programmer" needs to identify three things to the algorithm: (1) the *actions* an agent can take in the environment, (2) the *state variables* of the environment the agent should be concerned with, and (3) the reward function, or more simply, a *goal* that the agent should complete. That is, while RL *researchers* typically take actions, states, and goals as given and focus on how to design agents that can take actions to achieve goal states, RL *users* are the ones responsible for defining these parameters in the first place. Bad choices can lead to intractable learning problems because of either under-specification (critical aspects of the problem are not accessible by the learner) or over-specification (too many details are given to the learner, making learning and generalization difficult).

For some tasks and for some users, finding the right actions, states, and goals may be considerably easier than explicitly articulating the choices the agent should make. For others, perhaps not. Our research objective is understanding where that line might be.

Our baseline for comparisons is *trigger-action programming* [Ur et al. 2014, Huang and Cakmak 2015, Zhang et al. 2020], a methodology in broad use that allows end users to specify behaviors for data analysis, smart home devices, and other applications. A trigger-action program (TAP) consists of a set of rules, each of which has a trigger (something that has become true of the world) and action (an intervention that should be taken in response). Triggers can be primitive events or a primitive event combined with one or more conditions. An example TAP rule in the home-automation domain is: "**If** I arrive home (trigger event) **while** the indoor temperature is above 75 degrees (trigger condition) **then** turn on the AC (action)." Existing research shows that end users with no programming experience are able to construct and interpret TAPs, providing some support for the contention that this style of programming strikes a useful balance between ease of use and expressive power.

In this work, we observe that the actions in TAP are analogous to the actions in RL, while the triggers are akin to states. Thus, an RL task can be specified by a set of triggers (which constitute the learner's states), actions (which constitute the learner's actions), and a special trigger (which acts as the learner's goal).

Proposed Interface

Our interface is a modified version of the AutoTap project created by Zhang et al. (2019) to support the construction of TAPs. We have repurposed this framework to provide actions and triggers for a scenario in which a mobile robot is tasked with moving boxes throughout a house. We created two interfaces: TAP and RL. In both versions, users are greeted with a page allowing them to create new rules that guide agent behavior using the associated programming style.

In the TAP version, the user creates rules that pair a trigger and an action. (See Figure 1.)

In the RL version (see Figure 2), the user specifies a set of parts, each of which belongs to one of three selected categories: 'Consider doing:', 'Pay attention to:', and 'Get a 'yes' answer to:'. The part finishes the clause started by the corresponding category. Parts take the form of conditions that can be true or false given the current state of the envi-

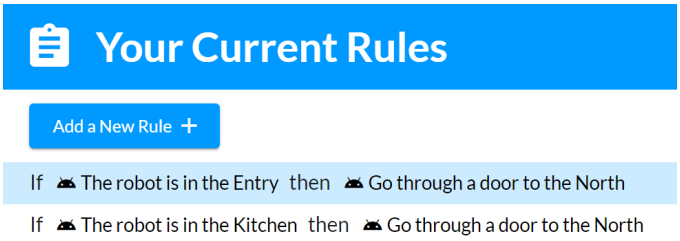

Figure 1: An example of a TAP ruleset in our system.

ronment or actions that can be performed by the agent. By combining a category with a part, a user specifies an action that the agent can take ('Consider doing:'), a state variable the agent should be concerned with ('Pay attention to:') or the goal of the agent ('Get a 'yes' answer to:').

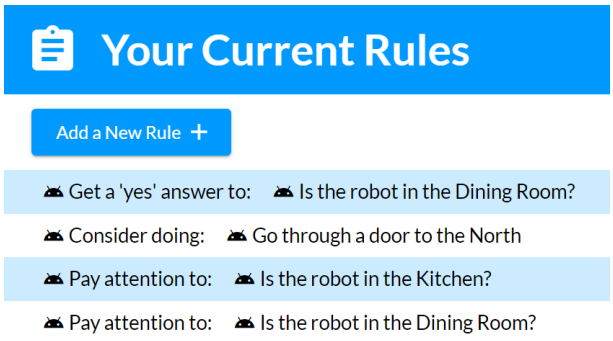

Figure 2: An example of an RL ruleset in our system.

Proposed Experiment Design

Our experiment will involve two groups of 20 participants with no prior experience in programming. One group will randomly be assigned to the TAP condition while the other will be assigned to the RL condition. Both groups will receive instructions on how to access the website with our program-creation framework.

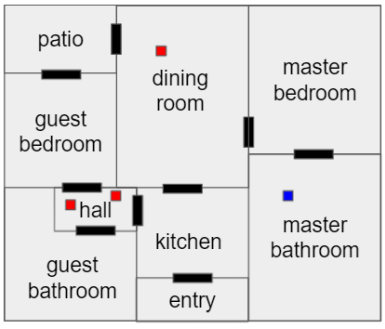

Figure 3: Floorplan of the house for our experiments.

Users would then be given four programming tasks of increasing difficulty and tasked with writing rules to control an agent to solve each task. The tasks involve a robot navigating through a house and moving items around in it (Figure 3).

Table 1: Proposed tasks for our experiments.

Instruction	*TAP rules*	*RL parts*
Starting from the entry, go to the kitchen.	1	3
Wherever you start, go to the master bedroom.	8	12
Starting from the entry, bring back the blue box from the master bathroom.	7	13
Starting from the entry, clear all of the red boxes out of the hall.	6	12

Triggers in the system include 'Robot is in room X', 'There is a red/blue block in room X', and 'Robot is holding a red/blue block'. Actions that the robot can take include 'Go through a door to the North/South/East/West', 'Pick up a red/blue block in your current room', and 'Put down held block'.

Table 1 provides an example set of tasks and the number of TAP rules or RL parts needed to solve them. Although the number of required RL parts is consistently larger than the TAP rule sets, the assembly of RL parts is performed by the learner, which may make it easier to select them.

After users finish writing their programs, they submit them to our database for analysis. Solutions that miss components that are necessary to complete the task or put agents into situations where they have no available actions will be deemed incomplete. Solutions will be graded according to a rubric determined in advance for each task.

We are interested in statistics such as: How often does each programming style succeed on each task? Is there any pattern to how the success rates change as the task difficulty increases? Is a certain programming style better suited to certain tasks? How often do participants include unnecessary components? How often do they miss required components? Are there consistent errors that participants make that might be reduced through careful interface design? Ultimately, we are interested in assessing if RL can be a viable programming language for end users.

References

Huang, J.; and Cakmak, M. 2015. Supporting Mental Model Accuracy in Trigger-Action Programming. In *ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp)*.

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

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.;

Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587): 484–489.

Sutton, R. S.; and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. The MIT Press.

Ur, B.; McManus, E.; Ho, M. P. Y.; and Littman, M. L. 2014. Practical Trigger-Action Programming in the Smart Home. In *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*.

Zhang, L.; He, W.; Martinez, J.; Brackenbury, N.; Lu, S.; and Ur, B. 2019. AutoTap: Synthesizing and repairing trigger-action programs using LTL properties. In *IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 281–291.

Zhang, L.; He, W.; Morkved, O.; Zhao, V.; Littman, M. L.; Lu, S.; and Ur, B. 2020. Trace2TAP: Synthesizing Trigger-Action Programs from Traces of Behavior. In *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, volume 4, 104:1–104:26.