Teaching a Robot Tasks of Arbitrary Complexity via Human Feedback

Guan Wang
Department of Computer Science
Brown University
Providence, Rhode Island
guan_wang@brown.edu

Carl Trimbach
Department of Computer Science
Brown University
Providence, Rhode Island
carl_trimbach@brown.edu

Jun Ki Lee
Department of Computer Science
Brown University
Providence, Rhode Island
jun_ki_lee@brown.edu

Mark K. Ho
Department of Computer Science
Princeton University
Princeton, New Jersey
mho@princeton.edu

Michael L. Littman
Department of Computer Science
Brown University
Providence, Rhode Island
mlittman@cs.brown.edu

ABSTRACT

This paper addresses the problem of training a robot to carry out temporal tasks of arbitrary complexity via evaluative human feedback that can be inaccurate. A key idea explored in our work is a kind of curriculum learning—training the robot to master simple tasks and then building up to more complex tasks. We show how a training procedure, using knowledge of the formal task representation, can decompose and train any task efficiently in the size of its representation. We further provide a set of experiments that support the claim that non-expert human trainers can decompose tasks in a way that is consistent with our theoretical results, with more than half of participants successfully training all of our experimental missions. We compared our algorithm with existing approaches and our experimental results suggest that our method outperforms alternatives, especially when feedback contains mistakes.

CCS CONCEPTS

• **Theory of computation → Reinforcement learning**; Modal and temporal logics; • **Computing methodologies → Learning from critiques**; • **Human-centered computing →** *Interaction paradigms.*

KEYWORDS

reinforcement learning; human-robot interaction; learning from human feedback; linear temporal logic

ACM Reference Format:
Guan Wang, Carl Trimbach, Jun Ki Lee, Mark K. Ho, and Michael L. Littman. 2020. Teaching a Robot Tasks of Arbitrary Complexity via Human Feedback. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction (HRI'20), March 23–26, 2020, Cambridge, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3319502.3374824

1 INTRODUCTION

Reinforcement learning is a powerful mechanism for creating robots with desired behaviors. However, autonomous learning systems can require prohibitive amounts and experience and require the target user to specify the agent's mission in the form of a reward function. In creating learning systems that can carry out a variety of behaviors for people, we can take inspiration from dog training. A trainer can convey a seemingly unbounded collection of tasks to a dog using essentially only evaluative feedback. One significant tool in the trainer's collection is expanding on learned tasks later in training. The site "Doggy Buddy"[1], provides instructions for training 52 tricks using this kind of curricular training. Learning to fetch a drink from the fridge, for example, builds upon first training the dog to complete a set of 8 other tricks.

Interactive reinforcement learning employs human trainers as a source of feedback [2, 9, 13, 16, 17, 24, 32, 33]. In one view, training can be viewed as a form of communication [11, 12] in which the trainer wishes to convey a target task to the learning agent and the agent wishes to infer this task and behave accordingly [10]. The work on SABL [23] makes this perspective explicit and we adopt it in the current work. Past work has also looked at training algorithms that can learn from evaluative feedback. For the most part, these algorithms concentrate on learning reward functions that map state and action to either a reward function or an action. Here, we focus on an internal representation of tasks that supports a boundless collection of meaningfully different tasks. Our approach also handles trainer error, which is prone to make traditional methods unstable [8].

Deriving agent behavior from scalar evaluative feedback is the primary concern of reinforcement learning [21, 31]. Feedback usually comes from a hand-designed or automatically-constructed [30] reward function. Reward functions are the most common representations for tasks, mapping state features to scalar values. They can be learned from expert demonstration via inverse reinforcement learning or IRL [1, 5, 26, 34]. Classical reward functions are history-independent, but representations for temporal tasks have been proposed, often using variants of temporal logic [6, 14]. Temporal-logic

[1]www.doggiebuddy.com/topics/Trainingtopics/traintopic3.html

representations of tasks are powerful because they can compositionally express tasks of unlimited complexity. Whereas Markov reward functions are limited to being able to express m^n distinct behaviors in an n-state, m-action environment, the number of distinct temporal tasks is countably infinite.[2]

Existing work for learning compositional, or logical, representations requires either an optimization procedure that can posit and recombine substructures [19] or a training procedure that applies feedback to separate subtasks [28]. It is the latter path we follow here. We build on recent work showing that human trainers can decompose complex training tasks into more tractable curriculum structures [27] and examine the problem of learning a complex task specification through a series of self-contained lessons.

2 PROBLEM DEFINITION

We represent an agent's environment as a Markov decision process (MDP) $M = \langle S, A, T\rangle$. In place of standard reward functions, objectives or *tasks* are represented by linear temporal logic (LTL) formulas. That is, a task Φ has a corresponding optimal policy π_Φ, possibly non-Markovian, that results in the agent moving through the state space in a way that maximizes the probability of satisfying Φ. A *mission* is a special task Φ^* whose execution is the overall goal of the training process. The problem we study is that of constructing an agent that is able to learn the desired behavior π_{Φ^*} efficiently via evaluative feedback from the trainer. We write $f_{\Phi,t} = 1$ if $a_t \in \pi_\Phi(s_t)$, 0 otherwise to capture the feedback expected from a trainer of task Φ if action a_t is taken by the agent in the state s_t visited at time t.

To separate the problem of learning the MDP M from the problem of learning the desired behavior, we assume M is known to both the agent and the trainer. We measure inefficiency in learning by counting the number of times the agent takes an action that is inconsistent with π_{Φ^*}.

To help the agent learn the correct task, a trainer gives either positive or negative feedback for each agent action. (We disallow neutral feedback or non-feedback in this work.) The trainer should respond with positive feedback if the agent's actions are consistent with the desired behavior and with negative feedback otherwise. Interactions take place in *rounds*, signaled by the trainer to the agent, in which the task being taught changes from round to round.

We make two key assumptions of trainers: (1) The majority of the evaluative feedback from the human trainers are accurate; and, (2) They can select tasks to teach at each round such that tasks are either one of a relatively small set of basic tasks or a relatively simple transformation of previously learned tasks where the final round's task is the mission Φ^*. We show that these assumptions are sufficient to learn arbitrarily complex missions in theory and also that users can carry out this curriculum-style training process successfully with an implemented agent and minimal prior instruction.

[2]One could argue that every possible reward function from the uncountably infinite set of reward functions does indeed represent a distinct task in that, for any pair of distinct reward functions, there exists an environment in which they induce different behaviors [3]. We do not undertake a formal analysis of the relative expressibility of rewards and temporal logic in the current work.

3 REPRESENTING TASKS

MDP reward functions can specify goal-seeking behavior, for example, by providing a large reward for visiting a goal state or by providing zero reward at the goal state and penalties at all other states. These task representations are well studied [18], but they do not provide a systematic compositional way of specifying complex tasks like the following: (1) Visit locations x and y. (2) Visit location x while avoiding location y. (3) Visit location x then y. (4) Do not visit location x unless location y has been visited exactly once.

One well-studied compositional representation for specifying temporal sequences is linear temporal logic (LTL). Although representations are possible, LTL has an extensive literature and has been shown to cover a significant set of tasks of interest [20].[3]

The value of using LTL in machine learning well studied. Shah et al. [29] introduced a probabilistic model for inferring task specifications as LTL formula. Kasenberg and Scheutz [15] inferred state-based and action-based objective functions from demonstrated behavior trajectories in MDPs. Neider and Gavran [25] reduced the learning task to a series of satisfiability problems and produced a smallest LTL formula. Camacho and McIlraith [7] applied LTL to constructing human-interpretable behavior models. Our work differs in that it learns LTL representations via interactive evaluative human feedback and curriculum design.

An LTL formula is a logical expression over propositional functions that is intended to hold starting from a given moment in time. In addition to the standard logical operations like $\wedge$ (and), $\vee$ (or), $\implies$ (implication), and $\neg$ (negation), LTL provides a set of operations for constraining future values of propositions as well. The operators we consider are $\Diamond$ (eventually), $\Box$ (always), and $\mathcal{U}$ (until). Briefly, $\Diamond\Phi$ means that formula Φ must be true at some point in the future, $\Box\Phi$ means that formula Φ must be true at all points in the future, and $\Phi \;\mathcal{U}\; \Phi'$ means that Φ must be true at every time point until Φ' becomes true and that Φ' must eventually become true.

A particularly powerful aspect of LTL is that the subformulas Φ themselves can include temporal operators. Here are representations of the example tasks:

(1) $(\Diamond x) \wedge (\Diamond y)$: At some point in the future, the agent should visit a state where x is true. In addition, the agent should at some point visit a state where y is true.
(2) $(\neg y) \;\mathcal{U}\; x$: The agent should visit a state where x is true, but must not visit a state where y is true until that happens.
(3) $\Diamond(x \wedge \Diamond y)$: The agent needs to reach a state in which x is true and, at that same point, y is true in the future.
(4) $((\Box\neg x) \vee ((\neg x) \;\mathcal{U}\; y)) \wedge \Box(y \implies (y \;\mathcal{U}\; (\Box\neg y)))$: Two conditions must hold. First, either the agent never visits a state where x is true or it avoids states where x is true until a state where y is true is reached. But, at the same time, it should always be the case that visiting a state where y is true results in always avoiding such states after leaving.

One challenge to using LTL in MDPs is that conditions such as $\Box x$ cannot be verified except in the context of an infinitely long state sequence. In addition, say we are trying to encourage an agent

[3]LTL is not sufficiently expressive to capture all possible tasks, however. For example, LTL cannot express standard reward maximization tasks or temporal tasks involving context-free structures [4].

to reach a goal state by putting a reward of +1 there. Looking at a pure sum of rewards, any sequence that reaches the goal, regardless of length, is equally good. A common approach to discouraging unbounded delays is to introduce temporal discounting, which can be viewed as introducing a per-step expiration probability. Specifically, the agent's reward accumulation continues after each step with probability γ and terminates otherwise. The result of introducing discount factor $0 < \gamma < 1$ is that rewards t steps in the future are only worth γ^t as much. Setting γ closer to zero has the effect of making an agent "impatient", gathering rewards as quickly as possible before task termination. Values close to one allow for more "patient" behavior. (We used $\gamma = 0.99999$ in our experiments.)

A similar idea has recently been introduced into LTL, which we use in our work—each temporal operator in the formula has a per-step probability of expiring [22]. If an $\Box$ operator has been satisfied up until the time it expires, it is considered to have been evaluated to be true. If an $\Diamond$ operator has not been satisfied up until the time it expires, it is considered to have been evaluated to be false. LTL formulas can be automatically compiled into an extended state space that allows standard MDP planning algorithms to compute policies that maximize the formula-satisfaction probability.

4 TRAINING ALGORITHM

We propose an iterative algorithm (Algorithm 1) that learns missions effectively and efficiently over a series of rounds, $i = 0, 1, \ldots, k$. Before the start of the first round, the agent is given K_0, a set of logical propositions in the domain. The set of initial hypotheses H_0 is generated by applying the transformations τ to K_0. After the training for round i, the agent identifies a set $L_i \subseteq H_i$ of learned tasks and sets $K_{i+1} = K_i \cup L_i$. In each subsequent round $i + 1$, the agent takes the set K_{i+1} and templates τ and generates $H_{i+1} = \tau(K_{i+1})$ from them to form new hypotheses for learning. This process continues until the trainer successfully conveys the mission Φ^*.

At round i, the trainer guides the agent to learn Φ_i by giving positive feedback when the agent's actions align with Φ_i and negative feedback otherwise. During training (LearnTask), the agent takes actions that elicit the most discriminative feedback possible from the trainer. Specifically, the agent takes an action such that it can rule out as close as possible to 50% of the policies currently under consideration. Given perfect feedback, whenever the agent sees feedback inconsistent with a given hypothesis, it can eliminate that hypothesis from consideration for the rest of the round. To be robust to occasional trainer errors, however, we instead have the agent allocate "strikes" to hypotheses that disagree with the feedback. When a round is ended, the agent is left with L, the set of hypotheses that had the fewest strikes against them. In our experiments, we direct the trainer to continue training if more than one hypothesis is in L. However, after 10 restarts, the algorithm simply returns two hypotheses selected at random from L.

Algorithm 1 Agent learning algorithm

```
Input: basic propositions $K_0$, templates $\tau$
Initialize $H_0 \leftarrow \tau(K_0)$, $i \leftarrow 0$
while trainer has not finished mission do
    $L_i \leftarrow$ LearnTask($H_t$)
    $K_{i+1} \leftarrow K_i \cup L_i$
    $H_{i+1} \leftarrow \tau(K_{i+1})$
    $i \leftarrow i + 1$
end while
return $L_{i-1}$ as learned mission

function LearnTask($X$)
    $t \leftarrow 0$
    restarts $\leftarrow 0$
    $L \leftarrow X$
    for $\phi \in X$ do
        $r_\phi \leftarrow 0$, initialize formula strike counter
    end for
    choose starting state $s_t$
    while task has not ended do
        observe current state $s_t$
        for all $a \in A$ do
            $c_a$ = # formula in $L$ with $a$ optimal in $s_t$
        end for
        execute $a_t = \arg\min_a |c_a - |L|/2|$
        if trainer gives feedback $f_t$ then
            for all $\phi \in X$ do
                if $f_t \neq f_{\phi,t}$ then
                    $r_\phi \leftarrow r_\phi + 1$
                end if
                $L = \{\phi \in X | r_\phi = \min_{\phi \in X} r_j\}$
            end for
        else if trainer attempts to end task then
            if $|L| = 1$ or restarts $\geq 10$ then
                break
            else
                restarts $\leftarrow$ restarts + 1
                choose starting state $s_{t+1}$
            end if
        end if
        $t \leftarrow t + 1$
    end while
    return $L$ (at most 2, selected at random)
end function
```

Theorem 1. *Consider a set of tasks X, where every $x \in X$ has a formula of length at most d and can be distinguished from the rest of the tasks in X using a trajectory of length m. Given that $|X|$ is polynomial in d and provided with evaluative feedback with at most n errors, a learning agent can successfully identify the target task with a number of interactions that is polynomial in m, n, and d.*

Theorem 1 shows us that, if we know a trainer will make only at most n mistakes, we can ensure that we find the single correct hypothesis. It would be difficult, if even possible, to know the maximum number of mistakes a particular human trainer might make, so this assumption is a strong one. We can calculate a bound on the error rate of the trainer that the naïve algorithm in the proof will tolerate. Since we can err up to n times safely on each of the tests with $(n + 1)m$ possible feedback signals, our acceptable error

	Known Formula		Derived Formula
	Atemporal transformations (X and Y are not necessarily temporal formulas)		
1.	X		$\Diamond X$
2.	X		$\Box X$
3.	X,	Y	$X \, \mathcal{U} \, Y$
	Temporal transformations (X, Y, are temporal formulas, x and y are not necessarily temporal formulas)		
4.	$X = \Diamond x$		$\Diamond \neg x$
5a.	$X = \Diamond x$,	$Y = \Diamond y$	$\Diamond(x \vee y)$
5b.			$\Diamond(x \wedge y)$
6.	$X = \Box x$		$\Box \neg x$
7a.	$X = \Box x$,	$Y = \Box y$	$\Box(x \vee y)$
7b.			$\Box(x \wedge y)$
8.	$X = \Box x$,	$Y = \Diamond y$	$x \, \mathcal{U} \, y$
9.	X		$\neg X$
10.	X,	Y	$X \wedge Y$
11.	X,	Y	$X \vee Y$
12.	X		$\Box \neg X$
13.	$X = \Diamond x$,	Y	$\Diamond(x \wedge Y)$
	Specialized transformations (X, Y, Z, x, y need not be temporal)		
14.	Z,	$X \, \mathcal{U} \, Y$	$X \, \mathcal{U} \, (Y \wedge Z)$
15.	$X = \Diamond x$		$\Diamond \Box x$
16.	$X = \Box x$		$\Box \Diamond x$
17.	$\Diamond(X \wedge \Diamond y)$		$\Diamond(X \wedge \Diamond \Box y)$

Figure 1: Templates used in constructing temporal formulas.

rate ρ is bounded by $\rho = \frac{n}{(2n+1)m} < \frac{1}{2m}$. Further, Algorithm 1 follows the same logic as the algorithm used to prove Theorem 1, except it more efficiently tests hypotheses by using feedback to gain information about multiple hypotheses at once. Requiring fewer feedback signals allows for the algorithm to be tolerant of higher rates of trainer error, in practice.

Similar to standard Boolean logic, LTL formulas can be represented as syntax trees with nodes for the additional LTL operators. We partition the space of all LTL formulas into two classes, *temporal* and *atemporal*. We define a temporal LTL formula to be one in which, on every root-to-leaf path in the tree representation, there exists at least one temporal operator ($\Diamond$, $\Box$, $\mathcal{U}$). Conversely, an atemporal LTL formula is one in which there exists at least one root-to-node path in the syntax tree that does not contain a temporal operator. Any task that is defined by a temporal formula is referred to as a temporal task and any task defined by an atemporal formula is referred to as an atemporal task. Figure 1 lists the templates we use in our algorithm. The templates were chosen by inferring what kinds of transformations participants seemed to be expecting in the context of a pilot study, but they were then significantly modified to both ensure all temporal tasks could be constructed and strike a balance between coverage and complexity.

The significance of temporal tasks is that they cannot be shown to be unsatisfied without at least a single temporal step, meaning

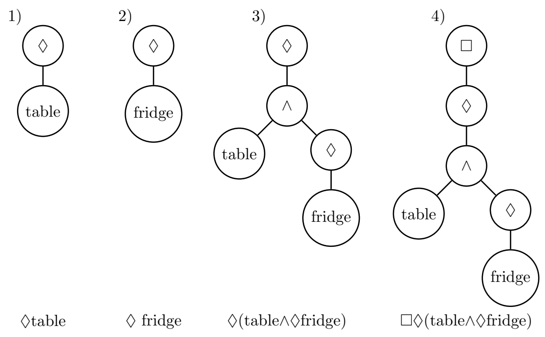

Figure 2: The training procedure for task 2E. Steps 1) and 2) use template 1 from Figure 1. Step 3) uses template 13. The final step, 4) uses template 2.

that a trainer will have an opportunity to provide feedback. Therefore, to show that a mission can be trained by our multi-round curricular algorithm, it must be a temporal task *and* it must be able to be constructed via the templates we define where every task en route to the mission is formulated as a temporal task.

Lemma 2. *Any temporal formula can be built up via application of these transformations starting from basic tasks that include simple temporal formulas of the domain's propositions. All of the intermediate formulas in the construction are themselves temporal.*

An intuitive description of the proof sketch for Lemma 2 is the following. Whenever we're training at a node that is an atemporal operator, we know, by definition, that both children must be temporal. So, we could have trained (bottom up) to this point without any issues via induction. When the current node is temporal, it is possible that one or both of its subtrees is atemporal. In that case, we can show that we can propagate the temporal operator down to the subtrees. If we train according to this new tree structure, the atemporal subtrees become temporal and therefore trainable. This training strategy is illustrated in Figure 2.

Theorem 3. *If a trainer can decompose a mission into tasks satisfying the lemma and can provide evaluative feedback with low error rate, they can train the algorithm to learn the temporal LTL task in time polynomial in the size of the formula with high probability.*

Theorem 3 follows from a combined application of Theorem 1 and Lemma 2. We show that the training procedure remains feasible with respect to the size of the formula for the overall mission.

In short, Theorem 1 shows that any temporal task we consider can be correctly learned via imperfect feedback. Lemma 2 explains that any temporal mission can be successfully decomposed and trained as a series of these smaller tasks discussed in Lemma 2. Finally, Theorem 3 proves that if a trainer is able to decompose tasks according to Lemma 2 and can provide feedback with low error as in Theorem 1, then Algorithm 1 can efficiently learn LTL missions. Having shown theoretically that our learning algorithm can be efficiently taught any temporal mission formula by an idealized, low-error trainer, we follow up by showing that real world users can approximate the idealized trainer sufficiently closely to convey complex tasks to the agent.

5 EXPERIMENTS AND RESULTS

We assessed the performance of our algorithm compared to existing approaches—TAMER and COACH.

5.1 Simulation study

In Study 1, we evaluated how well each algorithm performed in a simulated 5×5 grid world (Figure 4). In the grid world, there were four objects, each associated with its own atomic proposition: a table, a chair, a charger, and a fridge. Four tasks were tested: *(1A) Move all around, but don't bump into the chair:* $\square\neg chair$. *(1B) Go directly to the table:* $\lozenge table$. *(1C) Don't touch the table on your way to reaching the charger:* $\neg table\ \mathcal{U}\ charger$. *(1D) Start at the fridge and stay there:* $\square fridge$. The tasks were all in the initial set of hypotheses and thus could be learned directly without a curriculum strategy.

There were two types of trainers we simulated in this experiment: 1) an ideal trainer who only gives correct feedback; 2) a simulated non-expert trainer who gives feedback with an error rate of θ. For each type of trainer, we ran 10 rounds of training for each algorithm. During each round, the agent started from a random grid and learned from feedback given by the trainer until the task was complete, then chose another random position to start the next episode.

We measured how well the learner performed by the percentage of positive feedback its trajectory received from an ideal trainer. A *perfect* episode is one with a value of 100%. Once the agent gets three perfect episodes in a row, the round is finished. We counted the median number of episodes the agent had taken before the first perfect episode, the median number of feedback per episode, and the median number of the total feedback per round. For task 1D, the agent always started from the fridge because the task cannot be completed successfully otherwise.

Since the experimental tasks was temporal rather than state-based in nature, we augmented the state space available to TAMER and COACH so that they could learn the target behavior. The state consisted of the current position of the robot along with the order that the landmark object were visited up to that point. So, if the robot had visited the charger and the table, in that order, the state would be augmented with "1: charger, 2: table".

Our algorithm took a very small number of episodes to learn the tasks. Figure 3 presents the training time results for all three algorithms in two scenarios—error-free feedback and feedback with $\theta = 0.327$ errors. This value was chosen to align the simulation with our observation that 32.7% of feedback signals were observed to be in error in our user study. For each algorithm, we show the range of training times over the four tasks. The results show that our algorithm needs significantly fewer episodes to learn the tasks—the simulated trainer gave much more feedback per round to TAMER and COACH than to our algorithm. The performance difference between our algorithm and the other two became larger when the trainer made mistakes. (Note the log scale on the plot.)

We further tested the algorithms with a group of complicated missions: *(2A) Go to the table and then go to the charger and stay there:* $\lozenge(table \wedge \lozenge\square charger)$. *(2B) Go to your charger without colliding with either the chair or the table:* $\neg(chair \vee table)\ \mathcal{U}\ charger$. *(2C) Go to the table and then go to the fridge:* $\lozenge(table \wedge \lozenge fridge)$. *(2D) Go to the charger and then go to the chair without running into the table along the way:* $(\neg table)\ \mathcal{U}\ (charger \wedge \lozenge chair)$. *(2E) Go back and forth patrolling between the table and the fridge:* $\square\lozenge(table \wedge \lozenge fridge)$. Neither TAMER nor COACH could learn any of them. Our algorithm, in contrast, successfully learned from the missions being decomposing into simpler tasks, and tackling them one at a time.

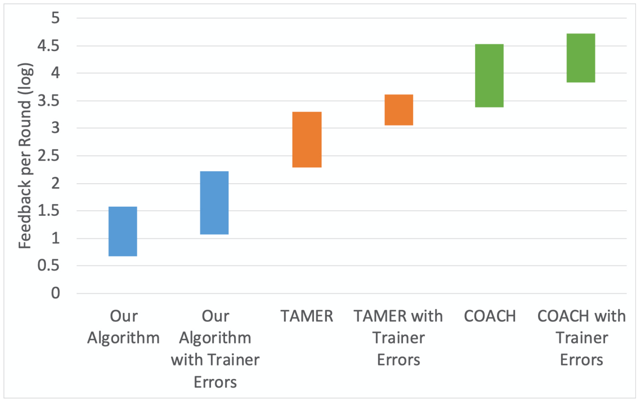

Figure 3: Training feedback needed for our algorithm, TAMER, and COACH across tasks 1A through 1D.

5.2 User study 1: Learning a single task

We then carried out two user studies to determine whether non-expert human trainers could 1) train a learning agent running our algorithm to execute basic tasks; 2) train a learning agent end to end on a complex mission by decomposing it into smaller tasks and providing evaluative feedback to convey each task. Participants for each study were recruited via Amazon Mechanical Turk (AMT).

In Study 1, participants were instructed to train agents on basic tasks. They were presented with a simulated robot (the learning agent) in a 5×5 grid world (Figure 4). A brief introduction of the user study environment can be found here: https://youtu.be/40uLW10UFzk During each round, participants were given a basic task that they needed to train the robot to execute via positive and negative feedback. Each participant was asked to complete four rounds, each with a different and independent task. The tasks were same as the four basic tasks in the simulated environment, and were given one by one in a random order to the participants.

At the beginning of each round, the participant was asked to place the robot by clicking on one grid cell in the map. The robot would then choose an action (moving up, right, down, left, or stay) based on our scheme. After taking an action, the agent would wait for the participant to give feedback. The participant could choose to give positive feedback by clicking on "NICE MOVE :)", or negative feedback by clicking on "BAD MOVE :(". Our algorithm would then learn from the feedback, ruling out formulas that did not match the feedback and choosing from the remaining formulas to make its next action selection. During training, the participant could change the agent's location by clicking "PICK UP AND REPLACE ROBOT"

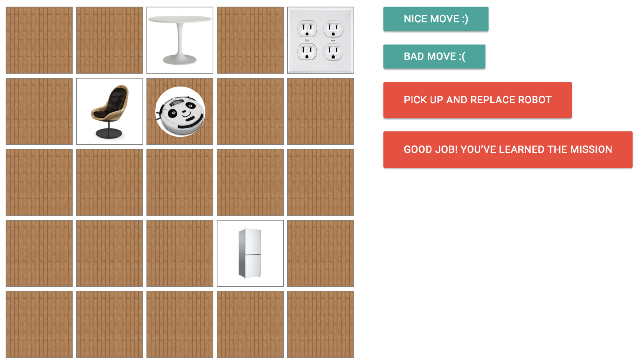
NICE MOVE :)
BAD MOVE :(
PICK UP AND REPLACE ROBOT
GOOD JOB! YOU'VE LEARNED THE MISSION

Figure 4: Experimental domain for training temporal tasks.

and choose another grid cell to continue. Once the participant was satisfied with the performance of the agent, he or she could click "GOOD JOB! YOU'VE LEARNED THE MISSION" to end the training. At the end of training, the remaining set of formulas, as derived by the algorithm, was considered to be the *learned formula* set.

We instructed participants 1) to give positive feedback if he or she believed that the most recent action was consistent with the target task and negative feedback otherwise; and 2) to end the training when he or she believed that the agent had learned the task.

5.3 User study 1 results

To measure task success, we evaluated whether the learned formula matched the target formula using a *similarity* score defined over the range $[0, 1]$. Consider a ground truth formula A of which the induced policy is π_A, and a learned formula B with policy π_B. For π_B, we create a set of trajectories J_B by starting the agent from each of the twenty-five positions on the grid and executing twenty random trajectories of π_B. Thus, J_B contains five hundred trajectories in total. The *similarity* of B to A is calculated by counting the percentage of positive feedback of all trajectories in J_B from an ideal trainer trying to teach A. It can be thought of as telling us how happy someone would be when they were wanting to train formula A and the robot exhibits behavior from formula B. If the agent learns more than one formula, then we conservatively report the minimum similarity for all learned formulas to the target formula.

In the 5×5 grid world, the similarity between two randomly generated formulas is 0.16 on average. To get a sense of the meaning of the scores, here are some examples: The similarity of formula A: $\Box$fridge to formula B: $\Diamond$charger is 0—they send the agent in very different directions. The similarity of formula B to formula C: $\Diamond$fridge is 0.35, because both draw the agent to the right side of the grid. The similarity of formula D: $(\neg$table$)\ \mathcal{U}$ charger to B is 0.98 because they produce nearly the same action in all states.

A training round was considered to be successful if learned formulas were above a specific similarity threshold to the mission. The success rate is the percentage of rounds that were successful. In this study, there were 80 training rounds (20 participants and 4 rounds each). Figure 5 shows the success rate for different similarity thresholds for each target task. The horizontal axis shows the

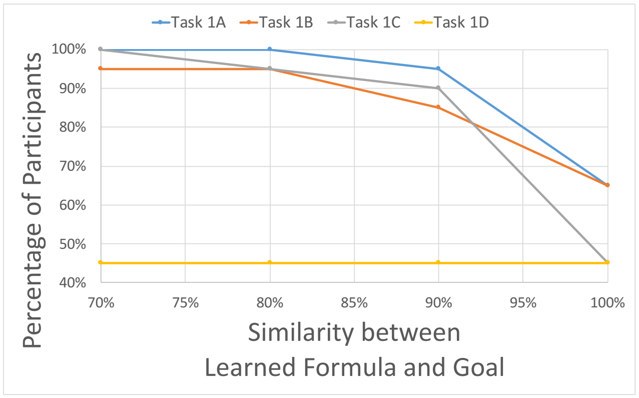

Figure 5: Mission success rate in user study 1.

similarity between the learned and target tasks. The vertical axis shows how many of the participants reached the similarity score in each task. The results support our hypothesis that the majority of people can teach simple temporal tasks, with the exception of Task 1D. An examination of the logs suggested that participants were unsure what feedback to give for this task when the robot did not start at the fridge. Some gave positive feedback to the robot for approaching the fridge, causing the learning algorithm to fail.

We found that the success rate for a participant was strongly correlated with the number of replacements used in training. At a similarity threshold of 0.9, trainers who successfully trained 3 or more rounds (among all 4 rounds) clicked "PICK UP AND REPLACE ROBOT" 21 times on average; in comparison, trainers who successfully trained only 2 or fewer rounds used replacement 9 times on average. This finding was likely a result of the fact that replacements expanded the size of the effective training set for the learning agent. Moreover, a good replacement (one that can efficiently help the agent distinguish between the remaining formulas) sped up the training and improved performance.

Based on this finding, we added functionality to our interface so that, whenever multiple formulas remained when the participant requested to end training, the agent would automatically move itself to a new start position to help disambiguate the remaining formulas. Note that the learning agent was not aware of the actual target of training, just that there was residual ambiguity in what it had learned. This form of active learning greatly simplified the training process for participants.

5.4 User study 2: Learning via decomposition

In Study 2, we had participants train an agent to carry out complex missions by decomposing each mission into a curriculum of simpler tasks and training the tasks one by one.

As in Study 1, we recruited 20 participants. The participants were presented with the same 5×5 grid world and the simulated robot. Participants received the same instruction as in Study 1 regarding how to give feedback and when to end a training round. The robot chose actions and waited for feedback. At the end of each round, if there were more than two formulas in the learned formula set, the robot would automatically move to a new start position. A training

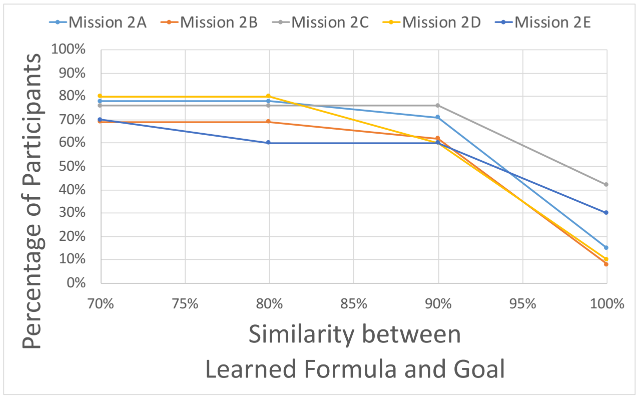

Figure 6: Mission success rate in user study 2.

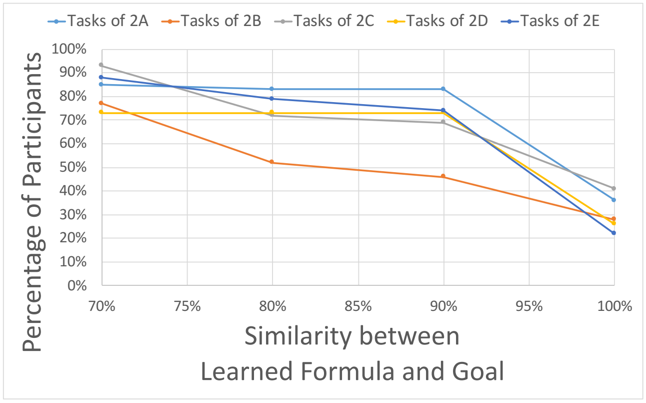

Figure 7: Task success rate in user study 2.

round could only be finished if the number of remaining potential formulas in H was one or two.

Unlike Study 1, we presented each participant with three independent complex missions chosen randomly from the group of five complicated missions in the simulated environment. After showing them a decomposition example, we asked them to first decompose each mission into a sequence of simple tasks, the last task being the mission itself. The tasks were trained consecutively in rounds. Within each round, the training process is executed as in Study 1. The participants can revise the sequence of tasks after each round. The English description of each decomposed task was recorded and later translated by the authors into LTL. The learned formula was derived in the same way as in Study 1. For the last task in each mission, the target task is the mission itself and the learned formula represents the learned formula for the entire mission.

5.5 User study 2 results

Figure 6 shows the success rate for different similarity thresholds for each mission. Figure 7 shows the success rate for different similarity thresholds for the decomposed tasks of each mission. The two figures were generated in the same way as Figure 5. The study result supports our hypothesis that the majority of human trainers can teach complex temporal missions via our algorithm. For all five missions, more than 60% of participants successfully taught the agent the formulas (similarity threshold 0.9).

Figure 2 shows the most common sequence of formulas trained by participants for Formula $2E$. All missions required a multi-step decomposition, and participants produced a formula with similarity over 0.8 with the target formula in more than half of all attempts. We define a decomposition as *sufficient* if the temporal formula for the mission can be built up via application of temporal transformations starting from the set of basic tasks. An *unnecessary* task in a sufficient decomposition is one that can be removed and the decomposition remains sufficient. A sufficient decomposition is *minimal* if none of its tasks are unnecessary. Among all the decompositions, 63.4% were sufficient and 50.1% were minimal. When the decomposition was insufficient or included unnecessary tasks, the mission could still be learned successfully with reasonably high probability as shown in Figure 6 and Figure 7. In 25 of the 59 runs, the similarity of the learned mission to the target mission was higher than the similarity of at least one of the learned intermediate tasks to the user's attempted intermediate task. For Missions 2D and 2E, the agent was able to learn the mission even when at least one of the decomposed tasks failed.

We examined how often trainer feedback is in error given our LTL translations of the tasks. Across all training sessions, 4739 of 14505 (32.7%) feedback signals were in error. While this error rate was higher than expected, it yields a promising success rate (at a similarity threshold of 0.9), which was at least 60% across all the five missions. The multi-round training procedure allows participants to successfully train tasks even in the face of high error rates for the individual feedback signals.

6 CONCLUSION AND FUTURE WORK

In this paper, we addressed the problem of learning complex tasks via evaluative feedback through a sequence of self-contained lessons. We provided theoretical results showing the effectiveness of our approach, then followed up with experiments in simulation. The results showed that our algorithm outperformed existing approaches—TAMER and COACH—by taking significantly fewer training episodes and feedback signals to finish training. In a user study, we invited non-expert participants to train simple tasks, and asked them to train complex missions through decomposition. The results show that, in spite of people making mistakes in feedback and decomposition, they were very often able to convey a formula very similar to the target mission. Future work will apply the algorithm in robotics and home automation environments and identify methods for better preparing users by improving their teaching skills.

7 ACKNOWLEDGMENTS

The authors are grateful to support from DARPA I2O (XAI, L2M, and LwLL programs). We thank Jordan Hartzell and Lan Wei for their contributions to the project.

A APPENDIX

Proofs of the main formal results follow.

A.1 Proof of Theorem 1

Proof. Consider the following naïve, but sufficient, algorithm. For each hypothesis, the agent will conduct a set of tests on that hypothesis by making decisions based on the policy induced by the hypothesis and observing feedback. During this test, whenever the trainer gives the agent a negative feedback, the agent counts a strike against that hypothesis. Strikes are the result of either the hypothesis under testing being an incorrect hypothesis (and making a bad decision) or the result of a trainer error in feedback.

For a hypothesis to be incorrect, the policy induced by that hypothesis must take at least 1 action that receives negative feedback. For the agent to be certain that the test hypothesis is not the target hypothesis, it must observe at least $n + 1$ strikes. After accumulating $n + 1$ strikes, only n strikes can be attributed to errors in the feedback signal, and at least 1 strike is a legitimate strike. So, the test hypothesis must be incorrect.

To be sure the agent has the opportunity to observe the $n + 1$ requisite strikes to disqualify the test hypothesis, the agent must run at most $2n + 1$ tests. The first n tests might not reveal the incorrect behavior because each of the negative feedback signals could be in error and reported as positive. With $2n + 1$ tests, the agent is sure to see at least $n + 1$ strikes for an incorrect hypothesis.

By testing each of the $|X|$ hypotheses, the agent can be guaranteed to find the single correct hypothesis since it will be the only test with at most n strikes. Since each set of tests requires $2n + 1$ runs of the length m trajectory, each hypothesis' tests can take at most poly(m, n) interactions. We know that $|X| = \text{poly}(d)$, so the overall time to complete the algorithm and test all hypotheses is poly(d, m, n). □

A.2 Proof of Lemma 2

Remark 1. *The set of* $\{\wedge, \neg, \mathcal{U}, \Box, \Diamond\}$ *is functionally complete over the set we define as temporal formulas and are included in our basic templates via Templates* 1–3, 9–11. *So, to prove Lemma 2, we need to show that we can construct any given temporal tree via these transformations.*

Proof. We can prove the claim using induction on the depth of the temporal formula tree. Since the basic templates cover all possible operators on propositions $\{\Diamond, \Box, \mathcal{U}\}$, the claim holds when the depth is 1. (We do not need to even consider $\wedge$, $\vee$, and $\neg$ since these operators used on propositions would create an atemporal function with a tree of depth 1.)

For induction, we assume that the claim holds for all depths $[1, k]$ for some arbitrary k. We need to show that the claim still holds for $k + 1$. We investigate two cases for a temporal formula tree of depth $k + 1$.

Case 1. *The root of the tree is a temporal operator.*

Since the root is temporal and, by definition, on the path to every leaf, the resulting tree will always be temporal. The subtrees below the root are either temporal or atemporal and are of depth $\leq k$. Temporal subtrees can be constructed via the induction hypothesis on subtrees of depth k. Atemporal subtrees must be handled more carefully. We first build the subsubtrees by attaching a temporal operator to them. We use $\Box$ or $\Diamond$ to match the root of the overall tree. If the root of the tree has the form $a\ \mathcal{U}\ b$, we append $\Box$ if we are constructing subtrees of a or $\Diamond$ if we are constructing subtrees of b. These altered subsubtrees are of depth $k - 1 + 1 = k$ after adding on the extra temporal operator as described. Therefore, they are temporal trees of depth k, and can now be constructed by the induction hypothesis. Once these altered subsubtrees are constructed, we can combine them to make our subtree by using them in Templates 4–8 or 13. After we have constructed the subtrees as stated above, we can construct the overall tree via templates 1–3.

Case 2. *The root of the tree is an atemporal operator.*

In this case, we know that both subtrees must be temporal. If even one subtree were atemporal, it would include a path to a leaf that includes no temporal operator and our original claim that the function is temporal is contradicted. Since both of the subtrees are temporal, all leaves in the overall tree have a path from the root that traverses a temporal operator. Further, since the subtrees are of depth $k + 1 - 1 = k$, they can be constructed via the induction hypothesis. The overall tree can then be constructed via Templates 9–12. □

A.3 Proof of Theorem 3

Proof. Successful training can be accomplished by using the Algorithm 1 with rounds as described by Lemma 2. Since the transformations given by the procedure in the proof of Lemma 2 do not increase the size of the formula tree by more than a factor of 2, the bounds on the description of the formula remains the same. Within each iteration of the algorithm, we increase the hypothesis set by $|\tau||L|$. Since we bound $|L|$ by a constant (we use 2) and $|\tau|$ is also constant, the overall size of the hypothesis space increases only polynomially over the course of the algorithm. As a result, training each round (even later ones) remains feasible and bounded, both in sample and computational complexity. Applying Theorem 1 to this approach to training proves our claim. □

REFERENCES

[1] Pieter Abbeel and Andrew Y Ng. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning.*

[2] Riad Akrour, Marc Schoenauer, and Michèle Sebag. 2011. Preference-based policy learning. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML-PKDD).* 12–27.

[3] Kareem Amin, Nan Jiang, and Satinder Singh. 2017. Repeated Inverse Reinforcement Learning. (2017). arXiv preprint arXiv:1705.05427.

[4] Davide Ancona, Angelo Ferrando, and Viviana Mascardi. 2016. Comparing trace expressions and linear temporal logic for runtime verification. In *Theory and Practice of Formal Methods.* Springer, 47–64.

[5] Monica Babes, Vukosi N. Marivate, Michael L. Littman, and Kaushik Subramanian. 2011. Apprenticeship Learning About Multiple Intentions. In *Proceedings of the International Conference on Machine Learning.* 897–904.

[6] Fahiem Bacchus, Craig Boutilier, and Adam Grove. 1996. Rewarding Behaviors. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence.* AAAI Press/The MIT Press, 1160–1167.

[7] Alberto Camacho and Sheila A McIlraith. 2019. Learning Interpretable Models Expressed in Linear Temporal Logic. ICAPS.

[8] Javier Ruiz-del-Solar Celemin, Carlos and Jens Kober. 2019. A fast hybrid reinforcement learning framework with human corrective feedback. In *Autonomous Robots.* Springer, 1173–1186.

[9] Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems.* 4302–4310.

[10] Dylan Hadfield-Menell, Stuart J Russell, Pieter Abbeel, and Anca Dragan. 2016. Cooperative inverse reinforcement learning. In *Advances in Neural Information Processing Systems.* 3909–3917.

[11] Mark K Ho, Michael L. Littman, Fiery Cushman, and Joseph L. Austerweil. 2015. Teaching with rewards and punishments: Reinforcement or communication?. In *Proceedings of the 37th Annual Meeting of the Cognitive Science Society.*

[12] Mark K. Ho, James MacGlashan, Michael L. Littman, and Fiery Cushman. 2017. Social is special: A normative framework for teaching with and learning from evaluative feedback. *Cognition* 167 (2017), 91–106.

[13] Charles Lee Isbell, Michael Kearns, Satinder Singh, Christian R Shelton, Peter Stone, and Dave Kormann. 2006. Cobot in LambdaMOO: An adaptive social statistics agent. *Autonomous Agents and Multi-Agent Systems* 13, 3 (2006), 327–354.

[14] Daniel Kasenberg and Matthias Scheutz. 2017. Interpretable Apprenticeship Learning with Temporal Logic Specifications. In *Proceedings of the 56th IEEE Conference on Decision and Control.*

[15] Daniel Kasenberg and Matthias Scheutz. 2017. Interpretable apprenticeship learning with temporal logic specifications. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC).* IEEE, 4914–4921.

[16] W Bradley Knox and Peter Stone. 2009. Interactively shaping agents via human reinforcement: The TAMER framework. In *Proceedings of the Fifth International Conference on Knowledge Capture.* 9–16.

[17] W Bradley Knox and Peter Stone. 2013. Learning non-myopically from human-generated reward. In *Proceedings of the 2013 International Conference on Intelligent User Interfaces.* 191–202.

[18] Sven Koenig and Reid G. Simmons. 1993. Complexity Analysis of Real-time Reinforcement Learning. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.* AAAI Press/MIT Press, Menlo Park, CA, 99–105.

[19] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press.

[20] H. Kress-Gazit, G.E. Fainekos, and G.J. Pappas. 2009. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Tans. on Robotics* 25 (2009), 1370–1381.

[21] Michael L. Littman. 2015. Reinforcement learning improves behaviour from evaluative feedback. *Nature* 521, 7553 (2015), 394–556.

[22] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. 2017. Environment-Independent Task Specifications via GLTL. (2017). arXiv preprint arXiv:1704.04341.

[23] Robert Loftin, James MacGlashan, Michael L. Littman, Matthew E. Taylor, David L. Roberts, and Jeff Huang. 2014. A Strategy-Aware Technique for Learning Behaviors from Discrete Human Feedback. In *Proceedings of the Twenty-Eighth Association for the Advancement of Artificial Intelligence Conference.*

[24] James MacGlashan, Mark K Ho, Robert Loftin, Bei Peng, Guan Wang, David L. Roberts, Matthew E. Taylor, and Michael L. Littman. 2017. Interactive Learning from Policy-Dependent Human Feedback. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning.*

[25] Daniel Neider and Ivan Gavran. 2018. Learning linear temporal properties. In *2018 Formal Methods in Computer Aided Design (FMCAD).* IEEE, 1–10.

[26] Andrew Y. Ng and Stuart Russell. 2000. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning.* 663–670.

[27] Bei Peng, James MacGlashan, Robert Loftin, Michael L. Littman, David L. Roberts, and Matthew E. Taylor. 2017. Curriculum Design for Machine Learners in Sequential Decision Tasks. In *Proceedings of the 16th International Conference on Autonomous Agents and Multiagent Systems.*

[28] Ronald L Rivest and Robert Sloan. 1994. A formal model of hierarchical concept-learning. *Information and Computation* 114, 1 (1994), 88–114.

[29] Ankit Shah, Pritish Kamath, Julie A Shah, and Shen Li. 2018. Bayesian inference of temporal task specifications from demonstrations. In *Advances in Neural Information Processing Systems.* 3804–3813.

[30] S Singh, R L Lewis, and A G Barto. 2009. Where do rewards come from?. In *Proceedings of the Annual Conference of the Cognitive Science Society.*

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

[32] Andrea L Thomaz and Cynthia Breazeal. 2008. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence* 172 (2008), 716–737.

[33] Christian Wirth and Johannes Fürnkranz. 2013. Preference-based reinforcement learning: A preliminary survey. In *Proceedings of the ECML/PKDD-13 Workshop on Reinforcement Learning from Generalized Feedback: Beyond Numeric Rewards.*

[34] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. 2008. Maximum Entropy Inverse Reinforcement Learning. In *AAAI*, Vol. 8. 1433–1438.