Appl Intell (2013) 39:267–278
DOI 10.1007/s10489-012-0412-6

Learning via human feedback in continuous state and action spaces

Ngo Anh Vien · Wolfgang Ertel · Tae Choong Chung

Published online: 2 February 2013

Abstract This paper considers the problem of extending Training an Agent Manually via Evaluative Reinforcement (TAMER) in continuous state and action spaces. Investigative research using the TAMER framework enables 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 tasks of training agents by only human feedback or combining human feedback with environment rewards. However, these methods are originally designed for discrete state-action, or continuous state-discrete action problems. This paper proposes an extension of TAMER to allow both continuous states and actions, called ACTAMER. The new framework utilizes any general function approximation of a human trainer's feedback signal. Moreover, a combined capability of ACTAMER and reinforcement learning is also investigated and evaluated. The combination of human feedback and reinforcement learning is studied in both settings: sequential and simultaneous. Our experimental results demonstrate the proposed method successfully allowing a human to train an agent in two continuous state-action domains: Mountain Car and Cart-pole (balancing).

N.A. Vien (✉) · W. Ertel
Institute of Artificial Intelligence, Ravensburg-Weingarten University of Applied Sciences, Weingarten 88250, Germany
e-mail: ngoa@hs-weingarten.de

W. Ertel
e-mail: ertel@hs-weingarten.de

T.C. Chung
Department of Computer Engineering, Kyung Hee University, Seoul, South Korea
e-mail: tcchung@khu.ac.kr

Keywords Reinforcement learning · Human-agent interaction · Interactive learning · Human teachers · Reward shaping · Continuous states · Continuous actions

1 Introduction

TAMER is the framework helping with the design of agents trained by a human trainer's feedback of negative and positive [13]. The TAMER framework has performed well in tasks in which a human already has significant knowledge. It does not require any prior technical or programming skills from the human in order to transfer knowledge to agents. In some situations, it could reduce the cost of an agent's learning progress without damaging the asymptotic performance. Some successful examples include Tetris, Mountain Car, and Cart-pole [13, 17]. The results display that TAMER helps agents to reduce the sample complexity when learning within a Markov Decision Process (MDP) [12, 13]. This is relevant for tasks having a sparse and time-shifted (delayed) MDP reward function.

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 [11, 14–17]. In the sequential combination scheme, the TAMER agent is first trained by a human trainer, then the TAMER agent's policy is used to shape an RL agent to achieve convergence rapidly. The simultaneous scheme allows a human to interactively train the agent at any time during the autonomous learning process of RL process. 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 the combined applicability with RL because there are many efficient RL algorithms in continuous state-action space.

This paper introduces ACTAMER an extension of TAMER using continuous state and action domains, called Actor-Critic TAMER. The new framework utilizes any general reinforcement function approximation for a human trainer's value function. The TAMER framework is implemented as the *critic* which can use any form of the human trainer's function approximator. The *actor* implements a policy gradient algorithm in which the trainer's preference policy is defined 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 enables RL to be easily combined with human feedback. A combined capability of the ACTAMER and RL is also investigated and evaluated. The combination of human feedback and RL is studied in both sequential and simultaneous settings. The 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). The proposed extension still preserves the 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 an RL agent which helps to accelerate the autonomous learning process or improve the performance.

2 Background

This section describes the background of MDP and RL, the original TAMER framework, tile coding, and actor-critic methods used in our proposed approaches. The tile-coding method will be used as a function approximator in discrete TAMER framework, which establishes a baseline method for comparison.

2.1 MDP and reinforcement learning

A discrete MDP is defined by a five tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a discrete state space, $\mathcal{A}$ is a discrete action space, $\mathcal{T}(s,a,s') = P(s'|s,a)$ defines the probability of a next state, $\mathcal{R}(s,a,s')$ defines an immediate reward.

Reinforcement learning [34] is a sub-area of machine learning concerned with how an agent ought to take actions in an environment so as to maximize some notion of long-term reward. A control algorithm in RL tries to find an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$ with an assumption of unknown environment dynamics in order to maximize the expected total discounted reward ρ, which is defined in Eq. (1).

$$\rho = \mathrm{E}\left(\sum_{t=0}^{\infty} \gamma^t R\big(s_t, a_t, s_t'\big)\right) \tag{1}$$

where s_t, a_t, s_t', and γ denote a state, an action, a next state, and a discount factor respectively.

One of the most important breakthroughs in RL was the development of the temporal difference learning (TD learning) method which combines the advantages of the Monte-Carlo method and dynamic programming [33]. It is similar to a Monte Carlo method because it learns by sampling, and is also related to dynamic programming because it approximately updates the current value based on previously learned estimates. More specifically, the state-value function V^π of a given policy π is defined as $V^\pi(s_t)$, which is defined in Eq. (2).

$$V^\pi(s_t) = \mathrm{E}\left(\sum_{i=0}^{\infty} \gamma^i r_{t+i} \big| s_t\right) \tag{2}$$

This can be rewritten as

$$\begin{aligned} V^\pi(s_t) &= \mathrm{E}\left(r_t + \sum_{i=1}^{\infty} \gamma^i r_{t+i} \big| s_t\right) \\ &= \mathrm{E}\left(r_t + \gamma \sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \big| s_t\right) \\ &= \mathrm{E}(r_t) + \gamma V^\pi(s_{t+1}) \end{aligned} \tag{3}$$

where $r_t = R(s_t, a_t, s_t')$ is an immediate reward. Thus, the TD method approximately updates the state-value function V^π after each step (s_t, a_t, s_{t+1}) as

$$\bar{V}^\pi(s_t) \leftarrow \bar{V}^\pi(s_t) + \alpha\big(r_t + \gamma \bar{V}^\pi(s_{t+1}) - \bar{V}^\pi(s_t)\big) \tag{4}$$

which is based on the estimate value $\bar{V}^\pi(s_{t+1})$ of the next state s_{t+1}, where α is a learning rate parameter.

RL has had some remarkable practical successes in various areas, including learning to play checkers [26], backgammon [37–39], job-scheduling [45], chess [3], dynamic channel allocation [29, 41], and others [6–8, 22, 23].

2.2 The TAMER framework

The TAMER framework is first proposed by Knox et al. [12, 13], in which an agent is trained by receiving feedback of an observing human. TAMER has been evaluated and shown some successes in many problems such as Tetris [10, 12, 15], Mountain Car [13], and Cart-pole [14]. The framework is also studied thoroughly in the field of social robotics in order to evaluate how a human trainer's feedback frequency and the computational agent's learned performance can be affected [10]. Many later researches have tried to combine TAMER with RL to obtain better performance such as Knox et al. [14, 17]. The combination techniques improve not only the performance of an RL algorithm but also its learning speed.

TAMER is defined in the context of sequential decision making tasks which is mathematically modeled as an MDP. However, the reward function is not used, then it would be defined as an MDP$\backslash\mathcal{R}$ [1]. The feedback is communicated as negative or positive signals using agent observation [24, 44]. The human's feedback is considered as an action value (similar to Q-value of RL). Specifically, it is modeled as a human reinforcement function $\mathrm{H} : \mathcal{S} \times \mathcal{A} \rightarrow \Re$ at each state-action pair. After an action is taken by the agent and feedback provided by the human trainer, function H is updated in real-time by a regression method. 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 Knox [13], assuming that the human's reinforcement function is parametrized by a linear model $\mathrm{H}(s, a) = w^\top \phi(s, a)$, where w is a parameter and $\phi(s, a)$ is a feature function. With delayed feedback, the agent is uncertain about the time of the feedback signal it has just received (at time t). Therefore, the feedback may be 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}, which is defined in Eq. (5).

$$c_{t-k} = \int_{t-k-1}^{t-k} f(x)dx, \quad \text{for } k = 0, 1, \ldots \tag{5}$$

If the agent receives feedback $h \neq 0$ at time t, the error of each update step weighted by credit assignments is shaped as δ_t, which is defined in Eq. (6).

$$\delta_t = h - \sum_k c_{t-k} w_t^\top \phi(s_{t-k}, a_{t-k}) \tag{6}$$

The parameter update using a gradient of the least square, which is also weighted by credit assignments, is w_{t+1}, which is defined in Eq. (7).

$$w_{t+1} = w_t + \alpha_t \delta_t \sum_k c_{t-k} \phi(s_{t-k}, a_{t-k}) \tag{7}$$

The temporal error computation in Eq. (6) is normally $\delta_t = h - w_t^\top \phi(s_t, a_t)$ 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. (7). Depending on tasks, the history windows of the credits can be pruned to only recent time steps. Then, the credit computation at each step will not become a burden of the algorithm's computational time. With the help of the TAMER framework, the optimal policy is only based on the trainer's preference.

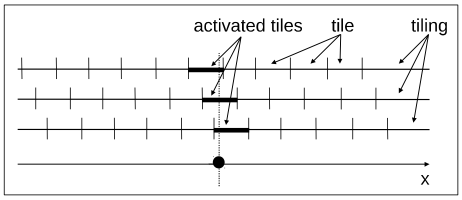


Fig. 1 Three-tiling example

2.3 Tile coding method

Tile coding [28, 32, 34] 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 a three-tiling example for the variable space consisting of a single continuous variable x. The weights of the tiles are highlighted for an indicated point.

Assuming that the state-action space is partitioned into a set of N tilings; $\{T_i; i = 1, \ldots, N\}$, each tiling T_i has N_i tiles $\{t_j; j = 1, \ldots, N_i\}$, 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

$$\mathrm{H}(s, a) = \sum_{i=1}^{N} \sum_{j=1}^{N_i} b_{ij} w_{ij} \tag{8}$$

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

$$\sum_{j=1}^{N_i} b_{ij} = 1, \quad \text{for all } i \tag{9}$$

2.4 Actor-critic algorithm

This section presents one of the popular RL techniques, the actor-critic algorithms. The actor-critic algorithms, first proposed by Barto [2] and Witten [43], implement both major techniques of RL algorithms. They maintain a value function as in Sarsa(λ) method, called the critic. Concurrently, they use a parametrized policy as in the policy gradient method 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 the policy's parameters. Under a compatible representation of the critic and actor, the algorithms are proved to converge to a local maximum [20, 35]. 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, making actor-critic methods the appropriate solution, which can learn an explicitly stochastic policy [30].

Sutton provides a very simple actor-critic method in RL [34] where the *critic* is implemented by a state-value function $V(s_t)$, and the *actor* is represented by the Gibbs softmax method which is defined in Eq. (10).

$$\pi_t(s,a) = \Pr\{a_t = a | s_t = s\} = \frac{e^{w^\top \phi(s,a)}}{\sum_{b\in\mathcal{A}} e^{w^\top \phi(s,b)}} \tag{10}$$

where the denominator sums over all the exponentials of the possible actions' value functions. After each action selection step and the observation of the next state s_{t+1}, the critic's evaluation is exactly a TD error δ_t which is defined in Eq. (11).

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{11}$$

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 recently taken is updated to know how its tendency must be weakened as described in Eq. (12).

$$\phi(s_t,a_t) = \phi(s_t,a_t) + \beta_t\delta_t \tag{12}$$

where β_t is the learning factor.

2.5 Related work

The proposed ACTAMER is based on the original TAMER framework [13], which is an instance of interactive machine learning. This family of interactive machine learning enables human trainers to interactively train an autonomous agent. Recently, there have been numerous techniques in this direction successfully leveraging the training tasks of the human such as the work of Judah [9], Subramanian [31], Taylor [36] and Thomaz [40]. Subramanian et al. [31] train an agent's options (macro actions) from demonstrations in an MDP framework. Taylor et al. [36] also use learning from demonstration (LfD) to initialize the policy, then apply RL to find the optimal policy. Knox et al. [11] discuss the difference between LfD and learning from human feedback. They also briefly argue some advantages of learning from human feedback over LfD. Judah et al. [9] make the learning agent alternate between practicing and critique receiving. In the critique stage, 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 [13], ACTAMER learns the human trainer's reinforcement function in continuous state and action domains. In order to extend TAMER to continuous problems, two well-known alternative methods in RL are used, tile-coding and the actor-critic. The actor-critic TAMER technique is implemented similar to the actor-critic RL algorithm discussed by Bhatnagar [4], where the authors proposed four actor-critic RL algorithms using linear function approximation. Meanwhile, TAMER uses a tile-coding function approximator similarly to Santamaria [27] in which the authors generalize previous RL algorithms to continuous state and action spaces. 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 [25]. However, the human feedback in this work is considered as immediate rewards which are different from the long-term effects in TAMER.

3 Tile-coding TAMER

A simple method to build the continuous TAMER framework is to use discretization of the state-action space. Thus, tile-coding is first used to build a baseline tile-coding TAMER to approximate the reinforcement function calculated in Eq. (8), where $\mathrm{H}(s,a)$ is a function of both continuous variables s and a. This is an application of the original TAMER using a different value function approximation. This simple extension will be used to compare with the main principle extension of the paper.

Tile-coding TAMER uses 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 described in Algorithm 1. The pseudo-code shown in line 7 computes the credit assignment of previous steps with respect to the current obtained feedback h_t. The pseudo-code shown 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 steps' gradient weighted by their corresponding credits, where α_t is a learning rate. The feature b_{ij}^{t-k} represents

[1]With respect to the actor's parameters.

Algorithm 1 Tile-coding TAMER

```
1:  Initialize weights $w^0_{ij}$, and an initial state $s_0$.
2:  while (1) do
3:      Choose an action $a_t = \text{argmax}_a \text{H}(s_t, a)$ (computed as in Eq. (8)),
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. (5)
8:          $\delta_t = h_t - \sum_k c_{t-k} \sum_{i=1}^{N} \sum_{j=1}^{N_i} b^{t-k}_{ij} w^t_{ij}$,
9:          for $i = 1$ to $N$ do
10:             for $j = 1$ to $N_i$ do
11:                 $w^{t+1}_{ij} = w^t_{ij} + \alpha_t \delta_t \sum_k c_{t-k} b^{t-k}_{ij}$
12:             end for
13:         end for
14:     end if
15: end while
```

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).

4 Continuous TAMER framework

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 with precision. Secondly, the degree of generalization is fixed according to the initial tile partition. Finally, if the state mapping into tiling features is non injective, then it would lose the convergence property of the regression algorithms. In practical applications, state and action spaces are often continuous or infinite, especially in robotic tasks such as motor primitive control [18, 19] and robotic grasping [5, 21]. In these tasks, the robot initially obtain knowledge using a demonstration of a human trainer. For continuous problems, the value functions of agents must be represented as function approximations. The proposed technique of ACTAMER is to use the policy gradient framework in which the human trainer's preference policy is represented in a family of randomized policies. The preference policy is then considered as the actor. In addition, the critic is implemented by the TAMER framework, to evaluate the actor's performance. As a result, the extension method is called a Actor-Critic TAMER (ACTAMER) framework.

ACTAMER operates in continuous state-action spaces. More specifically, the human trainer's reinforcement function is defined as the critic $\bar{\text{H}} : \mathcal{S} \rightarrow \Re$. ACTAMER uses a slightly different reinforcement function from TAMER's which depends only on state. Similarly, it is also updated in real-time by a regression algorithm. Assuming that the

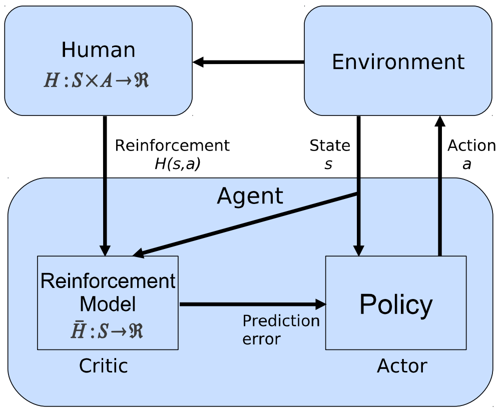


Fig. 2 ACTAMER framework

ACTAMER's critic is parametrized by a parameter space $\Theta = \{\theta_1, \ldots, \theta_m\}$. On the other hand, the ACTAMER's 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 2 shows the 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.

The trainer's reinforcement function is written as $\bar{\text{H}}(s)$, with the features represented by $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$, which is defined in Eq. (13).

$$\bar{\text{H}}(s) = \boldsymbol{\Theta}^\top \boldsymbol{\Phi} = \sum_n \theta_i \phi_i \tag{13}$$

Then, the critic's parameter update can be computed by modifying Eq. (7), also weighted by credit assignments, which is defined in Eq. (14).

$$\theta_{t+1} = \theta_t + \alpha_t \delta_t \sum_k c_{t-k} \boldsymbol{\Phi}(s_{t-k}) \tag{14}$$

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 RL algorithms in [4], if the feedback function $\text{H}(s, a)$ 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 2. The pseudo-code shown in line 8 computes the critic's regression error. The pseudo-code shown in lines 9 and 10 updates the critic and actor's parameters respectively. All of those computations also take the credit assignment into account.

The ACTAMER uses two-timescale stochastic approximation, then the learning parameters α_t and β_t would be

Algorithm 2 ACTAMER

```
1:  Initialize $\theta_0$, $w_0$, and an initial state $s_0$.
2:  while (1) do
3:      Choose an action $a_t \sim \mu(s_t, w_t)$,
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. (5)
8:          $\delta_t = h_t - \sum_k c_{t-k}\bar{\mathrm{H}}_t(s_{t-k})$,
9:          $\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t + \alpha_t\delta_t \sum_k c_{t-k}\nabla_{\boldsymbol{\Theta}}\bar{\mathrm{H}}_t(s_{t-k})$,
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
```

chosen to satisfy $\beta = o(\alpha_t)$. This is 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 $\mu(s, a)$.

5 Combining RL with ACTAMER

The ACTAMER framework could be extended to combine with RL for problems having both continuous state and action spaces. In this section, a technique among the eight listed in Knox et al. [14] is investigated in order to combine human feedback with RL, it is called control sharing. Because the other techniques are neither directly applicable to the continuous action space problem or they are inefficient as indicated in Knox et al. [14], the most four efficient alternative combining techniques Knox et al. [14, 17] are described there as:

Reward shaping: Because TAMER implicitly represents the policy in the value function $\hat{H}(s, a)$, so the reward shaping technique can easily replace the MDP reward with the sum of itself and the weighted value function $\hat{H}(s, a)$. However, ACTAMER explicitly represents the policy in a parametric form, thus using this form is difficult to shape the MDP reward function.

Q augmentation: This method requires the state action function $Q(s, a)$ to choose actions, which is not used in ACTAMER. ACTAMER chooses actions directly from the parameterized policy.

Action biasing: This method is similar to the *Q augmentation* method, which requires the state action function $Q(s, a)$.

Control sharing: This method allows choosing actions from two different sources of policy. With a specific probability β, actions are chosen normally according to the RL agent's policy, and with $(1 - \beta)$ actions are chosen according to the ACTAMER agent's policy. The parameter β will be annealed how to first utilize the knowledge from the human's policy, then finally only use the RL's optimal policy. Because the RL agent initially has no knowledge about the problem, exploiting the knowledge from the human's policy would speed up its learning process. Thus, this method is appropriate to the combining techniques of ACTAMER proposed in this paper and RL.

The ACTAMER framework embodies RL using both sequentially and simultaneously control sharing techniques.

5.1 Sequentially combining RL with ACTAMER

The sequential combination of ACTAMER+RL first enables 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: $P(a \sim \mu(s, w)) = \min(\beta, 1)$, otherwise uses the action selection mechanism of the base actor-critic RL algorithm, where β is a predefined combination parameter. This can be understood that the base actor-critic RL algorithm chooses exploration actions which are affected by the human's preference policy [14]. The parameter β decreases after each episode in order to gradually reduce the effect of the human's preference policy and converge to the RL algorithm's optimal policy.

5.2 Simultaneously combining RL with ACTAMER

Similar to Knox et al. [17], a simultaneous approach for combining ACTAMER and RL is proposed for continuous state and action domains. This modification allows a human trainer to intercept into an RL process to change its course of actions. The agent should learn from both types of rewards, human and MDP reward. 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 consistency of the learning behavior in the simultaneous approach, the combining method adopts the idea of balancing between following human trainer's preference policy and the policy learned by the 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*" [17]. 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

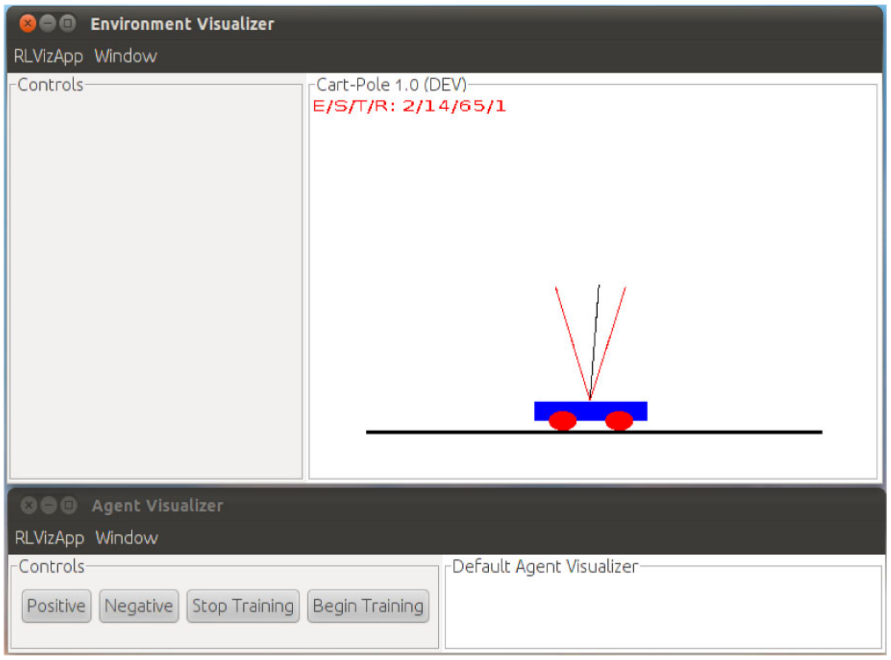


Fig. 3 Screenshot of the experiment interface from Cart-pole

time step's normalized feature vector. The eligibility module is used to make β a function of variables $\mathbf{e}, \mathbf{f}_t$, and c_s in range of $[0, c_s]$. More specifically, the influence of the human's policy is computed as β which is defined in Eq. (15).

$$\beta = c_s(\mathbf{e} \cdot \mathbf{f}_t)/\left(\|\mathbf{f}_t\|_1\right) \tag{15}$$

The eligibility trace uses accumulating traces as in TD(λ). The parameter β adaptively defines the influence of the human trainer's preference policy. It is large when there is recent feedback from the human trainer, and decreases in the absence of that interaction.

6 Experiment domains

In this section, the ACTAMER framework is evaluated on two popular domains in RL: Mountain Car and Cart-pole which are assumed to have continuous state and action spaces. The performance metric is the cumulative environmental MDP reward R. These two domains are chosen because they have previously been tested successfully with TAMER [13] and its RL derived extension [17]. With Mountain Car, both TAMER and standard RL algorithms such as SARSA(λ) and actor-critic can perform well. However with Cart-pole, SARSA(λ) can only find an optimal policy after 10000 episodes, and that is at least 800 episodes with an actor-critic RL algorithm. The bigger state space makes Cart-pole (4-dimensional state space) a more difficult problem than Mountain Car (2-dimensional state space) in finding an optimal policy. This will make a human trainer using ACTAMER rarely find a close optimal policy in Cart-pole domain, as reported in the next section. This is not the case in Mountain Car domain where a human trainer can quickly find a close optimal policy.

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. 3. 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, a uniform distribution $U(200, 900)$ is used as the credit assignment function in both domains.

6.1 Mountain Car

The well-known Mountain Car problem [42] 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 dynamic equations of

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

the car are described as in Eq. (16).

$$
\begin{aligned}
v_{t+1} &= v_t + 0.001 a_t - 0.0025 \cos(3 p_t) \\
p_{t+1} &= p_t + v_{t+1}
\end{aligned}
\tag{16}
$$

where the controlled action is a single continuous acceleration $a \in [-1.0; 1.0]$. The values of p and v is maintained bounded within their limits.

6.2 Cart-pole (balancing)

Cart-pole (Balancing) is another well-known benchmark in RL community. 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. The same setting in RL-Library is used, which implemented the discrete action Cart-pole example in [34]. The Cart-pole environment in RL-Library is modified to continuous actions. The pole length is $l = 0.5$ m, pole mass $m = 0.1$ kg, gravity $g = 9.8$ m/s^2, and cart mass $m_c = 1.0$ kg. 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 $a = F \in [-10$ N; 10 N]. 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.

6.3 Algorithm settings

ACTAMER and the combined techniques are compared to discrete TAMER (allowing only discrete actions as in the original paper of Knox et al. [13] in which the agent has only three discrete actions), tile-coding TAMER (using 25 equally distributed interval values for the actions), the standard Sarsa(λ) algorithm, and actor-critic RL as described in Bhatnagar et al. [4] (with only MDP reward). ACTAMER and actor-critic RL use the same representation for the actor and the critic.

Tile-coding TAMER: 48 tilings are used as a feature set for the function $\mathrm{H}(s, a)$. Each dimension is discretized by 12 intervals. There are 16 tilings based on three variables (p, v, a), uniformly offseted; 16 tilings based on two variables (p, v), uniformly offseted; 8 tilings based on one variable p, uniformly offseted; and 8 tilings based on one variable v, uniformly offseted. This feature set is also used to approximate the value function $Q(s, a)$ of Sarsa(λ). The implementation of tile-coding Sarsa(λ) is similar to [27].

ACTAMER: The critic, which is the human trainer's feedback function $\bar{\mathrm{H}}(s)$, is approximated by a linear approximator over Gaussian radial basis function (RBF) features. 16 and 256 uniformly centred RBF features within the limits of the state space are used in Mountain Car and Cart-pole respectively, four 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. A stochastic policy for the actor is generated 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: The combination parameter of the control sharing rule β is 1.0 for both problems, and is annealed by a factor of 0.98 after each episode.

Simultaneous ACTAMER+RL: The combination parameter of the control sharing rule β is tuned online using eligibility module as described in Sect. 5.2. The parameter c_s for Mountain Car and Cart-pole is set to 2.0 and 1.0 respectively.

The performance measured in both domains is the cumulative reward which sums all rewards received during one episode. The results of the best three and worse three are also provided as comparable variances. Because each human trainer's performance could be affected by his emotion and experience at the time of doing experiment.

7 Experimental Results

Fifteen people participated in training the Mountain Car and Cart-pole agents. Each trainer engaged in 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 Sect. 5.1.

7.1 Mountain Car

The comparisons for the Mountain Car domain are shown in the top panel of Fig. 4 when using ACTAMER-only. The results for the TAMER agents are averaged among the human trainers' results. The results of Sarsa(λ) and actor-critic RL are averaged over 100 trials. On average, both the ACTAMER agents give a comparable performance, but asymptotically outperform the Sarsa(λ) agent and are worse than actor-critic RL agent. However, ACTAMER found a suboptimal policy as quickly as actor-critic RL, and much faster than the Sarsa(λ). Thus, the ACTAMER can dramatically reduce sample complexity over the autonomous learning agents. On the other hand, the policy's behavior of ACTAMER agents is similar to discrete TAMER agents. The bottom panel of Fig. 4 shows the average performance received by agents trained by the best three trainers and worst three trainers for each algorithms. In this problem, ACTAMER's performance is not better than TAMER's because the optimal policy simply uses max and min acceleration which are in the action set of the discrete TAMER agent.

The results of the combination techniques of ACTAMER+RL are described in Fig. 5. 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 significantly improved the performance.

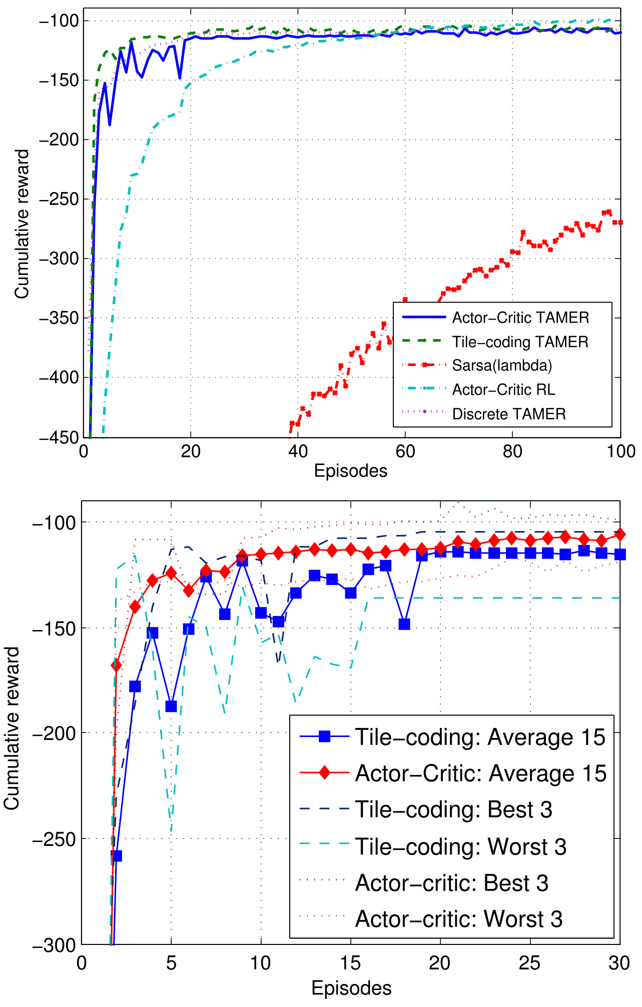


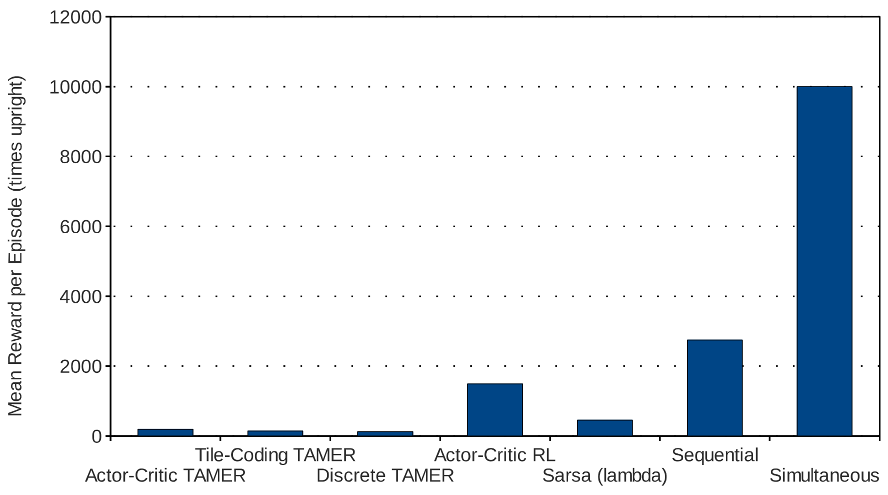
Fig. 4 Mountain Car: (*top*) The cumulative environmental rewards (steps to goal) of three algorithms; (*bottom*) The average cumulative environmental rewards, the best and worst three trainer's policy

7.2 Cart-pole (balancing)

The comparisons of ACTAMER against discrete TAMER for Cart-pole (balancing) domain are shown in the top panel of Fig. 6. The results for the TAMER agents are averaged among human trainers' results. In terms of the number of steps while balanced, ACTAMER has shown a significant advantage over the discrete TAMER. The bottom panel of Fig. 6 shows the average performance received by agents trained by the best three trainers and worst three trainers for each algorithms. In this more complicated domain, it's natural that the ACTAMER agent's asymptotic results are worse than autonomous learning agents such as Sarsa(λ) and actor-critic RL. Because the optimal policy needs to use small forces, which are not in the action set of the discrete TAMER agent (having only -10 N and 10 N), to yield smoother movement, this keeps the pole balanced with smaller angles.

The comparisons of ACTAMER and the combinations techniques against other RL algorithms are described in Fig. 7. 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). The results of the best three trainers (use the ACTAMER framework) are quite promising, they can train the agents to get over 300 times balancing

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



in within 50 episodes. Moreover, the results of these combination techniques ACTAMER+RL have outperformed other methods significantly. Especially, the simultaneous combination ACTAMER+RL successfully found an optimal policy after 150 episodes (which makes the pole balanced over 10000 steps) in comparison to at least 800 episodes of the only-RL agent.

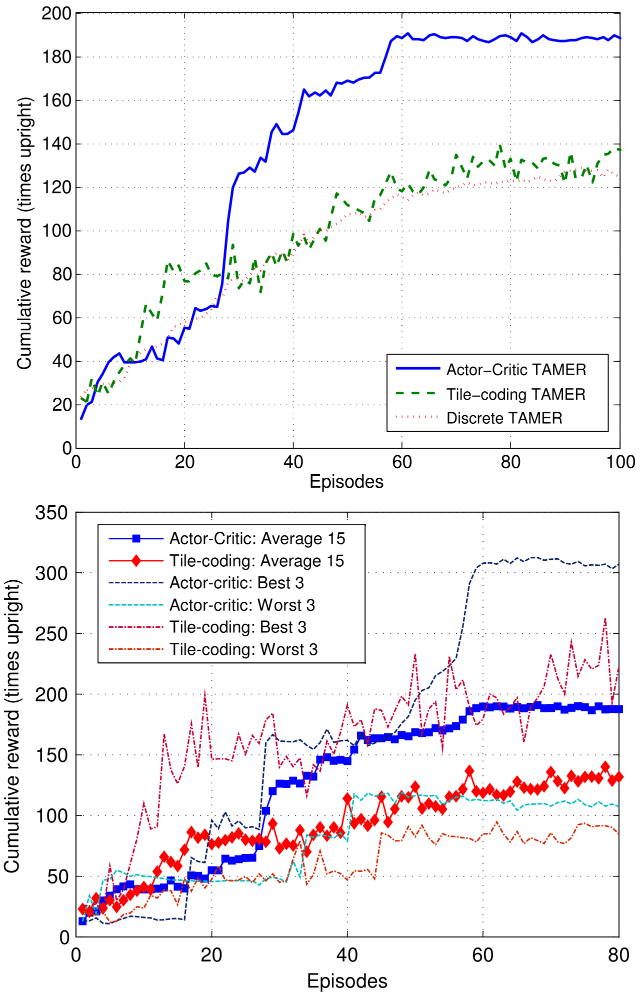


Fig. 6 Cart-pole (balancing): (*top*) The cumulative environmental reward (times upright); (*bottom*) The average cumulative environmental rewards, the best and worst three trainer's policy

8 Summary and future work

This paper proposes an extension for 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. The combination possibility of ACTAMER with an actor-critic method is also investigated. Two combination techniques are proposed, which combine sequentially or simultaneously human feedback with MDP reward in the continuous domains.

The proposed method is tested in two well-known domains in RL: 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 easily implemented, and accessible to non-technical human trainers. In terms of technical aspects, the new framework can still reduce sample complexity over autonomous learning algorithms. The preliminary results of the combination techniques (using only the control sharing rule) have shown their promising applicability in continuous domains, especially in robotics. The limitation of AC-

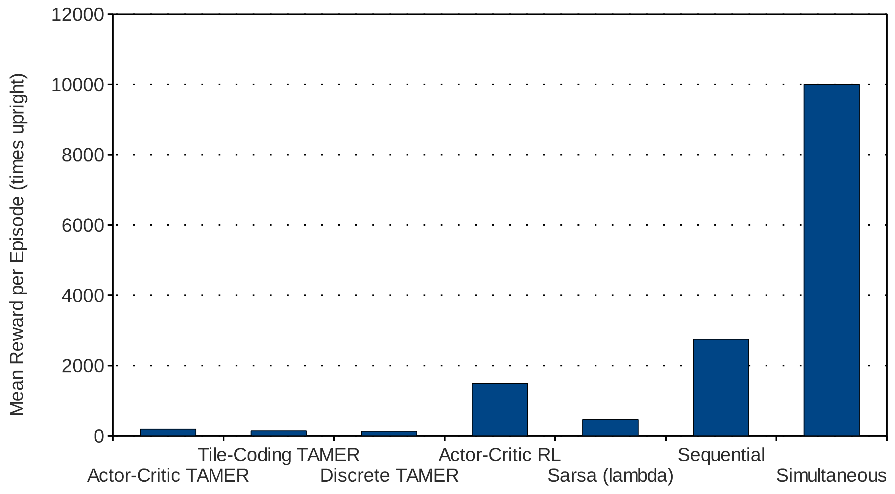


Fig. 7 Cart-pole (balancing): End-run performance of these algorithms for Cart-pole is averaged during the last five episodes (after 150 episodes of learning, capped at 10000 steps which is considered as successful balancing)

TAMER and TAMER as well is in application for more complex problems in which the human trainer does know how to give feedback, or his spent time for training an agent is limited. More investigations and applications of these techniques are suitable for future research.

Acknowledgements This work was supported by the Collaborative Center of Applied Research on Service Robotics (ZAFH Servicerobotik, http://www.zafh-servicerobotik.de) and the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science, and Technology (2010-0012609).

References

1. Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the twenty-first international conference on machine learning (ICML), pp 1–8
2. Barto AG, Sutton RS, Anderson CW (1983) Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans Syst Man Cybern 13(5):834–846
3. Baxter J, Tridgell A, Weaver L (2000) Learning to play chess using temporal differences. Mach Learn 40(3):243–263
4. Bhatnagar S, Sutton RS, Ghavamzadeh M, Lee M (2009) Natural actor-critic algorithms. Automatica 45(11):2471–2482
5. Detry R, Baseski E, Popovic M, Touati Y, Krüger N, Kroemer O, Peters J, Piater JH (2010) Learning continuous grasp affordances by sensorimotor exploration. In: From motor learning to interaction learning in robots, pp 451–465
6. Granmo OC, Glimsdal S (2012) Accelerated Bayesian learning for decentralized two-armed bandit based decision making with applications to the Goore game. Appl Intell
7. Hong J, Prabhu VV (2004) Distributed reinforcement learning control for batch sequencing and sizing in just-in-time manufacturing systems. Appl Intell 20(1):71–87
8. Iglesias A, Martínez P, Aler R, Fernández F (2009) Learning teaching strategies in an adaptive and intelligent educational system through reinforcement learning. Appl Intell 31(1):89–106
9. Judah K, Roy S, Fern A, Dietterich TG (2010) Reinforcement learning via practice and critique advice. In: Proceedings of the twenty-fourth AAAI conference on artificial intelligence, pp 481–486
10. Knox WB, Glass BD, Love BC, Maddox WT, Stone P (2012) How humans teach agents: a new experimental perspective. Int J Soc Robot 4(4):409–421
11. Knox WB, Setapen A, Stone P (2011) Reinforcement learning with human feedback in Mountain Car. In: AAAI 2011 spring symposium, pp 36–41
12. Knox WB, Stone P (2008) TAMER: training of an agent manually via evaluative reinforcement. In: IEEE 7th international conference on development and learning (ICDL-08), pp 292–297
13. Knox WB, Stone P (2009) Interactively shaping agents via human reinforcement: the TAMER framework. In: Proceedings of the 5th international conference on knowledge capture (K-CAP), pp 9–16
14. Knox WB, Stone P (2010) Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In: 9th international conference on autonomous agents and multiagent systems (AAMAS), pp 5–12
15. Knox WB, Stone P (2010) Training a Tetris agent via interactive shaping: a demonstration of the TAMER framework. In: 9th international conference on autonomous agents and multiagent systems (AAMAS), pp 1767–1768
16. Knox WB, Stone P (2011) Augmenting reinforcement learning with human feedback. In: 2011 ICML workshop on new developments in imitation learning
17. Knox WB, Stone P (2012) Reinforcement learning from simultaneous human and MDP reward. In: 11st international conference on autonomous agents and multiagent systems (AAMAS), pp 475–482
18. Kober J, Mohler BJ, Peters J (2010) Imitation and reinforcement learning for motor primitives with perceptual coupling. In: From motor learning to interaction learning in robots, pp 209–225
19. Kober J, Peters J (2011) Policy search for motor primitives in robotics. Mach Learn 84(1–2):171–203
20. Konda VR, Tsitsiklis JN (2003) On actor-critic algorithms. SIAM J Control Optim 42(4):1143–1166
21. Kroemer O, Detry R, Piater JH, Peters J (2010) Combining active learning and reactive control for robot grasping. Robot Auton Syst 58(9):1105–1116
22. Li J, Li Z, Chen J (2011) Microassembly path planning using reinforcement learning for improving positioning accuracy of a 1 cm^3 omni-directional mobile microrobot. Appl Intell 34(2):211–225
23. Pakizeh E, Palhang M, Pedram MM (2012) Multi-criteria expertness based cooperative Q-learning. Appl Intell
24. Phillips-Wren GE, Mørch AI, Tweedale J, Ichalkaranje N (2007) Innovations in agent collaboration, cooperation and teaming, part 2. J Netw Comput Appl 30(3):1085–1088
25. Pilarski PM, Dawson MR, Degris T, Fahimi F, Carey JP, Sutton RS (2011) Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In: IEEE international conference on rehabilitation robotics, pp 1–7
26. Samuel AL (1959) Some studies in machine learning using the game of checkers. IBM J Res Dev 3(3):210–229
27. Santamaria JC, Sutton RS, Ram A (1998) Experiments with reinforcement learning in problems with continuous state and action spaces. Adapt Behav 6(2):163–218
28. Sherstov AA, Stone P (2005) Function approximation via tile coding: automating parameter choice. In: Abstraction, reformulation and approximation, 6th international symposium (SARA), pp 194–205
29. Singh SP, Bertsekas D (1996) Reinforcement learning for dynamic channel allocation in cellular telephone systems. In: Advances in neural information processing systems (NIPS), pp 974–980
30. Singh SP, Jaakkola T, Jordan MI (1994) Learning without state-estimation in partially observable Markovian decision processes. In: Machine learning, Proceedings of the eleventh international conference (ICML), pp 284–292
31. Subramanian K, Isbell C, Thomaz A (2011) Learning options through human interaction. In: Workshop on agents learning interactively from human teachers at IJCAI
32. Sutton RS (1995) Generalization in reinforcement learning: successful examples using sparse coarse coding. In: Advances in neural information processing systems (NIPS), vol 8, pp 1038–1044
33. Sutton RS, Barto AG (1990) Technical note q-learning. In: Learning and computational neuroscience: foundations of adaptive networks, pp 497–537
34. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT Press, Cambridge
35. Sutton RS, McAllester DA, Singh SP, Mansour Y (1999) Policy gradient methods for reinforcement learning with function approximation. In: Advances in neural information processing systems, vol 12. NIPS conference, Denver, Colorado, USA, pp 1057–1063
36. Taylor ME, 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)
37. Tesauro G (1992) Practical issues in temporal difference learning. Mach Learn 8:257–277

38. Tesauro G (1994) Td-gammon, a self-teaching backgammon program, achieves master-level play. Neural Comput 6(2):215–219
39. Tesauro G (1995) Temporal difference learning and td-gammon. Commun ACM 38(3):58–68
40. Thomaz AL, 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
41. Vien NA, Viet NH, Lee S, Chung T (2009) Policy gradient SMDP for resource allocation and routing in integrated services networks. IEICE Trans 92-B(6):2008–2022
42. Vien NA, Yu H, Chung T (2011) Hessian matrix distribution for Bayesian policy gradient reinforcement learning. Inf Sci 181(9):1671–1685
43. Witten IH (1977) An adaptive optimal controller for discrete-time Markov environments. Inf Control 34(4):286–295
44. Wooldridge M (1997) Agent-based software engineering. In: IEE proceedings on software engineering, pp 26–37
45. Zhang W, Dietterich TG (1995) A reinforcement learning approach to job-shop scheduling. In: International joint conferences on artificial intelligence, pp 1114–1120