To appear in The Fifth International Conference on Knowledge Capture (K-CAP 09), Redondo Beach, California, USA, September 2009.

Interactively Shaping Agents via Human Reinforcement

The TAMER Framework

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

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

ABSTRACT
As computational learning agents move into domains that incur real costs (e.g., autonomous driving or financial investment), it will be necessary to learn good policies without numerous high-cost learning trials. One promising approach to reducing sample complexity of learning a task is knowledge transfer from humans to agents. Ideally, methods of transfer should be accessible to anyone with task knowledge, regardless of that person's expertise in programming and AI. This paper focuses on allowing a human trainer to interactively shape an agent's policy via reinforcement signals. Specifically, the paper introduces "Training an Agent Manually via Evaluative Reinforcement," or TAMER, a framework that enables such shaping. Differing from previous approaches to interactive shaping, a TAMER agent models the human's reinforcement and exploits its model by choosing actions expected to be most highly reinforced. Results from two domains demonstrate that lay users can train TAMER agents without defining an environmental reward function (as in an MDP) *and* indicate that human training within the TAMER framework can reduce sample complexity over autonomous learning algorithms.

Keywords
human-agent interaction, shaping, sequential decision-making, human teachers, learning agents

1. INTRODUCTION
As computational learning agents continue to improve their ability to learn sequential decision-making tasks, a central but largely unfulfilled goal of the field is to deploy these agents in real-world domains, making decisions that affect our lives. However, with real-world deployment comes real-world costs. For such a deployment to be viable, agents will not be able to use hundreds or thousands of learning trials to reach a good policy when each suboptimal trial is costly. For example, an autonomous driving agent should not learn to drive by crashing into road barriers and endangering the lives of pedestrians.

Fortunately, for many of these tasks, humans have domain knowledge that could speed the learning process, reducing costly sample complexity. Currently, most knowledge transfer from humans to agents occurs via programming, which is time-consuming and inaccessible to the general public. It is important to develop agents that can learn from natural methods of communication. The teaching technique of shaping is one such method. In this context, we define *shaping* as interactively training an agent through signals of positive and negative reinforcement.[1] In a shaping scenario, a human trainer observes an agent and reinforces its behavior through push-buttons, spoken word ("yes" or "no"), or any other signal that can be converted to a scalar signal of approval or disapproval. The key challenge, then, is to create agents that can be shaped effectively. Our problem definition is as follows:

The Shaping problem Within a sequential decision-making task, an agent receives a sequence of state descriptions ($s_1, s_2, ...$ where $s_i \epsilon S$) and action opportunities (choosing $a_i \epsilon A$ at each s_i). From a human trainer who observes the agent and understands a predefined performance metric, the agent also receives occasional positive and negative scalar reinforcement signals ($h_1, h_2, ...$) that are correlated with the trainer's assessment of recent state-action pairs. How can an agent learn the best possible task policy ($\pi : S \rightarrow A$), as measured by the performance metric, given the information

[1]We use the term "shaping" as it is used in animal learning literature (in which it was initially developed by B.F. Skinner). There, shaping is defined as training by reinforcing successively improving approximations of the target behavior [6]. 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 [14]. An important feature of our TAMER framework is that the two signals are not combined.

contained in the input?

In a Markov Decision Process (MDP), the environmental reward signal, along with the rest of the MDP specification (see [16] for details), unambiguously define a set of optimal policies. Since our goal is to allow the human trainer to fully control the agent's behavior, the environmental reward signal is not used. Following Abbeel and Ng's terminology [1], we call this an MDP\R.

Expected benefits of learning from human reinforcement[2] include the following.

1. Shaping decreases sample complexity for learning a "good" policy.
2. An agent can learn in the absence of a coded evaluation function (e.g., an environmental reward function).
3. The simple mode of communication allows lay users to teach agents the policies which they prefer, even changing the desired policy if they choose.
4. Shaped agents can learn in more complex domains than autonomous learning allows.

This paper addresses the first three of these benefits. In it, we review previous work on agents that can learn from a human teacher through natural methods of communication (Section 2). We treat shaping as a specific mode of knowledge transfer, distinct from (and probably complementary to) other natural methods of communication, including programming by demonstration and giving advice. Shaping only requires that a person can observe the agent's behavior, judge its quality, and send a feedback signal that can be mapped to a scalar value (e.g. by button press or verbal feedback of "good" and "bad"). We then present a novel method by which human trainers can shape agents (Section 3). This agent-trainer framework, called Training an Agent Manually via Evaluative Reinforcement (TAMER), makes use of established supervised learning techniques to model a human's reinforcement function and then uses the learned model to choose actions that are projected to receive the most reinforcement. The TAMER framework, the insights that motivate it, and an extension to delayed reinforcement are the key contributions of this paper. Furthermore, we describe two specific, fully implemented TAMER algorithms that provide proofs-of-concept for two contrasting task domains (Section 4). Experimental results in these domains show that TAMER agents, under the tutelage of human trainers, learn a "good" policy faster than effective autonomous learning agents (Section 5).

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

2. RELATED WORK: LEARNING FROM A HUMAN

Work on human-teachable agents has taken many forms, all with the aim of reducing the amount of programming required by an expert, using more natural modes of communication. In this section, we describe past work on agents that learn from humans. We argue that the relative strengths and weaknesses of our training system put it in a unique space that is not currently occupied by any other approach. Furthermore, we believe that many of the approaches we review are complementary to ours: an ideal learning agent might combine elements from several of them.

2.1 Learning from Advice

Advice, in the context of Markov Decision Processes (MDPs), is defined as suggesting an action when a certain condition is true. Maclin and Shavlik [12] pioneered the approach of giving advice to reinforcement learners. Giving advice via natural language could be an effective way for non-technical humans to teach agents. Kuhlmann et al. [11] 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 [12, 13] 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. [11] required manually labeled training samples. Moreover, work still remains on how to embed advice into agents that learn from experience.

Additionally, there will likely be times when the trainer knows that the agent has performed well or poorly, but cannot determine exactly why. In these cases, advice will be much more difficult to give than positive or negative feedback.

2.2 Learning from Demonstration

Another way for a human to teach an agent is to demonstrate a task via remote-control or with his own body, while the agent records state-action pairs, from which it learns a general policy for the task [3].

In Apprenticeship Learning [1], a type of learning from demonstration, the algorithm begins with an MDP\R (as does the TAMER framework; see Section 1). The algorithm learns a reward function R from a human's period of control, and then the agent trains on the MDP.

Considering the demonstration type in which a human controls the agent, there are some tasks that are 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. But as long as the human can judge the overall quality of the agent's behavior, then he or she should be able to provide feedback via TAMER regardless of the task's difficulty.

2.3 Learning from Reinforcement (Shaping)

Within the context of human-teachable agents, a human trainer shapes an agent by reinforcing successively improving approximations of the target behavior. When the trainer can only give positive reinforcements, this method is sometimes called *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.

Previous work on clicker training has involved teaching tricks to entertainment agents. Kaplan et al. [9] and Blumberg et al. [4] implement clicker training on robotic and simulated dogs, respectively. Blumberg et al.'s system is especially interesting, allowing the dog to learn multi-action sequences and associate them with verbal cues. Though significant 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 [18] 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.[3]

In another example of mixing human reinforcement with on-going reinforcement learning, Isbell et al. [8] enable a social software agent, Cobot, to learn to model human preferences in LambdaMOO. Cobot "uses reinforcement learning to proactively take action in this complex social environment, and adapts his behavior based on multiple sources of human [reinforcement]." Like Thomaz and Breazeal, the agent doesn't explicitly learn to model the human reinforcement function, but rather uses the human reinforcement as a reward signal in a standard RL framework.

The TAMER system is distinct from previous work on human-delivered reinforcement in that it is designed both for a human-agent team and to work in complex domains through function approximation, generalizing to unseen states. It also uniquely forms a model of the human trainer's intentional reinforcement, which it exploits for action selection in the presence or absence of the human trainer.

[3]As indicated in Footnote 1, combining human reinforcement and environmental reward into one reward signal is the narrower definition of shaping in reinforcement learning. As argued in [10], h and r are fundamentally different signals that contain different information and thus should be treated differently.

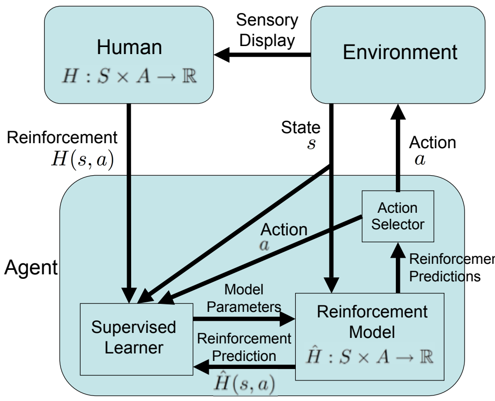

Figure 1: Framework for Training an Agent Manually via Evaluative Reinforcement (TAMER).

3. THE TAMER FRAMEWORK

Consider the Shaping problem description given in Section 1. The agent's role is by nature somewhat underconstrained: it must learn from reinforcement such that it maximizes performance (which it cannot measure). We hypothesize that the agent best fills its role by learning a model of the trainer's reinforcement and exploiting the model in action selection.

A TAMER agent seeks to learn the human trainer's reinforcement function $H : S \times A \to \mathbb{R}$. Presented with a state s, the agent consults its learned model $\hat{H}$ and, if choosing greedily, takes the action a that maximizes $\hat{H}(s, a)$. Figure 1 shows interaction between a human, the environment, and a TAMER agent within an MDP\R. 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.

3.1 Motivating Insights

When we speak of autonomous learning agents, we mean those that receive feedback in the form of environmental reward (i.e., reinforcement learning agents [16]). The principal challenge for such agents is, upon receiving environmental reward, to assign credit from that reward to the entire history of past state-action pairs. In domains where the only discriminating reward is at the end of the task, assigning this credit can be particularly difficult. A key insight of the TAMER framework is that the problem of credit assignment inherent in reinforcement learning is no longer present with an attentive human trainer. The trainer can evaluate an action or short sequence of actions, considering the long-term effects of each, and deliver positive or negative feedback within a small temporal window after the behavior. Assigning credit within that window presents a challenge in itself, which we discuss in Section 3.3, but it is much easier than assigning credit for an arbitrarily delayed reward signal as in reinforcement learning. Assuming, for now, that credit is properly assigned within the temporal window, we assert that a trainer can directly label behavior. Therefore, modeling the trainer's reinforcement function H is a supervised learning problem.

Intuition and some evidence [8] suggest that a human trainer's reinforcement function, H, is a moving target. Intuitively, it seems likely that a human trainer will raise his or her standards as the agent's policy improves, expecting more to get the same reinforcement. Though we have not experimentally studied this claim that a human's reinforcement function is a moving target, it is discussed in further detail in [10]. Considering our conjecture that H is a moving target function and that the TAMER agent should be able to exploit new data as it becomes available, the supervised learning algorithm should both be able to handle changes in the target function and allow incremental updates.[4] Additionally, for best results, the algorithm should generalize to all unseen samples (i.e., to states and possibly actions). Examples of such algorithms include gradient descent over the weights of a linear model (which we use in our experiments) and backpropagation in a multi-layer neural network.

As we will report in Section 5, compared to autonomous learning algorithms, TAMER dramatically reduces the number of episodes required to learn a good policy. One result of lowering the sample complexity is that if the incremental updates to the model use a small step size, the model's output will not move far enough to ever be the "true" human reinforcement function. But ideally the learned function $\hat{H}$ should map to the human reinforcement function H such that

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

In this case, the policies created by greedily exploiting $\hat{H}$ and H are equivalent.

3.1.1 Exploration

The TAMER framework is agnostic to exploration, leaving action selection as a black box to be filled by the agent designer. However, we have found that greedily choosing the action that is expected to receive the highest reinforcement provides sufficient exploration for many tasks, including those described in Section 5. This counterintuitive finding is easily justified. In a given state, a good trainer negatively reinforces any undesired action, eventually dropping its expected reinforcement below another action, resulting in a new (exploratory) action choice. A non-greedy action selection method is needed if there are actions of which the trainer is not aware or does not know the value. Otherwise, the trainer should be able to guide the TAMER agent's exploration sufficiently under greedy action selection.

[4]Supervised learning algorithms that have been used successfully as function approximators for reinforcement learning fit these two requirements and thus make good candidates for modeling the human.

Algorithm 1 A general greedy TAMER algorithm

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

3.1.2 Comparison with Reinforcement Learning

The TAMER framework for shaping agents shares much common ground with reinforcement learning, but there are some key differences that are important to understand in order to fully appreciate TAMER.

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 short-term reinforcement given by the human trainer. It does this because the trainer's reinforcement signal is a direct label on 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 the environmental reward within MDPs function dictates the optimal policy, so R is removed to make the human's reinforcement function the sole determinant of good and bad behavior.

Many reinforcement learning algorithms can be easily converted to TAMER algorithms. We encourage interested readers to follow our step-by-step instructions at http://www.cs.utexas.edu/users/bradknox/kcap09.

3.2 High-Level Algorithm

A high-level algorithm for implementing TAMER is described in Algorithm 1. After initialization of the reinforcement model *ReinfModel*, the state vector $\vec{s}$, and the feature vector $\vec{f}$ (lines 1-3), the algorithm begins a loop that occurs once per time step (line 4).

In the loop, the agent first obtains a scalar measurement of the human trainer's reinforcement since the previous time step (line 5). 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 (line 7). The model, left undefined for the general TAMER algorithm, can be instantiated in multiple ways as discussed below. The calculated error is then used, along with the previous feature vector, to update the reinforcement model (line 8). The update is not performed when $h = 0$ because the trainer may not be paying attention or might even have quit training altogether, satisfied with the trained policy.

After the agent obtains the new state description, it then greedily chooses the action a that, according to the human reinforcement model, yields the largest predicted reinforcement (lines 10-11). Although we have found that greedy action selection is sufficient in our work thus far, more sophisticated action selection methods may be needed in some situations, and would fit just as well within the TAMER framework. The agent calculates features $\overrightarrow{f}$ of the state and action of the current time step (line 12) and takes the chosen action (line 13) before restarting the loop.

3.3 Credit Assignment to a Dense State-Action History

In some task domains, the frequency of time steps is too high for human trainers to respond to specific state-action pairs before the next one occurs. In simulation, it is technically possible to lower the frequency of time steps. But doing so may change the character of the task, and in physical domains (e.g. helicopter flight) it may not be possible. Although we have not studied exactly where the frequency threshold is, our experience and studies of human response times (Figure 2) [7] have made it clear that having several seconds between time steps is enough for specific labeling and, conversely, having several time steps per second is too frequent for specific labeling.

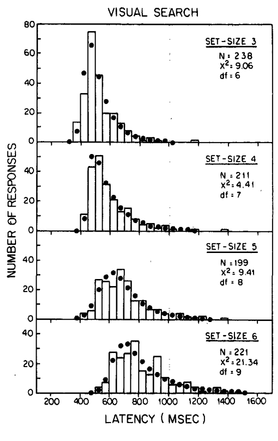

Figure 2: Hockley's study of the distribution of human response times for visual searches of different levels of complexity.

For these faster domains, credit from the human's reinforcement must be appropriately distributed across some subset of the previous time steps. Our approach for credit assignment with a linear model and gradient descent updates is shown in Algorithm 2, which is an extension of Algorithm 1 with the model inline. The error for an update is the difference between the projected reinforcement predicted by the linear model, where the input is a weighted sum of the state-action features for each preceding time step (lines 9-13), and the actual reinforcement received (line 14). The weight for any time step is its "credit". The model is then updated with that error and the weighted sum of features (line 15).

3.3.1 Credit Calculation

To assign credit, we assume that the reinforcement signal (received at time 0) is targeting a single state-action pair. This assumption will undoubtedly be wrong at times, but the effects of the assumption will probably be small since the credit is still spread out amongst recent events, as described below. There are n time steps that might be the target, beginning at times $t_1, t_2, ..., t_n$, where if $i < j$, then t_i occurred more recently than t_j. (Therefore the time step beginning at t_i ends at time t_{i-1} if $i > 1$.) We define credit c_t for a time step starting at time t_i[5] and ending at t_{i-1} (or 0 if i = 1) to be the probability that the reinforcement signal was given for the event (s, a) that occurred during that time step.

We create a probability density function f over the delay of the human's reinforcement signal. Then for any state-action pair, credit is calculated as the integral of f from t_{i-1} to t_i.

$$\begin{aligned} c_t &= P(\textit{event starting at } t_i \textit{ was targeted}) \\ &= P(\textit{target event between } t_{i-1} \textit{ and } t_i \textit{ sec. ago}) \\ &= \int_{t_{i-1}}^{t_i} f(x)dx \end{aligned}$$

(If $i = 1$, then $t_{i-1} = 0$ in the above equation.) We observe the constraint

$$\sum_{i=1}^{n} P(\textit{event starting at } t_i \textit{ was targeted}) = 1$$

An example probability density function is shown in Figure 3. In practice, we maintain a window of recent time steps, pruning out those time steps that are older and have near-zero probability (lines 10 and 21). We also always use functions that have zero probability before 0.2 seconds, assuming that human trainers cannot respond in less than 0.2 seconds (supported by Figure 2).

4. IMPLEMENTED ALGORITHMS

TAMER is fully implemented and tested in two contrasting domains. Though the TAMER agent actively seeks to maximize predicted human reinforcement H, the true objective of the human-agent system is to maximize some performance criteria, whether it be the trainer's subjective evaluation or an explicitly defined metric. For this reason, and because measuring directly against H is impractical to impossible, we evaluate TAMER using R to provide the performance metric, which is the cumulative MDP reward received throughout all episodes

[5]A time step can be thought of as a slice in time or an interval in time. Considering that a choice to take action a in state s changes the display seen by the human trainer until the next time step, we consider a time step to be an interval.

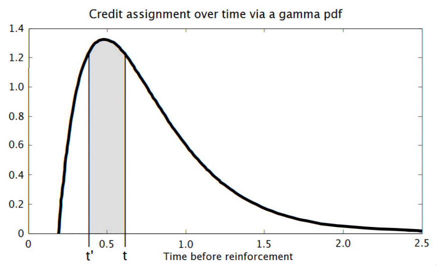

Figure 3: Probability density function $f(x)$ for a gamma(2.0, 0.28) distribution. Reinforcement signal h is received at time 0. If t and t' are times of consecutive time steps, credit for the time step at t is $\int_{t'}^{t} f(x)dx$. Note that time moves backwards as one moves right along the x-axis.

Algorithm 2 A greedy TAMER algorithm with credit assignment, using a linear model and gradient descent updates

```
Require: Input: stepSize, windowSize,
 1: Crediter.init(windowSize)
 2: $\overrightarrow{s} \leftarrow \overrightarrow{0}$
 3: $\overrightarrow{f} \leftarrow \overrightarrow{0}$
 4: $\overrightarrow{w} \leftarrow \overrightarrow{0}$
 5: while true do
 6:     Crediter.updateTime(clockTime())
 7:     $h \leftarrow$ getHumanReinfSincePreviousTimeStep()
 8:     if $h \neq 0$ then
 9:         $\overrightarrow{credFeats} \leftarrow 0$
10:         for all $(\overrightarrow{f_t}, t) \in$ Crediter.historyWindow do
11:             $c_t \leftarrow$ Crediter.assignCredit(t)
12:             $\overrightarrow{credFeats} \leftarrow \overrightarrow{credFeats} + (c_t \times \overrightarrow{f_t})$
13:         end for
14:         $error \leftarrow h - (\overrightarrow{w} \cdot \overrightarrow{credFeats})$
15:         $\overrightarrow{w} \leftarrow \overrightarrow{w} + (stepSize \times error \times \overrightarrow{credFeats})$
16:     end if
17:     $\overrightarrow{s} \leftarrow$ getStateVec()
18:     $a \leftarrow argmax_a(\overrightarrow{w} \cdot (getFeatures(\overrightarrow{s}, a)))$
19:     $\overrightarrow{f} \leftarrow$ getFeatures($\overrightarrow{s}$, a)
20:     takeAction(a)
21:     Crediter.updateWindow($\overrightarrow{f}$)
22:     wait for next time step
23: end while
```

in a run. In each domain, R (ignored by the agent) was easily communicated to the trainer, who was instructed to train his or her agent to earn the highest sum of environmental reward per trial episode.[6] Also, in each domain the state and action are displayed graphically to the human trainer. Both task domains were implemented within the RL-Library.[7] One domain is Tetris, a puzzle computer game with a large state space and a variable number of actions per time step (as is standard in the literature, a final placement of a falling Tetris piece is considered an action). The time step frequency in Tetris, at less than one action per second, allowed human trainers to easily reinforce individual actions.

The other domain is Mountain Car, in which a simulated car must get to the top of a hill. The car begins between two steep hills and must go back and forth to gain enough momentum to reach the goal. Mountain Car has a continuous, two-dimensional state space. In our experiments, actions occurred approximately every 150 milliseconds, preventing trainers from labeling specific actions. Also, agents started each episode in a random location within bounds around the bottom of the hill.

These two domains complement each other. Tetris has a complex state-action space and low time step frequency, and Mountain Car is simpler but occurs at a high frequency.

4.1 Tetris

The algorithm used to implement TAMER on Tetris is a specific implementation of Algorithm 1, using a linear model and gradient descent over the weights of each feature. The function *getFeatures*($\overrightarrow{s}$, *a*) on line 13 of the algorithm considers feature vectors derived from both the current state and the next state (before the new Tetris piece appears). The next state is merely the previous state with the new blocks in place (which the action dictates) with any full horizontal lines removed. The difference between the two feature vectors is returned and actually used by TAMER as the features over which to learn.

4.2 Mountain Car

The Mountain Car TAMER algorithm follows the algorithm given for TAMER with credit assignment described in Algorithm 2, also using a linear model with gradient descent updates. The function *getFeatureVec*($\overrightarrow{s}$, *a*) on lines 19 and 20 is implemented using 2-dimensional Gaussian radial basis functions as described in [16]. Credit assignment followed a Uniform(0.2, 0.6) distribution (described in more detail in Section 3.3).

5. RESULTS AND DISCUSSION

Our experiments test the first three of the the expected benefits of shaping that are described in the Introduction: shaping decreases sample complexity for learning a "good" policy, an agent can learn in the absence of a coded evaluation function, and lay users can teach agents the policies they prefer. For the experiments, human trainers observed the agents in simulation on a computer screen. Positive and negative reinforcement was given via two keys on the keyboard. The trainers were read instructions and were not told anything about the agent's features or learning algorithm. Nine humans trained Tetris agents. Nineteen trained Mountain Car agents. For each task, at least a fourth of the trainers did not know how to program a computer.

5.1 Tetris

Tetris is notoriously difficult for temporal difference learning methods that model a function such as a value or action-value function. In our own work, we were only able to get Sarsa(λ) [16] to clear approximately 30 lines per game with a very small step size α and after hundreds of games. Bertsekas and Tsitiklis report that they were unable to get optimistic TD(lambda) to make

[6]Our Tetris specification follows that described by Bertsekas and Tsitsiklis [2]. Experimental details and videos of agents before, during, and after training can be found at http://www.cs.utexas.edu/users/bradknox/kcap09.

[7]http://code.google.com/p/rl-library/

"substantial progress". RRL-KBR [15] does somewhat better, getting to 50 lines per game after 120 games or so.[8] The only successful approach that learns a value function (of those found by the authors) is λ-policy iteration [2], which reaches several thousand lines after approximately 50 games.

However, λ-policy iteration differs in two important ways from the rest of these value-based approaches. First, it begins with hand-coded weights that already achieve about 30 lines per game. Getting to that level of play is nontrivial, and some learning algorithms, such as Sarsa(λ), fail to even reach it when starting with all weights initialized to zero. Second, λ-policy iteration gathers many state transitions from 5 games and performs a least-squares batch update. All other successful learning methods likewise perform batch updates after observing a policy for many games (as many as 500 in the best-performing algorithms), likely because of the high stochasticity of Tetris. Additionally, of those that have the necessary data available, all previous algorithms that model a function are unstable after reaching peak performance, unlearning substantially until they clear less than half as many lines as their peak.

In our experiments, human trainers practiced for two runs. Data from the third run is reported. Of the algorithms that model an actual function, our gradient descent, linear TAMER is the only one that clearly does not unlearn in Tetris.[9] Of those that also perform incremental updates, TAMER learns the fastest and to the highest final performance, reaching 65.89 lines per game by the third game (Table 1 and Figure 4).[10]

Policy search algorithms, which do not model a value or reinforcement function, attain the best final performance, reaching hundreds of thousands of lines per game [5, 17], though they require many games to get there and do not learn within the first 500 games.

Within our analysis, TAMER agents learn much more quickly than all previously reported agents and reach a final performance that is higher than all other incrementally updating algorithms.

[8]We should note that Ramon et. al. rejected a form of their algorithm that reached about 42 lines cleared on the third game. They deemed it unsatisfactory because it unlearned by the fifth game and never improved again, eventually performing worse than randomly. Ramon et al.'s agent is the only one we found that approaches the performance of our system after 3 games.

[9]This statement is supported by training by one of the authors but not necessarily by the subjects, since the trainer subjects usually stopped training after the TAMER agent reached satisfactory performance.

[10]Most trainers stopped giving feedback by the end of the fifth game, stating that they did not think they could train the agent to play any better. Therefore most agents are operating with a static policy by the sixth game. Score variations come from the stochasticity inherit in Tetris, including the highest scoring game of all trainers (809 lines cleared), which noticeably brings the average score of game 9 above that of the other games.

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 [15]	5	50	120
Policy Iteration [2]	~ 0 (no learning until game 100)	3183	1500
Genetic Algorithm [5]	~ 0 (no learning until game 500)	586,103	3000
CE+RL [17]	~ 0 (no learning until game 100)	348,895	5000

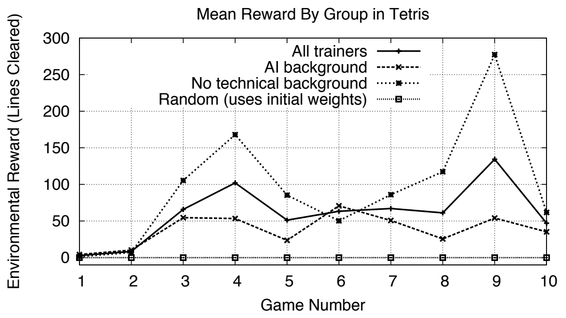

Figure 4: The mean number of lines cleared per game by experimental group.

5.2 Mountain Car

The Tetris results demonstrate TAMER's effectiveness in a domain with relatively infrequent actions. Conversely, our experiments in Mountain Car test its performance in a domain with frequent actions. The autonomous algorithm used for comparison was Sarsa(λ) [16] with the same function approximator (a linear model over Gaussian RBF features, using gradient descent updates), an algorithm that is known to perform well on Mountain Car. Two Sarsa(λ) agents were used: one tuned for total cumulative environmental reward across all previous episodes (Figure 6) after 3 episodes and one tuned for after 20 episodes (which we will refer to as Sarsa-3 and Sarsa-20, respectively). We tuned via a hill-climbing algorithm that varied one parameter (α, λ, or ϵ in the standard notation of Sarsa [16]) at a time, testing the agent's performance under each value for that parameter, taking the best performing value, and then repeating (for fifty or more iterations). The specific number of episodes (3 and 20) were chosen to exhibit different emphases on the trade-off between learning quickly and reaching the best asymptotic performance.

The TAMER agents were shaped for three runs of twenty episodes by each trainer. We consider the first run a practice run for the trainer and present the combined data from second and third runs. Results are shown in Figures 5 and 6. Figure 5(a) shows that the TAMER agents, on average, consistently outperformed the Sarsa-3 agent and outperformed the Sarsa-20 for the first five episodes, after which Sarsa-20 agents showed comparable performance. Under the guidance of the best trainers (Figure 5(b)), TAMER agents consistently

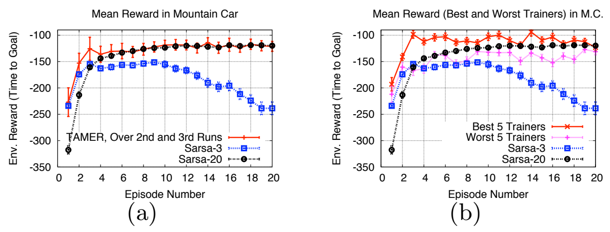

Figure 5: Error bars show a 95% confidence interval (with a Gaussian assumption). (a) The mean environmental reward (-1 per time step) received for the Mountain Car task for the second and third agents (runs) shaped by each trainer under TAMER and for autonomous agents using Sarsa(λ), using parameters tuned for best cumulative reward after 3 and 20 episodes. (b) The average amount of environmental reward received by agents shaped by the best five and worst five trainers, as determined over all three runs.

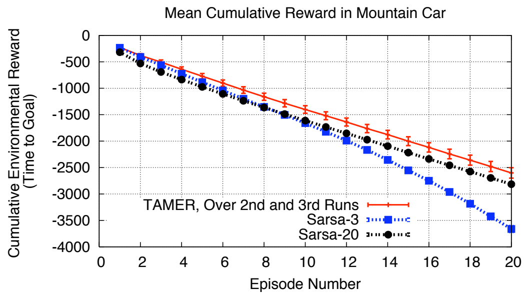

Figure 6: Mean cumulative environmental reward received for the Mountain Car task.

outperform any Sarsa agent, and, under the worst trainers, they perform somewhat worse than the Sarsa-20 agent. Most importantly, TAMER agents, on average, also outperformed each Sarsa agent in mean cumulative environmental reward through the length of a run (Figure 6). Since each time step incurred a -1 environmental reward, Figure 6 is also a measure of sample complexity. The four trainers who did not have a computer science background achieved performance as good or marginally better than the fifteen who did. Overall, the results, though less dramatic than those for Tetris, support our claim that TAMER can reduce sample complexity over autonomous algorithms.

6. CONCLUSION AND FUTURE WORK

The TAMER framework, which allows human trainers to shape agents via positive and negative reinforcement, provides an easy-to-implement technique that:

1. works in the absence of an environmental reward function,
2. reduces sample complexity, and
3. is accessible to people who lack knowledge of computer science.

Our experimental data suggests that TAMER agents outperform autonomous learning agents in the short-term, arriving at a “good” policy after very few learning trials. It also suggests that well-tuned autonomous agents are better at maximizing final, peak performance after many more trials.

Given this difference in strengths, we aim to explore how best to use both human reinforcement, H, and, when available, environmental reward, R, relying on the former more heavily for early learning and on the latter for fine-tuning to achieve better results than either method can achieve in isolation. We would also like to implement TAMER in tasks that are currently intractable for autonomous learning algorithms. Further, we wish to investigate how TAMER might need to be modified for tasks in which a trainer cannot evaluate behavior within a small temporal window (e.g., the behavior is hidden at times or the results of the behavior are delayed).

7. ACKNOWLEDGMENTS

We thank Ian Fasel for his contributions to previous work in this line of research. This research is supported in part by an NSF Graduate Research Fellowship and the DARPA IPTO Bootstrap Learning program.

8. REFERENCES

[1] P. Abbeel and A. Ng. Apprenticeship learning via inverse reinforcement learning. *ACM International Conference Proceeding Series*, 2004.

[2] D. Bertsekas and J. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[3] A. Billard, S. Calinon, R. Dillmann, and S. Schaal. Robot programming by demonstration. In B. Siciliano and O. Khatib, editors, *Handbook of Robotics.* Springer, 2008.

[4] B. Blumberg, M. Downie, Y. Ivanov, M. Berlin, M. Johnson, and B. Tomlinson. Integrated learning for interactive synthetic characters. *Proc. of the 29th annual conference on Computer graphics and interactive techniques*, 2002.

[5] N. Bohm, G. Kokai, and S. Mandl. Evolving a heuristic function for the game of Tetris. *Proc. Lernen, Wissensentdeckung und Adaptivitat LWA*, 2004.

[6] M. Bouton. *Learning and Behavior: A Contemporary Synthesis.* Sinauer Associates, 2007.

[7] W. E. Hockley. Analysis of response time distributions in the study of cognitive processes. *Journal of experimental psychology. Learning, memory, and cognition*, 10, 1984.

[8] C. Isbell, M. Kearns, S. Singh, C. Shelton, P. Stone, and D. Kormann. Cobot in LambdaMOO: An Adaptive Social Statistics Agent. *AAMAS*, 2006.

[9] F. Kaplan, P. Oudeyer, E. Kubinyi, and A. Miklósi. Robotic clicker training. *Robotics and Autonomous Systems*, 38(3-4), 2002.

[10] W. B. Knox, I. Fasel, and P. Stone. Design principles for creating human-shapable agents. In *AAAI Spring 2009 Symposium on Agents that Learn from Human Teachers*, March 2009.

[11] G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in RoboCup soccer. In *The AAAI-2004 Workshop on Supervisory Control of Learning and Adaptive Systems*, July 2004.

[12] R. Maclin and J. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1):251–281, 1996.

[13] D. Moreno, C. Regueiro, R. Iglesias, and S. Barro. Using prior knowledge to improve reinforcement learning in mobile robotics. *Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK*, 2004.

[14] A. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. *ICML*, 1999.

[15] J. Ramon and K. Driessens. On the numeric stability of gaussian processes regression for relational reinforcement learning. *ICML-2004 Workshop on Relational Reinforcement Learning*, pages 10–14, 2004.

[16] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

[17] I. Szita and A. Lorincz. Learning Tetris Using the Noisy Cross-Entropy Method. *Neural Computation*, 18(12), 2006.

[18] A. Thomaz and C. Breazeal. Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. *AAAI*, 2006.