Online $\ell 2$-regularized Reinforcement Learning via RBF Neural Network

Tianheng Song, Dazi Li*

School of Information Science and Technology, Beijing University of Chemical Technology, Beijing 100029
E-mail: lidz@mail.buct.edu.cn

Abstract: An $\ell 2$-regularized policy evaluation algorithm, termed RRC (Regularized RC), is proposed for applying in the reinforcement learning problems. RBF network is used to construct VFA, and its weight vector is solved based on RC algorithm. An additional recursive step is used to achieve a different effect from traditional recursive least-square-based $\ell 2$-regularized algorithm: the regularization term does not decrease throughout learning. Additionally, a fast counterpart algorithm with $O(n^2)$ complexity is also proposed, termed as Fast RRC (FRRC), which is more practical online algorithm than RRC. The convergence analysis and experiments results demonstrate the significant performances of RRC and FRRC.

Key Words: RBF neural networks, Reinforcement Learning, Polity Evaluation, Regularization

1 INTRODUCTION

Reinforcement learning (RL) methods contain a series of approximation algorithms for problems involving Markov decision processes (MDP) [1]. The two main tasks of RL [2] are policy evaluation, aiming to solve the value function as an evaluation of a certain policy, and control learning, aiming to find the optimal policy. Since policy evaluation has the basic position of RL, many researches focus on high effective policy evaluation algorithm [3]-[5] in the last two decades.

To solve large-scale or continuous MDP problems, the value function approximation (VFA) is used to construct the mapping from state to value function. Three-layer neural networks are desirable nonlinear mapping functions for constructing VFA [6]. However, the training methods for neural network-based VFA in RL are different from those in supervised learning problems. Least-squares-based and gradient-based temporal difference (TD) learnings are two effective series of algorithms to solve the linear parameters of this kind of VFA [7]. Furthermore, like in other machine learning methods, regularization in RL algorithms [8]-[10] can prevent over-fitting and improve generalization performance, especially with the sample noise and limited number of samples. However, in traditional recursive least-square-based TD methods, the regularization term decreases throughout learning, and thus the VFA parameters are unstable although the objective function converges.

To solve this problem, in this paper we focus on improving the online $\ell 2$-regulazied recursive least-square-based TD method. We proposed a different regularization procedure from other RLS-based method, and constructed the regularized RC algorithm (RRC). RRC can sustain the regularization effect throughout learning process. The VFA are constructed by an RBF neural network. Additionally, the convergence analysis, empirical results and some extension of RRC are also proposed.

The rest of paper is organized as follows. In Section 2 the RBF-based VFA is proposed. RRC algorithm and the extensions are proposed in Section 3. The convergence analysis is given in Section 4. The empirical results are presented in Section 5. Finally we conclude this paper in Section 6.

2 RBF-BASED VFA

For a brief expression, only finite MDPs are considered in this paper. However, the following analyses and algorithms can also be generalized in the cases of continuous MDPs.

A finite MDP is defined as a tuple (S, A, P_0, R, γ), where S is a finite set of states; A is a finite set of actions; P: $S\times A\times S\rightarrow[0,1]$ is a state transition kernel matrix; R is a reward function; and γ is a discounted factor. The state transitions in the MDP are determined by a stationary policy $\pi: S\rightarrow A$. The value function $V^\pi(s)$ is used to estimate the expected discounted sum of the future rewards in each state s:

$$V^\pi(s)=\mathbb{E}\left[\sum_{i=0}^{\infty}\gamma^i r_i \,\middle|\, s_0=s,\pi\right], \tag{1}$$

The vector V^π has $|S|$ dimensions, and should be a fixed point of the following Bellman equation:

$$V^\pi=T^\pi V^\pi=R+\gamma P^\pi V^\pi \tag{2}$$

The superscript π denotes that the matrix or vector are determined under the policy π, and we drop it for notational simplicity without ambiguity.

To deal with problems in large state spaces, a certain structured VFA is needed for learning and estimating the value function V^π. A three-layer RBF neural network is chosen as the VFA in this research, because RBF networks have good approximation performance and simple linear

This work is supported by National Nature Science Foundation under Grant 61573052

learning structure for deriving and analyzing. The input layer has the identical dimension to that of state vector s, used to mapping the current state directly to the hidden layer. Some artificially selected hidden nodes construct the hidden layer, and a bias node is added:

$$\phi(s)=\left[1,\exp\left(\frac{\|s-c_1\|}{-2\sigma^2}\right),\exp\left(\frac{\|s-c_2\|}{-2\sigma^2}\right),\cdots\exp\left(\frac{\|s-c_n\|}{-2\sigma^2}\right)\right] \tag{3}$$

where $c_i(i=1,2,\cdots,n)$ is the RBF center, and σ is the width parameter. The output vector of the hidden layer is denoted as $\phi(s)$, which provides the non-linear features mapping from the input space to the feature space.

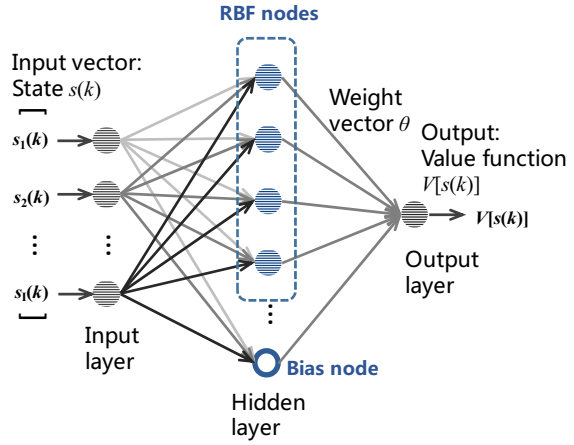

Fig 1. Construction of RBF-based VFA.

The output matrix of the hidden layer $\Phi(s)\in\mathbb{R}^{|S|\times n}$ is formed with all the ϕ^{T} for each state as each row. Defining the output linear weight vector as θ, the RBF-based VFA in matrix form can be obtained as follows:

$$V^{\pi}\approx V_{\theta}=\Phi\theta=\left[\phi^{\mathrm{T}}(s_1);\phi^{\mathrm{T}}(s_1);\cdots;\phi^{\mathrm{T}}(s_1)\right]\theta \tag{4}$$

The training process of the RBF network contains determining the centers c_i and width parameter σ in the hidden layer, and solving the weight vector θ to the output layer. The centers c_i can be determined using some unsupervised learning methods, such like k-means, or even selected manually in simple problems. As a single scalar, an appropriate σ can be determined by 1-D search. However, how to solve θ is main task. Least-square-based TD method is used to find θ as the TD fixed point, instead of error back propagation and pseudo-inverse methods in supervised learning.

3 REGULARIZED RC ALGORITHM

In this section we propose a technique to solve the weight vector θ in reinforcement learning, based on the RBF-based VFA mentioned in Section 2 and RC algorithm. RC algorithm (recursive least squares temporal difference with gradient correction) is a recursive version of LS-TDC algorithm [11], combining LSTD and TDC, and aims to minimize the following objective function termed MSPBE [12]:

$$\mathrm{MSPBE}(\theta)=\|V_{\theta}-\Pi TV_{\theta}\|_D^2 \tag{5}$$

As a gradient-descent algorithm, RC has the gradient residual equations as follows:

$$\begin{cases}(\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}+\varepsilon I)\theta=\tilde{\Phi}^{\mathrm{T}}(\gamma\tilde{\Phi}'u+\tilde{R})-\gamma\tilde{\Phi}'^{\mathrm{T}}\tilde{\Phi}\omega\\(\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}+\varepsilon I)\omega=\tilde{\Phi}^{\mathrm{T}}(\tilde{R}+\gamma\tilde{\Phi}'\theta-\tilde{\Phi}\theta)\end{cases} \tag{6}$$

The auxiliary modifiable parameter vector ω is used to estimate TD-error. The current and next time-step sample matrix $\tilde{\Phi}$ and $\tilde{\Phi}'$ are consisted as a sufficient observation sample trajectory of m samples, with the format of (s_i,r_i,s_i'), $i=1,2,\cdots,m$. $\tilde{\Phi}$ and $\tilde{\Phi}'$ are used to estimate the terms containing Φ and Φ'. these sample matrices and vectors are defined as follows:

$$\tilde{\Phi}\equiv\begin{bmatrix}\phi_1^{\mathrm{T}}\\\vdots\\\phi_m^{\mathrm{T}}\end{bmatrix},\tilde{\Phi}'\equiv\begin{bmatrix}\phi_1'^{\mathrm{T}}\\\vdots\\\phi_m'^{\mathrm{T}}\end{bmatrix}\text{ and }\tilde{R}\equiv\begin{bmatrix}r_1\\\vdots\\r_m\end{bmatrix} \tag{7}$$

The small nonnegative number ε in formula (6) is the regularization parameter. Many least-squares-based TD algorithms, such as LSTD, LSPE and RC, use ε to ensure the state matrix invertible. Another purpose of ε is to realize $\ell 2$-regularization. However, the state matrix sustains increasing throughout learning, whereas the regularization term εI is constant. Hence, the effect of regularization is asymptotically decreasing to zero with learning process. As the result, the norm of θ keeps increasing until the number of samples amount to large enough, and the effect of regularization barely exist afterwards.

A straight-forward method is proposed to sustain the effect of regularization throughout the learning process. That is, we change the joint linear system in formula (6) as

$$\begin{bmatrix}\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}-\gamma\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}' & \gamma\tilde{\Phi}'^{\mathrm{T}}\tilde{\Phi}\\\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}-\gamma\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}' & \tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}\end{bmatrix}\rho+\sum_{i=1}^{k}\bar{\varepsilon}I\rho=\begin{bmatrix}\tilde{\Phi}^{\mathrm{T}}\tilde{R}\\\tilde{\Phi}^{\mathrm{T}}\tilde{R}\end{bmatrix}, \tag{8}$$

where $\rho=[\theta;\omega]$. The only difference between formula (8) and (6) is that in every time-step a regularization diagonal matrix is added to the state matrix. The solution θ can be obtain by solved the linear system of formula (8) with computational complexity $O(n^3)$. However, the recursive counterpart with $O(n^2)$ complexity in each step is more practical for the online learning problems. The following lemma is a common technique to realize recursive matrix inverse:

Lemma 1 (Sherman-Morrison): assuming that matrix A is invertible and that the vectors u and v satisfy $1+v^{\mathrm{T}}A^{-1}u\neq 0$, then:

$$(A+uv^{\mathrm{T}})^{-1}=A^{-1}-A^{-1}u(1+v^{\mathrm{T}}A^{-1}u)^{-1}v^{\mathrm{T}}A^{-1}. \tag{9}$$

The incremental definition of formula (8) can be rewritten as

$$\begin{cases}\tilde{G}_k\equiv\sum_{i=1}^{k}\left(\begin{bmatrix}\phi_i(\phi_i-\gamma\phi_i')^{\mathrm{T}} & \gamma\phi_i'\phi_i^{\mathrm{T}}\\\phi_i(\phi_i-\gamma\phi_i')^{\mathrm{T}} & \phi_i\phi_i^{\mathrm{T}}\end{bmatrix}+\bar{\varepsilon}I\right)\\\tilde{h}_k\equiv\sum_{i=1}^{k}\begin{bmatrix}r_i\phi_i\\r_i\phi_i\end{bmatrix}\end{cases}. \tag{10}$$

The inverse P_k of state matrix $\tilde{G}_k$ is defined as

$$P_k \equiv \tilde{G}_k^{-1} = \Bigg\{ P_{k-1} + \underbrace{\begin{bmatrix}\phi_k \\ \phi_k\end{bmatrix}}_{u_{1,k}} \underbrace{\begin{bmatrix}(\phi_k - \gamma\phi_k')^{\mathrm{T}} & \mathbf{0}\end{bmatrix}}_{v_{1,k}^{\mathrm{T}}} + \underbrace{\begin{bmatrix}\gamma\phi' \\ \phi_k\end{bmatrix}}_{u_{1,k}} \underbrace{\begin{bmatrix}\mathbf{0} & \phi_k^{\mathrm{T}}\end{bmatrix}}_{v_{1,k}^{\mathrm{T}}} + \underbrace{\bar{\varepsilon}\mathcal{I}_{(1)}\mathcal{I}_{(1)}^{\mathrm{T}} + \bar{\varepsilon}\mathcal{I}_{(2)}\mathcal{I}_{(2)}^{\mathrm{T}} + \cdots + \bar{\varepsilon}\mathcal{I}_{(2n)}\mathcal{I}_{(2n)}^{\mathrm{T}}}_{\bar{\varepsilon}I} \Bigg\}^{-1} . \quad (11)$$

Here G_k in the kth time-step has been decomposed as the sum of a series of vector products, and in this case the inverse P_k can be solved by using Lemma 1 repeatedly. In RC algorithm only the first two vector products of G_k are considered, and the recursive computation procedure is shown as follows:

$$\begin{cases} \Psi_{1,k} = P_{k-1}u_{1,k} \\ K_{1,k} = \Psi_{1,k}(v_{1,k}^{\mathrm{T}}P_{k-1}) / (1 + v_{1,k}^{\mathrm{T}}\Psi_{1,k}) . \\ D_k = P_{k-1} - K_{1,k} \end{cases} \quad (12)$$

$$\begin{cases} \Psi_{2,k} = D_k u_{2,k} \\ K_{2,k} = \Psi_{2,k}(v_{2,k}^{\mathrm{T}}D_k) / (1 + v_{2,k}^{\mathrm{T}}\Psi_{2,k}) \\ P_k = D_k - K_{2,k} \end{cases} \quad (13)$$

A $2n$-step loop is employed to continue to deal with the last $2n$ terms in formula (11). In the jth step, the inverse $P_{k,(j)}$ containing the first j terms of $\bar{\varepsilon}I$ is solved as

$$\begin{aligned} P_{k,(j)} &= P_{k,(j-1)} - P_{k,(j-1)}\bar{\varepsilon}\mathcal{I}_{(j)}(1 + \mathcal{I}_{(j)}^{\mathrm{T}}P_{k,(j-1)}\bar{\varepsilon}\mathcal{I}_{(j)})^{-1}\mathcal{I}_{(j)}^{\mathrm{T}}P_{k,(j-1)} \\ &= P_{k,(j-1)} - \bar{\varepsilon}P_{k,(j-1)}(\cdot, j)(1 + \bar{\varepsilon}P_{k,(j-1)}(j, j))^{-1}P_{k,(j-1)}(j, \cdot) \end{aligned} \quad (14)$$

In online problems, the current parameter vector can be solved as $\rho_k = P_{k,(2n)}\tilde{h}_k$.

We call this algorithm as $\ell 2$-regularized RC (also as RRC). The procedure of RRC is presented as follows:

Algorithm 1 RRC

Initialization:

1. Parameters and samples:

$\bar{\varepsilon} \in \mathbb{R}_+$: regularization parameter

$\gamma \in [0,1]$: discounted factor

$\{c_i\}, \sigma$: centers and width of RBF network

$\{s_i, r_i, s_i'\}_{i=i}^{m}$: state transition and reward samples

2. Variables:

Set $\rho_k = \mathbf{0}$ and $\tilde{h}_0 = \mathbf{0}$

Set $\tilde{G}_0 = \bar{\varepsilon}I$

1: **Repeat**

2: Compute $u_{1,k}$, $u_{2,k}$, $v_{1,k}^{\mathrm{T}}$, and $v_{2,k}^{\mathrm{T}}$

3: Compute P_k as in formula (12) and (13)

4: **Repeat**

5: Compute $P_{k,(j)}$ as in formula (14)

6: Set $j \leftarrow j+1$

7: **Until** $j = 2n$

8: Compute $\rho_k = P_{k,(2n)}\tilde{h}_k$

9: Set $k \leftarrow k+1$

10: **Until** $k = m$

Return ρ_m (containing vector θ_m)

It can be seen from Algorithm 1 that the regularization loop from line 4 to 7 increases the computation highly. Accurately, this loop has $2n$ iterations that equals the number of hidden nodes of RBF, and each iteration contains an $O(n^2)$-complexity vector product. Hence, the computation complexity of this loop is $O(n^3)$. However, the complexity can still be reduced to $O(n^2)$ under one common assumption; that is, the number of samples is much larger than the number of hidden nodes. Accordingly, when a new sample comes, only one regularization term of one dimension is added:

$$P_k \equiv \tilde{G}_k^{-1} = \left\{ P_{k-1} + u_{1,k}v_{1,k}^{\mathrm{T}} + u_{2,k}v_{2,k}^{\mathrm{T}} + \bar{\varepsilon}\mathcal{I}_{(p)}\mathcal{I}_{(p)}^{\mathrm{T}} \right\}^{-1} . \quad (15)$$

And P_k can be obtained by solving formula (14) only once:

$$P_k = P_{k-1} - \bar{\varepsilon}P_{k-1}(\cdot, p)(1 + \bar{\varepsilon}P_{k-1}(p, p))^{-1}P_{k-1}(p, \cdot) \quad (16)$$

The periodic variable p changes from 1 to $2n$ in turn, and at the beginning p is initialized as 1. In this way, the regularization loop in Algorithm 1 (line 4 to 7) can be replaced by the following pseudo-code:

Pseudo-code: fast regularization step

1: Compute P_k as in formula (16)

2: Set $p \leftarrow p+1$

3: **If** ($p > 2n$), $p \leftarrow 1$

With this alternative step, Algorithm 1 becomes to be with an $O(n^2)$ complexity. This variant RRC is termed Fast RRC (FRRC) in our following analysis and experiments.

The idea proposed in this section to achieve a real online $\ell 2$-regularization can also be extended to other recursive least-square-based algorithms, such as the standard RLSTD and LSPE. With the similar derivation, the updating representation of the new online $\ell 2$-regularized RLSTD is given as follows:

$$\begin{cases} P_{k+\frac{1}{2}} = P_k - P_k\phi_i(\phi_i - \gamma\phi_i')^{\mathrm{T}}P_k / [1 + (\phi_i - \gamma\phi_i')^{\mathrm{T}}P_k\phi_i] \\ P_{k+1,(j)} = P_{k,(j-1)} - \bar{\varepsilon}P_{k+\frac{1}{2}}(\cdot, j)(1 + \bar{\varepsilon}P_{k+\frac{1}{2}}(j, j))^{-1}P_{k+\frac{1}{2}}(j, \cdot) \\ \theta_{k+1} = P_{k+1,(j)}\sum_{i=1}^{k} r_i\phi_i \\ P_{k+1} = P_{k+1,(j)} \end{cases} . \quad (17)$$

Note that although a faster $O(n^2)$ algorithm is obtained, the error emerges from the regularization of FRRC, because the replacement from formula (14) to (16) is only an

approximation. However, in the next two sections we show the convergence analysis and experiments to demonstrate that the error is negligible, when a large number of samples are possessed.

4 CONVERGENCE ANALYSIS

RRC and FRRC have the identical assumptions to LSTD for convergence. Considering that a RBF network is used the VFA, these assumptions are summarized as follows:

Assumption 1 (MDP) [12], [13]: the underlying policy is stationary; the MDP is finite and mixing; and the expectation of the reward function is bounded.

Assumption 2 (RBF): the output matrix of the hidden layer Φ is full rank; and for each state, $\mathbb{E}[\phi_k^2]$ is bounded.

Assumption 3 (Sherman-Morrison): all of the recursive inversing equations satisfy $1+v^{\mathrm{T}}A^{-1}u \neq 0$ in denominators.

Under these assumptions, RC algorithm converges to the unique TD fixed point with probability one. That is

$$\lim_{k\to\infty}\rho_k = \lim_{k\to\infty}\begin{bmatrix}\theta_k\\ \omega_k\end{bmatrix} = \begin{bmatrix}\theta^*\\ \mathbf{0}\end{bmatrix}. \tag{18}$$

However, the RRC and FRRC have different convergence to a different fixed point from RC, LSTD and other recursive least-squares-based algorithms.

We denote the four partitioned matrices of $\tilde{G}_k$ in formula (10) as $\tilde{A}_k$, $\tilde{B}_k$, $\tilde{C}_k$, and $\tilde{D}_k$, and their limits when $k\to\infty$ are calculated as

$$\begin{cases}\lim\limits_{k\to\infty}k^{-1}\tilde{A}_k = \lim\limits_{k\to\infty}\left[\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}-\gamma\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}'\right]+\lim\limits_{k\to\infty}k^{-1}\sum\limits_{i=1}^{k}\bar{\varepsilon}I\\ \qquad\qquad = \Phi^{\mathrm{T}}D\Phi-\gamma\Phi^{\mathrm{T}}DP\Phi'+\varepsilon I\\ \lim\limits_{k\to\infty}k^{-1}\tilde{B}_k = \lim\limits_{k\to\infty}\left[\gamma\tilde{\Phi}'^{\mathrm{T}}\tilde{\Phi}\right]\\ \lim\limits_{k\to\infty}k^{-1}\tilde{C}_k = \lim\limits_{k\to\infty}\left[\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}-\gamma\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}'\right]\\ \qquad\qquad = \lim\limits_{k\to\infty}k^{-1}\tilde{A}_k-\varepsilon I\\ \lim\limits_{k\to\infty}k^{-1}\tilde{D}_k = \lim\limits_{k\to\infty}\left[\tilde{\Phi}^{\mathrm{T}}\tilde{\Phi}\right]+\lim\limits_{k\to\infty}k^{-1}\sum\limits_{i=1}^{k}\bar{\varepsilon}I\\ \qquad\qquad = \Phi^{\mathrm{T}}D\Phi+\varepsilon I\end{cases} \tag{19}$$

The four limits in formula (19) are denoted as $\tilde{A}_\infty$, $\tilde{B}_\infty$, $\tilde{C}_\infty$, and $\tilde{D}_\infty$. The limit value of $\tilde{P}_\infty$ is then solved by using the rule for partitioned matrix inversion as follows:

$$P_\infty\tilde{h}_\infty \equiv \tilde{G}_\infty^{-1}\tilde{h}_\infty = \begin{bmatrix}\theta_\infty\\ \omega_\infty\end{bmatrix}$$
$$= \begin{bmatrix}\tilde{A}_\infty^{-1}H_\infty+\tilde{A}_\infty^{-1}\tilde{B}_\infty(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}\tilde{C}_\infty\tilde{A}_\infty^{-1}H_\infty\\ \qquad -\tilde{A}_\infty^{-1}\tilde{B}_\infty(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}H_\infty\\ -(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}\tilde{C}_\infty\tilde{A}_\infty^{-1}H_\infty+(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}H_\infty\end{bmatrix}. \tag{20}$$

Here $H_\infty = \lim\limits_{k\to\infty}k^{-1}\tilde{H}_k = \lim\limits_{k\to\infty}\left[\tilde{\Phi}\tilde{R}\right]$ is the limit value of both partitioned matrices of $\tilde{h}_\infty$. Now we focus on the weight vector θ_∞:

$$\begin{aligned}\theta_\infty &= \tilde{A}_\infty^{-1}H_\infty+\tilde{A}_\infty^{-1}\tilde{B}_\infty(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}(\tilde{C}_\infty\tilde{A}_\infty^{-1}H_\infty-H_\infty)\\ &= \tilde{A}_\infty^{-1}H_\infty+\underbrace{\tilde{A}_\infty^{-1}\tilde{B}_\infty(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}}_{M}(\tilde{C}_\infty\tilde{A}_\infty^{-1}H_\infty-H_\infty)\\ &= \tilde{A}_\infty^{-1}H_\infty+M(\tilde{C}_\infty\tilde{A}_\infty^{-1}H_\infty-H_\infty)\end{aligned} \tag{21}$$

where M is a bounded matrix. The first term $\tilde{A}_\infty^{-1}H_\infty$ is the unbiased evaluator of solution of the following objective function:

$$f(\theta^{**}) = \arg\min_u\frac{1}{2}\left\|\tilde{\Phi}u-(\tilde{R}+\gamma\tilde{\Phi}')\right\|^2+\bar{\varepsilon}\|u\|_2^2 \tag{22}$$

The solution θ^{**} is different from θ^*, because θ^* is the solution of other recursive least-square-based algorithms:

$$f(\theta^{*}) = \arg\min_u\frac{1}{2}\left\|\tilde{\Phi}u-(\tilde{R}+\gamma\tilde{\Phi}')\right\|^2+\lim_{k\to m}\frac{1}{k}\bar{\varepsilon}\|u\|_2^2 \tag{23}$$

That is why RRC and FRRC can sustain the regularization effect throughout the whole learning process.

Therefore, θ_∞ converges to the fixed point of formula (22):

$$\begin{aligned}\theta_\infty &= \tilde{A}_\infty^{-1}H_\infty+M(\tilde{C}_\infty\tilde{A}_\infty^{-1}H_\infty-H_\infty)\\ &= \theta^{**}+M(\tilde{C}_\infty\theta^{**}-H_\infty)\\ &\approx \theta^{**}+\mathbf{0}\end{aligned} \tag{24}$$

In addition, the auxiliary modifiable parameter vector ω also converges:

$$\begin{aligned}\omega_\infty &= -(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}\tilde{C}_\infty\tilde{A}_\infty^{-1}H_\infty+(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}H_\infty\\ &= \underbrace{(\tilde{D}_\infty-\tilde{C}_\infty\tilde{A}_\infty^{-1}\tilde{B}_\infty)^{-1}}_{N}(-\tilde{C}_\infty\theta^{**}+H_\infty)\\ &= N(\tilde{C}_\infty\theta^{**}-H_\infty)\\ &\approx \mathbf{0}\end{aligned} \tag{25}$$

Hence the limit value $\omega_\infty = \mathbf{0}$, identically to the conclusion in RC algorithm.

5 EMPIRICAL RESULTS

We choose a standard 20-state chain walk problem to demonstrate the different regularization performances between the proposed algorithm and others. This MDP consist 20 states, two action (go left and right), and the reward of one given at both end of the chain. Other details can be found in [14]. In both of the following experiments, 10000 samples are generated in a run under the independent identically-distributed sampling method (see [15]), and the learning process is set as an on-policy learning.

5.1 Comparison between RRC and RC

The first experiment aims to show difference regularization performance between RRC and RC. We chose 11 centers of the RBF network, being spread uniformly over the whole chain, and the width is set as 7 by 1-D search. The

regularization parameter of RC is set as 10^{-1}, and that of RRC is 0.5×10^{-4}.

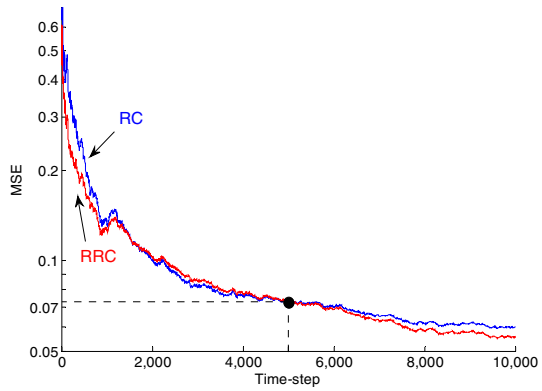

Fig 2. Comparison of MSEs of RRC and RC in 20-state chain walk MDP with 12 RBF hidden nodes.

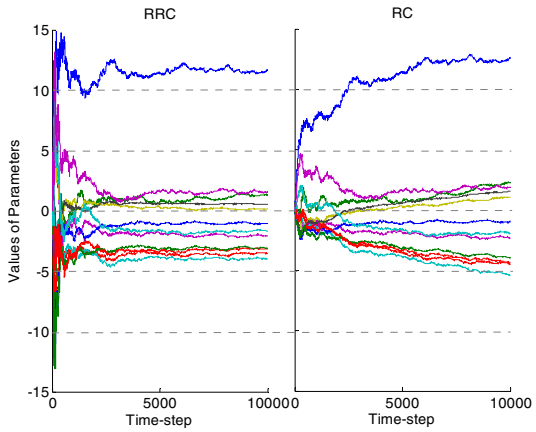

Fig 3. Comparison of parameters of RRC and RC in 20-state chain walk MDP with 12 RBF hidden nodes.

In Figure 2, the two algorithms demonstrates obvious different parameter convergent trends: the norm of RC's parameters sustains increasing, whereas that of RRC's is becoming stable. In Figure 3, the two MSE curves intersect at time-step 5000, the midpoint of the whole learning, because at time-step 5000 both algorithm has the same current regularization parameter 0.5×10^{-4} (note that RRC's is constant). Form the result of this comparison, it can be seen that RRC can sustain regularizing the parameter vector throughout the learning and converge to the regularized TD fixed point expected, whereas RC is gradually losing regularization effect and only converges to the original TD fixed point.

5.2 Comparison between RRC, FRRC and R-RLSTD

In the second experiment, we compare the performance of the three algorithms proposed in this paper. For a brief and clear result of comparison, in this experiment only 5 centers of the RBF are used, which spread uniformly over the whole chain. Then the dimension of the weight vector is 6 with 5 centers and a constant bias. The other settings are identical to those in the first experiment.

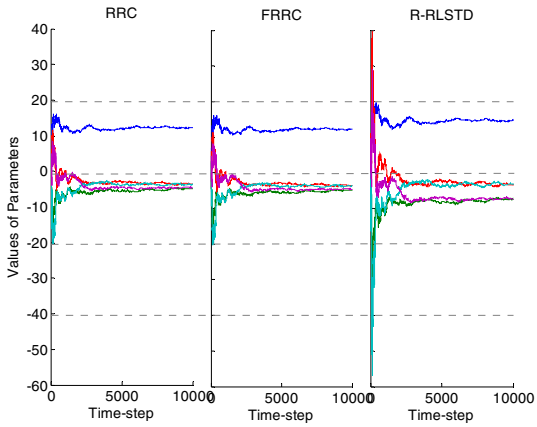

Fig 4. Comparison of parameters of RRC, FRRC and R-RLSTD in 20-state chain walk MDP with 5 RBF hidden nodes.

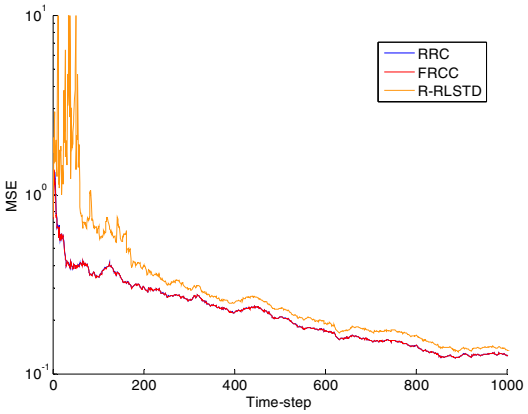

Fig 5. Comparison of parameters of RRC, FRRC and R-RLSTD in 20-state chain walk MDP with 5 RBF hidden nodes.

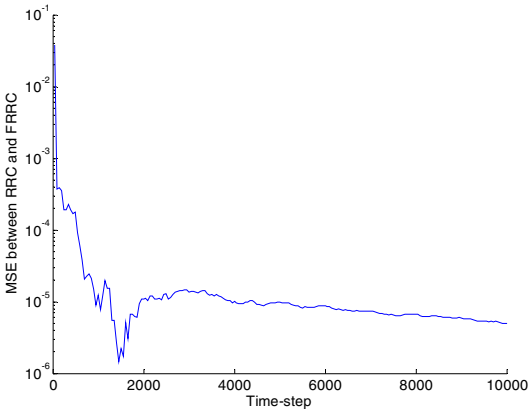

Fig 6. The 2-norm between MSE curves of RRC and FRRC in Figure 5.

Figure 4 shows the curves of parameters of these three algorithms. It can be seen that all of them sustains the regularization effect throughout learning. Further, RRC and FRRC have almost the identical convergent curves of each parameter. However, R-RLSTD is less stable than these two RC-based algorithms, especially in the beginning stage of learning. The MSEs are compared in Figure 5, where we can also see that the MSE curves of RRC and FRRC almost

intersect, and that of R-RLSTD has much high amplitudes in the beginning stage of learning. The 2-norm between MSE curves of RRC and FRRC is shown in Figure 6. This norm decreases fast, and the final value is very small. These results demonstrate that the performance of FRRC is extremely close to that of RRC.

Accordingly, the experiment results are summarized as follows:

RRC and FRRC can realize online $\ell 2$-regularized policy evaluation problems, and keep the identical regularization effect throughout the whole learning process.

The error between FRRC and RRC can be negligible. Therefore, with less complexity, FRRC is a more practical algorithm.

6 CONCLUSION

In this paper, the RBF-based VFA provided a structural model for analysis and derivation, and $\ell 2$-regularized policy evaluation algorithm (RRC) and its fast online counterpart FRRC were proposed to apply in reinforcement learning problem. Both of them can sustain regularization effect throughout learning. The future work contains extending these algorithms with eligibility traces, and generating in off-policy learning problems. Further, other VFA structure can be studied to improve the performance the RBF-VFA used.

REFERENCES

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

[2] X. Xu, L. Zuo, and Z. H. Huang, Reinforcement learning with function approximation: Recent advances and applications, Information Science, Vol.261, No.1, 1-31, 2014.

[3] C. Szepesvári, Algorithms for Reinforcement Learning, Morgan and Claypool Publishers, CA, USA, 2010.

[4] M. Wiering, and M. Otterlo, Reinforcement Learning: State-of-the-art, Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2012.

[5] L. Busoniu, R. Babuska, B. Schutter, and D. Ernst, Reinforcement Learning and Dynamic Programming Using Function Approximators, CRC Press, 2010.

[6] S. O. Haykin, Neural Networks and Learning Machines (Third Edition), Prentice Hall Press, 2008.

[7] C. Dann, G. Neumann, and J. Peters, Policy evaluation with temporal differences: A survey and comparison, Journal of Machine Learning Research, Vol.15, 809-833, 2014.

[8] Z. Qin, W. Li, and F. Janoos, Sparse reinforcement learning via convex optimization, Proceedings of 31th International Conference on Machine Learning, 424–432, 2014

[9] J. Z. Kolter and A. Ng, Regularization and feature selection in least-squares temporal difference learning, Proceedings of the 26th International Conference on Machine Learning, 521-528, 2009.

[10] B. Liu, S. Mahadevan and J. Liu, Regularized off-Policy TD-Learning, Advances in Neural Information Processing Systems 25, 845-853, 2012.

[11] T. H. Song, D. Z. Li, L. L. Cao and K. Hirasawa, Kernel-based least squares temporal difference with gradient correction, IEEE Trans. Neural Networks and Learning Systems, in press.

[12] R. S. Sutton, H. Maei, D. Precup, S. Bhatnagar, D. Silver, C. Szepesvári, and E. Wiewiora, Fast gradient-descent methods for temporal-difference learning with linear function approximation, Proceedings of 26th International Conference on Machine Learning, 993-1000, 2009.

[13] J. N. Tsitsiklis and B. Van Roy, An analysis of temporal-difference learning with function approximation, IEEE Trans. on Automatic Control, Vol. 42, No. 5, 674–690, 1997.

[14] M. G. Lagoudakis and R. Parr, Least-squares policy iteration, Journal of Machine Learning Research, Vol. 4, 1107–1149, 2003.

[15] R. S. Sutton, C. Szepesvári, and H. R. Maei, A convergent $O(n)$ temporal-difference algorithm for off-policy learning with linear function approximation, Proceedings of Advanced Neural Information and Process Systems, 1609–1616, 2008.