Reinforcement Learning Combined with Human Feedback in Continuous State and Action Spaces

Ngo Anh Vien
Institute of Artificial Intelligence
Ravensburg-Weingarten University of Applied Sciences
Weingarten 88250, Germany
Email: ngo@hs-weingarten.de

Wolfgang Ertel
Institute of Artificial Intelligence
Ravensburg-Weingarten University of Applied Sciences
Weingarten 88250, Germany
Email: 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 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. We propose an extension of TAMER to allow both continuous states and actions, called ACTAMER. The new framework extends the original TAMER to allow using any general function approximation of a human trainer's reinforcement signal. Moreover, we investigate a combination capability of the ACTAMER and reinforcement learning (RL). The combination of human feedback and RL is studied in both settings: sequential and simultaneous. Our experimental results show the proposed method successfully allowing a human to train an agent in two continuous state-action domains: Mountain Car, Cart-pole (balancing).**

I. 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 [1]. The TAMER framework has performed well in tasks in which the human already has significant knowledge. It does not require from the human 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 [1], [2]. 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 curve climb up. There are many possible sequential and simultaneous combinations between TAMER and RL [2], [3]. 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 interestingly, the simultaneous scheme allows a human 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 continuous state and action domains, called ACTAMER. The new framework 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 form of 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 extension still allows reinforcement learning to easily combine with human feedback. Moreover, we investigate a combination capability of the ACTAMER and reinforcement learning (RL). The combination of human feedback and RL is studied in both settings: sequential and simultaneous. Our experimental results show the proposed method successfully allowing a human to 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 extension still preserves key properties of the discrete TAMER framework: Easy to implement, accessible to non-technical human trainers, and quickly finding an *acceptable* policy. Moreover, the combination of ACTAMER+RL efficiently enables the human feedback to transfer human knowledge to a reinforcement learning which helps to accelerate the autonomous learning process or improve the performance.

II. Background

This section describes the original TAMER framework, and the actor-critic methods used in our proposed approaches. The original TAMER framework is originally designed only for discrete state-action, or continuous state-discrete action tasks.

A. The TAMER framework

The TAMER framework is defined in the context of sequential decision making tasks which is 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 MDP$\backslash\mathcal{R}$ [4].

The TAMER framework is first proposed by [5], [1], 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 an action value (similar to Q-value of reinforcement learning). Moreover, it is modeled as a human reinforcement function $\mathrm{H} : \mathcal{S} \times \mathcal{A} \to \Re$ 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 $a = \mathrm{argmax}_a \mathrm{H}(s, a)$. The human feedback is usually delayed due to high frequency of time steps. This problem is solved by using credit assignment as in [1], assuming that the human's reinforcement function is parametrized by a linear model $\mathrm{H}(s, a) = w^\top \phi(s, a)$, 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: $t-1, t-2, \ldots, t-n, \ldots$. 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's feedback signal, then the credit for each previous time step is computed as

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

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

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

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

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

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 sub-optimal policy is based only on the trainer.

B. Actor-critic

The actor-critic algorithms, first proposed by [6], [7], 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[1] 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 [8], [9]. 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, then actor-critic methods is the solution which can learn an explicitly stochastic policy [10].

We describe a very simple actor-critic method in reinforcement learning as presented in [11]. Suppose the critic is a state-value function $V(s_t)$, and the actor is represented by the Gibbs softmax method:

$$\pi_t(s, a) = \Pr\{a_t = a | s_t = s\} = \frac{e^{\phi(s,a)}}{\sum_{b \in \mathcal{A}} e^{\phi(s,b)}} \tag{4}$$

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

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{5}$$

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.

$$\phi(s_t, a_t) = \phi(s_t, a_t) + \beta_t \delta_t \tag{6}$$

where β_t is a learning factor.

In the next section, we apply this scheme to the continuous TAMER framework. The reinforcement function $\mathrm{H}(s, a)$ 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.

III. 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 requires to be represented in some forms of function approximation. Our 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

[1] with respect to the actor's parameters

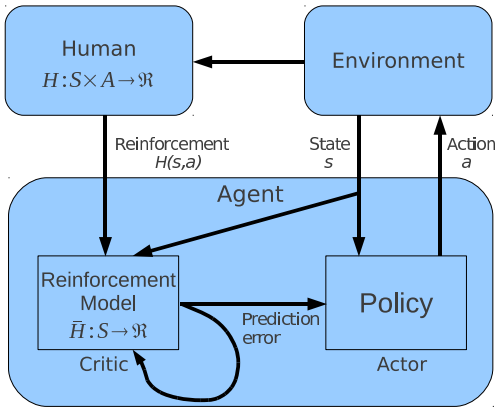

Fig. 1. ACTAMER framework.

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, our extension method is called an ACTAMER framework. This extension of TAMER would work in continuous state-action spaces. More specifically, we still use the human trainer's reinforcement function as the critic $\bar{\mathrm{H}} : \mathcal{S} \rightarrow \Re$. We use a slightly different reinforcement function which depends only on state. Similarly, it is also updated in real time by regression. Assuming that the TAMER critic is parametrized by a parameter space $\Theta = \{\theta_1, \ldots, \theta_m\}$. On the other hand, the TAMER actor is a randomized 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 1 shows interaction between a human, the environment, and an ACTAMER agent through the ACTAMER 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 $\bar{\mathrm{H}}(s)$, with the features represented by $\Phi = \{\phi_1, \ldots, \phi_m\}$. Thus, the function $\bar{\mathrm{H}}(s)$ is written as

$$\bar{\mathrm{H}}(s) = \Theta^\top \Phi = \sum_n \theta_i \phi_i \tag{7}$$

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

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \sum_k c_{t-k} \Phi(s_{t-k}) \tag{8}$$

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 [12], if the feedback function $\mathrm{H}(s, a)$ at pair (s, a) is approximated linearly with compatible features $\psi_{sa} = \nabla \log \mu(s, a)$, then the ACTAMER method would be derived as in Algorithm 1. 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 ACTAMER uses two-timescale stochastic approximation, then the learning parameters α_t and β_t would be chosen to satisfy $\beta = o(\alpha_t)$. Because, the slower convergence of α_t makes the approximation of the value function $\bar{\mathrm{H}}_t(s)$ having uniformly higher increments than the approximation of the policy $\mu(s, a)$.

Algorithm 1 ACTAMER

1: Initialize θ_0, w_0, and an initial state s_0.
2: **while** (1) **do**
3: Choose an action $a_t \sim \mu(s_t, w_t)$,
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: $\delta_t = h_t - \sum_k c_{t-k} \bar{\mathrm{H}}_t(s_{t-k})$,
9: $\Theta_{t+1} = \Theta_t + \alpha_t \delta_t \sum_k c_{t-k} \nabla_\Theta \bar{\mathrm{H}}_t(s_{t-k})$,
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**

IV. ACTAMER+RL

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. In this section, we investigate one technique among 8 listed in [3] in order to combine human feedback (by ACTAMER framework) with reinforcement learning, it is called control sharing. Because the other techniques are not either applicable directly to continuous action space problem or efficient as indicated in [3]. Similar to [3], [2], we combine ACTAMER framework and reinforcement learning in both ways: sequentially and simultaneously. We assume to also use the actor-critic RL method as the base RL algorithm, because this method is more directly applicable for continuous state and action problems.

A. Sequential ACTAMER+RL

The sequential combination of ACTAMER+RL first lets the human policy $\mu(s, w)$ to be trained using the ACTAMER framework. This human training stage must be finished prior to the next stage of an actor-critic RL algorithm. Then an actor-critic RL agent is autonomously trained, it chooses actions according to the control sharing rule: $P\big(a \sim \mu(s, w)\big) = \min(\beta, 1)$, otherwise uses the action selection mechanism of the base actor-critic RL, where β is a predefined combination parameter. This can be understood that the base actor-critic RL chooses exploration actions toward human-favored policy [3].

B. Simultaneous ACTAMER+RL

Similar to [2], we propose a simultaneous approach for ACTAMER+RL in continuous state and action domains. This modification could let a human trainer to intercept into an RL process to change its course of actions. The agent should learn from both kind of rewards: human and MDP rewards. The actions are still chosen using the control sharing rule. In the sequential combination technique, the human policy $\mu(s, w)$ does not affect the learning's behavior consistency, because the actions taken from $\mu(s, w)$ are considered as exploration actions. In order to maintain the learning's behavior consistency with the simultaneous approach, we inherit the idea of balancing between following human trainer's policy and the policy learned by MDP reward. This idea is implemented by using the eligibility module to determine the immediate influence of the human's policy.

The general idea is to keep an eligibility trace for each state-action feature, which is normalized to $[0, 1]$. This represents the "*recency of training while that feature was active*" [2]. Then, the combination parameter β is defined by the measure of the recency of training multiplied by a constant scaling parameter c_s. The measure of the recency of training in similar feature vectors is easily computed using the eligibility traces and the current time step's feature vector. Assuming that $\mathbf{e}$ is a trace vector, and $\mathbf{f}_t$ is the current time step's normalized feature vector. The eligibility module is use to make β a function of $\mathbf{e}, \mathbf{f}_t$, and c_s in range of $[0, c_s]$. More specifically, the influence of the human's policy is computed as $\beta = c_s(\mathbf{e} \cdot \mathbf{f}_t)/(\|\mathbf{f}_t\|_1)$. The eligibility trace uses accumulating traces as in TD(λ).

V. Experiment Domains

In this section, we test the ACTAMER framework 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 [13]. Both experimental environments use the implementations with graphical interfaces within the RL-Library[2]. The human trainers observe the graphically simulated agents on the computer screen as in Fig. 2. There are two accepted keys representing positive and negative feedback signals. Two additional buttons ("*Stop Training*" and "*Begin Training*") allow a trainer to begin/stop to step into the RL process in progress. 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.

A. 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

$$\begin{aligned} v_{t+1} &= v_t + 0.001a_t - 0.0025\cos(3p_t) \\ p_{t+1} &= p_t + v_{t+1} \end{aligned} \tag{9}$$

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

B. 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 [11]. 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.

C. Algorithm Settings

The ACTAMER is compared to discrete TAMER (allowing only discrete actions as in the original paper [1]), tile-coding TAMER (using 25 equally distributed interval values for the actions), the standard Sarsa(λ), and actor-critic RL (in [12]) algorithms (with only MDP rewards). ACTAMER and actor-critic RL use the same representation for the actor and the critic.

ACTAMER: The critic, which is the human trainer's feedback function $\bar{\mathrm{H}}(s)$, 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. A mean policy is defined 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 $\mu(s, a) = \mathcal{N}(a|w^\top \Psi(s), \sigma^2)$.

Sequential ACTAMER+RL: We tune the combination parameter of the control sharing rule $\beta = 1.0$.

Simultaneous ACTAMER+RL: The combination parameter of the control sharing rule β is tuned online using eligibility

[2]http://library.rl-community.org/

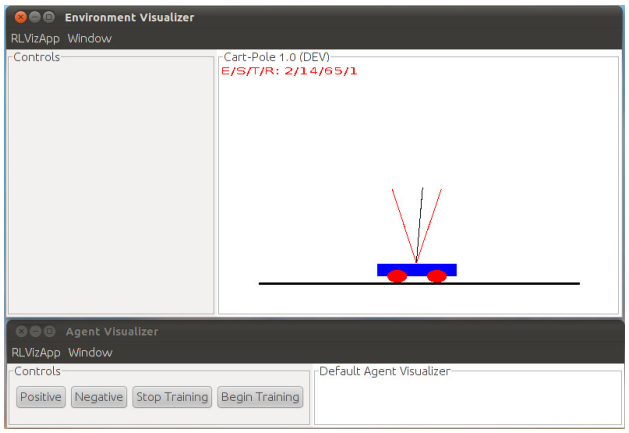

Fig. 2. Screenshot of the experiment interface from Cart-pole.

module as described in Subsection IV-B. The parameter c_s is set to 1.0.

VI. Experimental Results

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. These policies are later used to affect the base actor-critic RL algorithm in sequential combination as described in Subsection IV-A.

A. Mountain Car

The comparisons for the Mountain Car domain are shown in the top panel of Fig. 3 when using ACTAMER-only. The results for the TAMER agents are averaged among human trainers' results. The results of Sarsa(λ) and actor-critic RL are averaged over 100 trials. On average, both the ACTAMER agent gives a comparable performance, but asymptotically outperforms the Sarsa agent and worse than actor-critic RL agent. Moreover, ACTAMER has found a suboptimal policy as quickly as actor-critic RL, but much faster than the Sarsa(λ). Thus, the ACTAMER can dramatically reduce sample complexity over autonomous learning agents. On the other hand, the policy behavior of ACTAMER is similar to discrete TAMERs. The bottom panel of Fig. 3 shows the average performance received by agents trained by the best 3 trainers and worst 3 trainers for each algorithms.

The results of the combination techniques of ACTAMER+RL are described in Fig. 4. For the result of simultaneous ACTAMER+RL, the human training is after 20 episodes of actor-critic RL algorithm-only, and occurs for 30 episodes, then switches back to actor-critic RL algorithm-only. The primary results of these combination techniques have improved significantly the performance.

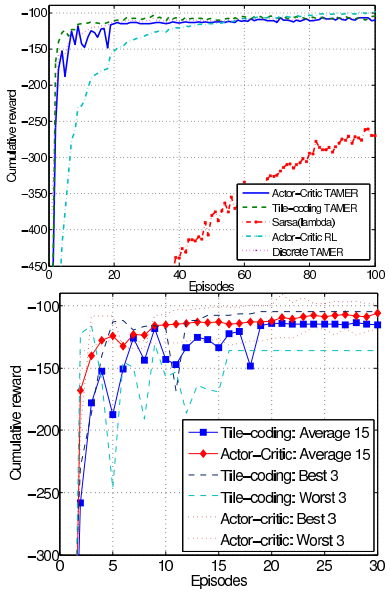

Fig. 3. Mountain Car: (top) The cumulative environmental rewards (steps to goal) of three algorithms; (bottom) The average cumulative environmental rewards, the best and worst 3 trainer's policy.

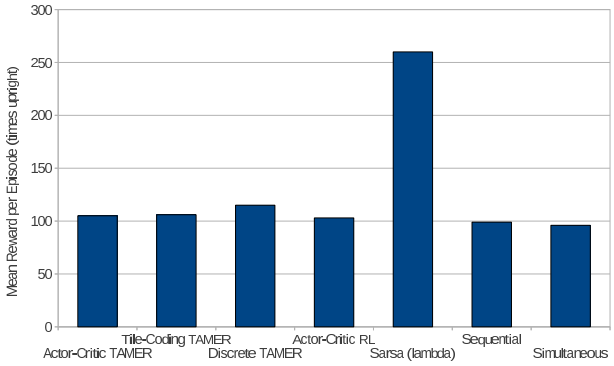

Fig. 4. Mountain Car: End-run performance of comparing algorithms is averaged during the last 5 episodes (after the first 100 episodes of learning) (in term of the number of steps to goal).

B. Cart-pole (Balancing)

The comparisons of ACTAMER against discrete TAMERs for Cart-pole (balancing) domain are shown in the top panel of Fig. 5. The results for the TAMER agents are averaged among human trainers' results. In term of the number of steps while balanced, ACTAMER has shown a significant advantage over the discrete TAMERs. The right 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 of ACTAMER and the combinations techniques against other RL algorithms are described in Fig. 6. The result of the combinations techniques (sequential and simultaneous), Sarsa(λ) and actor-critic RL are averaged over 100 trials after 150 episodes (the cumulative rewards are capped at 10000). In this more complicated domain, it's natural that the ACTAMER agent's asymptotic results are

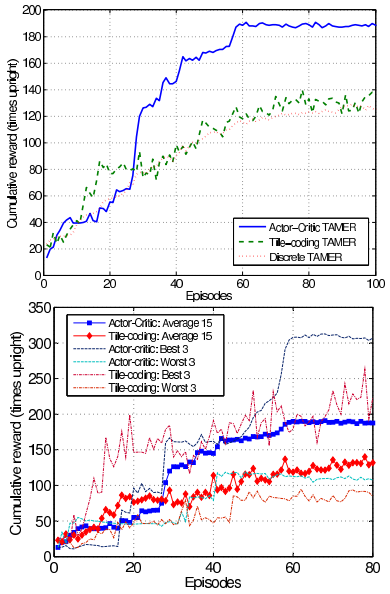

Fig. 5. Cart-pole (balancing): (top) The cumulative environmental reward (times upright); (bottom) The average cumulative environmental rewards, the best and worst 3 trainer's policy.

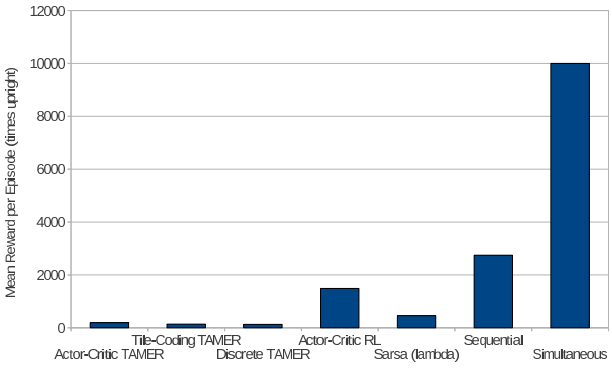

Fig. 6. Cart-pole (balancing): End-run performance of these algorithms for Cart-pole is averaged during the last 5 episodes (after 150 episodes of learning, capped at 10000 steps which is considered as successful balancing).

worse than autonomous learning agents such as Sarsa(λ) and actor-critic RL. However, the results of the best 3 trainers (use the ACTAMER framework) are quite promising, they can train the agents to get over 300 times balancing in within 50 episodes. Moreover, the primary results of these combination techniques have outperformed significantly over other methods' performance.

VII. Summary and Future Work

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 method, named Actor-Critic TAMER (ACTAMER), 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 critic's prediction. Moreover, we have investigated a combination possibility of ACTAMER with an actor-critic method, which could allow to combine sequentially or simultaneously human feedback with MDP reward in the continuous domains.

The proposed method is 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 show the feasibility of the two methods in the task of approximating the human trainer's preference policy. Similar to the original TAMER agent, the proposed framework is easy to be implemented, and accessible to non-technical human trainers. In term of technical aspects, the new framework can reduce sample complexity over autonomous learning algorithms. Moreover, our preliminary results of the combination techniques (only use control sharing rule) have shown their promising applicability in continuous domains, especially in robotics. More investigations and applications of these techniques are under our current research.

References

[1] W. B. Knox and P. Stone, "Interactively shaping agents via human reinforcement: the TAMER framework," in *Proceedings of the 5th International Conference on Knowledge Capture (K-CAP 2009)*, 2009, pp. 9–16.

[2] ——, "Reinforcement learning from simultaneous human and MDP reward," in *11st International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2012.

[3] ——, "Combining manual feedback with subsequent MDP reward signals for reinforcement learning," in *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2010, pp. 5–12.

[4] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004)*, 2004.

[5] W. B. Knox and P. Stone, "TAMER: Training of an agent manually via evaluative reinforcement," in *IEEE 7th International Conference on Development and Learning (ICDL-08)*, 2008, pp. 292–297.

[6] I. H. Witten, "An adaptive optimal controller for discrete-time markov environments," *Information and Control*, vol. 34, no. 4, pp. 286–295, 1977.

[7] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, & Cybernetics*, vol. 13, no. 5, pp. 834–846, 1983.

[8] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA]*, 1999, pp. 1057–1063.

[9] V. R. Konda and J. N. Tsitsiklis, "On actor-critic algorithms," *SIAM J. Control and Optimization*, vol. 42, no. 4, pp. 1143–1166, 2003.

[10] S. P. Singh, T. Jaakkola, and M. I. Jordan, "Learning without state-estimation in partially observable markovian decision processes," in *Machine Learning, Proceedings of the Eleventh International Conference (ICML)*, 1994, pp. 284–292.

[11] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[12] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Natural actor-critic algorithms," *Automatica*, vol. 45, no. 11, pp. 2471–2482, 2009.

[13] F. S. Melo and M. Lopes, "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)*, 2008, pp. 66–81.