IEEE International Symposium on Industrial Electronics (ISIE 2009)
Seoul Olympic Parktel, Seoul, Korea July 5-8, 2009

Online Support Vector Regression based Value Function Approximation for Reinforcement Learning

Dong-Hyun Lee[1], Vo Van Quang[2], Sungho Jo[2], Ju-Jang Lee[2]

[1]Robotics Program, KAIST, Daejeon, Korea
(E-mail: leedh97@kaist.ac.kr)
[2]School of Electrical Engineering and Computer Science, KAIST, Daejeon, Korea
(E-mail: vovquang@kaist.ac.kr, shjo@cs.kaist.ac.kr, jjlee@ee.kaist.ac.kr)

Abstract— **This paper proposes the online Support Vector Regression (SVR) based value function approximation method for Reinforcement Learning (RL). This approach conserves the Support Vector Machine (SVM)'s good property, the generalization which is a key issue of function approximation. Online SVR can do incremental learning and automatically track variation of environment with time-varying characteristics. Using the online SVR, we can obtain the fast and good estimation of value function and achieve RL objective efficiently. Throughout simulation tests, the feasibility and usefulness of the proposed approach is demonstrated by comparison with SARSA and Q-learning.**

I. Introduction

By using RL, an autonomous agent that interacts with the environment can learn how to take a reasonable action for a specific situation [1]. Because of this merit, RL has been of interests not only in machine learning, but also in control engineering and other related fields [1], [2], [6]–[9]. RL provides a general methodology to solve complex uncertain sequential decision problems, which are very challenging in many real-world applications. The RL problem is, in many applications, modeled as a Markov Decision Process (MDP), which has been popularly studied [1], [2], [6]–[9]. An RL agent is assumed to learn the optimal or near-optimal policies from its experiences without knowing the parameters of the MDP.

To find the optimal or near-optimal policy, a value function should be defined to specify the total reward an agent can expect at its current state. RL algorithms estimate the value function usually by observing data generated from interaction with the environment. Various value-function estimation techniques for RL have been proposed. The temporal difference (TD) algorithm suggested by Sutton [2] is one of popularly used techniques in these days. Especially finite-state MDPs have been approached throughout tabular method and tile coding or CMAC under the condition that states and actions are discrete. As for continuous state and discrete action problems, least square methods have proposed to estimate value functions using radial basis functions or kernels [7], [8]. For continuous state and action problems, gradient descent method and actor-critic network approach have been applied for value function approximation [1], [9]. In this paper, we propose an algorithm to handle the problems with continuous states and discrete actions.

Generalization property is an important factor to determine prediction performance with function approximation. SVM is known to have good properties over the generalization [5]. Therefore, applying SVR, which is the regression method of SVM, could be a good approach to estimate value functions. [6] proposes a method using SVR for state value function approximation used in RL. They use the TD error to apply SVR to RL. However, their approach is inattractive to solve online learning problem. Because it estimates a value function after visiting all states using the uniform random policy. In their approach, an RL agent can neither cumulate its experiences continuously nor adapt itself to the changing environment readily. An RL agent should generally be able to learn from data obtained sequentially from interaction with the environment.

This paper proposes to use online SVR to more effectively and quickly approximate value functions used for RL. The idea about the online SVR is originated from the online SVM proposed by Cuwenberghs and Poggio [4]. The online SVR can be equivalent to approaches obtained by applying exact methods such as quadratic programming, However, the online SVR enables the incremental addition of new sample vectors and removal of existing sample vectors, and its process is quicker. A key idea of the online SVR consists in finding the appropriate Karush-Kuhn-Tucker (KKT) conditions for new or updated data by modifying their influences in the regression function while maintaining consistence in the KKT conditions for the rest of data used for learning.

Our approach applies the TD error-based online SVR to estimate state-action value functions, which are directly used in RL. It does not require exact computation like quadratic programming. Therefore, computation load is low and learning speed is fast.

An introduction to RL and online SVR is summarized in Section II, and our method is presented in Section III. In Section IV, some simulation results are shown to evaluate

978-1-4244-4349-9/09/$25.00 ©2009 IEEE

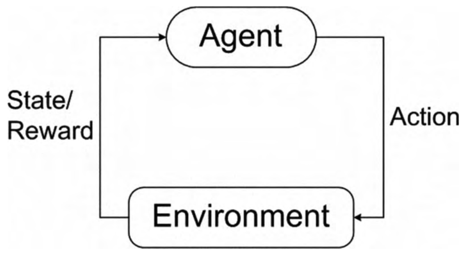


Fig. 1. The agent-environment interaction in RL.

the performance of the proposed method. Section V draws conclusions.

II. Background

A. Reinforcement learning

As mentioned previously, a RL agent learns to find an optimal policy by interacting with the environment (Fig. 1) The optimal policy is defined as follows [1].

$$\pi^*(s) = \arg\max_{a \in A(s)} Q(s,a) \tag{1}$$

where s represents a state, a an action by the agent, $A(s)$ a set of all possible actions, $\pi^*(s)$ the optimal policy and $Q(s,a)$ the state-action value function.

If we know the state-action value function, we can easily find the optimal policy. But the state-action value function requires information on future rewards, so that it should generally be estimated. TD method is popularly used for the value function approximation. The method uses an error defined as follows [2].

$$r + \gamma Q(s',a') - Q(s,a) \tag{2}$$

where s and s' represent the current and next states respectively, a and a' current and next actions respectively, r reward at current state, and γ discount factor.

There are many value function approximation methods using the above TD error such as tabular method, radial basis function/kernel function method, least square method, etc.

B. Online SVR

ϵ-insensitive SVR problem is to find optimal values of weights w such that a parametric function $f(x) = \langle w, x\rangle + b$ can approximate output y within previously set error bound ϵ. It can be formulated as follows [3].

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{3}$$

subject to

$$y_i - \langle w, x_i\rangle - b \le \epsilon$$
$$\langle w, x_i\rangle + b - y_i \le \epsilon$$

where $i = 1, ...l$, l is the number of training data, b represents the bias term, and the constant $C > 0$ determines the trade-off between the flatness of $f(x)$ and the amount up to which deviations larger than ϵ are tolerated [3].

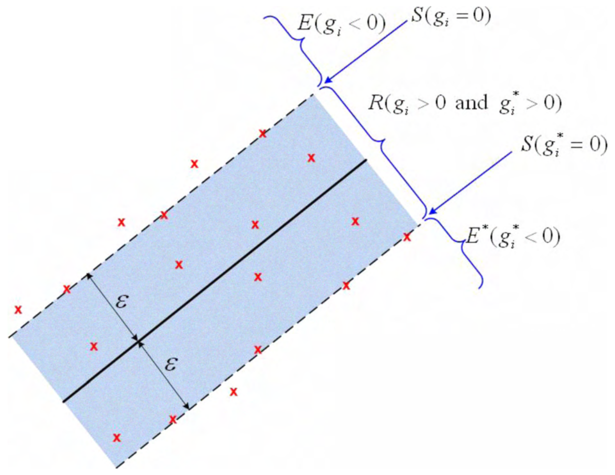


Fig. 2. Data set seperation (this figure is adapted from [5]).

SVR generally calculates the optimal weights by using exact methods such as quadratic programming. However, online SVR find optimal weights by reflecting the influences of new data or removal data readily not requiring exact computations while maintaining the KKT conditions.

From the dual formulation for ϵ-insensitive SVR to find values for α and α^* (lagrange multipliers introduced to derive the dual form), we can obtain the following quadratic cost function [5].

$$\begin{aligned} W = {} & \frac{1}{2}\sum_{ij}^{l}(\alpha_i - \alpha_i^*)Q_{ij}(\alpha_j - \alpha_j^*) - \sum_{i}^{l} y_i(\alpha_i - \alpha_i^*) \\ & + \epsilon\sum_{i}^{l}(\alpha_i + \alpha_i^*) + b\sum_{i}^{l}(\alpha_i - \alpha_i^*) \end{aligned} \tag{4}$$

subject to

$$0 \le \alpha_i, \alpha_i^* \le C$$
$$\sum_{i}^{l}(\alpha_i - \alpha_i^*) = 0$$

where Q is the positive definite kernel matrix whose element is $Q_{ij} = K(x_i, x_j)$ ($K(\cdot,\cdot)$ is a kernel function).

From partial derivatives of W with respect to α, α^*, and b, the following equations can be obtained.

$$\begin{aligned} g_i &\triangleq \frac{\partial W}{\partial \alpha_i} = \sum_{j}^{l} Q_{ij}\beta_j - y_i + \epsilon + b \\ g_i^* &\triangleq \frac{\partial W}{\partial \alpha_i^*} = -\sum_{j}^{l} Q_{ij}\beta_j + y_i + \epsilon - b = -g_i + 2\epsilon \\ \frac{\partial W}{\partial b} &= \sum_{j}^{l} \beta_j = 0 \end{aligned} \tag{5}$$

where $\beta_i \triangleq \alpha_i - \alpha_i^*$

The first two gradient functions in (5) lead to the KKT conditions, that will allow the reformulation of SVR by dividing the whole training data set D into the following sets: margin support vectors S (where $g_i = 0$ or $g_i^* = 0$), error support vectors E (where $g_i < 0$), error star support vectors E^* (where $g_i^* < 0$), and the remaining vectors R. Fig. 2 shows how data space is divided. Specifically centering on g_i , the KKT conditions are:

$$\begin{aligned} 2\epsilon < g_i &\rightarrow g_i^* < 0, \beta_i = -C, i \in E^* \\ g_i = 2\epsilon &\rightarrow g_i^* = 0, -C < \beta_i < 0, i \in S \\ 0 < g_i < 2\epsilon &\rightarrow 0 < g_i^* < 2\epsilon, \beta_i = 0, i \in R \\ g_i = 0 &\rightarrow g_i^* = 2\epsilon, 0 < \beta_i < C, i \in S \\ g_i < 0 &\rightarrow g_i^* > 2\epsilon, \beta_i = C, i \in E \end{aligned} \tag{6}$$

A new vector c is added by inspecting g_c and g_c^*. If both values are positive, c is added as a R vector because that means that the new vector lays inside the ϵ-tube and it does not affect on the existing data in D. When g_c or g_c^* are negative, a new vector is added by setting its initial influence on the regression (β_c) to 0. Then this value is carefully modified (incremented when $g_c < 0$ or decremented when $g_c^* < 0$) until its g_c, g_c^* and β_c values become consistent with respect to the KKT conditions (that is, $g_c < 0$ and $\beta_c = C$, or $g_c^* < 0$ and $\beta_c = -C$, or $0 < \beta_c < C$ and $g_c = 0$, or $-C < \beta_c < 0$ and $g_c^* = 0$). Fig.3 (a) shows the whole process of adding new vector. $\Re$ in the algorithm is a matrix to be maintained and updated to calculate the β_i's (see equations (12) and (13)). The procedure for removing vector from the data set uses the similar principles for adding new vector as in Fig. 3 (b). The details can be found in [5].

III. Online SVR to RL

We propose to use online SVR to provide the state-action value function approximation to RL. Applications of online SVR so far have used output approximation error to find optimal weights, but we propose to use the TD error. Therefore, our online SVR aims to find optimal weights to make the TD error less than the allowed maximum deviation ϵ. In our method, the state-action value function is approximately set to $\hat{Q}(s,a) = w^T\phi(s,a) + b$ where $\phi(s,a)$ is the feature vector (radial basis functions can be chosen as the feature for example). With this approximation, the problem formulation is modified from (3) as follows.

$$\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{7}$$

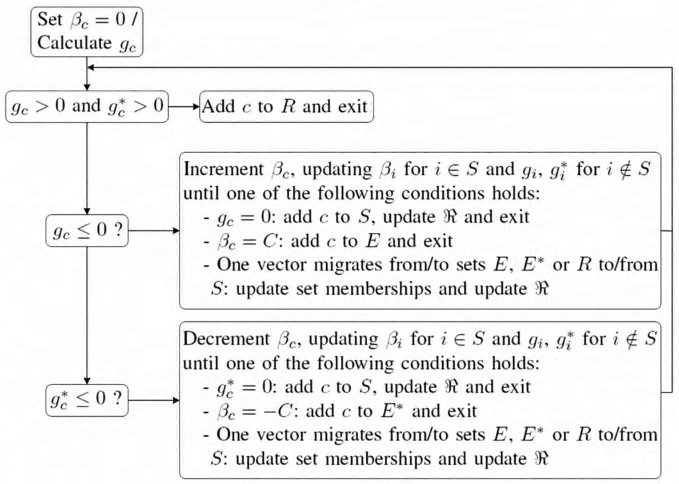
subject to

$$\begin{aligned} r_i + \gamma\hat{Q}(s_i', a_i') - \hat{Q}(s_i, a_i) &\le \epsilon + \xi_i \\ -r_i - \gamma\hat{Q}(s_i', a_i') + \hat{Q}(s_i, a_i) &\le \epsilon + \xi_i^* \end{aligned}$$

By applying the definition of $\hat{Q}$, the constraints can be rewritten to be:

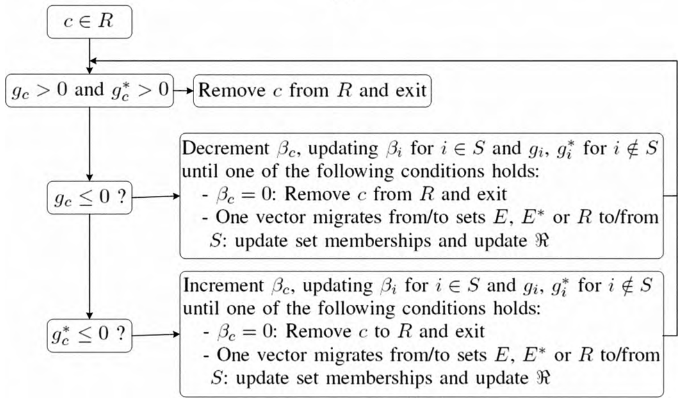
$$\begin{aligned} r_i + \gamma w^T(\phi(s_i', a_i') - \phi(s_i, a_i)) - (1-\gamma)b &\le \epsilon + \xi_i \\ -r_i - \gamma w^T(\phi(s_i', a_i') - \phi(s_i, a_i)) + (1-\gamma)b &\le \epsilon + \xi_i^* \end{aligned} \tag{8}$$

(a)



Set $\beta_c = 0$ / Calculate g_c
$g_c > 0$ and $g_c^* > 0$
Add c to R and exit
$g_c \le 0$?
Increment β_c, updating β_i for $i \in S$ and g_i, g_i^* for $i \notin S$ until one of the following conditions holds:
- $g_c = 0$: add c to S, update $\Re$ and exit
- $\beta_c = C$: add c to E and exit
- One vector migrates from/to sets E, E^* or R to/from S: update set memberships and update $\Re$
$g_c^* \le 0$?
Decrement β_c, updating β_i for $i \in S$ and g_i, g_i^* for $i \notin S$ until one of the following conditions holds:
- $g_c^* = 0$: add c to S, update $\Re$ and exit
- $\beta_c = -C$: add c to E^* and exit
- One vector migrates from/to sets E, E^* or R to/from S: update set memberships and update $\Re$

(b)



$c \in R$
$g_c > 0$ and $g_c^* > 0$
Remove c from R and exit
$g_c \le 0$?
Decrement β_c, updating β_i for $i \in S$ and g_i, g_i^* for $i \notin S$ until one of the following conditions holds:
- $\beta_c = 0$: Remove c from R and exit
- One vector migrates from/to sets E, E^* or R to/from S: update set memberships and update $\Re$
$g_c^* \le 0$?
Increment β_c, updating β_i for $i \in S$ and g_i, g_i^* for $i \notin S$ until one of the following conditions holds:
- $\beta_c = 0$: Remove c to R and exit
- One vector migrates from/to sets E, E^* or R to/from S: update set memberships and update $\Re$

Fig. 3. Procedure for (a) adding and (b) removing a vector.

Then, the corresponding cost function is as follows.

$$\begin{aligned} W = &\frac{1}{2}(\alpha - \alpha^*)^T\mathcal{D}\mathcal{D}^T(\alpha - \alpha^*) + \epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) \\ &- \sum_{i=1}^{l} r_i(\alpha_i - \alpha_i^*) + (1-\gamma)b\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) \end{aligned} \tag{9}$$

subject to

$$0 \le \alpha_i, \alpha_i^* \le C$$
$$\sum_{i}^{l}(\alpha_i - \alpha_i^*) = 0$$

where

$$\mathcal{D}_{ij} = \gamma\phi_j(s_i', a_i') - \phi_j(s_i, a_i)$$

From (9), new gradient functions can be obtained as follows.

$$\begin{aligned} g_i &= \mathcal{D}_i\mathcal{D}^T\beta + \epsilon - r_i + (1-\gamma)b \\ g_i^* &= -\mathcal{D}_i\mathcal{D}^T\beta + \epsilon + r_i - (1-\gamma)b = -g_i + 2\epsilon \end{aligned} \tag{10}$$

The main difference between (5) and (10) is that y_i and b is replaced by r_i and $(1-\gamma)b$ respectively. When a new vector with influence β_c is added, the variation in g_i, g_i^*, and β_i can be calculated by (11) without migration of vectors between sets S, E, E^* and R.

$$\begin{aligned}\Delta g_i &= \mathcal{Q}_{ic}\Delta\beta_c + \sum_{j\in S}\mathcal{Q}_{ij}\Delta\beta_j + (1-\gamma)\Delta b\\ \Delta g_i^* &= -\Delta g_i \\ \Delta\beta_c &+ \sum_{j\in S}\Delta\beta_j = 0\end{aligned} \tag{11}$$

Because $\Delta g_i = 0$ for $i \in S$, the influence of the new vector can be calculated by the following equations.

$$\begin{bmatrix}\Delta b(1-\gamma)\\ \Delta\beta_{S_1}\\ \vdots \\ \Delta\beta_{S_l}\end{bmatrix} = -\Re\begin{bmatrix}1\\ \mathcal{Q}_{S_{1c}}\\ \vdots\\ \mathcal{Q}_{S_{lc}}\end{bmatrix}\Delta\beta_c = \begin{bmatrix}\delta\\ \delta_{S_{1c}}\\ \vdots\\ \delta_{S_{lc}}\end{bmatrix}\Delta\beta_c \tag{12}$$

where

$$\Re = \begin{bmatrix}0 & 1 & \cdots & 1\\ 1 & \mathcal{Q}_{S_1,S_1} & \cdots & \mathcal{Q}_{S_1,S_l}\\ \vdots & \vdots & \ddots & \vdots\\ 1 & \mathcal{Q}_{S_l,S_1} & \cdots & \mathcal{Q}_{S_l,S_l}\end{bmatrix}^{-1} \tag{13}$$

The influence of the new vector for $i \notin S$ can be calculated by the following equations.

$$\begin{aligned}\Delta g_i &= \mathcal{Q}_{ic}\Delta\beta_c + \sum_{j\in S}\mathcal{Q}_{ij}\Delta\beta_j + (1-\gamma)\Delta b\\ &= (\mathcal{Q}_{ic} + \sum_{j\in S}\mathcal{Q}_{ij}\delta_j + \delta)\Delta\beta_c\\ &= \gamma_i\Delta\beta_c\end{aligned} \tag{14}$$

Equations (12) and (14) are valid while vectors do not migrate from set S, E, $E^*$and R to another one. But in order to satisfy KKT conditions consistently for the new vector c, it could be necessary to change first the membership of some vectors to these sets or to update matrix $\Re$ in some cases. β_c is modified incrementally or decrementally until one migration is forced. If the migration occurrs, membership change for the data should be executed, and then the variation of β_c continues. The whole process of adding new vector or removing existing vector is according to Fig. 3. Online SVR can be trained from sequentially added data using this algorithm and the approximated state-action value function can be calculated to be:

$$\begin{aligned}\hat{Q}(x) &= w^T\phi(x) + b\\ &= -\sum_{i=1}^{l}\beta_i\left(\gamma\phi(x_i') - \phi(x_i)\right)\phi(x) + b\\ &= -\sum_i \beta_i(\gamma K(x_i', x) - K(x_i, x)) + b\end{aligned} \tag{15}$$

where x represents a new state-action pair and x_i' and x_i means the next state-action pair and current state-action pair of i$^{\text{th}}$ sample respectively.

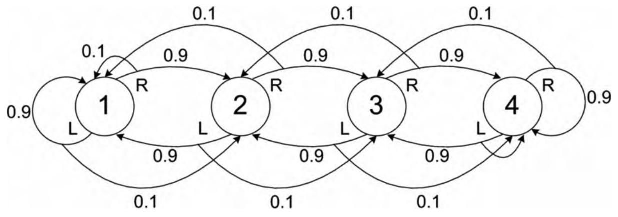


Fig. 4. 4-state chain walk problem (this figure is adapted from [7]).

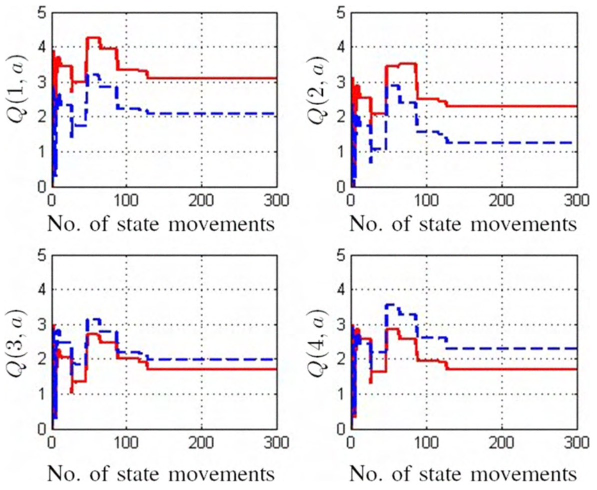


Fig. 5. State-action value function estimate for 4-state chain walk (the solid line indicates $Q(s,R)$, the dashed line indicates $Q(s,L)$).

IV. Simulation

We apply proposed method to two problems. The first one is the 4-state chain walk problem, which is a simple toy problem introduced in [7]. The other one is cart pole balancing problem, which has been popularly used for algorithm evaluation [1], [7]. In both cases, the state-action value function is estimated by the proposed method.

A. 4-state Chain Walk

Fig. 4 shows the 4-state chain walk. There are 2 actions available (L: going to the left and R: going to the right) and 4 states. The rewards over states are 0, +1, +1, and 0 respectively. If one action is selected, the agent takes that action with probability 0.9 and the other with probability 0.1. Radial basis kernel is used to estimate the state-action value function. The optimal policy of this problem is (R, R, L, L). In Fig. 5 the state-action values are depicted. The solid line represents the state-action value for action R and the dashed represents that for action L. We can see the state-action values converges eventually after 128 state movements in Fig. 5.

B. Cart-Pole Balancing Problem

Fig. 6 shows the cart-pole balancing system. An agent applies forces to the cart in appropriate directions to maintain the pole's upright position on the cart. There are two actions: pushing the cart to the left with force $f = -10N$ and to the

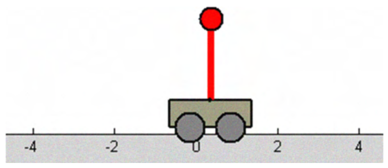


Fig. 6. Cart-pole Balancing System.

TABLE I
PARAMETERS FOR THE DOUBLE POLE BALANCING PROBLEM

Symbol	Description	Value
x	Position of cart on track	[-2.4, 2.4] m
θ	Angle of pole from vertical	[-45, 45] deg.
f	Force applied to cart	-10 N or 10 N
l	Half length of pole	$l = 0.5m$
M	Mass of cart	$1.0kg$
m	Mass of pole	$0.1kg$

right with $f = 10N$. The dynamics of the system is described in (16), but unknown to the agent. Table I describes variables and parameters.

$$\begin{aligned} \ddot{\theta} &= (gsin\theta - Fcos\theta)/\left(\frac{4}{3}l - mlcos^2\theta/(M+m)\right) \\ \ddot{x} &= F - ml\ddot{\theta}cos\theta/(M+m) \\ F &= (f + ml\dot{\theta}^2 sin\theta)/(M+m) \end{aligned} \quad (16)$$

4 states (cart position, cart velocity, pole angle, and pole angular velocity) are used to train the online SVR. Radial basis kernel is used to estimate to the state-action value function. If the cart position is less than -2.4 m or greater than 2.4 m, the system returns reward -1. Also, if the angle of pole is less than -45 deg or greater than 45 deg, it also returns -1. In the other states, the reward is 0. If the reward is -1, that episode terminates and a new episode starts. The initial state is (0, 0, 0, 0). We assume that the task is successfully achieved if the cart-pole system still maintains balancing in 3000 steps. The simulation results of online SVR based RL are depicted in Fig. 7 (a). To evaluate the performance, the results of applying SARSA and Q-learning for the same problem are presented in Fig. 7 (b) and (c) respectively. In SARSA and Q-learning, tabular method is used to estimate state-action values. There are 324 state-action values (3 cart positions $\times$ 3 cart velocities $\times$ 6 pole angles $\times$ 3 pole angular velocities $\times$ 2 actions). Fig. 7 (a) to (c) depicts the best, the worst, and the average performance of each method. we can see the proposed online SVR based RL learns and finds an optimal policy much more quickly than SARSA and Q-learning. In the best performance case, SARSA achieves the task successfully after 200 episodes and Q-learning does after 170 episodes. But, online SVR based RL succeeds the task after 3 episodes only. In both SARSA and Q-learning, the tabular method cannot estimate state-action values properly until the states are visited. Furthermore, it affects negatively for RL agents to estimate other state-action

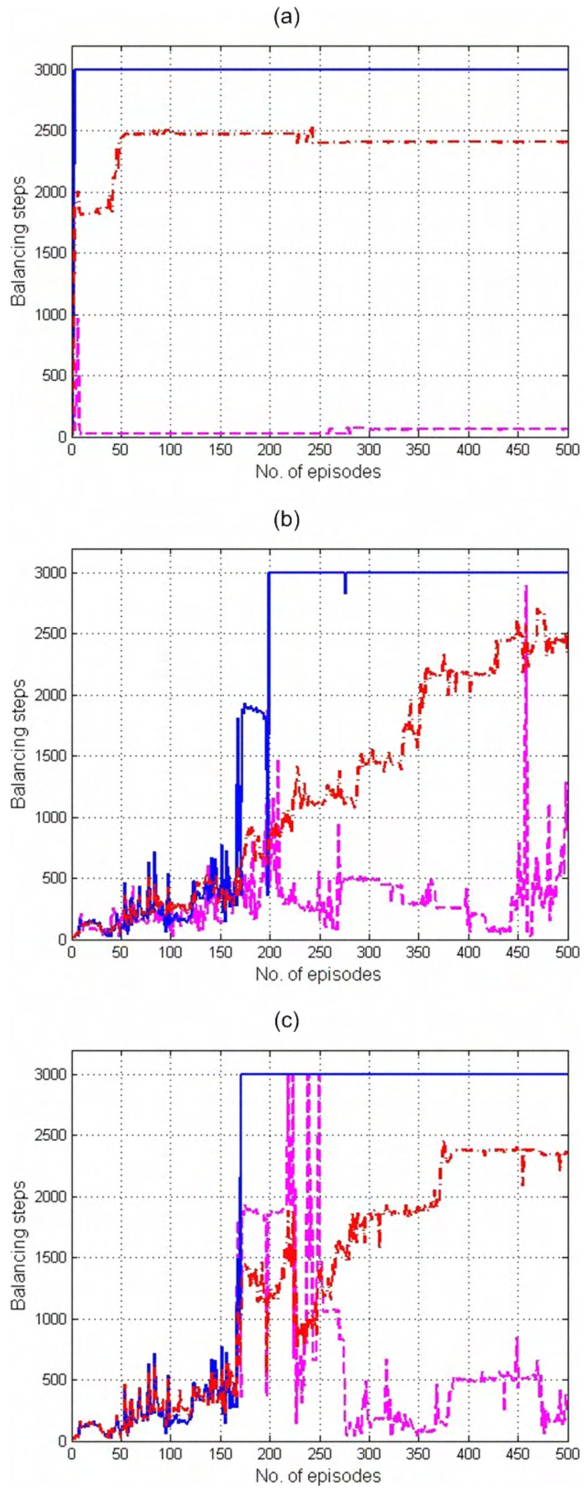


Fig. 7. Balancing steps of cart-pole system using (a) proposed method, (b) SARSA, and (c) Q-learning (the solid, the dashed, and the dash-dotted lines indicate the best, the worst, and the average performances respectively).

values and select actions. RL agents should visit most state-action space to generate appropriate estimates. On the other hands, the online SVR based method computes estimates of state-action values quickly from continuously obtained data. Because of the generalization properties of SVR, RL agents can estimate state-action values of unvisited states reasonably from experience with other states and actions. Radial basis or other kernel function methods can be applied instead of tabular method. But center positions of basis or kernel functions should be assigned appropriately. The assignment is critical to their performance. Therefore, required is a decision algorithm on number of required basis functions and their center positions. In [8], a sparsification procedure is considered to resolve the additional burden. However, our online SVR based RL does not need such consideration. During online learning, Data to estimate the state-action value function are automatically managed throughout additions and removals. Useful data is kept and useless data is removed from data set.

V. Conclusion

In this paper, online SVR based value function approximation method for RL application is proposed. The SVR generally has some limits to be applied directly to RL. However, the online SVR can be properly applicable using the TD error. The online SVR can provide quickly the function approximation with good generalization properties. Furthermore, it enables a RL agent to change its behavior adaptively in interaction with environment. It is known that tabular method or radial basis function methods can be used to solve continuous state problems such as the cart pole balancing. But, to use tabular method, the state discretization is necessary and seriously influential in its performance. Furthermore, the dicretization generally generates a large number of states, which may cause slow learning and heavy computation. Radial basis function method is another choice for function approximation, but generally needs to select center positions. Its performance depends critically on the selection. The proposed online SVR based RL suffers no such problems because it updates its data set and removes useless data automatically throughout online learning. The remaining samples in data set are used to estimate state-action values. The good generalization property of SVR and the adaptation ability of TD error based online scheme make a RL agent learn and find an optimal policy effectively and quickly.

References

[1] R. Sutton and A. Barto, *Reinforcement Learning, an Introduction*, Cambridge, MA:MIT Press, 1998

[2] R. Sutton, "Learning to predict by the method of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp.9-44, 1988.

[3] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression", *Statistics and Computing 14*, pp.199-222, 2004

[4] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," *Advances in Neural Information processing Systems 13*, pp. 409-415, MIT Press, 2001.

[5] M. Martin, "On-Line Support Vector Machine Regression," *Machine Learning: ECML 2002*, pp.173-198.

[6] G. Hu, Y. Qiu, and L. Xiang, "Kernel-Based Reinforcement Learning," *Lecture Notes in Computer Science*, vol. 4113, pp.757-766, 2006

[7] M. G. Lagoudakis and R. Parr, "Least-square policy iteration," *J. Mach. Learn. Res.*, vol. 4, pp.1107-1149, 2003

[8] X. Xu, D. Hu, and X. Lu, "Kernel-Based Least Squares Policy Iteration for Reinforcement Learning", *IEEE Trans. Neural Networks*, vol. 18, no.4, pp.973-991, July 2007

[9] P. He and S. Jagnnathan, "Reinforcement Learning Neural-Network-Based Controller for Nonlinear Discrete-Time Systems With Input Constraints", *IEEE Trans. SMC*, vol. 37, no. 2, pp.425-436, April 2007