Selector-Actor-Critic and Tuner-Actor-Critic Algorithms for Reinforcement Learning

Ala'eddin Masadeh, Zhengdao Wang, Ahmed E. Kamal
Iowa State University (ISU), Ames, IA 50011, USA,
emails: {amasadeh,zhengdao,kamal}@iastate.edu

***Abstract*—This work presents two reinforcement learning (RL) architectures, which mimic rational humans in the way of analyzing the available information and making decisions. The proposed algorithms are called selector-actor-critic (SAC) and tuner-actor-critic (TAC). They are obtained by modifying the well known actor-critic (AC) algorithm. SAC is equipped with an actor, a critic, and a selector. The role of the selector is to determine the most promising action at the current state based on the last estimate from the critic. TAC is model based, and consists of a tuner, a model-learner, an actor, and a critic. After receiving the approximated value of the current state-action pair from the critic and the learned model from the model-learner, the tuner uses the Bellman equation to tune the value of the current state-action pair. Then, this tuned value is used by the actor to optimize the policy. We investigate the performance of the proposed algorithms, and compare with AC algorithm to show the advantages of the proposed algorithms using numerical simulations.**

***Index Terms*—Reinforcement learning, model-based learning, model-free learning, actor-critic**

I. Introduction

In the framework of artificial intelligence (AI), one of the main goals is to implement intelligent agents with high level of understanding and making correct decisions. These agents should have the capability to interact with their environments, collect data, process it, and improve their performance with time. Implementing autonomous agents capable of learning effectively has been a challenge for a long time. One of the milestones that have contributed in this field is reinforcement learning (RL). It is considered as a principled mathematical framework for learning experience-driven autonomous agents [1]. RL has been widely used to implement autonomous agents, e.g., [2]–[5].

RL refers to algorithms enable the agents to optimize their behaviors and improve their performance over time. The agents work in unknown environments, and learns from trial and error [6]. RL methods are categorized into two classes, which are model-free learning and model-based learning. Model-free learning updates the value function after interacting with the environment without learning the underlying model. On the other hand, model-based learning estimates the dynamics (the model) of an environment, which is used later to optimize the policy [7]. Each learning class has its own advantages, and suffers from a number of weaknesses. Model-based learning is characterized by its efficiency in learning [8], but at the same time, it struggles in complex problems [9]. On the other hand, model-free learning has strong convergence guarantees under certain situations [1], but the value functions change slowly over time [10], especially, when the learning rate is small.

In this work, we present two RL algorithms, which aim at providing efficient learning methods that utilize the available information efficiently. These architectures are called selector-actor-critic (SAC) and tuner-actor-critic (TAC). The main idea is to combine methods from the two learning classes to implement intelligent agents and to overcome the weaknesses of these two classes.

A. Related Work

Combining methods from both learning classes has been discussed in many works, e.g., [2], [3], [11]–[14].

In [2], a method called model-guided exploration is presented. This method integrates a learned model with an off-policy learning. The learned model is used to generate good trajectories using trajectory optimization, and then, these trajectories are mixed with on-policy experience to learn the agent. To overcome the weaknesses of the model-guided exploration approach, another method called imagination rollout was designed [2]. It was proposed for applications that need large amounts of experience, or when undesirable actions are expensive. In this approach, synthetic samples are generated from the learned model that are called the imagination rollouts. These rollouts, the on-policy samples, and the optimized trajectories from the learned model are used with various mixing coefficients to evaluate each experiment.

In [11], an algorithm called approximate model-assisted neural fitted Q-iteration was proposed. Using this algorithm, virtual trajectories are generated from a learned model to be used for updating the Q function. This work mainly aims at reducing the amount of real trajectories required to learn a good policy.

Actor-critic (AC) is a model-free RL method, where the actor learns a policy that is modeled by a parameterized distribution, while the critic learns a value function and evaluates the performance of the policy optimized by the actor. In [3], the framework of human-machine nonverbal communication was discussed. The goal is to enable machines to understand people intention from their behavior. The idea of integrating AC with model-based learning was proposed. The learned dynamics of the underlying model are used to control over

the temporal difference (TD) error learned by the critic, and the actor uses the TD error to optimize the policy for exploring different actions.

In [12], two learning algorithms were designed. The first one is called model learning actor-critic (MLAC), while the second one is called reference model actor-critic (RMAC). The MLAC is an algorithm that combines AC with model-based learning. In this algorithm, the gradient of the approximated state-value function $\hat{V}(s)$ with respect to the state s, and the gradient of the approximated model $s' = f(s, a)$ with respect to the action a are calculated. The actor is updated by calculating the gradient of $\hat{V}(s)$ with respect to a using the chain rule and the previously mentioned two gradients. However, using RMAC, two functions are learned. The first function is the underlying model. The second one is the reference model $s' = R(s)$, which maps state s to the desired next state s' with the highest possible value. Then, using the inverse of the approximated underlying model, the desired action can be found. The integrated paradigm of the reference model and the approximated underlying model serves as an actor, which maps states to actions.

Off-policy learning has been investigated in many works [15], [16], [17], [18]. Q-learning is considered as the most well known off-policy RL method [16]. This method evaluates an action at a state using its current value, the received immediate reward resulting from this action, and the value of the best action at the next state. This expected best action at the next state is selected independently of the currently executed policy, which is the reason for classifying this method as an off-policy learning method.

Combining off-policy learning with AC was studied in [17]. Using this algorithm, a stream of data (a sequence of states, rewards, and actions) is generated according to a fixed behavior policy. The critic learns off-policy estimates of the value function for the current actor policy. Then, these estimates are used by the actor for updating the weights of the actor policy, and so on. Using off-policy data (generated data from past interaction with the environment) to estimate the policy gradient accurately was investigated in [15]. In [19], an analytic expression for the optimal behavior policy (off-policy) is derived. This expression is used to generate trajectories with low variance estimates to improve the learning process by estimating the direction of the policy gradient efficiently.

B. Contributions

The main contribution in this paper is to introduce two new algorithms for RL, namely SAC and TAC:

1) In SAC, the newly added component, compared to the vanilla AC, is a selector. In conventional AC architecture, the actor uses the action selected by the current policy at the current state to optimize the policy's parameters. However, the selector in SAC determines the most promising action at the current state, which is used by the actor to optimize the policy's parameters.
2) TAC has two more elements added to AC, which are a model learner and a tuner. The model learner approximates the dynamics of the underlying environment, while the tuner tunes the value of the current state-action pair using the Bellman equation, the learned model, and the learned value function by the critic. The actor uses the tuned value of the current state-action pair to optimize the policy's parameters.

Our SAC method uses the idea of off-policy greedy action selection of Q-learning, and apply it to the AC algorithm to yield an off-policy AC algorithm.

The main differences between TAC and the most-related model based algorithm in [3] are summarized as follows:

- In [3], the critic approximates the state-value function to evaluate the system performance, while the critic in TAC approximates the action-value function.
- In [3], the policy uses a preference function for selecting actions, which indicates the preference of taking an action at a state. The preference function of the current state-action pair is updated by adding its old value to the current TD error learned by the critic. On the other hand, the actor in TAC uses stochastic parameterized policies to select actions, and uses policy gradient to optimize these policies.
- TAC uses the approximated underlying model, the approximated action-value function learned by the critic, and the Bellman equation to tune the value of the current state-action pair. In contrast, [3] uses the approximated underlying model to find the expected TD error for the current state. The value of the current state is updated by adding its previous value to the expected TD error.

The remainder of the paper is organized as follows. The formulated problem and the actor-critic architecture are presented in Section II. The proposed architectures are described in Section III. Numerical simulation results are presented in Section IV. Finally, the paper is concluded in Section V.

II. ACTOR-CRITIC

This part reviews the basic AC algorithm for RL. We use standard notation that is consistent with that used in e.g., [7]. Specifically, s and s' denote states, a and a' denote actions, $Q(s, a)$ denotes the action-value function, and $V(s)$ denote the state-value function. The function $\pi(a|s, \boldsymbol{\theta})$ denotes a stochastic policy function, parameterized by $\boldsymbol{\theta}$.

In AC, the actor generates stochastic actions, and the critic estimates the value function and evaluate the policy optimized by the actor. Figure 1 shows the interaction between the actor and the critic in the AC architecture, e.g., [20].

In this context, the critic approximates the action-value function $q^{\pi}(s, a) \approx Q(s, a)$, and evaluates the currently optimized policy using state-action-reward-state-action (SARSA), which is given by

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r(s,a,s') + \gamma Q(s',a') - Q(s,a)] \quad (1)$$

where α is the learning rate used to update $Q(s, a)$, γ is the discount factor, and $r(s, a, s')$ is the expected immediate reward resulting from taking action a at state s and transiting to state s'.

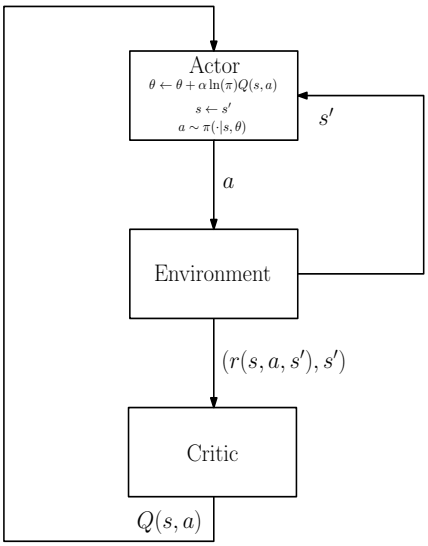

Fig. 1. Actor-critic architecture.

The actor uses policy gradient to optimize a parameterized stochastic policy $\pi(a|s,\boldsymbol{\theta})$. Using policy gradient, the policy objective function $J(\boldsymbol{\theta})$ takes one of three forms, which are

- The value of the start state in episodic environments

$$J_1(\boldsymbol{\theta}) = V^{\pi}(s_1) \tag{2}$$

- The average value in continuing environments

$$J_{\text{avV}}(\boldsymbol{\theta}) = \sum_s d^{\pi}(s) V^{\pi}(s) \tag{3}$$

- The average reward per time-step in continuing environments also

$$J_{\text{avR}}(\boldsymbol{\theta}) = \sum_s d^{\pi}(s) \sum_a \pi(a|s,\boldsymbol{\theta}) r(s,a) \tag{4}$$

where $d^{\pi}(s)$ is the steady-state distribution of the underlying Markov Decision Process (MDP) using policy π, and $r(s,a)$ is the expected immediate reward resulting from taking action a at state s. The goal is to maximize $J(\boldsymbol{\theta})$ [7]. The updating rule for $\boldsymbol{\theta}$ is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{5}$$

where $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is the gradient of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, and β is the step-size used to update the gradient of the policy.

One of the main challenges in this optimization problem is to ensure improvement during changing $\boldsymbol{\theta}$. This is because changing $\boldsymbol{\theta}$ changes two functions at the same time, which are the policy and the states' distribution. The other challenge is that the effect of $\boldsymbol{\theta}$ on the states' distribution is unknown, which makes it difficult to find the gradient of $J(\boldsymbol{\theta})$. Fortunately, policy gradient theorem provides an expression for the gradient of $J(\boldsymbol{\theta})$ that does not involve the derivative of the states' distribution with respect to $\boldsymbol{\theta}$ [7]. According to policy gradient theorem, for any differentiable policy and for any of the policy objective function, the policy gradient is

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx E_{\pi}[\nabla_{\boldsymbol{\theta}} \ln(\pi(a|s,\boldsymbol{\theta}))\, Q(s,a)] \tag{6}$$

Due to the difficulty of finding the expectation, a stochastic estimate $\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})}$ is used to approximate $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ [7], [21]. The new updating rule of $\boldsymbol{\theta}$ is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})} \tag{7}$$

where

$$\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})} = \nabla_{\boldsymbol{\theta}} \ln(\pi(a|s,\boldsymbol{\theta}))\, Q(s,a) \tag{8}$$

III. The Proposed Algorithms

A. Selector-Actor-Critic

We present a proposed off-policy policy-gradient method, where the policy being followed is optimized using the most promising action at state s. The idea is to approximate the most promising action (i.e., the optimal action) at state s by the greedy action a_g. To the best of our knowledge, it is the first work using the most promising action a_g to optimize stochastic parameterized policies using policy gradient methods. The goal is to optimize the policy in the direction that maximizes the probability of selecting a_g, and increase the speed of learning a suboptimal $\boldsymbol{\theta}$.

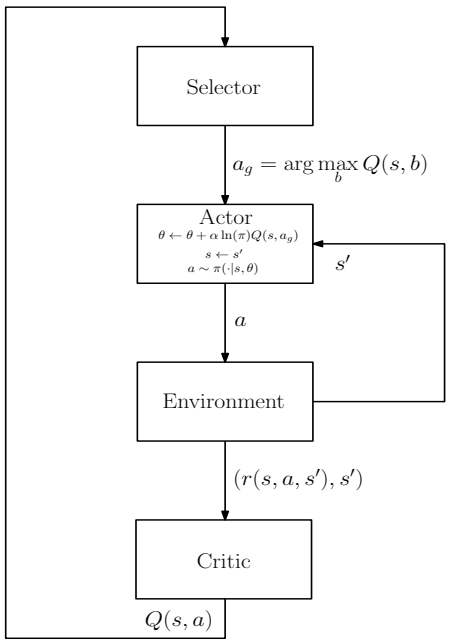

Fig. 2. Selector-actor-critic architecture.

To achieve this goal, a selector is added to the conventional AC. Figure 2 depicts the SAC model, and the interaction between its components. The selector determines a_g at the current state greedily according to

$$a_g = \underset{b}{\text{argmax}}\, Q(s,b), \quad \forall b \text{ at } s, \tag{9}$$

where b indicates each possible action at state s. After determining a_g by the selector, it is used by the actor to optimize the policy. The action a selected by the policy being followed in (8) is replaced by a_g. The new updating rule of $\boldsymbol{\theta}$ is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta\, \nabla_{\boldsymbol{\theta}} \ln(\pi(a_g|s,\boldsymbol{\theta}))\, [Q(s,a_g)]. \tag{10}$$

After selecting an action by the actor and interacting with the environment, the critic updates the action-value function according to

$$Q(s,a) \leftarrow Q(s,a)+\alpha[r(s,a,s')+\gamma Q(s',a')-Q(s,a)]. \quad (11)$$

B. Tuner-Actor-Critic

The TAC algorithm aims at improving the learning process through integrating a tuner and a model-learner with AC.

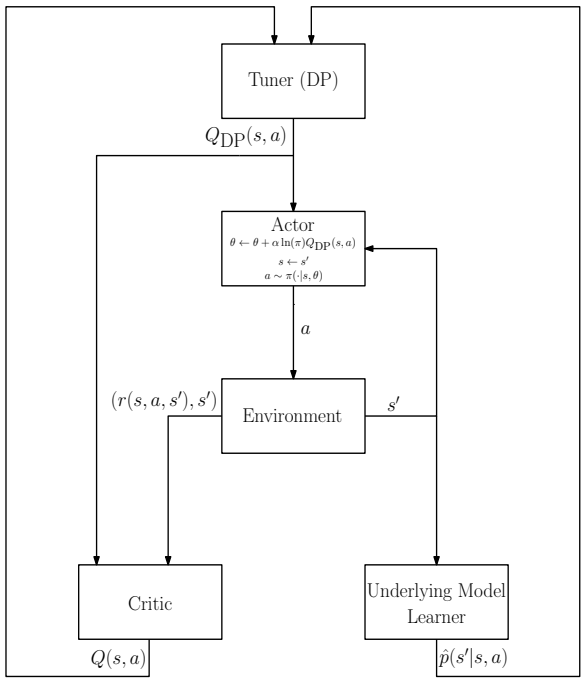

Fig. 3. Tuner-actor-critic architecture.

As shown in Figure 3, the newly added components to AC architecture are the tuner and the model learner. Starting from the values received from the critic and the model learner, the tuner uses the Bellman equation to tune the value of the state-action pair received from the critic. The tuner tunes the value of (s,a) state-action pair according to

$$Q_{\mathrm{DP}}(s,a) = \sum_{s'} \hat{p}(s'|s,a)\{r(s,a,s') + \gamma V(s')\}, \quad (12)$$

where $\hat{p}(s'|s,a)$ is the approximated probability for transiting from the current state s to next possible state s' given action a is taken, and $V(s') = \sum_{a'} \pi(a'|s',\boldsymbol{\theta}) Q(s',a')$ is the approximated value of s'. The critic replaces the value of the current state-action pair, (s,a), by the value computed by the tuner

$$Q(s,a) \leftarrow Q_{\mathrm{DP}}(s,a). \quad (13)$$

The actor updates $\boldsymbol{\theta}$ using

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \beta \nabla_{\boldsymbol{\theta}} \ln(\pi(a|s,\boldsymbol{\theta}))\,[Q_{\mathrm{DP}}(s,a)]. \quad (14)$$

After selecting an action and interacting with the environment, the critic evaluates the current policy using

$$Q(s,a) \leftarrow Q(s,a)+\alpha[r(s,a,s')+\gamma Q(s',a')-Q(s,a)]. \quad (15)$$

IV. Experimental Results

This section evaluates the proposed architectures. To evaluate these architectures, we use Value iteration (VI) to find the optimal solution as a benchmark when possible using the true underlying model [22]. AC algorithms from [3], [7] are also compared with.

A. Experimental Set-up

Two scenarios are considered in the simulation; a simple scenario with small number of states, and a scenario with large number of states. The simple scenario is used to evaluate and compare the proposed architectures with the optimal performance and AC. For the large scenario, the proposed models are only compared with AC, where it is difficult to find the optimal solution.

In all scenarios, the discount factor γ is set to 0.9. The learning rate α used by the critic is set to 0.1. The step-size learning parameter β used in policy gradient is set to 0.1. All the simulations started with an initial policy selecting the available actions uniformly. The approximated transition model was initialized with zero transition probabilities.

To evaluate the performance of the considered architectures, a number of MDP problems with different number of states and different dynamics were considered. The goal is to maximize the discounted return, where the discounted return following time t, G_t, is given by

$$G_t = \sum_{i=t}^{T-1} \gamma^{i-t} R_{i+1} \quad (16)$$

where t is the starting time for collecting a sequence of rewards, T is a final time step of an episode.

In the simulated environments, the simple scenario is modeled by an MDP with 18 states. Three actions are available with different immediate rewards and random transition probabilities. The second scenario is modeled using 354 states. The available actions are 7 with different immediate rewards and random transition probabilities. All the results were averaged over 500 runs. The starting state is selected randomly, where all the states have equal probability to be the starting state. All mentioned parameters were used in all experiments unless otherwise stated.

B. Exponential Softmax Distribution

The exponential softmax distribution [7] is used as a stochastic policy to select actions. The policy is given by

$$\pi(a|s,\boldsymbol{\theta}^s_a) = \frac{\exp(h(s,a,\boldsymbol{\theta}^s_a))}{\sum_b \exp(h(s,b,\boldsymbol{\theta}^s_b))} \quad (17)$$

where $\exp(x)$ is the base of the natural logarithm, $h(s,a,\boldsymbol{\theta}^s_a) \in \mathbb{R}$ is the parameterized preference for (s,a) pair, and $\boldsymbol{\theta}^s_a$ is the policy's parameter related to action a at state s. For discrete and small action spaces, the parameterized preferences can be allocated for each state-action pair [7].

The parameterized preferences are functions of feature functions $\boldsymbol{\phi}(s,a)$ and the vector $\boldsymbol{\theta}^s_a$, which are used to determine the preference of each action at each state. The action with the

highest preference at a state will be selected with the highest probability, and so on [7]. These preferences can take different forms. One of the simple forms is that when the preference is a linear function of the weighted features, which is given by

$$h(s,a,\boldsymbol{\theta}_a^s) = \boldsymbol{\theta}_a^{s\top}\boldsymbol{\phi}(s,a) \tag{18}$$

The $\nabla_{\boldsymbol{\theta}_a^s} \ln(\pi(a|s,\boldsymbol{\theta}_a^s))$ is given by

$$\begin{aligned}\nabla_{\boldsymbol{\theta}_a^s} \ln(\pi(a|s,\boldsymbol{\theta}_a^s)) &= \frac{\nabla\pi(a|s,\boldsymbol{\theta}_a^s)}{\pi(a|s,\boldsymbol{\theta}_a^s)} \\ &= \boldsymbol{\phi}(s,a) - \pi(a|s,\boldsymbol{\theta}_a^s)\,\boldsymbol{\phi}(s,a)\end{aligned} \tag{19}$$

The feature function $\boldsymbol{\phi}(s,a)$ for (s,a) pair is used for representing the states and actions in an environment. Feature functions should correspond to aspects of the state and action spaces, where the generalization can be implemented properly [7]. This work uses binary feature functions. Feature function for a state-action pair is set to one if action a satisfies the feasibility condition at state s, otherwise, it is set to zero.

C. Comparisons

In this experiment, the discounted return G_t of each architecture was evaluated. The optimal performance uses the optimal policy from the first time slot. It requires a priori statistical knowledge about the environment, which is unavailable to the remaining architectures. Value iteration was used to find the optimal policy to find the upper-bound [22].

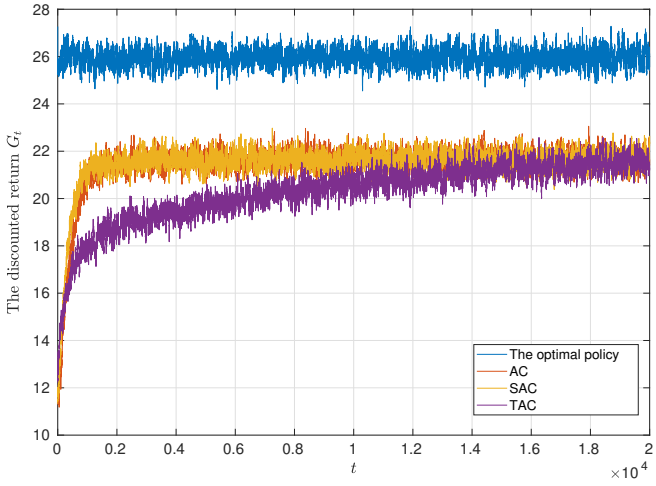

Fig. 4. The discounted return G_t versus t for 18-state system

Figure 4 shows the discounted return G_t versus t for the considered architectures. As expected, the discounted return of the optimal policy takes a near-constant pattern from the first time slot. This is due to using one policy all the time, and the discount factor γ which restricts the discounted return to a certain value. The discounted returns of the RL architectures increase with experience significantly, in the beginning. As the time increases, they start taking a near-constant pattern, which results from learning policies that could not be improved any more, and γ that restricts the discounted return of the architectures to certain values.

AC experiences an action at the current state, and then, it optimizes the policy based on the approximated value of the current state-action pair. TAC just tunes the approximated value of the experienced action using the learned underlying model, then, this tuned value is used by the actor to optimize the policy. It is clear that both AC and TAC do not exploit the available information about the remaining actions at the current state to optimize the policy. SAC experiences an action at the current state, and then, based on its approximated value and the approximated values of other actions, it optimizes the policy. It is observed that for the simulated scenario, SAC is able to converge faster to a high return value, and TAC converges more slowly compared to AC and AC.

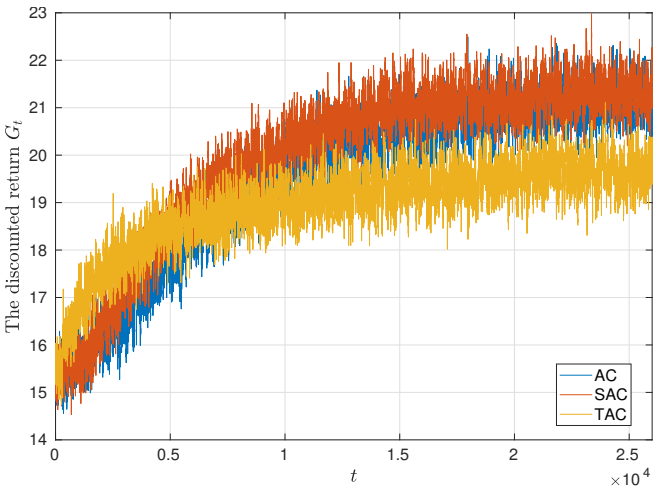

Fig. 5. The discounted return G_t versus t.

Fig. 5 shows the performances of AC, SAC, and TAC when there are 364 states. It can be observed that TAC has a faster initial learning rate, and SAC converges to a higher return, both compared to AC.

D. Rarely visited states

We investigate the performance of different models in the case of having rarely visited good states. For such cases, the opportunity to increase the cumulative reward is small. Also, experiencing bad actions would be very expensive. Optimizing the policy and selecting an optimal action at rarely visited states is difficult due to lack of experience at these states. So, the available information should be utilized efficiently to make correct decisions. This leaves room for improving the performance based on previous experience.

Fig. 6 and Fig. 7 show the performance of different architectures when good states are visited rarely, for 18- and 364-state systems, respectively. Regarding to the quality and the speed of finding a suboptimal policy, the results show the superiority of SAC compared to regular AC. Also TAC has slight advantage in the beginning of the learning.

V. Conclusions

In this paper, two new RL algorithms, named selector-actor-critic and tuner-actor-critic are proposed by adding compo-

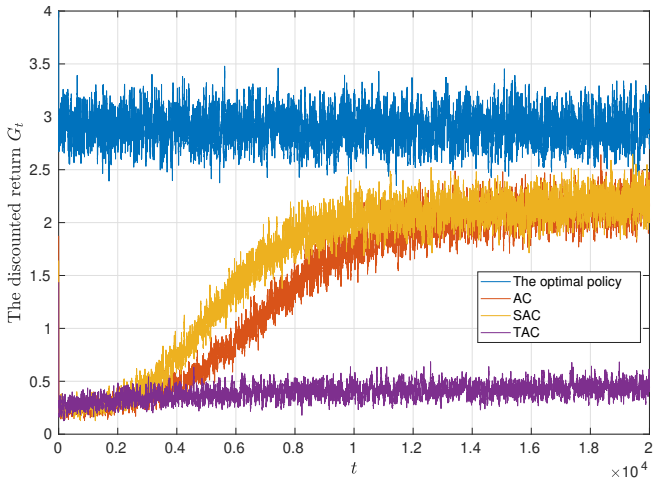

Fig. 6. The discounted return G_t versus t for 18-state system with rarely visited states.

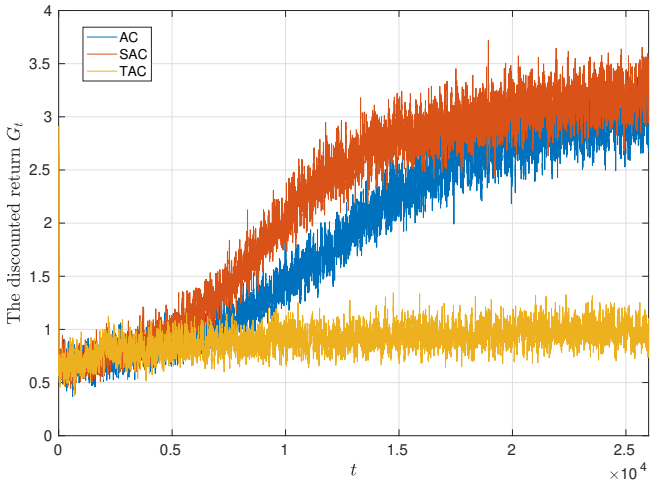

Fig. 7. The discounted return G_t versus t for 364-state system with rarely visited states.

nents to the conventional AC algorithm. Instead of using on-policy policy gradient as in AC, SAC uses off-policy policy gradient. TAC aims at improving the learned value function by adding a model-learner and a tuner to improve the learning process. The tuner tunes the approximated value of the current state-action pair using the learned underlying model and the Bellman equation. AC, SAC, and TAC experience an action, and then, they optimize their policies based on the value of the experienced action. Based on numerical simulations, SAC seems to offer higher converged returns, and TAC is preferred if faster initial learning rate is desirable.

ACKNOWLEDGEMENT

This work was supported in part by NSF Grant 1711922 and NSF Grant 1827211.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT Press, 1998.

[2] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep q-learning with model-based acceleration," in *Proc. of the International Conference on Machine Learning*, New York, NY, USA, June 2016, pp. 2829–2838.

[3] M. A. T. Ho, Y. Yamada, and Y. Umetani, "An HMM-based temporal difference learning with model-updating capability for visual tracking of human communicational behaviors," in *Proc. of the IEEE International Conference on Automatic Face and Gesture Recognition*, Washington, DC, USA, May 2002, pp. 170–175.

[4] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, Oct. 2017.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, Jan. 2016.

[6] T. Mannucci, E.-J. van Kampen, C. de Visser, and Q. Chu, "Safe exploration algorithms for reinforcement learning controllers," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 4, pp. 1069 – 1081, Apr. 2018.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT Press, 2018.

[8] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, Aug. 2013.

[9] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning," in *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, QLD, Australia, May 2018, pp. 7559–7566.

[10] Q. J. Huys, A. Cruickshank, and P. Seriès, "Reward-based learning, model-based and model-free," in *Encyclopedia of Computational Neuroscience*, Mar. 2015, pp. 2634–2641.

[11] T. Lampe and M. Riedmiller, "Approximate model-assisted neural fitted q-iteration," in *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, Beijing, China, July 2014, pp. 2698–2704.

[12] I. Grondman, M. Vaandrager, L. Busoniu, R. Babuska, and E. Schuitema, "Efficient model learning methods for actor–critic control," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 3, pp. 591–602, June 2012.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[14] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Machine Learning Proceedings*, Austin, Texas, USA, June 1990, pp. 216–224.

[15] J. P. Hanna and P. Stone, "Towards a data efficient off-policy policy gradient," in *Proc. of the AAAI Spring Symposium on Data Efficient Reinforcement Learning*, Palo Alto, CA, Mar. 2018, pp. 320–323.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[17] T. Degris, M. White, and R. S. Sutton, "Off-policy actor-critic," *arXiv preprint arXiv:1205.4839*, 2012.

[18] H. R. Maei, C. Szepesvári, S. Bhatnagar, and R. S. Sutton, "Toward off-policy learning control with function approximation," in *Proc. of the International Conference on Machine Learning (ICML)*, Haifa, Israel, June 2010, pp. 719–726.

[19] J. P. Hanna, P. S. Thomas, P. Stone, and S. Niekum, "Data-efficient policy evaluation through behavior policy search," *arXiv preprint arXiv:1706.03469*, 2017.

[20] X. Xu, D. Hu, and X. Lu, "Kernel-based least squares policy iteration for reinforcement learning," *IEEE Transactions on Neural Networks*, vol. 18, no. 4, pp. 973–992, July 2007.

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

[22] T. Wang, C. Jiang, and Y. Ren, "Access points selection in super wifi network powered by solar energy harvesting," in *Proc. of the IEEE Wireless Communications and Networking Conference (WCNC)*, Doha, Qatar, Apr. 2016, pp. 1–5.