To appear in *AAAI Spring 2009 Symposium on Agents that Learn from Human Teachers* (AAAI-SS09), Palo Alto, United States, March 2009.

Design Principles for Creating Human-Shapable Agents

W. Bradley Knox
Department of Computer Sciences
University of Texas at Austin
bradknox@cs.utexas.edu

Ian Fasel
Department of Computer Sciences
University of Texas at Austin
ianfasel@cs.utexas.edu

Peter Stone
Department of Computer Sciences
University of Texas at Austin
pstone@cs.utexas.edu

Abstract

In order for learning agents to be useful to non-technical users, it is important to be able to teach agents how to perform new tasks using simple communication methods. We begin this paper by describing a framework we recently developed called Training an Agent Manually via Evaluative Reinforcement (TAMER), which allows a human to train a learning agent by giving simple scalar reinforcement[1] signals while observing the agent perform the task. We then discuss how this work fits into a general taxonomy of methods for human-teachable (HT) agents and argue that the entire field of HT agents could benefit from an increased focus on the *human* side of teaching interactions. We then propose a set of conjectures about aspects of human teaching behavior that we believe could be incorporated into future work on HT agents.

Introduction

In order for learning agents to be broadly useful outside the AI laboratory, it is important for everyday (non-technical) human users to be able to teach agents to solve new tasks without expert programming knowledge. Although they may not be expert programmers and may not even have full knowledge of how the task *should* be solved by the agent, human users often do have useful knowledge about the task at hand. We aim to make it easy for them to give that knowledge to the agent.

One reason we want to be able to give humans a way to shape agents is that *tabula rasa* learning agents, i.e., agents that start from scratch, often require hundreds to hundreds of thousands of learning trials to achieve satisfactory performance on a task. Unfortunately, in real-world tasks, such as autonomous driving or financial investment, it is often the case that even a few learning trials can be very expensive. For instance, a robotic car cannot take one thousand attempts to learn how to drive on mountain roads, and an investment agent cannot learn how to invest by recklessly spending all of its firm's capital. While available computing power tends to follow Moore's law, increasing exponentially as time progresses, the cost of obtaining n training samples does not follow such a law. A robot that can act ten times per second cannot reduce the time it takes to get ten thousand samples, regardless of what clock speed its processor is running. Consequently, within the learning agent community there is often a greater focus on sample complexity than computational complexity.

Recently, a number of researchers have been interested in the general topic of *transfer learning*, which is the transfer of knowledge attained from a previous learning task to speed up or otherwise improve the learning on the current task (e.g., see (Taylor 2008)). When developing agents that can receive knowledge from humans, a key question is how such knowledge can be communicated from the human to the agent.

Reviewing the literature on transfer from a human to an agent, three basic modes of communication seem to have emerged. The first might be broadly described as *showing* the agent what to do, often referred to as *imitation learning* and *programming by demonstration* (Billard et al. 2008), in which the human gives the agent some form of examples of the task which the agent is meant to copy (and hopefully generalize to new situations). The second can be described as *telling* (or *advising*) the agent about good choices of action in particular states, sometimes in the form of guideline rules provided using natural language as in (Kuhlmann et al. 2004). In this paper, we focus largely on a third and less explored method— shaping a learning agent by giving positive and negative reinforcement.[2] To shape an agent, a human trainer needs only to be able to indicate his or her approval of the agent's behavior in such a way that it can be converted to a scalar signal for the agent (e.g., +2 might indicate a positive evaluation). Communication options include pushing buttons or speaking through a speech-to-text program.

Imitation learning is a very useful method for human-to-

Copyright © 2009, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1] In this paper, we distinguish between human reinforcement and environmental reward within an MDP. To avoid confusion, human feedback is always called "reinforcement".

[2] We use the term "shaping" as it is used in animal learning literature. There, shaping is defined as training by reinforcement of successively improving approximations of the target behavior (Bouton 2007). In reinforcement learning literature, it is sometimes used as in animal learning, but more often "shaping" is restricted to methods that combine the shaping reinforcement signal and the reward signal of the environment into a single signal (Ng, Harada, and Russell 1999). We later argue in the fifth conjecture that the two signals should be kept separate.

Table 1: Results of various Tetris agents.

Method	Mean Lines Cleared		Games
	at Game 3	at Peak	for Peak
TAMER	**65.89**	65.89	3
RRL-KBR (2004)	5	50	120
Policy Iteration (1996)	0 (random until game 100)	3183	1500
Genetic Alg. (2004)	0 (random until game 500)	586,103	3000
CE+RL (2006)	0 (random until game 100)	348,895	5000

agent transfer, however it requires more of the trainer than does shaping. In imitation learning, a teacher must take control of the agent and demonstrate a policy. In contrast, a shaping trainer only needs to criticize the agent. Not everyone is an expert, but as they say, "everyone's a critic." Additionally, the cognitive load of giving positive and negative evaluation is arguably less than that of remote-controlling an agent (provided it is even possible). Agents that receive natural language advice require an understanding of natural language, a feat thus far accomplished only in specific domains after an extensive hand-labeling of language samples (Kuhlmann et al. 2004).

In our work on creating human-shapable agents, we developed a general framework called TAMER (for Teaching an Agent Manually via Evaluative Reinforcement). As a proof-of-concept, we created a specific algorithm within the TAMER framework (Knox and Stone 2008) that allows human trainers to teach a learning agent to play Tetris, a puzzle game. On average, the agents learned good policies an order of magnitude faster than tabula rasa agents, clearing more than 50 lines per game by the end of the third game (see Table 1).

In this position paper, we first describe the TAMER framework and a generic high-level algorithm for the framework in the following section. Then we give a review of methods by which a human can train an agent in the section entitled "Taxonomy". Finally, in a section called "Conjectures", we state a number of conjectures about designing a shapable agent. A common theme is that knowledge of how a human delivers reward should guide the design of any shaping algorithm. The conjectures have not been experimentally tested, but we argue for each using what evidence is available.

TAMER Framework

At the highest level of description, the TAMER framework allows a human to train an agent to perform a task via positive and negative reinforcement signals. From the agent's point of view, its goal is to model the human's reinforcement while exploiting that model to earn as much immediate reinforcement as possible. A key factor is that human teachers are treated differently from the environment in general (see Figure 1 and the fifth conjecture). By exploiting knowledge about human behavior, we can make learning more efficient than if we only had access to, or treated human feedback as just an additional source of, environmental reward.

Specification and Overview

The TAMER framework is designed to work within Markov Decision Processes (MDPs) that have the reward function

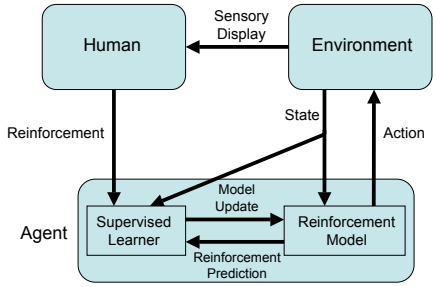

Figure 1: Framework for Training an Agent Manually via Evaluative Reinforcement (TAMER) in an MDP\R.

R unspecified or removed. Following Abbeel and Ng's terminology (Abbeel and Ng 2004), we call this specification an MDP\R. A description of MDPs can be found in (Sutton and Barto 1998).

A TAMER agent seeks to learn the human trainer's reinforcement function $H : S \times A \rightarrow \mathbb{R}$. Presented with a state s, the agent consults its learned model $\hat{H}$ and chooses the action a that maximizes $\hat{H}(s, a)$. Since the agent seeks only to maximize human reinforcement, the optimal policy is defined solely by the trainer, who could choose to train the agent to perform any behavior that its model can represent. When the agent's performance is evaluated using an objective metric, we expect its performance to be limited by the information provided by the teacher.[3]

A key insight of the TAMER framework is that the problem of credit assignment that is inherent in the field of reinforcement learning (Sutton and Barto 1998) is no longer present when a human trainer, evaluating recent state-action pairs with a model of their long-term consequences, can label those pairs within a temporal window of when they occur (argued later in the first conjecture). Assigning credit within that window presents a challenge to the agent, which we discuss later in the second conjecture, but it is much easier than assigning credit for an arbitrarily delayed reward signal as in reinforcement learning. Because a trainer can directly label behavior, modeling the trainer's reinforcement function H is a supervised learning problem.

If we assume that the human trainer desires to teach a specific policy described by H, then it is sufficient for the agent to learn a function $\hat{H}$ that maps to H such that

$$\hat{H}(s, a_1) > \hat{H}(s, a_2) \iff H(s, a_1) > H(s, a_2).$$

If this relation holds, the policies created by greedily exploiting $\hat{H}$ and H are equivalent. However, from our subjective observations, human trainers do not usually have one specific policy that satisfies them. Rather, there is a set of acceptable policies that are deemed "good enough". The set may change during the course of training (e.g., enlarging if the trainer gets bored with the interaction, or shrinking if the agent starts performing better than expected).

Comparison with Reinforcement Learning The TAMER framework for shaping agents shares much common ground with temporal difference reinforcement

[3] In principle, the agent could use both human and environmental feedback while learning. But in its initial version, TAMER focuses entirely on H.

learning, but there are some key differences.

In reinforcement learning, agents seek to maximize return, which is a discounted sum of all future reward. In contrast, a TAMER agent does not seek to maximize a discounted sum of all future *human* reinforcement. Instead, it attempts to directly maximize the immediate expected reinforcement given by the human trainer. It does so under the assumption that the trainer's reinforcement signal is a direct label on the value of recent state-action pairs.

Correspondingly, the human's reinforcement function H is not an exact replacement for a reward function R within an MDP. Although it may be possible for a reinforcement learning algorithm to use H in lieu of a reward function, it would be unnecessary extra computation, since H already defines a policy. We use MDP\Rs because an environmental reward function dictates the optimal policy, so R is removed to make the human's reinforcement function the sole determinant of good and bad behavior from the agent's perspective.

High-Level Algorithm

A high-level algorithm for implementing TAMER is described in Algorithm 1. After some initialization steps (lines 1-4), the algorithm begins a loop that occurs once per time step (line 5).

In the loop, the agent first obtains a scalar measurement of the human trainer's reinforcement since the previous time step (line 6). If the reinforcement value is nonzero, then the error is calculated as the difference between the actual reinforcement and the amount predicted by the agent's reinforcement model (denoted *ReinfModel* in the pseudocode; line 8). The model, left undefined for the general TAMER algorithm, can be instantiated in multiple ways as discussed below. The calculated error and the most recent feature vector is then used to update the reinforcement model (line 9).

The agent then obtains the current state description (line 11) and greedily takes the action a that, according to the human reinforcement model, yields the largest predicted reinforcement (line 12). The new feature vector $\overrightarrow{f}$ is calculated (line 13) and the chosen action is taken (line 14) before the loop restarts.

Taxonomy

Work on human-teachable agents has taken many forms, all with the aim of reducing the amount of programming required by an expert and instead allowing a human to transfer knowledge to an agent in a more natural way. As noted by Atkeson & Schaal (1997), many works prior to the mid-90s for teaching robots focused on an automatic programming process, driven by showing a robot a demonstration of the task, segmenting it into primitive actions and relationships, and then submitting these to a symbolic reasoning process. Bakker & Kuniyoshi (1996) give an overview of many of these earlier approaches (most of which came from a symbolic-AI perspective). In this paper, we focus on the most prominent approaches since that time, which we briefly categorize into three types:

1. *advising* (or simply telling) to the agent what it should do
2. *demonstrating* the agent examples of what it should do
3. *shaping* the agent by rewarding actions it takes as it runs autonomously

Advice: Advice, in the context of MDPs, is defined as suggesting an action when a certain condition is true. Maclin and Shavlik pioneered the approach of giving advice to reinforcement learners (Maclin and Shavlik 1996). Giving advice via natural language could be an effective way for non-technical humans to teach agents in some cases. This method was demonstrated by Kuhlmann et al. (Kuhlmann et al. 2004), who created a domain-specific natural language interface for giving advice to a reinforcement learner.

The informational richness of advice is clearly powerful, however there still remain a number of technical challenges. The first of these is that general natural language recognition is unsolved, so many current advice-taking systems (Maclin and Shavlik 1996; Moreno et al. 2004) require that the human encode her advice into a scripting or programming language, making it inaccessible to non-technical users. The natural language unit of Kuhlmann et al. (Kuhlmann et al. 2004) required manually labeled training samples. Moreover, work still remains on how to embed advice into agents that learn from experience.

Learning a general state-action mapping from examples: One of the most straightforward ways to teach an agent is simply to remote-control it using whatever interface is most convenient to the human, while the agent records state-action pairs (note that the recorded state variables, to be used by the learner, may be different from the state variables that the human operator has access to). One of the most celebrated examples of this type of teaching is ALVINN, a neural network based autonomous car (Pomerleau 1989), in which a human drove a car while it recorded the steering wheel position and camera sensors periodically. A neural network was then trained to map camera input to a distribution over output directions, and the chosen action was the center of mass of the output direction activations.

Many subsequent methods of teaching by demonstration have followed a similar approach, albeit using more sophisticated methods to learn an input-output mapping. For in-

Algorithm 1 RunAgent() for TAMER

Require: *Input: stepSize*
1: *ReinfModel.init(stepSize)*
2: $\overrightarrow{s} \leftarrow \overrightarrow{0}$
3: $\overrightarrow{f} \leftarrow \overrightarrow{0}$
4: $a \leftarrow -1$
5: **while** *true* **do**
6: $h \leftarrow$ *getHumanReinfSincePreviousTimeStep()*
7: **if** $h \neq 0$ **then**
8: *error* $\leftarrow$ *h - ReinfModel.predictReinf(*$\overrightarrow{f}$*)*
9: *ReinfModel.update(*$\overrightarrow{f}$*, error)*
10: **end if**
11: $\overrightarrow{s} \leftarrow$ *getStateVec()*
12: $a \leftarrow argmax_a$*(ReinfModel.predict(getFeatureVec(*$\overrightarrow{s}$*, a)))*
13: $\overrightarrow{f} \leftarrow$ *getFeatureVec(*$\overrightarrow{s}$*, a)*
14: *takeAction(a)*
15: wait for next time step
16: **end while**

stance, Grollman & Jenkins (2007b; 2007a) and Atkeson et al. (1997) have used methods such as locally weighted projection regression (Vijayakumar and Schaal 2000) and sparse online Gaussian processes (Csató and Opper 2002) to learn this mapping.

A slightly different form of learning from human demonstrations is *Apprenticeship Learning* (Abbeel and Ng 2004). In this method, the algorithm begins with an MDP\R and a human temporarily controls the agent. Instead of learning a policy that directly mimics the human, apprenticeship learning infers a reward function from the human-provided examples, and then uses reinforcement learning on an MDP consisting of a simulator (which models the state transitions) and the learned reward function. The goal is therefore to maximize the long-term discounted reward rather than to make a direct input-output mapping via supervised methods.

Considering demonstrations for which a human controls the agent, some tasks may be too difficult for a human trainer. This might be because the agent has more actuators than can be put in a simple interface (e.g., many robots) or because the task requires that the human be an expert before being able to control the agent (e.g., helicopter piloting in simulation). In these cases, a demonstration is infeasible.

Shaping:
Within the context of human-teachable agents, a human trainer shapes an agent by reinforcing successively improving approximations of the target behavior. Another name for this method is clicker training, which comes from a form of animal training in which an audible clicking device is previously associated with reinforcement and then used as a reinforcement signal itself to train the animal.

Most clicker training has involved teaching tricks to simulated or robot dogs. Kaplan et al. (2002) and Blumberg et al. (2002) respectively implement clicker training on a robotic and a simulated dog. Blumberg et al.'s system is especially interesting, allowing the dog to learn multi-action sequences and associate them with verbal cues. While interesting in that they are novel techniques of teaching pose sequences to their respective platforms, neither is evaluated using an explicit performance metric, and it remains unclear if and how these methods can be generalized to other, possibly more complex MDP settings.

Thomaz & Breazeal (Thomaz and Breazeal 2006) interfaced a human trainer with a table-based, Q-learning agent in a virtual kitchen environment. Their agent seeks to maximize its discounted total reward, which for any time step is the sum of human reinforcement and environmental reward.[4]

In another example of mixing human reward with ongoing reinforcement learning, Isbell et al. (Isbell et al. 2006) enable a social software agent, Cobot, to learn to model human preferences in the virtual world LambdaMOO. Cobot "uses reinforcement learning to proactively take action in this complex social environment, and adapts his behavior based on multiple sources of human reward." Like Thomaz & Breazeal, the agent doesn't explicitly learn to model the human reward function, but rather uses the human reward as a reward signal in a standard RL framework.

The TAMER system described above uses this shaping form of communication. The goal of TAMER is to allow human instruction to be as simple as possible, eliminating the need to create example data or use any kind of *programming* (even one based on natural language). It is unnecessary for the human trainer to give specific feedback about how bad or good an action is or exactly what went wrong – the human must only accurately indicate *when* things are generally good or bad. TAMER is distinguished from previous work on human-delivered reward in that it is designed to work in complex domains through function approximation.

Of particular interest is the fact that TAMER does not simply incorporate the human as part of the environment and treat human reinforcement as additional environmental reward. Instead, TAMER forms a model of human reinforcement and uses the model for greedy action selection, thus far reaching empirical success. The remainder of this paper discusses design principles for developing human-shapable agents, four of which are used in TAMER and then two more that we would eventually like to incorporate into the system.

Conjectures: using knowledge of human behavior to design human-teachable agents

The TAMER system, which models human behavior, is meant to be as simple for humans to use as possible. To design a shaping agent, we believe that it is important to make explicit use of knowledge about human behavior and incorporate it into the choice and use of the model.

In this section we list and discuss a series of conjectures about the behavior of human trainers that can serve as guidance in creating systems that use shaping feedback. These conjectures are not meant to be taken as statements of fact, but rather are meant to serve as hypotheses to guide our development and testing as we build more effective agents that learn from humans.

1. Modeling the reinforcement of a human trainer largely removes the problem of credit assignment. Supervised learning, not temporal difference learning, is therefore the correct approach.

In a typical Markov Decision Process, reward is given sparsely. A canonical example is a checkers game in which the reward signal is 0 until the end of the game (Mitchell 1997). In these MDPs, temporal difference learning and similar reinforcement learning approaches are used to infer the value (in terms of expected return) of non-reward states (or state-action pairs) given only this very sparse feedback.

We conjecture that a human that trains an agent in an MDP\R does not pose as challenging of a credit assignment problem. In particular, we suppose that the trainer's feedback reflects (in some way) their internal beliefs about the long-term effects of actions, and therefore human feedback is more closely related to the long-term value[5] of actions

[4] As indicated in footnote 2, combining human reinforcement and environmental reward into one reward signal is the narrower definition of shaping in reinforcement learning.

[5] "Value" is used here in the general sense of the word. We are not claiming that the human's mind contains a numerical repre-

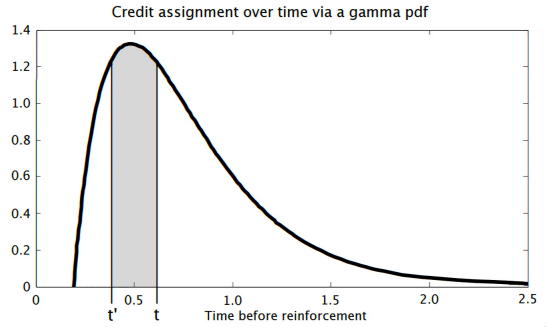

Figure 2: Probability density function $f(x)$ for a gamma(2.0, 0.28) distribution. Reinforcement signal h is received at time 0. If t and t' are times of consecutive time steps, credit for the time step at t is $h \times \int_{t'}^{t} f(x)dx$.

rather than the *immediate* goodness of actions. For instance, a human trainer may give negative feedback about a particular checkers move because the trainer knows that it will ultimately lead to poor results, even if it does no immediate harm in terms of number of pieces on the board.

Therefore we propose that a human trainer's reinforcement signal should be considered a label of the overall value of the recent state-action trajectory, and therefore it does not carry the full credit assignment problem that a traditional MDPs reward signal does. There is still some ambiguity in how to temporally assign credit, but we believe this problem is much simpler (as we explain in the next conjecture). Our positive results from applying TAMER to Tetris seem to support this first conjecture.

2. Assignment of credit from reinforcement by a human trainer should be similar to the Gamma probability density function shown in Figure 2.

Some task domains, such as Tetris, can occur at a rate of one action (i.e., a piece placement) per two seconds, or slower, without seeming unnatural. In such domains, a human trainer can easily give feedback to individual actions in the time between time steps. However, many interesting domains require actions at a much faster rate. Tasks that require visually fluid motion are examples of this faster class of domains. For such domains, the human trainer cannot accurately label individual state-action pairs in real time, and so a credit assignment scheme that distributes the reinforcement across recent time steps is required.

The psychology literature contains quite a large number of studies of human reaction time for decision making. An important one is William Hockley's study of the time required for a human subject to make a judgment about n items (requiring a visual search) and push a button to communicate their decision (Hockley 1984). Reacting during a visual search task is similar to the task that a human trainer confronts — watching behavior and evaluating it. Histogram results of Hockley's reaction time study are shown in Figure 3. As shown in Figure 2, a gamma distribution can be used to effectively approximate these histograms.[6] Note that there is a zero credit region from 0 seconds to 0.3 seconds, which represents the minimum delay of human response for this kind of task.

Therefore, we propose using a gamma distribution function to choose weights for credit assignment when using a human teacher in relatively fast, continually running tasks. Because we do not know the exact time since the event being responded to, we propose assigning credit based on how likely the human reward was given in response to each previous time step (illustrated in Figure 2). Specifically, let time 0 be when a reinforcement h is given. For a single time step that occurred t seconds before reinforcement and the time step immediately before it, occurring t' before reinforcement, credit for the time step at t is $h \times \int_{t'}^{t} f(x)dx$.

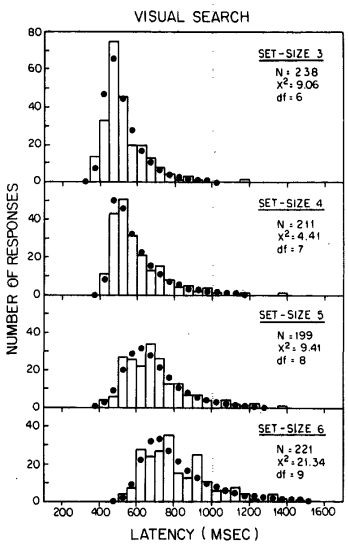

Figure 3: Hockley's study (Hockley 1984) of the distribution of human's negative response times to visual searches of different levels of complexity.

In Algorithm 2, an extension of Algorithm 1 with credit assignment added, we show a method for applying credit in incremental updates of a linear model.[7] Specifically, credit takes the form of a weight $credit$ between zero and one on the step size parameter α. Provided we have a way to use credit weights, we now are left with a question of what credit weighting function is best for making use of human reward signals.

Differences in individual response times, the complexity of the task, and other factors will likely affect the actual distribution of responses in any one training episode. One idea for an appropriate calibration is to perform a Bayesian update using response time data from the trainer while he or she performs a visual search task. The prior would be a distribution over the parameters of the Gamma distribution that is denser near the parameters we used to roughly approximate Hockley's results, and the likelihood would be calculated from the trainer's samples.

3. A human trainer's reinforcement function is a moving target.

Though much research has been done on how animals are best shaped (Ramirez 1999), relatively little work has examined the trainer and how he or she delivers reinforcement.

However, consider this example. Imagine training a dog to jump and touch a colored circle on a wall. At first you

sentation of the discounted sum of expected reward, as "value" is defined in reinforcement learning.

[6] Another good candidate is the lognormal distribution.

[7] A detailed explanation of this algorithm can be found at www.cs.utexas.edu/users/bradknox/papers/linearTAMER.pdf.

Algorithm 2 RunAgent() for TAMER with credit assignment using a linear model

Require: *Input: stepSize, windowSize,*
ReinfModel.init(stepSize)
Crediter.init(windowSize)
$\overrightarrow{s} \leftarrow \overrightarrow{0}$
$\overrightarrow{f} \leftarrow \overrightarrow{0}$
$a \leftarrow -1$
while *true* **do**
  Crediter.updateTime(clockTime())
  $h \leftarrow$ *getHumanReinfSincePreviousTimeStep()*
  if $h \neq 0$ **then**
    $\overrightarrow{credFeats} \leftarrow 0$
    for all $(\overrightarrow{f_t}, t) \in$ *Crediter.historyWindow* **do**
      $credit \leftarrow$ *Crediter.assignCredit(t)*
      $\overrightarrow{credFeats} \leftarrow \overrightarrow{credFeats} + (credit \times \overrightarrow{f_t})$
    end for
    error $\leftarrow h -$ *ReinfModel.predict*$(\overrightarrow{credFeats})$
    ReinfModel.update$(\overrightarrow{credFeats}$, *error)*
  end if
  $\overrightarrow{s} \leftarrow$ *getStateVec()*
  $a \leftarrow argmax_a$*(ReinfModel.predict(getFeatureVec(*$\overrightarrow{s}$*, a)))*
  $\overrightarrow{f} \leftarrow$ *getFeatureVec(*$\overrightarrow{s}$*, a)*
  takeAction(a)
  Crediter.updateWindow($\overrightarrow{f}$*)*
  wait for next time step
end while

might reinforce him for merely approaching the wall. Once the dog reliably does that, you would raise your standards, giving fewer and less emphatic rewards for approaching the wall and saving larger rewards for touching the wall. As the dog learns to touch the wall, you raise the standards again, making the dog touch higher on the wall. As a trainer, you would raise the standards for reinforcement as performance increases until the target behavior has been learned.

Using the sensible principle that teaching methods generally fit learning methods, we conjecture that as animals can be shaped by rewarding successive approximations of goal behavior (Bouton 2007), humans will naturally reward accordingly. Rewarding successive approximations means that the level of performance needed to receive a certain amount of reward increases as the performance increases. Improvement is rewarded, not absolute quality of performance. We do not punish a bright 5th grade math student for not knowing calculus, and we do not constantly reward a high school senior for knowing long division. Humans positively reinforce improvement. We negatively reinforce declines in performance. Thus the "reinforcement function" of the teacher changes as the agent learns and improves.

Though the authors are unaware of this conjecture previously being studied directly, the previously mentioned research on the social agent Cobot (Isbell et al. 2006) did find that a human reinforcement function is a moving target within their domain. Isbell et al. chose to mix both explicit training rewards with implicit rewards (e.g., from hugs) and run a reinforcement learning algorithm to learn socially adaptive behavior. They found that human reinforcement was a moving target. They report: "In all cases, as time progresses, Cobot is receiving less feedback... despite the fact that Cobot is learning [preferential] policies."

Therefore the designer of a system that allows human trainers to shape learning agents should choose a supervised learning algorithm that performs well with moving target functions. Certain recency-weighted function approximation algorithms used in temporal difference learning are thus good candidates and have been effective in recent work (Knox and Stone 2008). Examples of such algorithms include gradient descent over the weights of a linear model (which we use) and backpropagation in a multi-layer neural network.

4. Shaping reinforcement from a human trainer provides feedback on state-action pairs or state-next state pairs, not on state alone.

For most types of reinforcement learning agents, the programmer must decide whether the agent will learn a state-value function, $V : S \rightarrow \mathbb{R}$, or a state-action-value function, $Q : S \times A \rightarrow \mathbb{R}$. Though in this context "value" means a discounted sum of expected future reward, a designer of a human-shapable agent might wonder whether to model the trainer's reinforcement as a function of the state alone or the state-action pair.

In our work on TAMER, we found that modeling the reinforcement function by state alone, $H : S \rightarrow \mathbb{R}$, is ineffective, producing no appreciable learning. Modeling by state-action pair, $H : S \times A \rightarrow \mathbb{R}$, works well. Additionally, modeling by state-next state pair, $H : S \times S' \rightarrow \mathbb{R}$, may also be effective. The Tetris TAMER agent modeled a similar function with good results.

Considering the task of the trainer, this result is not surprising for MDPs that have a much larger state space than action space (as most commonly used MDPs do). For expository purposes, assume that the trainer has exactly one policy in mind that he or she wants to teach the agent to follow. Say the agent is modeling the human's reinforcement as a function of only state. To communicate the desired policy, the trainer must give reinforcement such that the model yields the correct preference ordering over all states. For even relatively small state spaces, ordering them correctly would be extremely difficult. Now say the agent is modeling a function of state-action pairs. Then, for any given state, the trainer merely has to give reinforcement such that the best action has the highest expected reinforcement.

Another disadvantage of modeling the trainer's reinforcement as a function of the state is that the agent would need a model of the transition function, $H : S \times A \times S' \rightarrow \mathbb{R}$, to choose a next state. Learning a transition function is a challenging problem and an active area of research.

5. If the environment is an MDP, a human-shapable agent should treat the human trainer differently from the rest of the environment.

How to formally include the reinforcement signal from a human trainer in a description of the agent's environment is an open question. If the environment without the human trainer