

# Learning via Human Feedback in Continuous State and Action Spaces

Ngo Anh Vien and Wolfgang Ertel

Institute of Artificial Intelligence

Ravensburg-Weingarten University of Applied Sciences

Weingarten 88250, Germany

{ngo,ertel}@hs-weingarten.de

## Abstract

We consider the problem of extending manually trained agents via evaluative reinforcement (TAMER) in continuous state and action spaces. The early work TAMER framework allows a non-technical human to train an agent through a natural form of human feedback, negative or positive. The advantages of TAMER have been shown on applications such as training Tetris and Mountain Car with only human feedback, Cart-pole and Mountain Car with human feedback and environment reward (augmenting reinforcement learning with human feedback). However, those methods are originally designed for discrete state-action, or continuous state-discrete action problems. In this paper, we introduce an extension of TAMER to allow both continuous states and actions. The new scheme, actor-critic TAMER, extends the original TAMER to allow using any general function approximation of a human trainer's reinforcement signal. Our extension still allows reinforcement learning to be easily combined with human feedback. The experimental results show that the proposed method helps a human trainer successfully train an agent in two continuous state-action domains: Mountain Car, and Cart-pole (balancing).

## Introduction

Training an Agent Manually via Evaluative Reinforcement (TAMER) is a framework helping with the design of agents trained by human trainer's feedbacks of negative and positive signals (Knox and Stone 2009). The TAMER framework has performed well in tasks in which the human trainer already has significant knowledge. It does not require from the human trainer any prior technical or programming skills in order to transfer knowledge to agents. In some situations, it could reduce the cost of the agent's learning progress without damaging the asymptotic performance. Some successful examples can be named as Tetris, Mountain Car, Cart-pole (Knox and Stone 2009; 2012). The results show that TAMER helps agents to reduce the sample complexity when learning within a Markov Decision Process (MDP).

Another application of TAMER is to combine it with reinforcement learning (RL) algorithms to make its learning

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

curve climb up. There are many possible sequential and simultaneous combinations between TAMER and RL (Knox and Stone 2010a; 2010b; Knox, Setapen, and Stone 2011; Knox and Stone 2011; 2012). In the sequential combination scheme, the TAMER agent is first trained by a human trainer, then reinforcement learning is used as an autonomous learning agent which is shaped in many ways by the TAMER agent's optimal value function. More interesting, the simultaneous scheme allows a human trainer to interactively train the agent at any time during an autonomous learning process of reinforcement learning. However, TAMER is originally designed only for discrete state-action, or continuous state-discrete action problems. This property could limit the widespread applicability of the TAMER framework, especially in the combination with reinforcement learning which has provided many efficient algorithms in continuous state-action space domains.

In this paper, we introduce an extension of TAMER to make it applicable to continuous state and action domains. The new scheme, actor-critic TAMER, extends the original TAMER to allow for general reinforcement function approximation of a human trainer. The TAMER framework is implemented as the critic which can use any forms of the human trainer's function approximator. The actor implements a policy gradient algorithm in which the trainer's preference policy is in a parametric form. The critic is used to evaluate the actor's performance, and its temporal prediction error is used to update the actor's parameters. The proposed extension still allows reinforcement learning to easily combine with human feedback. The experimental results show that the proposed method helps a human trainer successfully train an agent in two continuous state-action domains: Mountain Car, and Cart-pole (balancing). Through experimental results, we want to claim that the proposed extensions still preserve key properties of the discrete TAMER framework: Easy to implement, accessible to non-technical human trainers, and quickly finding an *acceptable* policy.

## Background

This section describes the original TAMER framework, the tile-coding TAMER as a baseline method for comparison, and the actor-critic method used in our proposed approach. The original TAMER framework is originally designed only for discrete state-action, or continuous state-discrete action

tasks.

## The TAMER framework

The TAMER framework is defined in the context of sequential decision making tasks which are mathematically modeled as a Markov decision process (MDP). A finite MDP is defined by the tuple  $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{T}_0, \mathcal{R}\}$ , where  $\mathcal{S}$  is a set of states,  $\mathcal{A}$  is a set of allowed actions,  $\mathcal{T}$  is a transition function,  $\mathcal{T}_0$  is the distribution of initial states, and  $\mathcal{R}$  is a reward function. In the TAMER framework, the reward function is not used, then it would be defined as an  $\text{MDP} \setminus \mathcal{R}$  (Abbeel and Ng 2004).

The TAMER framework was first proposed by (Knox and Stone 2008; 2009), in which an agent is trained by receiving feedbacks of an observing human. The feedbacks are encoded as negative or positive signals over agent's behaviors. The human feedback is considered as a direct (positive/negative) label on recent state-action pairs. Moreover, it is modeled as a human reinforcement function  $H : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$  at each state-action pair. After an action taken by the agent and a feedback given by a human trainer, the function  $H$  is updated in real time by regression. An action of the next step is chosen greedily as  $a = \arg\max_a H(s, a)$ . The human feedback is usually delayed due to high frequency of time steps. This problem is solved by using credit assignment as in (Knox and Stone 2009), assuming that the human reinforcement function is parametrized by a linear model  $H(s, a) = w^\top \phi(s, a)$ , and the agent is uncertain about the time of the feedback signal it has just received (at time  $t$ ). Therefore, the feedback can be actually given to the actions at any time prior to  $t: t-1, t-2, \dots, t-n, \dots$ . The credits  $c_i$  are defined to be the probability of the signal given at a time step  $i$ . If a probability density function  $f(t)$  is given to define the delay of the human feedback signal, then the credit for each previous time step is computed as

$$c_{t-k} = \int_{t-k-1}^{t-k} f(x) dx, \text{ for } k = 0, 1, \dots \quad (1)$$

If the agent receives a feedback signal  $h \neq 0$  at time  $t$ , the error of each update step weighted by credit assignments is shaped as

$$\delta_t = h - \sum_k c_{t-k} w_t^\top \phi(s_{t-k}, a_{t-k}) \quad (2)$$

The parameter update using the gradient of the least square, which is also weighted by credit assignments, is

$$w_{t+1} = w_t + \alpha_t \delta_t \sum_k c_{t-k} \phi(s_{t-k}, a_{t-k}) \quad (3)$$

The temporal error computation in Eq. (2) normally is  $\delta_t = h - w_t^\top \phi(s_t, a_t)$  without credit assignments. Otherwise, the second term on the right would be the sum of the previous time step reinforcement values weighted by the credits assigned. The same explanation is applied to the parameter update in Eq. (3). Depending on tasks, the history windows of the credits can be pruned to only some recent time steps. Then, the credit computation at each step does not become a burden of the algorithm's computational time. With the help of the TAMER framework, the optimal policy is based only on the trainer.

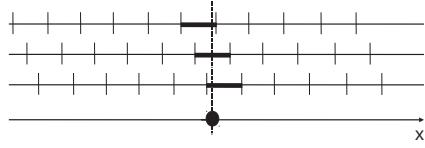


Figure 1: Three tiling organization for the variable space consisting of a single continuous variable  $x$ . The tilings are overlapping. The weights of the tiles are highlighted for an indicated point.

## Tile-coding

Tile-coding (Sutton 1995; Sutton and Barto 1998; Sherstov and Stone 2005) is a function approximation technique which uses sparse coarse-coded memory. It represents the value function in a set of exhaustive partitions of the state-action space, called tilings. All tilings may or may not be partitioned in the same way. Additionally, those tilings are overlapping and slightly offset. Each element of a tiling is called a tile which is weighted. So, each input activates one tile per tiling in which it is contained. The value of an input is the sum of weights of the activated tiles. Figure 1 illustrates one tile-coding example for the variable space consisting of a single continuous variable  $x$ .

Assuming that the state-action space is partitioned into a set of  $N$  tilings;  $\{T_i; i = 1, \dots, N\}$ , each tiling  $T_i$  has  $N_i$  tiles  $\{t_j; j = 1, \dots, N_i\}$ , and each tile has a weight  $w_{ij}$ . For one state-action input  $(s, a)$ , there is only one tile activated per tiling in which the input is contained. As a consequence, the evaluation value of the input is

$$H(s, a) = \sum_{i=1}^N \sum_{j=1}^{N_i} b_{ij} w_{ij} \quad (4)$$

where  $b_{ij} = 1$  if tile  $j$  in tiling  $i$  is active;  $b_{ij} = 0$  otherwise. Therefore,

$$\sum_{j=1}^{N_i} b_{ij} = 1, \text{ for all } i$$

## Actor-critic

The actor-critic algorithms, first proposed by (Witten 1977; Barto, Sutton, and Anderson 1983), implement both major families of reinforcement learning algorithms. They maintain a value function of the value-function based methods, called the critic. Meanwhile, they use a parametrized policy of the policy-based methods to select actions, called the actor. The policy is represented explicitly and independently from the value function. Thus, the actor is used to choose actions, the critic is used to evaluate the performance of the actor. The critic's evaluation provides a gradient estimate of a specific performance measure<sup>1</sup> to improve the actor by updating its parameters. Under a compatible representation of the critic and actor, the algorithms are proved to converge to a local maximum (Sutton et al. 1999; Konda and Tsitsiklis

<sup>1</sup>with respect to the actor's parameters

2003). Actor-critic methods have shown the following two major apparent advantages:

- The action selection step is implemented on an explicit policy instead of a value function which would reduce computation.
- In competitive and non-Markov cases, a stochastic policy may be useful, thus actor-critic method is the solution which can learn an explicitly stochastic policy (Singh, Jaakkola, and Jordan 1994).

We describe a very simple actor-critic method in reinforcement learning as presented in (Sutton and Barto 1998). Suppose the critic is a state-value function  $V(s_t)$ , and the actor is represented by the Gibbs softmax method:

$$\pi_t(s, a) = \Pr\{a_t = a | s_t = s\} = \frac{e^{\phi(s, a)}}{\sum_{b \in \mathcal{A}} e^{\phi(s, b)}} \quad (5)$$

After each action selection step and observing a next state  $s_{t+1}$ , the critic's evaluation is an TD error:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \quad (6)$$

where  $r_{t+1}$  is an MDP environmental reward. The TD error offers a suggestion to the policy update, if it is negative, the action just taken is updated how its tendency must be weakened.

$$\phi(s_t, a_t) = \phi(s_t, a_t) + \beta_t \delta_t \quad (7)$$

where  $\beta_t$  is a learning factor.

In the next section, we apply this scheme to the continuous TAMER framework. The reinforcement function  $H(s, a)$  will be implemented as the TAMER critic. Its temporal error at each time step is used to update the parameters of a policy which functions as the TAMER actor.

## Related Works

Our work is an extension of the TAMER framework (Knox and Stone 2009), called Training an Agent Manually via Evaluative Reinforcement, which is an instance of interactive Machine learning. This family of methods allows human trainers to interactively train an autonomous agent. Recently, there have been numerous techniques in this family successfully leveraging the training tasks of the human (Thomaz and Breazeal 2006; Subramanian, Isbell, and Thomaz 2011; Taylor and Chernova 2010; Judah et al. 2010). In (Subramanian, Isbell, and Thomaz 2011), the authors train an agent's options (macro actions) from demonstrations in an MDP framework. In (Taylor and Chernova 2010), the authors also use learning from demonstration (LfD) to initialize the policy, then apply reinforcement learning to find the optimal policy. For the difference between LfD and learning from human feedback, we want to refer readers to the paper (Knox, Setapen, and Stone 2011) in which the authors also briefly argue some advantages of learning from human feedback over LfD. In (Judah et al. 2010), the learning agent alternates between practicing and critique receiving. In the critique state, a human trainer observes the agent's trajectories, then criticizes each possible state-action pair as good or bad. In the practice stage, the agent uses an RL algorithm

to learn autonomously while taking the human trainer's critique into account.

Different from the TAMER framework (Knox and Stone 2009), our framework learns the human trainer's reinforcement function in continuous state and action domains. In order to do this extension, we use two previous well-known methods in continuous reinforcement learning, which are tile-coding and the actor-critic methods. The actor-critic TAMER implemented the actor-critic style as (Bhatnagar et al. 2009), where the authors proposed four actor-critic reinforcement learning algorithms using linear function approximation. Meanwhile, the tile-coding TAMER implemented a tile-coding function approximator similar to (Santamaria, Sutton, and Ram 1998) in which the authors discussed how to generalize previous reinforcement learning algorithms to continuous state and action spaces using tile-coding function approximators. Tile-coding TAMER is used as a baseline method for comparison. There is a previous work also using actor-critic RL for learning via human feedback in continuous state and action spaces (Pilarski et al. 2011). However, the human feedback in this work is considered as immediate rewards which are different from the long-term effects in TAMER.

## Continuous TAMER Framework

In practical applications, state and action spaces are often continuous or infinite, especially in robotic tasks. As a consequence, the value functions representing the optimal policy must be represented by some forms of function approximation. In reinforcement learning, tile-coding is one simple and easy-to-implement function approximation technique which showed many successes in such continuous state-action tasks (Sutton 1995), (Santamaria, Sutton, and Ram 1998). So, we take tile-coding as the first approximation method to learn the human trainer's reinforcement function  $H(s, a)$ . We call this baseline method as a tile-coding TAMER method. Our major proposed technique is to use the policy gradient framework in which we represent the human trainer's preference policy in a family of randomized policies. The preference policy is then considered as the actor. In addition, we maintain the critic, which is implemented by a TAMER framework, to evaluate the actor's performance. As a result, the second method is called an actor-critic TAMER framework.

### Tile-coding TAMER

A simple method to build the continuous TAMER framework is to use discretization of the state-action space. Thus, we first use tile-coding to build a baseline tile-coding TAMER to approximate the reinforcement function  $H(s, a)$  of both continuous variables  $s$  and  $a$ . This is an application of the original TAMER in which we use a different value function approximation. Our intention is to use this simple extension for comparison with our main principled extension.

Like in Background section, we use  $N$  tilings, and each has  $N_i$  tiles associated with weights  $w_{ij}$ . As a consequence, it's very straightforward to derive the tile-coding TAMER algorithm as in Algorithm 1. The code in line 7 computes the

credit assignments of previous steps with respect to the current obtained feedback  $h_t$ . The code in line 8 computes the regression error in which the evaluation at the current time step is the sum of previous time step values weighted by their corresponding credits. In line 11, the weights are updated along the sum of previous time step gradients weighted by their corresponding credits, where  $\alpha_t$  is a learning rate. The feature  $b_{ij}^{t-k}$  represents whether tile  $(i, j)$  is active with respect to the state  $s_{t-k}$  at time step  $t-k$  (Because the matrix  $b$  is actually the feature function in the tile-coding method).

---

**Algorithm 1** Tile-coding TAMER

---

```

1: Initialize weights  $w_{ij}^0$ , and an initial state  $s_0$ .
2: while (1) do
3:   Choose an action  $a_t = \text{argmax}_a H(s_t, a)$  (computed
      as in Eq. (4)),
4:   Take the action  $a_t$ , and observe a next state  $s_{t+1}$ ,
5:   Receive a feedback  $h_t$ ,
6:   if  $h_t \neq 0$  then
7:     Compute credits  $c_{t-k}$  as in Eq. (1)
8:      $\delta_t = h_t - \sum_k c_{t-k} \sum_{i=1}^N \sum_{j=1}^{N_i} b_{ij}^{t-k} w_{ij}^t$ ,
9:     for  $i = 1$  to  $N$  do
10:    for  $j = 1$  to  $N_i$  do
11:       $w_{ij}^{t+1} = w_{ij}^t + \alpha_t \delta_t \sum_k c_{t-k} b_{ij}^{t-k}$ 
12:    end for
13:  end for
14: end if
15: end while

```

---

## Actor-critic TAMER

Though the tile-coding method is simple and computationally efficient, it has some limitations. Firstly, it needs a human designer of the tile partition. In some hard problems, the tiles must be partitioned very finely and carefully. Secondly, the degree of generalization is fixed according to the initial tile partition. And thirdly, if the state mapping into tiling features is non injective, then it would lose the convergence property of regression algorithms. In this section, we introduce another extension of TAMER, which would also work in continuous state-action spaces, the actor-critic method. More specifically, we still use the human trainer's reinforcement function as the critic  $\bar{H} : \mathcal{S} \rightarrow \mathbb{R}$ . We use a slightly different reinforcement function which depends only on state. Similar to the original TAMER framework, it is also updated in real time by regression. Assuming that the TAMER critic is parametrized by a parameter space  $\Theta = \{\theta_1, \dots, \theta_m\}$ . On the other hand, the TAMER actor is a stochastic policy  $\mu : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ , parametrized by a parameter space  $w$ . The function  $\mu_w(s, a)$  defines the probability of choosing an action  $a$  at a state  $s$  depending on a parameter  $w$ . In this modeling, the TAMER critic would use any function approximation method as regression. Figure 2 shows interaction between a human, the environment, and an actor-critic TAMER agent through the actor-critic TAMER framework.

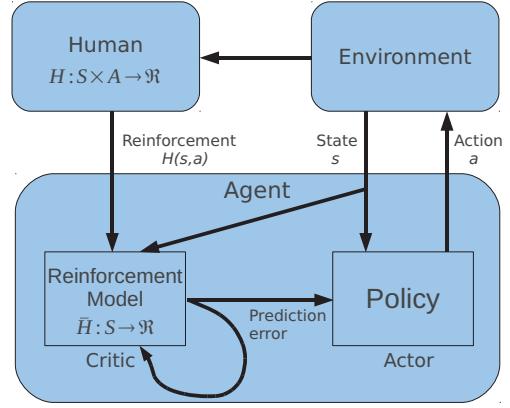


Figure 2: Actor-critic TAMER framework.

The optimal policy with respect to the trainer's perspective is approximated by an actor.

Assuming that we use a linear approximator for the trainer's reinforcement function  $\bar{H}(s)$ , with the features represented by  $\Phi = \{\phi_1, \dots, \phi_m\}$ . Thus, the function  $\bar{H}(s)$  is written as

$$\bar{H}(s) = \Theta^\top \Phi = \sum_m \theta_i \phi_i \quad (8)$$

Then, the critic's parameter update can be computed similarly to Eq. (3), which is also weighted by credit assignments, as

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \sum_k c_{t-k} \Phi(s_{t-k}) \quad (9)$$

In actor-critic style, the TAMER critic evaluates the TAMER actor's performance and advises the update tendency for it. It updates the actor's parameters using the temporal error of the critic's prediction. Similar to the actor-critic reinforcement learning algorithms in (Bhatnagar et al. 2009), if the feedback function  $H(s, a)$  at pair  $(s, a)$  is approximated linearly with compatible features  $\psi_{sa} = \nabla \log \mu(s, a)$ , then the actor-critic TAMER method would be derived as in Algorithm 2. The code in line 8 computes the critic's regression error. The code in lines 9 and 10 updates the critic and actor's parameters respectively. All of those computations also take the credit assignments into account.

The actor-critic TAMER uses two-timescale stochastic approximation, then the learning parameters  $\alpha_t$  and  $\beta_t$  would be chosen to satisfy  $\beta = o(\alpha_t)$ . Because, the slower convergence of  $\alpha_t$  makes the approximation of the value function  $\bar{H}_t(s)$  having uniformly higher increments than the approximation of the policy  $\mu(s, a)$ .

## Experiment Domains

In this section, we test the proposed continuous TAMER algorithm on two popular domains in reinforcement learning: Mountain Car and Cart-pole which we assume to have continuous state and action spaces. The performance metric is the cumulative environmental MDP reward  $R$ . The continuous action and state Mountain Car is described as in (Melo

---

**Algorithm 2** Actor-critic TAMER

---

```

1: Initialize  $\theta_0$ ,  $w_0$ , and an initial state  $s_0$ .
2: while (1) do
3:   Choose an action  $a_t \sim \mu(s_t, w_t)$  (computed as in
   Eq.(8)),
4:   Take action  $a_t$ , and observe  $s_{t+1}$ ,
5:   Receive feedback  $h_t$ ,
6:   if  $h_t \neq 0$  then
7:     Compute credits  $c_{t-k}$  as in Eq. (1)
8:      $\delta_t = h_t - \sum_k c_{t-k} \bar{H}_t(s_{t-k})$ ,
9:      $\Theta_{t+1} = \Theta_t + \alpha_t \delta_t \sum_k c_{t-k} \nabla_\Theta \bar{H}_t(s_{t-k})$ ,
10:     $w_{t+1} = w_t + \beta_t \delta_t \sum_k c_{t-k} \nabla_w \log \mu_t(s_{t-k}, a_{t-k})$ .
11:   end if
12: end while

```

---

and Lopes 2008). We choose those domains because the discrete TAMER has been demonstrated to perform well, and we want to show why an algorithm for the continuous action space is needed. By these experiments, we observe that the proposed extension continues preserving the properties of the discrete TAMER in which it helps reduce the sample complexity of learning a reasonably good policy. Additionally, an agent can learn without defining any hand-coded environment reward signal. This allows a human trainer to teach agents the policies according to their preference.

Both experimental environments use the implementations with graphical interfaces within the RL-Library<sup>2</sup>. We made some small modifications to make them work with continuous actions and receive human trainers' feedback signals by pressing on the keyboard. The human trainers observe the graphically simulated agents on the computer screen. There are two accepted keys representing positive and negative feedback signals. These two domains are high time step frequency, which is set at approximately 200 milliseconds. For the delay distribution function, we use a Uniform(200,900) distribution as credit assignment function for both domains.

### Mountain Car

The well-known Mountain Car problem is the task of driving an underpowered car situated at the bottom of a valley to the top of the steep hill on the right. With a limited acceleration, it can not climb up the hill shortly, rather it must go back and forth to gain enough momentum to go up. An episode terminates when the car reaches the top of the hill on the right. The 2-dimension continuous state space  $s = (p, v)$  consists of the current position  $p \in [-1.2; 0.5]$  and velocity  $v \in [-0.07; 0.07]$ . The controlled action is a single continuous acceleration  $a \in [-1.0; 1.0]$ . The dynamic equations of the car are described as

$$\begin{aligned} v_{t+1} &= v_t + 0.001 a_t - 0.0025 \cos(3p_t) \\ p_{t+1} &= p_t + v_{t+1} \end{aligned} \quad (10)$$

The values of  $p$  and  $v$  is maintained bounded within their limits.

---

<sup>2</sup><http://library.rl-community.org/>

### Cart-pole (Balancing)

In this section, we apply the proposed algorithms for another well-known benchmark for reinforcement learning, which is Cart-pole (balancing) domain. The goal is to keep the pole balancing on top of the cart by accelerating the cart. The cart can not go too far to the left or right boundary. We use the same setting in RL-Library which implemented the discrete action Cart-pole example in (Sutton and Barto 1998). We modified the Cart-pole environment in RL-Library to continuous actions. The pole length is  $l = 0.5m$ , pole mass  $m = 0.1kg$ , gravity  $g = 9.8m/s^2$ , and cart mass  $m_c = 1.0kg$ . The continuous state is chosen by  $s = [p, \dot{p}, \theta, \dot{\theta}]$  which is the cart's current position  $p \in [-2.4; 2.4]$ , the cart's velocity, the pole's angle, and the pole's angular velocity. The control action is the force applied on the cart  $a = F \in [-10N; 10N]$ . An episode terminates when the pole angle exceeds a threshold of (-12.0;12.0) degrees, or if the cart moves out of the boundary of the track.

### Algorithm Settings

Both algorithms are compared to discrete TAMER (Knox and Stone 2009) (The action space of the domains is implemented to be discrete), the standard Sarsa( $\lambda$ ), and actor-critic RL (in (Bhatnagar et al. 2009)) algorithms (with environmental rewards and without human trainer feedback). Sarsa( $\lambda$ ) and tile-coding TAMER use tile-coding as function approximators of  $Q(s, a)$  and  $H(s, a)$ , respectively. Both Sarsa( $\lambda$ ) and tile-coding TAMER use 25 equally distributed interval values for the acceleration (within its limits) to perform  $\epsilon$ -greedy and only greedy action selection, respectively. Actor-critic TAMER and actor-critic RL also use the same representation for actor and critic. In the following, we describe each algorithm's setting in detail.

*Tile-coding TAMER:* We use 48 tilings as a feature set for the function  $H(s, a)$ . Each dimension is discretized by 12 intervals. There are 16 tilings based on  $(p, v, a)$ , uniformly offset; 16 tilings based on  $(p, v)$ , uniformly offset; 8 tilings based on  $p$ , uniformly offset; and 8 tilings based on  $v$ , uniformly offset. This feature set is also used to approximate the value function  $Q(s, a)$  of Sarsa( $\lambda$ ). The implementation of tile-coding Sarsa( $\lambda$ ) is similar to (Santamaria, Sutton, and Ram 1998).

*Actor-critic TAMER:* The critic, which is the human trainer's feedback function  $\bar{H}(s)$ , is approximated by a linear approximator over Gaussian RBF features. We use 16 and 256 uniformly centred RBF features within the limits of the state space in Mountain Car and Cart-pole respectively, 4 intervals for each dimension. We define a mean policy as  $a = w^\top \Psi(s)$  with parameter vector  $w$  and a Gaussian RBF feature function  $\Psi(s)$ . In both domains, the mean policy also uses 4 RBF basis functions for a one-dimensional action space. We generate a stochastic policy for the actor by adding a small exploration term  $\epsilon \sim \mathcal{N}(\epsilon|0, \sigma^2)$ . Then, the actor can be written as  $\mu(s, a) = \mathcal{N}(a|w^\top \Psi(s), \sigma^2)$ .

### Experimental Results

The experiments were implemented with assumptions that the human trainers are instructed how to give feedback with-

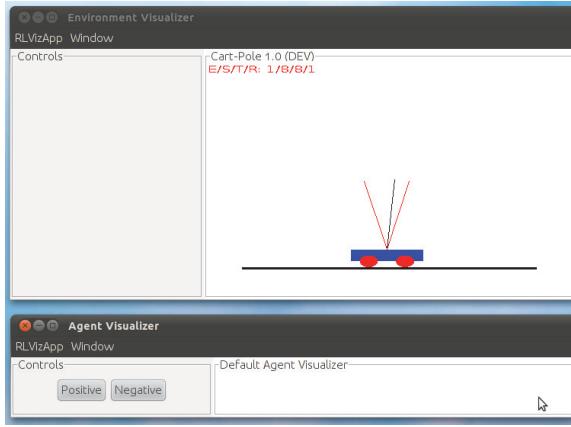


Figure 3: Screenshot of the experiment interface from Cart-pole.

out knowing the agent's feature function or learning algorithms. The screenshot of the experiment interface is shown as in Fig. 3. Fifteen people participated in training the Mountain Car and Cart-pole agents. Each trainer did three teaching trials of up to 60 episodes (almost all of them stopped after only 20-30 episodes). Average training time of each person is about 10 minutes. After the training stage finished, the agent continued running (testing stage) until 100 episodes were reached. The best result among three trials is used as a report for that trainer.

### Mountain Car

The comparisons for the Mountain Car domain are shown in the top panel of Fig. 4. The environmental reward is -1.0 for each step. The results for the TAMER agents are averaged among human trainers' results. The results of Sarsa( $\lambda$ ) and actor-critic RL are averaged over 100 trials. On average, both continuous TAMER agents give comparable performances, but asymptotically outperforms the Sarsa agent and worse than actor-critic RL agent. Moreover, TAMER agents have found a suboptimal policy as quickly as actor-critic RL, but much faster than the Sarsa( $\lambda$ ). Thus, the continuous TAMER agents can dramatically reduce sample complexity over autonomous learning agents such as Sarsa( $\lambda$ ). The policy behavior of continuous TAMER agents is similar to discrete TAMER's. Because, both TAMER frameworks (discrete and continuous) found the suboptimal policy of max (+1.0) and min acceleration (-1.0) actions. The bottom panel of Fig. 4 shows the average performance received by agents trained by the best 3 trainers and worst 3 trainers for each algorithms.

### Cart-pole (Balancing)

The comparisons of continuous TAMER agents against discrete TAMER for Cart-pole (balancing) domain are shown in the top panel of Fig. 5. The environmental reward is 1.0 for each step in which the pole is kept upright  $\theta \in (-12.0, 12.0)$ . The results for the TAMER agents are averaged among human trainers' results. In Car-pole domain,

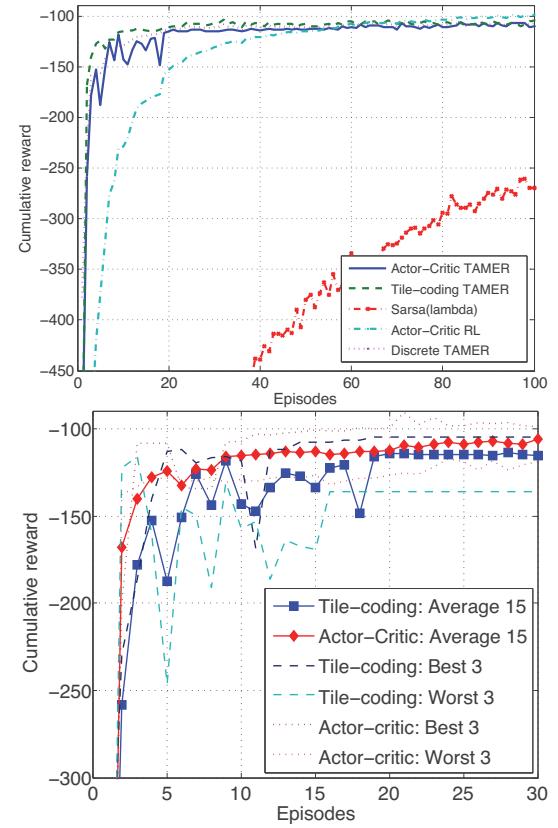


Figure 4: Mountain Car: (top) The cumulative environmental rewards (steps to goal) of three algorithms; (bottom) The average cumulative environmental rewards, the best and worst 3 trainer's policy.

the applied forces on the pole could range from  $-10(N)$  to  $+10(N)$ . If there are only 3 actions: left ( $-10N$ ), neutral ( $0N$ ), and right ( $10N$ ) like in discrete TAMER's domain, the movement of the pole is less smooth than using continuous actions. If using continuous actions, the optimal policy helps the agent only choose very small actions to make the pole balance smoothly. So, if we compare the number of steps while balanced, two continuous TAMER settings have shown a significant advantage over the discrete TAMER. The bottom panel of Fig. 5 shows the average performance received by agents trained by the best 3 trainers and worst 3 trainers for each algorithms.

The comparisons against other RL algorithms are described in Fig. 6. The result of Sarsa( $\lambda$ ) and actor-critic RL are averaged over 100 trials after 50 episodes. In this more complicated domain, it's natural that the TAMER agents' asymptotic results are worse than autonomous learning agents such as Sarsa( $\lambda$ ) and actor-critic RL. However, the results of the best 3 trainers (use the actor-critic TAMER framework) are quite promising, they can train the agents to get over 300 times balancing in within 50 episodes. In this experiment, we observe that the actor-critic TAMER agent outperforms over the tile-coding one. This is due to the ef-

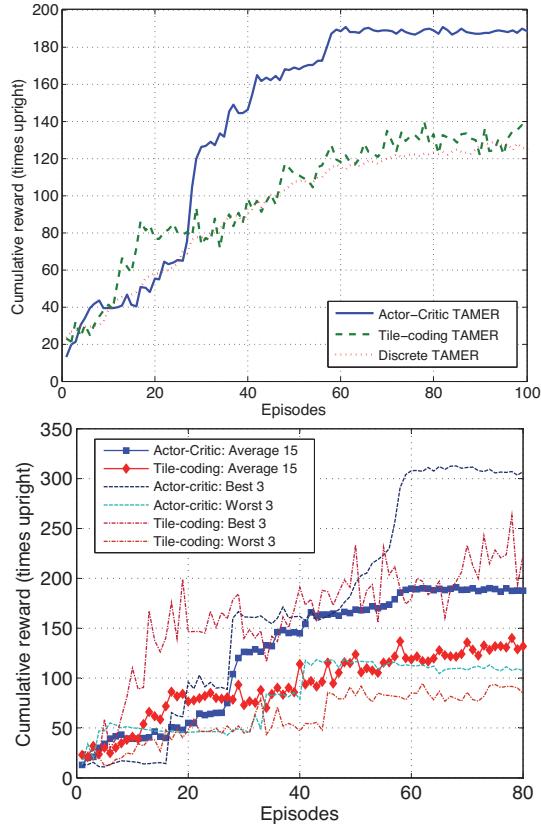


Figure 5: Cart-pole (balancing): (top) The cumulative environmental reward (times upright); (bottom) The average cumulative environmental rewards, the best and worst 3 trainer’s policy.

fect of preallocation in tile-coding function approximators which clearly does not perform good enough in the region of the stable states (keep the pole balanced). In such a situation, one solution could be using a higher resolution in the region of the stable states which needs very small adjustments in acceleration to keep the pole balanced. The observation demonstrates the benefits and flexibility of having a continuous action space. Thus, a continuous TAMER framework is needed.

## Summary and Future Work

### Summary

This paper proposes an extension for the Training an Agent Manually via Evaluative Reinforcement Framework (TAMER) which allows human trainers to train agents in continuous state and action domains. The proposed actor-critic TAMER framework implements the actor-critic style to approximate the human trainer’s preference policy. The trainer’s preference policy, which is the actor, is parametrized by a parameter space. The critic is used to evaluate the actor’s performance and can be implemented by using any state value function approximators. The actor’s parameters are updated by following the temporal error of the

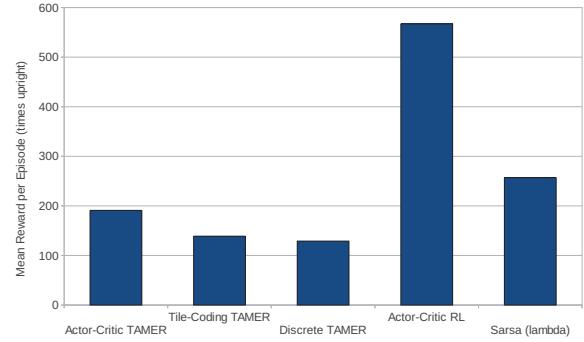


Figure 6: Cart-pole (balancing): End-run performance of these algorithms for Cart-pole is averaged during the last 5 episodes (The results for Actor-Critic and Sarsa ( $\lambda$ ) are after 50 episodes).

critic’s prediction.

The proposed method was tested in two well-known domains in reinforcement learning: Mountain Car and Cart-pole (balancing) which have continuous state and action spaces. The experimental results obtained shows the feasibility of the two methods in the task of approximating the human trainer’s preference policy. Similar to the original TAMER agent, the new algorithm is easy to implement, and accessible to a non-technical human trainer. In term of technical aspects, it still can reduce sample complexity over autonomous learning algorithms. Moreover, the proposed method suggested that continuous extension of TAMER framework is needed, because the continuous extensions would make TAMER more efficient, and applicable for continuous domains.

### Future Work

The extension to continuous action problems could open a new applicability for the TAMER framework, especially in robotics whose problems are often both continuous state and action spaces. Investigation of the continuous TAMER framework in robotics problems is also our future work. On the other hand, the learning can be speeded up by combining TAMER with reinforcement learning algorithms. Thus, this extension can bring more ways of combination with other existing continuous reinforcement learning algorithms. Similar to (Knox and Stone 2010a; 2011; 2012), we are investigating the combination capability of the proposed continuous TAMER approaches with reinforcement learning. We plan to use one technique among 8 listed in (Knox and Stone 2010a). The other techniques are not either applicable directly to continuous action space problem or efficient as indicated in (Knox and Stone 2010a; 2011). The technique we want to use is called control sharing, as numbered 7 in (Knox and Stone 2010a). This technique lets the combining agent effectively choose the RL agent’s action or TAMER agent’s action. This combination technique can be easily implemented either sequentially or simultaneously in actor-critic TAMER framework.

## References

- Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004)*.
- Barto, A. G.; Sutton, R. S.; and Anderson, C. W. 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, & Cybernetics* 13(5):834–846.
- Bhatnagar, S.; Sutton, R. S.; Ghavamzadeh, M.; and Lee, M. 2009. Natural actor-critic algorithms. *Automatica* 45(11):2471–2482.
- Judah, K.; Roy, S.; Fern, A.; and Dietterich, T. G. 2010. Reinforcement learning via practice and critique advice. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*.
- Knox, W. B., and Stone, P. 2008. TAMER: Training of an agent manually via evaluative reinforcement. In *IEEE 7th International Conference on Development and Learning (ICDL-08)*, 292–297.
- Knox, W. B., and Stone, P. 2009. Interactively shaping agents via human reinforcement: the TAMER framework. In *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009)*, 9–16.
- Knox, W. B., and Stone, P. 2010a. Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 5–12.
- Knox, W. B., and Stone, P. 2010b. Training a tetris agent via interactive shaping: a demonstration of the TAMER framework. In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 1767–1768.
- Knox, W. B., and Stone, P. 2011. Augmenting reinforcement learning with human feedback. In *2011 ICML Workshop on New Developments in Imitation Learning*.
- Knox, W. B., and Stone, P. 2012. Reinforcement learning with human and MDP reward. In *11st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*.
- Knox, W. B.; Setapen, A.; and Stone, P. 2011. Reinforcement learning with human feedback in mountain car. In *AAAI 2011 Spring Symposium*, 36–41.
- Konda, V. R., and Tsitsiklis, J. N. 2003. On actor-critic algorithms. *SIAM J. Control and Optimization* 42(4):1143–1166.
- Melo, F. S., and Lopes, M. 2008. Fitted natural actor-critic: A new algorithm for continuous state-action MDPs. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD (2)*, 66–81.
- Pilarski, P. M.; Dawson, M. R.; Degris, T.; Fahimi, F.; Carey, J. P.; and Sutton, R. S. 2011. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *IEEE International Conference on Rehabilitation Robotics*, 1–7.
- Santamaria, J. C.; Sutton, R. S.; and Ram, A. 1998. Experiments with reinforcement learning in problems with continuous state and action spaces. *Adaptive Behavior* 6(2):163–218.
- Sherstov, A. A., and Stone, P. 2005. Function approximation via tile coding: Automating parameter choice. In *Abstraction, Reformulation and Approximation, 6th International Symposium (SARA)*, 194–205.
- Singh, S. P.; Jaakkola, T.; and Jordan, M. I. 1994. Learning without state-estimation in partially observable markovian decision processes. In *Machine Learning, Proceedings of the Eleventh International Conference (ICML)*, 284–292.
- Subramanian, K.; Isbell, C.; and Thomaz, A. 2011. Learning options through human interaction. In *Workshop on Agents Learning Interactively from Human Teachers at IJCAI*.
- Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.
- Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA]*, 1057–1063.
- Sutton, R. S. 1995. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8 (NIPS)*, 1038–1044.
- Taylor, M. E., and Chernova, S. 2010. Integrating human demonstration and reinforcement learning: Initial results in human-agent transfer. In *Proceedings of the Agents Learning Interactively from Human Teachers workshop (at AAMAS-10)*.
- Thomaz, A. L., and Breazeal, C. 2006. Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference*.
- Witten, I. H. 1977. An adaptive optimal controller for discrete-time markov environments. *Information and Control* 34(4):286–295.