Online Reinforcement Learning by Bayesian Inference

Zhongpu Xia, Dongbin Zhao
The State Key Laboratory of Management and Control of Complex Systems,
Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China
Email: zhongpu.xia@gmail.com, dongbin.zhao@ia.ac.cn

***Abstract*—Policy evaluation has long been one of the core issues of the online reinforcement learning, especially in the continuous state domain. In this paper, the issue is addressed by employing Gaussian processes to represent the action value function from the probability perspective. By modeling the return as a stochastic variable, the action value function can sequentially update according to observed variables such as state and reward by Bayesian inference during the policy evaluation. The update rule shows that it is a temporal difference learning method with the learning rate determined by the uncertainty of a collected sample. Incorporating the policy evaluation method with the ϵ-greedy action selection method, we propose an online reinforcement learning algorithm referred as to Bayesian-SARSA. It is tested on some benchmark problems and the empirical results verifies its effectiveness.**

I. Introduction

Reinforcement learning simulates the self-learning mechanism of an organism which learns from its interaction with the environment [1]. It is able to learn an available control policy based on the collected data. This metric makes it a research hotspot in the control community [2]–[4]. Nevertheless, how to calculate the value function for a given control policy from collected samples is the core issue in its application, especially in the continuous state domain [5], [6].

In the continuous state domain, it is infeasible to represent the value function explicitly, and so a class of parametric approximations such as multi-layer perceptron [7]–[9], linear approximation architecture [10], [11] is mainly adopted to approximate the value function. However, features selection and architectures are hard to manipulate, and require experience and repeated testing on collected data. Otherwise, it will result in underfitting or overfitting [12].

Fortunately, these problems can be avoided with a non-parametric approximation architecture since the number of its features grows along with the collection of data. Moreover, non-parametric approximation architectures have been proven to be consistent with mild assumptions for the reinforcement learning [13], [14]. Gaussian process is one of the kernel machines and provides a state-of-the-art nonparametric Bayesian regression framework commonly used in machine learning [15]. It is able to represent not only the estimation of the target value, but also the uncertainty of the estimation. Rasmussen *al. et.* [16]–[18] employed Gaussian processes to identify the model of an underlying system and the value function thus can be solved in a closed form, but extra computation is required to identify the model. Represented the value function directly by Gaussian Processes, Engel *al. et.* proposed the Gaussian process temporal difference (GPTD) learning algorithm [19]. Furthermore, they illustrated that GPTD inherited the metrics of the temporal difference learning which was endowed with a low variance and that Monte Carlo learning could efficiently use the collected data. Previous works commonly focused on addressing the policy evaluation problem from the perspective of regression.

In this paper, Gaussian processes are employed to represent the action value function to solve the issue of the policy evaluation from the probability perspective. The return of the collected data is modeled as a stochastic variable with its expectation as the action value function. Based on such a style of modeling, the action value function can sequentially update according to the collected data through Bayesian inference. It is the first contribution of this paper. Moreover, an online reinforcement learning algorithm, referred to as Bayesian SARSA, is proposed by incorporating the update rule of the action value function with the ϵ-greedy action selection method. It is the second contribution of this paper. At last, the effectiveness of the algorithm is tested and verified by several different tasks.

The rest of the paper is mainly organized as follows. Section II introduces the fundamentals of reinforcement learning and Gaussian processes. Section III presents an approach to update the action value function by Bayesian inference. Section IV proposes the online reinforcement learning algorithm. Section V tests and demonstrates the performance of the algorithm by several different tasks, and then the conclusion is given in section VI.

II. Background

A. Markov Decision Process

In reinforcement learning, an agent learns a control policy by interacting with an underlying system. The interaction can be modeled as a Markov decision process (MDP), which can be described as 5-tuples of the form $\{S,\ A,\ P(s'|s,a),\ r(s,a),\ \gamma\}$, where S is the state space; A is the action set; $P(s'|s,a)$ is the probability of a state transition from s to s' by taking action a and a reward $r(s,a)$ is given to the agent during such a state transition; γ is the discount factor which ranges in $[0,\ 1]$. The discounted return is defined as the sum of exponentially discounted rewards collected along

This work is supported by National Natural Science Foundation of China (NSFC) under Grants No. 61273136 and No. 61034002.

978-1-4799-1959-8/15/$31.00 ©2015 IEEE

a trajectory from state s,

$$R(s) = \sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s. \tag{1}$$

The control policy is a mapping from a state to an action selection probability $\pi(s) = p(a|s)$. Its performance over the state and action space can be defined as the expectation of the discounted return by taking action a from state s and thereafter following a policy π:

$$Q^{\pi}(s,a) = \mathbf{E}\left\{r(s,a) + \gamma R^{\pi}(s')\right\}. \tag{2}$$

Consequentially, the greedy action on a state can be directly selected by maximizing the action value function over the action space

$$a_{greedy}(s) = \arg\max_{a \in \mathbb{A}} Q^{\pi}(s,a). \tag{3}$$

The action value function is usually applied in model-free reinforcement learning as control policies can be improved directly. It can be represented explicitly in the discrete state and action domains by using Q-table, but it is impractical or impossible for underlying systems with large or infinite state spaces. Thus, a function approximation becomes necessary for the policy evaluation for such cases. Here, Gaussian processes are chosen to model the action value function.

B. Gaussian Processes

It is always assumed that Gaussian processes are a collection of random variables with respect to inputs x, any finite subsets of which have joint Gaussian distributions with a latent function $f_0(x)$ and a positive semidefinite covariance function $k(x, x')$, also called a kernel [15]. The squared exponential (SE) function is commonly chosen as the kernel

$$k(x,x') = \alpha_0^2 \exp\left(-\frac{1}{2}(x-x')^{\mathsf{T}}\mathbf{L}^{-1}(x-x')\right) \tag{4}$$

which reflects the prior brief that we expect the latent function to be smooth [15]. The element in $\mathbf{L} = diag\left([l_1, l_2, \cdots, l_n]\right)$ is the parameter of length scale which determines the degree of correlation for each dimension of the input vector. Hence, the latent function with respect to each input has a prior Gaussian distribution $\mathcal{N}(f_0,\, \alpha_0^2)$.

Following the standard setup, the observations $\mathbf{y}$ are the sum of the latent function $f(\cdot)$ of inputs $\mathbf{x}$ plus independent zero-mean Gaussian noises $\mathcal{N}(0, \sigma_n^2)$. $\mathbf{x}$ is the collected input data, also named bias vector. The kernel matrix $\mathbf{K}$ with elements $\mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$ is the covariance matrix of the collected data. While a new sample x' is collected, it has a covariance vector $k(\mathbf{x}, x')$ related to the bias vector, and thus the corresponding latent function value distribution can be calculated as a conditional probability distribution of the bias vector with the mean and variance:

$$\mu(x') = f_0 + k(\mathbf{x}, x')^{\mathsf{T}}(\mathbf{K} + \sigma_n^2\mathbf{I})^{-1}(\mathbf{y} - \mathbf{f}_0) \tag{5a}$$

$$\sigma^2(x') = k(x', x') - k(\mathbf{x}, x')^{\mathsf{T}}(\mathbf{K} + \sigma_n^2\mathbf{I}))^{-1}k(\mathbf{x}, x'). \tag{5b}$$

III. Bayesian Policy Evaluation

In reinforcement learning, the model of an underlying system is always unavailable, namely, its state transition probability $P(s'|(s,a))$ and the reward function are unknown. A reinforcement learning agent has to learn a control policy by interacting with the underlying system. The information collected during the interaction at one time can be denoted as a 4-tuple (s_t, a_t, r_t, s_{t+1}), also called a sample, meaning the agent takes action a_t when the underlying system stays in a state s_t at time t, then a reward r_t is received and the underlying system transits to a next state s_{t+1}. Such samples can be collected all through the operation of an underlying system. We denote the collected samples till time t as $\mathbf{D}_t = \{(s_i, a_i, r_i, s_{i+1}) \,|i = 1, 2, 3, \cdots, t\}$.

A. Modeling Action Value function

Policy evaluation is to calculate the action value function of a control policy from the collected samples. By modeling the collected samples as the observed variables and the action value functions as the latent variables, the policy evaluation is transformed into a problem of inferring the latent variables from the observed variables based on the probability theory. Thus, the action value function can be solved by Bayesian inference,

$$p(Q|\mathbf{D}_t) = \frac{p(\mathbf{D}_t|Q)p(Q)}{P(\mathbf{D}_t)}. \tag{6}$$

Through such a transformation, the goal is to determine the probability distributions of prior $p(Q)$ and likelihood $p(\mathbf{D}_t|Q)$.

Gaussian processes would be an elegant solution to model the action value function as a stochastic variable, $Q(s,a) = f(s,a)$ with prior expectation $\mathbf{E}(f(s,a)) = f_0$ and prior covariance $\mathbf{E}(f(s,a)f(s',a'))) = k\left([s,a],[s',a']\right)$ for all $s, s' \in S$ and $a, a' \in A$. For clarity, we will collect a state-action pair into a vector $x_t = [s_t, a_t]$, and define the bias vector of this Gaussian process as $\mathbf{x}_t = \{[s_i, a_i],\ i = 1, 2, 3, \cdots, t\}$. Therefore, the covariance matrix of the bias vector is $[\mathbf{K}_t]_{i,j} = k\left([\mathbf{x}_t]_i, [\mathbf{x}_t]_j\right)$, kernel vector is $[\mathbf{k}_t]_i = k\left(x_t, [\mathbf{x}_t]_i\right)$ and $k_{t,t} = k(x_t, x_t)$. $[\mathbf{f}_t]_i = f(x_i)$ and $f_t = f(x_t)$ for clarity, too.

By assuming that the action value function is a Gaussian process, we setup the relationship between the whole state-action pairs and the collected state-action pairs by the covariance matrix. The action value function can be estimated from the collected samples

$$p(f|\mathbf{D}_t) = \int p(f|\mathbf{f}_t)p(\mathbf{f}_t|\mathbf{D}_t)d\mathbf{f}_t. \tag{7}$$

Following the formulation of [19], the discounted return R at each episode is decomposed into its mean $\mathbf{E}(R)$ and a random, zero-mean Gaussian noise with variance σ_R^2,

$$R = \mathbf{E}\{R\} + \mathcal{N}(0, \sigma_R^2) \tag{8}$$

where the expectation is the action value function as discussed in (1), namely $\mathbf{E}\{R\} = f$. The expectation is deterministic and no longer random in the classical probability theory, but here it is viewed as a random entity by assigning it additional randomness that is due to the subjective uncertainty regarding the underlying system in reinforcement learning.

Till now, we have set up the relationships between the return, collected samples and the action value function by (1), (7) and (8). Besides, (7) can be solved easily by the property of the Gaussian process regression associated with (5). Hence, the following will focus on how to update the action value function of the collected samples. It is detailed as:

$$p(\mathbf{f}_t|\mathbf{D}_t) = \frac{\int p(\mathbf{D}_t|R)p(R|f)p(f|\mathbf{f}_t)p(\mathbf{f}_t)dRdf}{P(\mathbf{D}_t)}. \quad (9)$$

B. Online Updating

Here we are interested in updating the action value function in a sequential form, in which a new observation is incorporated at each time instant a sample of the underlying system collected. Instead of recomputing the $p(\mathbf{f}_t|\mathbf{D}_t)$ at every time instant, the action value function is updated recursively for a low-cost as follows:

$$\begin{aligned} &p(\mathbf{f}_t|\mathbf{D}_t) \\ &= p(\mathbf{f}_{t-1}, f_t|\mathbf{D}_{t-1}, r_t) \quad (10a) \\ &= \frac{\int p(r_t|R_{t+1}, R_t)p(R_{t+1}, R_t|\mathbf{f}_{t-1})p(\mathbf{f}_{t-1}|\mathbf{D}_{t-1})dR_t dR_{t+1}}{p(r_t|\mathbf{D}_{t-1})} \quad (10b) \end{aligned}$$

The equation (10b) follows from (10a) by expanding the relationships of the variables according to (9), and then by applying Bayesian inference. For simplicity, only r_t from the sample (s_t, a_t, r_t, s_{t+1}) is expressed explicitly in the above formula. s_t, a_t, s_{t+1} are expressed implicitly, but they will be explicitly detailed below.

According to (10), a new posterior probability $p(\mathbf{f}_t|\mathbf{D}_t)$ including the most recent action value function $f(x_t)$ is calculated after a new sample is included based on a prior probability $p(\mathbf{f}_{t-1}|\mathbf{D}_{t-1})$. The calculated distribution $p(\mathbf{f}_t|\mathbf{D}_t)$ will become a prior probability at the next update. This process repeats when a new sample is collected. In order to get the solution of the equation, it is assumed that $p(\mathbf{f}_{t-1}|\mathbf{D}_{t-1})$ is a known Gaussian process

$$p(\mathbf{f}_{t-1}|\mathbf{D}_{t-1}) = \mathcal{N}\left(\mathbf{f}_{t-1}|\boldsymbol{\mu}_{t-1}, \Sigma_{t-1}\right). \quad (11)$$

After a new sample (s_t, a_t, r_t, s_{t+1}) is observed, the distribution of the action value function on the new observes (s_t, a_t) and $(s_{t+1}, \pi(s_{t+1}))$ conditioned on $\mathbf{f}_{t-1}$ is still a joint Gaussian distribution according to the property of Gaussian processes

$$\begin{aligned} &p(f_t, f_{t+1}|\mathbf{f}_{t-1}) \\ &\quad = \mathcal{N}\left(\begin{bmatrix} f_t \\ f_{t+1} \end{bmatrix} | \begin{bmatrix} \hat{f}_t \\ \hat{f}_{t+1} \end{bmatrix}, \quad \begin{bmatrix} \sigma^2_{t,t} & \sigma^2_{t,t+1} \\ \sigma^2_{t,t+1} & \sigma^2_{t+1,t+1} \end{bmatrix}\right) \quad (12) \end{aligned}$$

where

$$\begin{cases} \hat{f}_i = f_0 + \mathbf{q}_i^\mathsf{T}(\mathbf{f}_{t-1} - \mathbf{f}_0) \\ \sigma^2_{i,j} = k_{i,j} - \mathbf{q}_i^\mathsf{T}\mathbf{k}_j \end{cases}$$

and $\mathbf{C}_{t-1} = \mathbf{K}_{t-1}^{-1}$, $\mathbf{q}_i = \mathbf{C}_{t-1}\mathbf{k}_i$ with $i, j = t, t+1$.

According to (8), the likelihood of the return dependent on the action value function is calculated:

$$p(R_t|f_t) = \mathcal{N}\left(R_t|f_t, \ \sigma_R^2\right). \quad (13)$$

Similar to the distribution of R_t, we also can obtain the likelihood of R_{t+1}

$$p(R_{t+1}|f_{t+1}) = \mathcal{N}\left(R_{t+1}|f_{t+1}, \ \sigma_R^2\right). \quad (14)$$

According to (1), $r_t = R_t - \gamma R_{t+1}$. Thus, the likelihood of reward r_t dependent on R_t, R_{t+1} can be calculated

$$p(r_t|R_t, R_{t+1}) = \mathcal{N}(r_t|R_t - \gamma R_{t+1}, \ \sigma_n^2). \quad (15)$$

Here it is assumed that the measured reward is perturbed by a zero-mean Gaussian noise $\mathcal{N}\left(0, \sigma_n^2\right)$.

The denominator of (10b), which corresponds to the marginal likelihood, provides the predictive distribution of a new observation r_t given the past samples. According to the characteristics of multivariate Gaussian distributions: the linear combinations, the marginal and the conditional distributions of Gaussian distributions are again Gaussian distributions. Hence, it can be calculated by combining (12), (13), (14) with (15).

$$\begin{aligned} &p(r_t|\mathbf{D}_{t-1}) \\ &= \int p(r_t|R_t, R_{t+1})p(R_t, R_{t+1}|\mathbf{f}_{t-1})p(\mathbf{f}_{t-1}|\mathbf{D}_{t-1})dRd\mathbf{f}_{t-1} \\ &= \mathcal{N}(r_t|\hat{r}_t, \hat{\sigma}^2_{r_t}) \end{aligned} \quad (16)$$

with the mean and the variance of the distribution being

$$\hat{r}_t = (1-\gamma)f_0 + \Delta\mathbf{k}_t^\mathsf{T}\mathbf{C}_t(\mathbf{f}_{t-1} - \mathbf{f}_0) \quad (17a)$$

$$\begin{aligned} \hat{\sigma}^2_{r_t} = \ & k_{t,t} + \gamma^2 k_{t+1,t+1} - 2\gamma k_{t,t+1} \\ & + \Delta\mathbf{k}_t^\mathsf{T}\mathbf{M}_{t-1}\Delta\mathbf{k}_t + \sigma_n^2 + (1+\gamma^2)\sigma_R^2 \end{aligned} \quad (17b)$$

where $\Delta\mathbf{k}_t = \mathbf{k}_t - \gamma\mathbf{k}_{t+1}$, $\mathbf{M}_{t-1} = \mathbf{C}_{t-1}\Sigma_{t-1}\mathbf{C}_{t-1} - \mathbf{C}_{t-1}$.

The state-action pair $x_t = (s_t, a_t)$ is added to the bias vector $\mathbf{x}_t = \mathbf{x}_{t-1} \cup x_t$ at the instant the new sample is collected. Combining (11) with (12), the predictive distribution of $f(x_t)$ given the collected samples can be calculated as

$$\begin{aligned} p(f_t|\mathbf{D}_{t-1}) &= \int p(f_t|\mathbf{f}_{t-1})p(\mathbf{f}_{t-1}|\mathbf{D}_{t-1})d\mathbf{f}_{t-1} \\ &= \mathcal{N}(f_t|\hat{f}_t, \ \hat{\sigma}^2_{\hat{f}_t}) \end{aligned} \quad (18)$$

with the mean and the variance being

$$\begin{cases} \hat{f}_t = f_0 + \mathbf{k}_t^\mathsf{T}\mathbf{C}_{t-1}(\mathbf{f}_{t-1} - \mathbf{f}_0) \\ \hat{\sigma}^2_{\hat{f}_t} = k_{t,t} + \mathbf{k}_t^\mathsf{T}\mathbf{M}_{t-1}\mathbf{k}_t \end{cases}.$$

After the new sample is incorporated into the bias vector, the inverse of the kernel matrix will be updated by

$$\mathbf{K}_t^{-1} = \mathbf{C}_t = \begin{bmatrix} \mathbf{C}_{t-1} & \mathbf{0} \\ \mathbf{0}^\mathsf{T} & 0 \end{bmatrix} + \frac{1}{\sigma^2_{\hat{f}_t}} \begin{bmatrix} \mathbf{q}_t \\ -1 \end{bmatrix} \begin{bmatrix} \mathbf{q}_t \\ -1 \end{bmatrix}^\mathsf{T}. \quad (19)$$

Therefore, all involved distributions (12), (13), (14) and (15) appearing in (10b) are Gaussian distributions. Based on these relations, the posterior distribution can be calculated after the new sample is collected, and expressed as

$$p(\mathbf{f}_t|\mathbf{D}_t) = \mathcal{N}\left(\boldsymbol{\mu}_t, \ \Sigma_t\right) \quad (20a)$$

$$\boldsymbol{\mu}_t = \begin{bmatrix} \boldsymbol{\mu}_{t-1} \\ \hat{f}_t \end{bmatrix} + \frac{r_t - \hat{r}_t}{\hat{\sigma}^2_{r_t}}\mathbf{h}_t \quad (20b)$$

$$\Sigma_t = \begin{bmatrix} \Sigma_{t-1} & \Sigma_{t-1}\mathbf{C}_{t-1}\mathbf{k}_t \\ \mathbf{k}_t^\mathsf{T}\mathbf{C}_{t-1}\Sigma_{t-1} & \hat{\sigma}^2_{\hat{f}_t} \end{bmatrix} - \frac{1}{\hat{\sigma}^2_{r_t}}\mathbf{h}_t\mathbf{h}_t^\mathsf{T} \quad (20c)$$

where $\mathbf{h}_t = \begin{bmatrix} \Sigma_{t-1}\mathbf{C}_{t-1}\Delta\mathbf{k}_t \\ k_{t,t} - \gamma k_{t,t+1} + \Delta\mathbf{k}_t^\mathsf{T}\mathbf{M}_{t-1}\mathbf{k}_t \end{bmatrix}$ is the covariance vector between $\mathbf{f}_t$ and r_t.

The update rule is depicted in (20b). It shows the action value function learns from the temporal difference $r_t - \hat{r}_t$, with the learning rate as $\mathbf{h}_t/\hat{\sigma}_{r_t}^2$. It means that the learning rate is determined by the uncertainty of the collected sample, different from the original SARSA learning algorithm [2] in which the learning rate is required to be determined in advance. Besides, it updates the action value function by Bellman residual [7] rather than predictions based on the bootstrap method [2]. Such an update rule makes an efficient use of the collected samples, which has also been verified in the method of least-square temporal difference learning [10], [11], [20].

When the first sample (x_1, r_1, x_2) is collected, we can get the joint distribution $p(f_1, f_2)$ based on the prior assumption of Gaussian processes. Consequentially, the recursion updates can be initialized

$$p(f_1|r_1) = \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1) \tag{21}$$

with

$$\boldsymbol{\mu}_1 = f_0 + \frac{r_1 - (1-\gamma)f_0}{\hat{\sigma}_{r_1}^2}(k_{1,1} - \gamma k_{1,2}) \tag{22a}$$

$$\Sigma_1 = k_{1,1} - \frac{1}{\hat{\sigma}_{r_1}^2}(k_{1,1} - \gamma k_{1,2})(k_{1,1} - \gamma k_{1,2})^\mathsf{T} \tag{22b}$$

$$\mathbf{C}_1 = \frac{1}{k(x_1, x_1)} \tag{22c}$$

where $\hat{\sigma}_{r_1}^2 = k_{1,1} - 2\gamma k_{1,2} + \gamma^2(k_{2,2} + \sigma_t^2) + \sigma_n^2 + (1+\gamma^2)\sigma_R^2$.

C. Sparsification

As mentioned above, the dimension of the bias vector of Gaussian processes grows at each time step when a new sample is incorporated. Unchecked, the growth would result in the unbounded increase in the computation and memory requirements. Engel *al. et.* [21] sparsified Gaussian processes by transforming the data into a high-dimensional reproducing kernel Hilbert space and thus data can be represented in a linear form. The data which are approximately linear dependent on the data in the bias vector will be removed, and so the representative data are retained in the bias vector. The criterion for approximately linear dependence is expressed as

$$\delta_t^2 = k_{t,t} - \mathbf{k}_t^\mathsf{T}\mathbf{C}_{t-1}\mathbf{k}_t. \tag{23}$$

And a threshold ν is used for the criterion. If $\delta_t^2 > \nu$, the new sample will be added into the bias vector. Otherwise, it can be removed with less information loss.

IV. Online Reinforcement Learning

The previous section presents the sequential update rule for the action value function from the collected samples via the way of Bayesian inference. Namely, the policy evaluation can be undertaken when a new sample is collected. It provides a way to implement the reinforcement learning in an online form. More than that, a reinforcement learning agent has to collect samples over the whole state-action space, in order to give a valid evaluation for the control policy. Failing to do that, the agent may never find the optimal control policy.

Here, the classic action selection method of ϵ-greedy is employed to explore an underlying system. It selects the action with the maximized action value function with a probability of $1-\epsilon$ and selects a random action uniformly in the action space with a probability of ϵ as shown in (24). The random action can prompt the agent to explore the inexperienced parts of the underlying system.

$$\pi(s_t) \leftarrow \begin{cases} \arg\max\limits_{a \in A} Q(s_t, a) & \text{with prob. } 1-\epsilon \\ a \in A & \text{with prob. } \epsilon \end{cases} \tag{24}$$

The value of ϵ is employed to manage the degree of the exploration. It diminishes during the interaction to ensure a sufficient exploration of the environment at the beginning and gradually shift to the exploitation as the collected samples increase, as it is shown in the following equation:

$$\epsilon = \epsilon_0 \epsilon_d^\kappa \tag{25}$$

where ϵ_d is the decayed factor. The parameter κ is used to determine the time of exploration decaying. The exploration rate may decay after a number of samples are collected, or after an episode is terminated.

The above content explicitly presents the approach of the policy evaluation by Bayesian inference and the diminishing ϵ-greedy action selection method. Integration of the two will form an online reinforcement learning algorithm, named Bayesian SARSA. It continuously updates the action value function for the behavior control policy and at the same time the policy updates toward greediness with respect to the action value function, meaning it is an on-policy algorithm.

V. Empirical Results

In this section Bayesian SARSA will be tested and demonstrated in two different tasks. To precisely illustrate its performance, the online version of least-square policy iteration (online LSPI) [10], [11] which can make an efficient use of the samples are brought in for comparisons. The algorithm is selected because it is also an on-policy and online model-free approach.

A. Mountain Car Task

The mountain car task is to drive a car up to a steep mountain road from the bottom of a valley [2]. The difficulty is that the car is underpowered to accelerate up the steep slope directly from the start position, and therefore a control policy should be learned to drive the car up as fast as possible. There are three actions $\{-1, 0, 1\}$, which respectively represents full throttle forward, zero throttle and full throttle backward. The reward is -1 for all time steps before the car moving past the goal position $p_{goal} = 0.5$, and the reward is 0 when the car reaches the goal. (26) is the state update of the car.

$$\begin{cases} p_{t+1} &= p_t + v_{t+1} \\ v_{t+1} &= v_t + 0.001a_t - 0.0025\cos(3p_t) \end{cases} \tag{26}$$

where p_t is the position of the car limited in the range of $[-1.2, 0.5]$ and v_t is the velocity limited in the range of $[-0.07, 0.07]$, a_t is the action. The car starts from the bottom

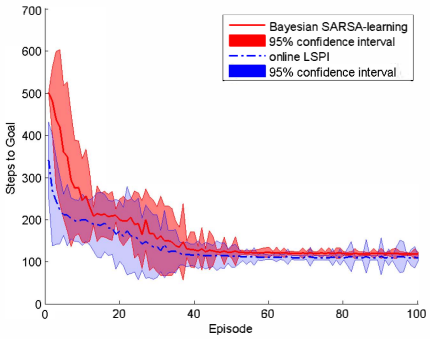

Fig. 1. Steps to the goal averaged over 30 trials and the corresponding 95% confidence interval.

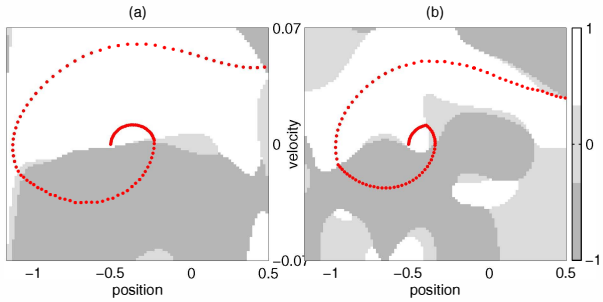

Fig. 2. Learned control policy at the end of one trial: (a) by Bayesian SARSA; (b) by online LSPI. The three different gray-scales represent the three actions, respectively. The red dotted lines represent the state trajectories from the start to the goal region.

of the valley with a state of $d_0 = -\pi/6$ and $v_0 = 0$ in each episode. When it gets to the left side, velocity is set to 0; when it gets to the right side, the goal is reached and the episode is terminated.

The kernel of the Gaussian processes is the SE function with $\alpha_0 = 1$ and $l_1 = 0.1$, $l_2 = 0.007$, the sparsification factor is $\nu = 10^{-3}$ and $\gamma = 0.99$. The radial bias function is chosen for online LSPI with $7 * 5$ centers distributed uniformly in the state space. In the simulation, each episode has a maximum time step of 500, and 100 episodes are taken in each trial. The initial exploration factor is $\epsilon_0 = 0.6$ and decays every episode with $\epsilon_d = 0.96$, which makes sure the exploration rate at the end of each trial is below 0.01.

The two different algorithms are applied to the mountain car task. The steps to the goal with respect to each episode averaged over 30 trials are shown in Fig. 1. Both Bayesian SARSA and online LSPI can search the goal within 5 episodes and learn within 50 episodes a near-optimal control policy which can drive the car up to the goal within 120 steps. These results argue that Bayesian SARSA can make the same efficient use of data with online LSPI. In addition, after 50 episodes the steps to the goal by Bayesian SARSA have smaller confidence intervals and less fluctuations than those by online LSPI, indicating it is more robust. This argument is also supported by the comparison of the corresponding control policies respectively learned by both algorithms, which is shown in Fig. 2. It depicts that Bayesian SARSA learns the near-optimal control policy which is almost available in the whole state space, while online LSPI learns that only available in the vicinity of the trajectory at the end of the trials.

B. Puddle World Task

Puddle world [22] is a two dimensional continuous grid world with two puddles. In the world, an agent moves along four different directions (East, South, West and North) with distance 0.05. It starts from an initial state $(0.2, 0.6)$ to search for a fast path to the goal region in the northeast corner while trying to avoid two puddles. The puddles are 0.1 in radius and located at the center points $(0.1, 0.75)$ to $(0.45, 0.75)$ and $(0.45, 0.4)$ to $(0.45, 0.8)$. The movements of the agent are perturbed by a Gaussian noise with a standard deviation of 0.01 along both dimensions. The reward in this problem is -1 until the agent reaches the goal region, and an additional penalty will be given if the agent trapped into the puddle. The value of the penalty is -200 times the distance to the nearest edge. When the agent reaches the goal region, the reward is 0 and the episode is terminated.

The kernel of the Gaussian processes is the SE function with $\alpha_0 = 1$ and $l_1 = 0.05$, $l_2 = 0.05$, the sparsification factor is $\nu = 10^{-3}$ and $\gamma = 0.98$. The online LSPI algorithm selects radial bias functions as the bias functions with $9 * 9$ centers distributed uniformly in the state space. In the experiment, each episode has a maximum time step of 200, and 100 episodes are taken in a trial. The parameter settings for the ϵ-greedy method are the same as the ones in mountain car task.

Both algorithms are tested on this task and the steps from the start to the goal region averaged over 30-trials are shown in Fig. 3. Bayesian SARSA performs much better than online LSPI in both the steps to the goal region and the corresponding confidence interval. Some significant fluctuations exist in the confidence interval as the actions are perturbed by noises. Taking a look inside, Bayesian SARSA can learn a near-optimal control policy at the end of each trial, while online LSPI successes within only 1/5 of the trials. This stems from the fact that a sudden change happened in the value of reward results in the lost of experience for online LSPI, when the agent 'traps' into the puddle. While Bayesian SARSA can avoid the sudden changes as it incorporates a prior information and learns from the temporal difference.

Furthermore, a comparison of the two learned control policies is shown in Fig. 4. The first one is randomly selected from the trials by Bayesian SARSA and the second one is selected from the trials which has successfully reached the goal region by online LSPI. Similar to the results of the mountain car task, the control policy learned by Bayesian SARSA is almost near-optimal in the whole state space, while the one learned by online LSPI is only available in the vicinity of the trajectory.

VI. Conclusions

In this paper, a Bayesian SARSA algorithm is proposed to study the policy evaluation issue of the reinforcement learning in the continuous state domain from the probability perspective. It is implemented by modeling the action value function as a stochastic variable via Gaussian processes and then updating it according to Bayesian inference. The algorithm was tested and demonstrated on two different benchmark

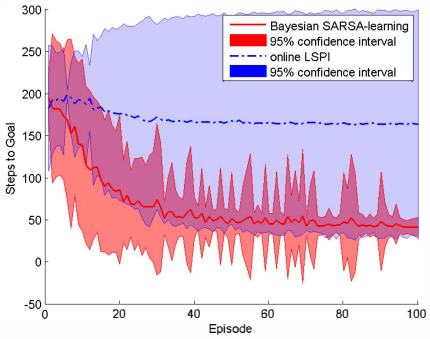

Fig. 3. Steps to the goal averaged over 30 trials and the corresponding confidence interval.

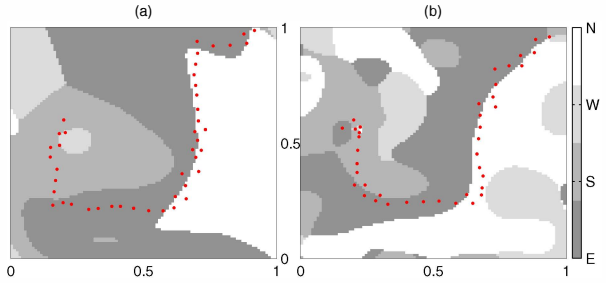

Fig. 4. Learned control policy at the end of one trial: (a) by Bayesian SARSA; (b) by online LSPI. The four different gray-scales represent the four actions, respectively. The red dotted lines represent the state trajectories from the start to the goal region.

problems by comparing with online LSPI. Empirical results show that Bayesian SARSA can make an efficient use of data and learn fast the near-optimal control policy. Moreover, it outperforms the online LSPI algorithm in the performance of robustness and optimality.

In order to learn an optimal control policy, a reinforcement learning agent always has to explore the uncertainty part of the state-action space. As mentioned previously, Bayesian SARSA also learns the uncertainty of each state-action pair, not just the action value function. Hence, how to efficiently explore an underlying system according to such an uncertainty will be a promising research area in the future work.

REFERENCES

[1] F. L. Lewis and D. Vrabie, "Reinforcement learning and adaptive dynamic programming for feedback control," *IEEE Circuits and Systems Magazine*, vol. 9, no. 3, pp. 32–50, 2009.

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

[3] F. Wang, H. Zhang, and D. Liu, "Adaptive dynamic programming: an introduction," *IEEE Computational Intelligence Magazine*, vol. 4, no. 2, pp. 39–47, 2009.

[4] D. Liu, D. Wang, and H. Li, "Decentralized stabilization for a class of continuous-time nonlinear interconnected systems using online learning optimal control approach," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 2, pp. 418–428, 2014.

[5] C. Dann, G. Neumann, and J. Peters, "Policy evaluation with temporal differences: A survey and comparison," *Journal of Machine Learning Research*, vol. 15, pp. 809–883, 2014.

[6] D. Zhao and Y. Zhu, "MEC—a near-optimal online reinforcement learning algorithm for continuous deterministic systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 2, pp. 346 – 356, 2014.

[7] D. Zhao, Z. Xia, and D. Wang, "Model-free optimal control for affine nonlinear systems with convergence analysis," *IEEE Transactions on Automation Science and Engineering*, no. 99, pp. 1–8, 2014.

[8] D. Zhao, Z. Hu, Z. Xia, C. Alippi, Y. Zhu, and D. Wang, "Full-range adaptive cruise control based on supervised adaptive dynamic programming," *Neurocomputing*, vol. 125, pp. 57–67, 2014.

[9] D. Zhao, X. Bai, F.-Y. Wang, J. Xu, and W. Yu, "DHP method for ramp metering of freeway traffic," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 990–999, 2011.

[10] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[11] L. Busoniu, D. Ernst, B. De Schutter, and R. Babuska, "Online least-squares policy iteration for reinforcement learning control," in *Proceedings of American Control Conference*, 2010, pp. 486–491.

[12] Z. Xia, D. Zhao, and H. Tang, "Model-free adaptive dynamic programming for optimal control of discrete-time affine nonlinear system," in *Proceedings of International Federation of Automatic Control World Congress*, South Africa, 2014, pp. 7049–7054.

[13] D. Ormoneit and Ś. Sen, "Kernel-based reinforcement learning," *Machine learning*, vol. 49, no. 2-3, pp. 161–178, 2002.

[14] N. Jong and P. Stone, "Kernel-based models for reinforcement learning," in *ICML Workshop on Kernel Machines and Reinforcement Learning, Pittsburgh, PA, USA*, 2006.

[15] C. E. Rasmussen, *Gaussian processes for machine learning*. Citeseer, 2006.

[16] C. E. Rasmussen and M. Kuss, "Gaussian processes in reinforcement learning," in *Advances in Neural Information Processing Systems 16*. MIT Press, 2004, pp. 751–758.

[17] M. P. Deisenroth, *Efficient reinforcement learning using gaussian processes*. KIT Scientific Publishing, 2010.

[18] T. Jung and P. Stone, "Gaussian processes for sample efficient reinforcement learning with RMAX-like exploration," in *Machine Learning and Knowledge Discovery in Databases*. Springer, 2010, pp. 601–616.

[19] Y. Engel, S. Mannor, and R. Meir, "Reinforcement learning with gaussian processes," in *Proceedings of International Conference on Machine learning*. ACM, 2005, pp. 201–208.

[20] J. A. Boyan, "Least-squares temporal difference learning," in *Proceedings of International Conference on Machine learning*. Citeseer, 1999, pp. 49–56.

[21] Y. Engel, S. Mannor, and R. Meir, "The kernel recursive least-squares algorithm," *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.

[22] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," in *Advances in Neural Information Processing Systems 8*. MIT Press, 1996, pp. 1038–1044.