2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Online Learning Control Based on Projected Gradient Temporal Difference and Advanced Heuristic Dynamic Programming

Jian Fu, Sujuan Wei
School of Automation
Wuhan University of Technology
Wuhan, Hubei, China 430070
Email: fujian@ whut.edu.cn
Email: weisujuan@whut.edu.cn

Haibo He
Department of Electrical,Computer and Biomedical Engineering
University of Rhode Island
Kingston,Rhode Island, USA 02881
Email: he@ele.uri.edu

Shengyong Wang
WISDRI(WuHan) Automation Co.,Ltd
Wuhan, Hubei, China 430070
Email: 10118@wisdri.com

***Abstract*—We present a novel online learning control algorithm (OLCPA) which comprises projected gradient temporal difference for action-value function (PGTDAVF) and advanced heuristic dynamic programming with one step delay (AHDPOSD). PGTDAVF can guarantee the convergence of temporal difference(TD)-based policy learning with smooth action-value function approximators, such as neural networks. Meanwhile, AHDPOSD is a specially designed framework for embedding PGTDAVF in to conduct online learning control. It not only coincides with the intention of temporal difference but also enables PGTDAVF to be effective under nonidentical policy environment, which results in more practicality. In this way, the proposed algorithms achieve the stability and practicability simultaneously. Finally, simulation of online learning control on a cart pole benchmark demonstrates practical control capability and efficiency of the presented method.**

I. Introduction

Arising as a promising and effective approach, Adaptive dynamic programming (ADP) has made great success in solving the complex sequential decision problem in large and infinite space, performing perfect autonomous/optimal control in noisy, nonlinear, and model free environments [1]–[8].

Briefly speaking, the core idea of ADP is built on the Bellman equation to achieve optimization over time. Given a system with a performance cost function, the objective of dynamic programming is to seek the optimal policy π^* (generating a control sequence $u(x(k))$ denoted as u_k) [9].

$$J^*(x_k, u_k^{\pi^*}) = \min[r(x_k, u_k^{\pi^*}) + \alpha J^*(x_{k+1}, u_{k+1}^{\pi^*})] \quad (1)$$

so that the expected total discount reward J in infinite horizon is minimized.

$$\begin{aligned} J^*(x_k, u_k^{\pi^*}) &= \min \sum_{l=0}^{\infty} \alpha^l r_{l+k} \\ &= \min \sum_{l=0}^{\infty} \alpha^l r(x_{l+k}, u_{l+k}^{\pi^*}) \end{aligned} \quad (2)$$

where k is the kth discrete time step, $x_k = x(k)$ is the state vector of the system, $u_k^{\pi} = u^{\pi}(k) = \pi(x_k)$ is the control action under policy π in the time step k, r_k is the immediate cost in the time step k, and α is a discount factor. Here, we just list the formula in deterministic condition for the sake of simplicity. A more general formula in the stochastic case is easy to deduce.

Typical online ADP employs universal approximator, such as neural network, to approach and generalize state-value function $J(x)$ and policy function $u(x)$ based on online samples (trajectory) to address the curse of dimension and generalization ability when the state of Markov decision process is large and infinite. Together with policy iteration, it could asymptotically and almost-surly converge to optimal/ suboptimal policy according to the given functional criteria from the viewpoint of Hamilton-Jacobi-Bellman equation [10].

However, temporal difference $(TD(\lambda))$, which could be roughly regarded as alternative version of bellman equation in stochastic environment without modeling, is known to converge only under condition of linear function approximating [11], [12]. Specifically, $TD(\lambda)$ may diverge when it is with nonlinear functional approximator. Bair [12] presented residual gradient which can alleviate the problem to some degree, but its convergent value may bias. Recently, Maei [13] proposed a theoretical strategy, gradient temporal difference with smooth value function approximation (GTDSVF), to untangle this long pending issue for the first time. However, its practical algorithm is absent even though a theoretical algorithm framework was given. Also it only estimates the value function under stationary policy instead of non-stationary policy which happens during the procedure of online learning control. So how to develop it to conduct online learning control is an interesting and challenging issue.

Heuristic dynamic programming with one step delay (HDPOSD) is a powerful algorithm of ADP proposed firstly by Si [14]. It stores the previous J value at first. One step later, it conducts the temporal difference calculation in training period together with current J value. So the model or analogous component which is indispensable in previous ADP or ACD is not required any more. Fu, He and Ni developed the original version to generalize multiple-inputs-multiple-outputs (GMIMO) ADP and three networks form associated with adaptive internal immediate reward representation [15], [16]. Nevertheless, Storing the previous J value simply doesn't completely match the definition of cost function and orig-

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

inal intention of TD. Since the policy between consecutive moments may change during online learning control process, so the previous J value should be recalculated based on new policy instead of being constant. That limits its capabilities to a certain extent.

In the paper, we present a novel online learning control algorithm (OLCPA) which comprises stochastic gradient temporal difference for action-value function (PGTDAVF) and advanced Heuristic dynamic programming with one step delay (AHDPOSD). The two components, together with their seamless cooperation, to solve the above problem appropriately. To our best knowledge, this is the first practical online learning control algorithms which guarantee the convergence of TD-based policy evaluation learning with smooth action-value function approximators. Finally, simulation on a cart pole benchmark demonstrates the practical control capability and efficiency for OLCPA.

The rest of this paper is organized as follows. In section II, we present the outline of online learning control algorithm (OLCPA), and focus on introducing two components AHDPOSD and SGTDAVF. In section III, we provide complete implementation of OLCPA, which integrates AHDPOSD and SGTDAVF seamlessly by means of a special queue, to perform online learning control in model free environment. In section IV, the convergence of OLCPA is investigated. In section V, we apply the proposed strategy to cart pole balance benchmark. Detailed experimental settings and results are presented and analyzed. Finally, conclusions are given in section VI.

II. Schematic Diagram Of Propose Online Learning Control Algorithms (OLCPA)

The schematic diagram of our proposed online learning control algorithm (OLCPA) is presented in Fig.1. At first glance, OLCPA works like heuristic dynamic programming (The sketch of flowchart is given in Fig.2). Critic network is responsible for policy prediction, and action network is in charge of policy improvement. The main iteration comprises applying control, subcyle for policy evaluation and subcyle for policy improvement. The reinforcement signal r is provided from the external environment which we mark as the reinforcement signal explicitly. All discussions in this paper have assumed that neural network with multi-layer perception (MLP) architecture is used in both the critic network and the action network design.

In OLCPA, we regard the policy during the process from different perspectives, which is shown in Fig.3. As a whole, no doubt there is a nonidentical policy during online learning control(e.g. from time point k to $k+j-1$ in Fig.3). However, we also can treat the policy as constant within specific interval(e.g. from time point k to $k+i-1$ in Fig.3). So for any two consecutive samples, they were obtained either via identical policy or nonidentical policy, as the case maybe. For example, as shown in Fig.3, two consecutive samples share identical policy in the case of A. However, in the case of B, they were obtained via different policies.

In fact, we design advanced heuristic dynamic programing with one set delay (AHDPOSD) which takes advantage of consecutive samples under identical policy instead of just storing previous J value. Besides, we introduce projected

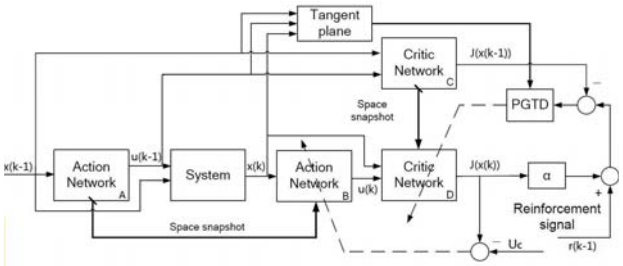

Fig. 1. Schematic diagram for online learning control(OLCPA)

gradient temporal difference for action-value function (PGTDVAF) which employs MSPBE (mean square projected bellman error) criterion instead of the quasi-MSE (mean square error) criterion to calculate stochastic gradient descent.

I note that, for the schematic OLCPA, there is diagram blending with spatial and temporal information together. Specifically, action network with tag A (in the lower right corner) is identical to action network with tag B in physical space. However, their parameters may vary along the time dimension. That why there is an arrow between tag A and tag B to indicate that they are space snapshot of action network along consecutive time point. And so are two critic networks with tag C and D. I will depict the procedure later in detail.

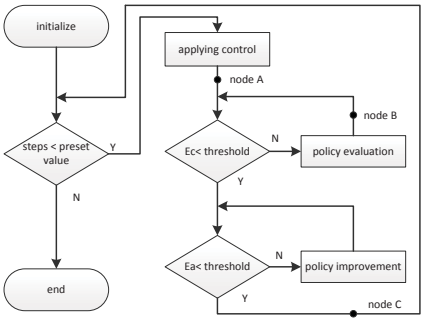

Fig. 2. Flowchart of online learning control(OLCPA)

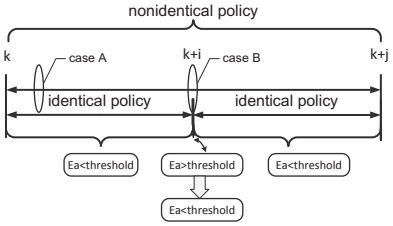

Fig. 3. Policy from different prospectives in OLCPA

As for critic neural network (CNN), the output can be defined as follows:

$$q_i(k) = \sum_{j=1}^{N_{cin}} w_{c_{ij}}^{(1)}(k).x_j(k), i = 1, \ldots, N_{ch} \tag{3}$$

$$p_i(k) = \frac{1 - e^{-q_i(k)}}{1 + e^{-q_i(k)}}, i = 1, \ldots, N_{ch} \tag{4}$$

$$J(k) = \sum_{i=1}^{N_{ch}} w_{c_i}^{(2)}(k) p_i(k) \tag{5}$$

Where q_i is the ith hidden node input of the critic network, p_i is the corresponding output of the hidden node q_i, N_{cin} is the total number of the input nodes in the critic network including N_x inputs from the system states and N_{out} inputs from the output nodes in the action network, and N_{ch} is the total number of the hidden nodes in the critic network.

Similar to critic, the associated equations for the action neural network (ANN) are:

$$h_i(k) = \sum_{j=1}^{N_{ain}} w_{a_{ij}}^{(1)}(k).x_j(k), i = 1, \ldots, N_{ah} \tag{6}$$

$$g_i(k) = \frac{1 - e^{-h_i(k)}}{1 + e^{-h_i(k)}}, i = 1, \ldots, N_{ah} \tag{7}$$

$$v_m(k) = \sum_{i=1}^{N_{ah}} w_{a_{mi}}^{(2)}(k).g_i(k), m = 1, \ldots, N_{aout} \tag{8}$$

$$u_m(k) = \frac{1 - e^{-v_m(k)}}{1 + e^{-v_m(k)}}, m = 1, \ldots, N_{aout} \tag{9}$$

Where h_i is the ith hidden node input of the action network, g_i is the ith hidden node output of action network, v_m is the mth output node outputs of action network, u_m is the mth output node of the action network, N_{ain} is the total number of the input nodes in the action network, N_{ah} is the total number of the hidden nodes in the action network, and N_{aout} is the total number of the output nodes in the action network.

A. Projected gradient temporal difference for action-value function (PGTDAVF)

We firstly investigate the PGTDAVF, which obtains a fix point by projecting the bellman error of action value function (BEVAF). As we know, the action value function moves on a nonlinear face rather than a hyperplane when a nonlinear action value function approximator is adopted. However, we can simplify the projecting onto a nonlinear manifold by projecting BEAVF on to the tangent plane at the given point assuming that the parameter changed very little in one step. In this way, we extend the value function to the action value function, and obtain a similar solution for minimum square projected bellman error (MSPBE), which is a natural extension for PGTDSVF [13]. The intuitive idea lies in that we can deduce action value function $J(x, u)$ from value function $V(x)$ and prompt action $u = \pi(x)$ given that policy π is stationary.

Then we certify it briefly. For any parametrized state-value function $J_{w,k} = J_w(x_k, u_k^{\pi})$, where w is the adjustable parameters set regarded as a vector. We can get tangent plane spanned by the component of gradient

$$\nabla J_{w,k} = \frac{\partial J_{w,k}}{\partial w} = \phi_k \tag{10}$$

Since we discuss the action-value function, so w is w_c where CNN is employed. Then we tend to acquire a fixed point where the projected gradient temporal difference lies in by minimizing MSPBE as following.

$$\begin{aligned} MSPBE(w) &\triangleq M(w) \\ &= \|J_{w,k} - \Pi T J_{w,k}\|^2 \\ &= E(\delta_k \phi_k)^T E\left(\phi_k \phi_k^T\right)^{-1} E(\delta_k \phi_k) \end{aligned} \tag{11}$$

where

$$\delta_k = r_k + \alpha J_{w,k+1} - J_{w,k} \tag{12}$$

and T is Bellman operator, Π is projection operator. Usually, we refer to $J_{w,k}$ as J_{prev} and $J_{w,k+1}$ as J in time step $k + 1$, respectively. In practice, we ordinarily evaluate the distance of two value functions J_1 and J_2 based on samples as $(\bar{x}_1, \bar{x}_2, \cdots, \bar{x}_s)$as follows.

$$\begin{aligned} \|\mathbf{J_1} - \mathbf{J_2}\|_d^2 &= \sum_{i=1}^{s} d\left(\bar{x}_i\right) \left[J_1\left(\bar{x}_i\right) - J_2\left(\bar{x}_i\right)\right]^2 \\ &= \left[\mathbf{J_1} - \mathbf{J_2}\right]^T \mathbf{D} \left[\mathbf{J_1} - \mathbf{J_2}\right] \end{aligned} \tag{13}$$

where $\mathbf{D} = diag\left[d\left(\bar{x}_1\right), d(\bar{x}_2), \cdots, d(\bar{x}_s)\right]$ is the weight matrix, $\mathbf{J}_p = \left[J_p\left(\bar{x}_p\right), J_1(\bar{x}_p), \cdots, J_p(\bar{x}_s)\right]^T, p = 1, 2$.

So (11) can be expressed in application as

$$\begin{aligned} &MSPBE(w) \\ &= \|\mathbf{J}_{w,k} - \Pi T \mathbf{J}_{w,k}\|_d^2 \\ &= \|\Pi\left(\mathbf{J}_{w,k} - T\mathbf{J}_{w,k}\right)\|_d^2 \\ &= \left[\mathbf{\Phi}_k^T \mathbf{D}(T\mathbf{J}_{w,k} - \mathbf{J}_{w,k})\right]^T \left(\mathbf{\Phi}_k^T \mathbf{D} \mathbf{\Phi}_k\right)^{-1} \\ &\quad \cdot \left[\mathbf{\Phi}_k^T \mathbf{D}(T\mathbf{J}_{w,k} - \mathbf{J}_{w,k})\right] \end{aligned} \tag{14}$$

where $\mathbf{\Phi}_k = \begin{bmatrix} \phi_k^T(\bar{x}_1) & \phi_k^T(\bar{x}_2) & \ldots & \phi_k^T(\bar{x}_s) \end{bmatrix}^T$ is a matrix which includes all feasible samples $\bar{x}_i = (x_i, u_i)$.

Next we will calculate the gradient of $M(w)$.

Taking partial derivative of $M(w)$ with respect to w_i (the element of w) as

$$\begin{aligned} -\frac{1}{2}\frac{\partial M(w)}{\partial w_i} &= -\left[\frac{\partial}{\partial w_i}\left(E(\delta_k \phi_k)\right)\right] E\left(\phi_k \phi_k^T\right)^{-1} E(\delta_k \phi_k) \\ &\quad + \frac{1}{2} E(\delta_k \phi_k)^T \frac{\partial}{\partial w_i}\left[E\left(\phi_k \phi_k^T\right)^{-1}\right] E(\delta_k \phi_k), \end{aligned} \tag{15}$$

we can obtain gradient as the following:

$$\begin{aligned} -\frac{1}{2}\nabla M(w) &= E\left((\phi_k - \alpha\phi_{k+1})\phi_k^T \xi\right) - h \\ &= E(\delta_k \phi_k) - \alpha E\left[\phi_{k+1} \phi_k^T \xi\right] - h \end{aligned} \tag{16}$$

where

$$\begin{cases} \xi = E\left(\phi_k \phi_k^T\right)^{-1} E(\delta_k \phi_k) \\ h = E\left[\left(\delta_k - \phi_k^T \xi\right) \nabla^2 J_{w,k} \xi\right] \end{cases} \tag{17}$$

The formula is the product of multiple expected values which cannot be sampled from a single experience because of correlation. This can be solved by updating an additional parameter vector ξ . Since in practice

$$\xi = \left(\mathbf{\Phi}_k^T \mathbf{D} \mathbf{\Phi}_k\right)^{-1} \mathbf{\Phi}_k^T \mathbf{D} \left(T\mathbf{J}_{w,k} - \mathbf{J}_{w,k}\right) \tag{18}$$

is identical to the solution of least-square problem

$$\min \|\mathbf{\Phi}_k \xi - (T\mathbf{J}_{w,k} - \mathbf{J}_{w,k})\|_d^2. \tag{19}$$

The problem can be represented in abstract terms as

$$\min \left\|\phi_k^T \xi - (TJ_{w,k} - J_{w,k})\right\|_d^2 \tag{20}$$

So we can obtain ξ via stochastic gradient descent method

$$\xi_{k+1} = \xi_k + \beta\big(\delta_k - \phi_k^T \xi_k\big)\phi_k. \tag{21}$$

Next we apply stochastic gradient descent method again to (11), together with substituting (21) into (16) and embedding special map Γ. As a result, we finally gain

$$w_{k+1} = \Gamma\Big(w_k + \lambda\left[(\phi_k - \alpha\phi_{k+1})\big(\phi_k^T \xi_k\big)\right] - h_k\Big) \tag{22}$$

or

$$w_{k+1} = \Gamma\Big(w_k + \lambda\left[\delta_k\phi_k - \alpha\phi_{k+1}\big(\phi_k^T \xi_k\big)\right] - h_k\Big) \tag{23}$$

obtain the desired w_{k+1} satisfying (14). where

$$h_k = \big(\delta_k - \phi_k^T w_k\big)\nabla^2 J_{w,k}\xi. \tag{24}$$

Besides Γ denotes the mapping from any domain to a compact set C and β, λ are coefficients. Here, (22) denotes OLCPA1 and (23) denotes OLCPA2(a variation).

Actually, we in the paper adopt a piecewise function to implement Γ as

$$w_{k+1} = \begin{cases} w_k + \lambda\left[(\phi_k - \alpha\phi_{k+1})\big(\phi_k^T w_k\big)\right] \\ -\xi_k \quad \max\big(|w_i|\big) \le T \\ \frac{w_{k+1}}{\max\big(|w_i|\big)} \quad \max\big(|w_i|\big) > T \end{cases} \tag{25}$$

or

$$w_{k+1} = \begin{cases} w_k + \lambda\left[\delta_k\phi_k - \alpha\phi_{k+1}\big(\phi_k^T w_k\big)\right] \\ -\xi_k \quad \max\big(|w_i|\big) \le T \\ \frac{w_{k+1}}{\max\big(|w_i|\big)} \quad \max\big(|w_i|\big) > T \end{cases} \tag{26}$$

which realizes the self-learning and avoids the divergence.

Also, we employ CNN to approximate the action-value function [15], so we have $\phi_k = \left[\left(\frac{\partial J_{w_c,k}}{\partial \mathbf{w}_c^{(1)}}\right)^T \quad \left(\frac{\partial J_{w_c,k}}{\partial \mathbf{w}_c^{(2)}}\right)^T\right]^T$, where

$$\begin{cases} \dfrac{\partial J_{w_c,k}}{\partial w_{c_{ij}}^{(1)}} = \dfrac{\partial J_{w_c,k}}{\partial p_i}\dfrac{\partial p_i}{\partial q_i}\dfrac{\partial q_i}{\partial w_{c_{ij}}^{(1)}} \\ \qquad\qquad = w_{c_i}^{(2)}(k)\left[\dfrac{1}{2}\big(1 - p_i^2(k)\big)\right]x_j(k) \\ \dfrac{\partial J_{w_c,k}}{\partial w_{c_i}^{(2)}} = p_j(k) \end{cases} \tag{27}$$

Further, we can calculate $\nabla^2 J_{w_c,k}$ as

$$\begin{cases} \dfrac{\partial^2 J}{\partial w_{c_{ij}^{(1)}}\partial w_{c_{kl}^{(1)}}} = \begin{cases} -0.5w_{c_i}^{(2)}(k)x_j(k)\big(1 - p_i^2(k)\big) \\ \cdot p_i(k)x_l(k) & i = k \\ 0 & i \ne k \end{cases} \\ \dfrac{\partial^2 J}{\partial w_{c_{ij}^{(1)}}\partial w_{c_k^{(2)}}} = 0 \end{cases} \tag{28}$$

$$\begin{cases} \dfrac{\partial^2 J}{\partial w_{c_i^{(2)}}\partial w_{c_{kl}^{(1)}}} = \begin{cases} -0.5\big(1 - p_i^2(k)\big)x_l(k) & i = k \\ 0 & i \ne k \end{cases} \\ \dfrac{\partial^2 J}{\partial w_{c_i^{(2)}}\partial w_{c_j^{(2)}}} = 0 \end{cases} \tag{29}$$

Apparently, $\nabla^2 J_{w_c,k}$ is a sparse matrix and easy to calculate.

In section *B*, we expand PGTDSVF to PGTDAVF which is fit for expressing policy evaluation in nonidentical policy environment. Since action-value function $J(x, u^{\pi_1})$ and $J(x, u^{\pi_2})$ can respectively denote the value in the same state x with different policy π_1 and policy π_2. However, state-value function $V(x)$ [13] is difficult to do the same thing. In addition, we simplify the solving process of PGTDAVF via special designed piecewise function Γ and approximator(CNN) for the action-value function.

B. Advanced heuristic dynamic programming with one step delay (AHDPOSD)

We investigate the AHDPOSE in this part. It is a framework specially designed to conduct online learning control. Obviously, it involves policy evaluation based on the samples generated by a changing policy. Though PGTDAVF described in part*A* has laid the foundation for tackling this problem, but we cannot apply it directly. The main problem we faced is the nonidentical policy which conflicts with the requirement of AHDPOSE. However, Based on the perspective in Fig.3, we can treat policy constant within specific interval(case*A* in Fig.3), though it is nonidentical as a whole. In the case of B, we can convert three consecutive actual samples generated by two different policies to four data points in two groups via sample adjusting. In each group, there are data points produced under identical policy. Between the group, there are data points produced under different policies. In this way, we can apply PGTDAVF under nonidentical policy environment. This is an intuitive idea of AHDPOSE.

Next, we will describe the AHDPOSE in detail. Specifically, we will discuss the implementation of AHDPOSE in each key nodes(node A, B and C shown in Fig.2) in the case of the A or B (shown in Fig.3). Before we start, we introduce the quintuple $(x_k, u_k^\pi, r_k, x_{k+1}, u_{k+1}^\pi)$ which is constructed to load the data point under identical policy between successive time points. We note that π and π' in the quintuple indicate whether current policy has been changed. Also, we adopt several specific symbols here. Subscript k denotes the index of the main iteration. Apparently, it is the time step k as well. $J^{(l)}$ indicate the cost-to-go value in the lth subcycle for policy evaluation. $l = 0$ implies the cost-to-go value before the subcycle for policy evaluation. And $l = \cdot$ implies the cost-to-go value after the subcycle for policy evaluation. As for $\pi^{(l)}$, it is the behavior policy in the lth subcycle for policy improvement. $l = 0$ and $l = \cdot$ have similar meaning as well. Now, we can implement PGTDVAF to estimate the cost-to-go value based on them.Specifically,

1) Node A – Between applying control and the policy evaluation: We always have $(x_k, u_k^\pi, r_k, x_{k+1}, u_{k+1}^\pi)$ in this node. Apparently, $u_k^\pi = \pi_k^{(\cdot)}(x_k)$ and $u_{k+1}^\pi = \pi_{k+1}^{(0)}(x_{k+1})$ are under the identical policy. That means $\pi_k^{(\cdot)} = \pi_{k+1}^{(0)}$.

As a result, we have the following data and formulas in node A:

$$\begin{cases} (x_k, u_k^\pi, r_k, x_{k+1}, u_{k+1}^\pi), u_{k+1}^\pi \leftarrow \pi_k^{(\cdot)}(x_{k+1}) \\ J_{prev} = J_{k+1}^{(0)}(x_k, u_k^\pi), J = J_{k+1}^{(0)}(x_{k+1}, u_{k+1}^\pi) \end{cases} \tag{30}$$

where $\pi_k^{(.)}$ denotes policy after policy improvement in time step k, $\pi_{k+1}^{(0)}$ indicates policy before policy improvement in time step $k+1$. $J_{k+1}^{(0)}(x_k, u_k^\pi)$ denotes cost-to-go value called by action-value function which is before policy evaluation in time step $k+1$.

2) Node B – During the policy evaluation: we have the following data and formulas in node B.

$$\begin{cases} (x_k, u_k^\pi, r_k, x_{k+1}, u_{k+1}^\pi), u_{k+1}^\pi \leftarrow \pi_k^{(.)}(x_k) \\ J_{prev} = J_{k+1}^{(l)}(x_k, u_k^\pi), J = J_{k+1}^{(l)}(x_{k+1}, u_{k+1}^\pi) \end{cases} \tag{31}$$

where $J_{k+1}^{(l)}(x_k, u_k^\pi)$ denotes action-value function in lth iteration during policy evaluation in main iteration with index $k+1$. Especially, We note that J_{prev} varies during the policy evaluation instead of remaining constant in common HDPOSE.

3) Node C – After the policy improvement: The situation in this node seems a little bit complicate. On condition that there is no policy improvement subcycle taking place in this main iteration(case A), data and formulas are the same of (31). In the case of the policy improvement subcycle having happened(case B), the current behavior policy is changed to π' instead of π. So we cannot keep the two consecutive samples in the quintuple any more. In other words, we change $(x_k, u_k^\pi, r_k, x_{k+1}, u_{k+1}^{\pi'})$ into two groups known as sample adjusting. Thus one is $(x_k, u_k^\pi, r_k, x_{k+1}, u_{k+1}^\pi)$, the other is $(x_{k+1}, u_{k+1}^{\pi'}, \varnothing, \varnothing, \varnothing)$, where $\varnothing$ indicates null.

we have following data and formulas:

$$\begin{cases} (x_{k+1}, u_{k+1}^{\pi'}, \varnothing, \varnothing, \varnothing), u_{k+1}^{\pi'} \leftarrow \pi_{k+1}^{(.)}(x_{k+1}) \\ J_{prev} = J_{k+1}^{(.)}(x_{k+1}, u_{k+1}^{\pi'}), J = \varnothing \end{cases} \tag{32}$$

where $J_{k+1}^{(.)}(x_{k+1}, u_{k+1}^{\pi'})$ denotes cost-to-go value called by action-value function after policy evaluation in time step $k+1$.

Next, we will explain why we design AHDPOSD in above pattern from theoretical prospective. As we known, the expected total discount reward in infinite horizon under stationary policy has the form as

$$J(x_k, u_k^\pi) = \sum_{l=0}^{\infty} \alpha^l r_{l+k} = \sum_{l=0}^{\infty} \alpha^l r(x_{l+k}, u_{l+k}^\pi) \tag{33}$$

given that the system is deterministic. And it is obvious that

$$J(x_l, u_l^\pi) = r(x_l, u_l^\pi) + \alpha J(x_{l+1}, u_{l+1}^\pi), \forall x \in D, l \geq k \tag{34}$$

where D is trajectory from start point $x(k)$.

If there is a policy evaluation taking place in time step k with J changed to J', we have total discount reward in infinite horizon under stationary policy as

$$J'(x_k, u_k^\pi) = \sum_{l=0}^{\infty} \alpha^l r_{l+k} = \sum_{l=0}^{\infty} \alpha^l (x_{l+k}, u_{l+k}^\pi) \tag{35}$$

similarly, we have

$$J'(x_l, u_l^\pi) = r(x_l, u_l^\pi) + \alpha J'(x_{l+1}, u_{l+1}^\pi), \forall x \in D, l \geq k \tag{36}$$

where D is trajectory from start point $x(k)$. Please note that $J_{prev} = J(x_l, u_l^\pi)$ in (34) is not equal to $J_{prev} = J'(x_l, u_l^\pi)$ in (36). So we cannot simply store the value of J_{prev}, we need store the (x_l, u_l^π) instead. Then we calculate the J_{prev} based on current action-value function J as AHDPOSD does. It seems more reasonable.

However, in HDPOSD, We store the previous J_{prev} at first. One step later, we conduct the temporal difference calculation in training together with current J value as:

$$J_{prev} = r(x_l, u_l^\pi) + \alpha J, \forall x \in D, l \geq k \tag{37}$$

It almost has the same result as (34) or (36), unless, during the policy evaluation, it doesn't coincide with (34) or (36) any more. Since

$$J(x_k, u_k^\pi) \neq r(x_k, u_k^\pi) + \alpha J'(x_{k+1}, u_{k+1}^\pi), \forall x \in D \tag{38}$$

where D is corresponding trajectory. However, our proposed calculation covering all three stages is always in accord with (34) or (36).

Besides, we conduct stochastic gradient decent in the standard pattern of temporal difference. I.e. for parametrized action value function $J_w(x_k, u_k^\pi)$, we take derivative of temporal difference error

$$E_c(x_k) = \frac{1}{2}\delta_k^2 = \frac{1}{2}(r_k + \alpha J_w(x_{k+1}, u_{k+1}^\pi) - J_w(x_k, u_k^\pi)) \tag{39}$$

with respect to w as

$$\frac{\partial E_c}{\partial w} = -1.\frac{\partial J_w(x_k, u_k^\pi)}{\partial w}.\delta_k \tag{40}$$

instead of

$$\frac{\partial E_c}{\partial w} = \alpha.\frac{\partial J_w(x_{k+1}, u_{k+1}^\pi)}{\partial w}.\delta_k \tag{41}$$

or

$$\frac{\partial E_c}{\partial w} = \left(\alpha.\frac{\partial J_w(x_{k+1}, u_{k+1}^\pi)}{\partial w} - 1.\frac{\partial J_w(x_k, u_k^\pi)}{\partial w}\right).\delta_k \tag{42}$$

We note that the TD performs prediction via bootstrapping. Alternatively, we regard $r_k + \alpha J_w(x_{k+1}, u_{k+1}^\pi)$ as a stochastic approximation for $J(x_k, u_k^\pi)$ that does not depend on w. So neither moving $\alpha J_w(x_{k+1}, u_{k+1}^\pi)$ closer to $J_w(x_k, u_k^\pi) - r_k$ as (41)(HDPOSE) nor moving two items in both directions as (42) (Residual Algorithms) seem reasonable. Adapting the parameters to move $J_w(x_k, u_k^\pi)$ closer to $r_k + \alpha J_w(x_{k+1}, u_{k+1}^\pi)$ as (40) appears rational.

III. Algorithms Implementation Of OLCPA

In this section, we describe in detail how to integrate PGTDAVF into AHDPOSD framework to achieve online learning control – OLCPA. Above all, we introduce a special queue (denoted as B) with elements (triples) in two consecutive time points $(k, k+1)$ shown in Fig.4. Usually, we access to a specific position of B via the combination of row index and column index. For example, $B[t, B]$ indicates u_t^π. And $B[t, \cdot]$ indicates the row with the index of t.

I would like to note that u_k^π and u_{k+1}^π are actions under identical policy π. It is crucial for the following algorithm. In previous AHDPOSD, we have discussed the methods to guarantee the data between successive time points are under identical policy though policy improvement is in operation. Besides, owing to applying AHDPOSD here, so the current

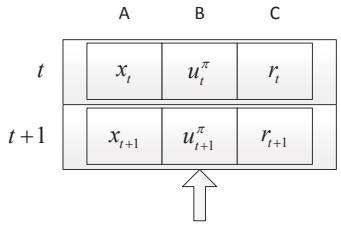

Fig. 4. Diagram of special queue(buffer)

time point is $k+1$ instead of k generally. We preset detail algorithms of OLCPA as following:

Algorithm: OLCPA

/*$u_k = ANN(w_{a,k}, x_k)$ is a neural network for action output calculation in time step k and $\pi_k(x_k)$ is a shorthand notation of it;
$x_k = x(k)$: state vector;
$w_{a,k} = w_a(k)$: weight of ANN;
$u_k = u(k)$: control output of ANN;
u_k^π: an alternative express of value for $ANN(w_{a,k}, x_k)$;

$J_k = CNN(w_{c,k}, x_k, u_k)$ is a neural network for calculate cost-to-go approximately in time step k and $J_k(w_{c,k}, u_k^\pi)$ is a shorthand notation of it;
$w_{c,k} = w_c(k)$ weight of CNN;

$MaxStepNum$: maximum elapsed step times for each trial ;
T_ξ, T_c and T_a: threshold for ξ estimation, policy evaluation and policy improvement respectively;
N_{erit}, N_{crit} and N_{act}: maximum loops for ξ estimation, policy evaluation and policy improvement respectively;
T_w: threshold of w_c

For other notations, please see description in section III.*/;

initiate x_0, initiate $B \Leftarrow null$
initiate $w_{c,0} = w_c(0)$ and $w_{a,0} = w_a(0)$ randomly
$u_0^\pi \leftarrow ANN(w_{a,0}, x_0), \pi \leftarrow (w_{a,0}, ANN)$
$J_0(x_0, u_0^\pi) \leftarrow CNN(w_{c,0}, x_0, u_0^\pi)$
store $x_0 \Rightarrow B[t, A], u_0^{\pi_0} \Rightarrow B[t, B]$
$k = 1$;
while $k \leq$ MaxStepNum **do**
  if $B[t+1, \cdot] \neq null$ **then**
    move $B[t, \cdot] \Leftarrow B[t+1, \cdot]$
    clear $B[t+1, \cdot]$
  end if
  fetch $x_{k-1} \Leftarrow B[t, A]$, $u_{k-1}^\pi \Leftarrow B[t, B]$
  apply control $x_k \leftarrow (x_{k-1}, u_{k-1}^\pi)$via system
  immediate reward $r_{k-1} \leftarrow (x_k)$via environment
  store $r_{k-1} \Rightarrow B[t, C]$,
  $u_k^\pi \leftarrow \pi(x_k)$, $\pi \leftarrow (w_{a,k}, ANN)$;
  store $x_k \Rightarrow B[t+1, A], u_k^\pi \Rightarrow B[t+1, B]$
  $J_k \leftarrow (w_{c,k}, CNN)$
  $J_{prev} \leftarrow J_k(B[t, A], B[t, B])$
  $J \leftarrow J_k(B[t+1, A], B[t+1, B])$
  fetch $r \Leftarrow B[t, C]$
  calculate $\phi_{k-1} = \nabla J_{prev}$
  $\delta_{k-1} = aJ - J_{prev} + r$
  initiate ξ randomly
  $E_\xi = 0.5(\delta_k - \phi_{k-1}^T \xi)\phi_{k-1}$
  $cyc = 0$
  while $(E_\xi > T_\xi)\&(cyc \leq Nerit)$ **do**
    $\xi = \xi + \beta(\delta_k - \phi_{k-1}^T \xi)\phi_{k-1}^T$
    $E_\xi = 0.5(\delta_k - \phi_{k-1}^T \xi)\phi_{k-1}^T$
    $cyc = cyc + 1$
  end while
  $E_c = 0.5\delta_{k-1}^2$
  $l = 0$
  while $(E_c > T_c)\&(l \leq Ncrit)$ **do**
    calculate $\phi_{k-1} = \nabla J_{prev}, \nabla^2 J_{prev}$ and $\phi_k = \nabla J$
    $h = (\delta_{k-1} - \phi_{k-1}^T \xi)\nabla^2 J_{prev} \xi$
    $w_t \leftarrow w_{c,k} + \lambda[(\phi_{k-1} - a\phi_k)(\phi_{k-1}^T \xi)] - h$
    or
    $w_t \leftarrow w_{c,k} + \lambda\left[\delta_{k-1}\phi_{k-1} - \alpha\phi_k\left(\phi_{k-1}^T \xi\right)\right] - h$
    if $\max(|w_{t_{i,j}}|) > T_w$ **then**
      $w_t \leftarrow w_t / \max\left(\left|w_{t_{i,j}}\right|\right)$
    end if
    $J_k^{(l)} \leftarrow (w_{c,k} = w_t, CNN)$
    $J = J_k^{(l)}(B[t+1, A], B[t+1, B])$
    $J_{prev} = J_k^{(l)}(B[t, A], B[t, B])$
    $E_c \leftarrow (\delta_{k-1} = aJ - J_{prev} + r)$
    $l = l + 1$
  end while
  $J_k \leftarrow (w_{c,k} = w_t, CNN)$
  $J = J_k(B[t+1, A], B[t+1, B])$
  $E_a = 0.5J^2$
  $\pi_{k-1} = \pi$
  $m = 0$
  while $(E_a > T_a)\&(m \leq Nact)$ **do**
    $w_{a,k} \leftarrow E_a$ via back-propagation
    $\pi' \leftarrow \pi_k^{(m)} \leftarrow (w_{a,k}, ANN)$
    $u_k^{\pi'} \leftarrow ANN(w_{a,k}, x_k)$
    store $u_k^{\pi'} \Rightarrow B[t+1, B]$
    $J = J_k(B[t+1, A], B[t+1, B])$
    calculate E_a
    $m = m + 1$
  end while
  $\pi_k = \pi \leftarrow (w_{a,k}, ANN)$
  if $\pi_k \neq \pi_{k-1}$ **then**
    $u_k^\pi \leftarrow \pi(x_k)$
    update $u_k^\pi \Rightarrow B[t+1, B]$
    move $B[t, \cdot] \Leftarrow B[t+1, \cdot]$
    clear $B[t+1, \cdot]$
  end if
end while

Generally speaking, the proposed method looks like the combination of SARSA and greedy policy seeking. PGTDVAF expedite SARSA learning with convergence guarantee because there are always on-policy learning periods alternated with the policy improvement periods in OLCPA. And AHDPOSD makes the algorithms practical and unbiased. We would like to note that special queue (B) is a data structure which is used to store and manipulate sample tuples $(x_k, u_k^\pi, r_k, x_{k+1}, u_{k+1}^\pi)$ effectively and efficiently, which link PGTDVAF and AHD-

POSD seamlessly.

IV. Analytical Characteristic Of OLCPA

This section is dedicated to expositions of analytical properties of the OLCPA approach. Generally speaking, the online learning control framework AHDPOSD could be considered as a generalized policy iteration approach with sequential updating of CNN (policy evaluation) and ANN (policy improvement). In the stage of policy evaluation, it solves the Hamiltonian through iterative PGTDAVF with current policy. Meanwhile it achieves the policy which minimizes the current function in the stage of policy improvement stage.

Firstly, we assume the system under control is deterministic. Standard policy iteration algorithms [17] can converge to the optimal control policy π^* with corresponding cost V^* under condition that the demand of initial admissible policy $\pi^{(0)}$ is met. In the case of approximation for value function, specifically, we use MLP NN to solve the cost function $V_k^{\pi}(\cdot)$, and there exists a complete independent basis set $\{\varphi_i(x)\}$ such that cost functional $V(\cdot)$ and its gradient are uniformly approximated by the Weierstrass higher-order approximate theorem. So it is justified to assume there exist CNN weights W_1 such that the value function $V(x)$ is approximated as

$$V(x) = W_1^T \phi(x) + \epsilon(x) = \sum_{i=1}^{N} w_{1i} \varphi_i(x) + \epsilon(x) \tag{43}$$

where N is the number of neurons in the hidden layer, and $\epsilon(x)$ is the NN approximation error. Adam [18] showed that $V_1(x) = W_1^T \phi(x)$ converges uniformly in Sobolev norm $W^{1,\infty}$ to the exact solution $V(x)$ as $N \to \infty$. As for ANN, there is a similar conclusion [19].

Next, we assume the system under control is stochastic. In this situation, we attempt to find zeroes or extrema of functions which only estimated via noisy observations. Base the relevant analysis, we learn that ANN is actually converging to a (local) minimum in a statistical average sense by means of Robbins-Monro stochastic approximation theorem [14], [15]. However, the theorem is valid on condition that regression function is fixed though we even do not know it, such as the case of supervised learning. TD performs prediction via bootstrapping, that results in convergence problem when nonlinear function approximator is adopted.

As for our proposed PGTDAVF, we conduct convergence analysis similar to the work based on two-timescale convergence analysis. Consider the ODE

$$\dot{\omega} = \hat{\Gamma}\left(-\frac{1}{2}\nabla J\right)(\omega) \tag{44}$$

where $\Gamma(j(\omega, \hat{\pi}(\omega)))$ is the projection of $j(\omega, \pi(\omega))$ to the tange space of $\frac{\partial}{\partial w_i} C$ at $\Gamma(w)$. Let K be the set of all asymptotically stable fixed points of (44). Apparently, we have $U \cap C \subset K$, where

$$U = \{w | \delta(\omega) \nabla J_{\omega}(x, \pi(x)) = 0\} \tag{45}$$

is the set of TD(0) solution. Thus if ω is a TD(0)-solution that lies in C, then it is an asymptotically stable fixed point of (44). Furthermore, the iterates produced by nonlinear OLCPA converge to K with probability one under some technical conditions. More detail analysis can be referenced in [13].

V. Case Study With Cart Pole Balancing

The proposed OLCPA has been implemented on a single cart-pole problem. The objective is to balance a single pole mounted on a cart, which can move either to the right or to the left on a bounded, horizontal track. The goal for the controller is to provide a force (applied to the cart) of a fixed force in either the right or the left direction so that the pole stands balanced and avoids hitting the track boundaries. This kind of pendulum is generally used to evaluate the performance of practical control strategies. Here we consider the same system model as

The cart-pole system used in the current study is described as

$$\frac{d^2\theta}{dt^2} = \frac{g\sin\theta + \cos\theta\left[\frac{-F - m(\dot{\theta}^2 \sin\theta + \mu_c sgn(\dot{x}))}{m_c + m}\right] - \frac{\mu_p \dot{\theta}}{ml}}{l\left(\frac{4}{3} - \frac{m\cos^2\theta}{m_c + m}\right)} \tag{46}$$

$$\frac{d^2x}{dt^2} = \frac{F + ml[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta] - \mu_c sgn(\dot{x})}{m_c + m} \tag{47}$$

TABLE I. THE PARAMETERS OF CART-POLE SYSTEM

Parameter	Value	Description
g	$9.8kg/s^2$	acceleration due to gravity
m_c	$1.0kg$	mass of cart
m	$0.1kg$	mass of pole
l	$0.5m$	half-pole length
μ_c	$0.0005m$	coefficient of friction of cart on track
μ_p	$0.000002m$	coefficient of friction of pole on cart
$F \pm 10$	Newtons	force applied to carts center of mass

This model provides four state variables:1) position of the cart on the track; 2) angle of the pole with respect to the vertical position;3) cart velocity;4) angular velocity. In our current study, a run consists of a maximum of 1000 consecutive trials. It is considered successful if the last trial(trial number less than 1000) of the run has lasted 6000 time steps. Otherwise, if the controller is unable to learn to balance the cart-pole within 1000 trials (i.e., none of the 1000 trials has lasted over 600 000 time steps), then the run is considered unsuccessful. In our simulations, we have used 0.02 s for each time step, and a trial is a complete process from start to fall. A pole is considered fallen when the pole is outside the range of [12 12]°or the cart is beyond the range of [2.4 2.4]m in reference to the central position on the track.

Specifically, 50 runs were performed to obtain the results reported as Table II. “Success of rate” indicates the number of successful runs divided by the total number of runs. “No of trials” means how many trials was experienced before it can run successfully. Here, it is an average value based on the data from 50 runs. Apparently, the good algorithms is the one with a high percentage of successful runs as well as a low average number of trials to learn. “Standard deviation” implies how far the data of “No of trials” spread out from their mean. The adjustable parameters of OLCPA have been listed in Table III, by which the above results come out.

VI. Conclusion

In this paper, we propose an online learning control based on projected gradient temporal difference and advanced heuristic dynamic programming (OLCPA). Detailed

TABLE II. PERFORMANCE EVALUATION OF HDPOSD, OLCPA1 AND OLCPA2 METHOD

HDPOSD			OLCPA1			OLCPA2		
Success rate	No of trail	Standard deviation	Success rate	No of trial	Standard deviation	Success rate	No of trial	Standard Deviation
100%	7.4	11.5	100%	6.5	5.7	100%	4.3	4.2

TABLE III. SUMMARY OF THE PARAMETERS

Parameter	$l_c(0)$	$l_a(0)$	$l_c(f)$	$l_a(f)$	T_w	β^*	λ^*
Value	0.3	0.3	0.005	0.005	0.025	0.34	0.21
Parameter	$N_c(0)$	$N_a(0)$	T_c	T_a	N_h	$\beta^\dagger$	$\lambda^\dagger$
Value	50	100	0.05	0.005	6	0.04	0.2

* : OLCPA1
† : OLCPA2

architecture components PGTDAVF and AHDPOSD are depicted. PGTDAVF can guarantee the convergence of temporal difference(TD)-based policy learning with smooth action-value function approximators, such as neural networks. Meanwhile, AHDPOSD is a specially designed framework for embedding PGTDAVF in to conduct online learning control. It not only coincides with the intention of temporal difference but also enables PGTDAVF to be effective under nonidentical policy environment, which results in more practicality. Later we investigate the algorithms of OLCPA which integrated the former components seamlessly. To demonstrate the efficiency and practical control capability of our approach, we evaluated the performance of the proposed approach based on a cart pole balance benchmark. Simulation of online learning control on a cart pole benchmark demonstrates practical control capability and efficiency of the presented method.

There are a few interesting future research directions along this topic. First, in this paper, we considered only multi-layer perception as smooth nonlinear approximator. It will be a natural extension to apply neural network in other forms. Second, our current approach in this paper investigate the application in the noise free environment. The related application in noise environment is an interesting and challenging topic.

ACKNOWLEDGMENT

The authors would like to thank the support in part from the Self-determined and Innovation Research Fund of WHUT(Grant No.2011-IV-129) and National Natural Science Foundation of China(Grant No.51177114).

REFERENCES

[1] S. G. Khan, G. Herrmann, F. L. Lewis, T. Pipe, and C. Melhuish, "Reinforcement learning and optimal adaptive control: An overview and implementation examples," *Annual Reviews in Control*, vol. 36, no. 1, pp. 42–59, 2012.

[2] X. Xu, L. Zuo, and Z. Huang, "Reinforcement learning algorithms with function approximation: Recent advances and applications," *Information Sciences*, vol. 261, pp. 1–31, 2014.

[3] H. He, *Self-adaptive systems for machine intelligence*. John Wiley & Sons, 2011.

[4] Z. Ni, H. He, and J. Wen, "Adaptive learning in tracking control based on the dual critic network design," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 6, pp. 913–928, 2013.

[5] W. B. Powell and J. Ma, "A review of stochastic algorithms with continuous value function approximation and some new approximate policy iteration algorithms for multidimensional continuous applications," *Journal of Control Theory and Applications*, vol. 9, no. 3, pp. 336–352, 2011.

[6] Q. Wei and D. Liu, "Adaptive dynamic programming for optimal tracking control of unknown nonlinear systems with application to coal gasification," *Automation Science and Engineering, IEEE Transactions on*, vol. PP, no. 99, pp. 1–17, 2013.

[7] S. Hu, Y.-D. Yao, and Z. Yang, "Mac protocol identification approach for implement smart cognitive radio," in *Communications (ICC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 5608–5612.

[8] Z. Huang, X. Xu, L. Ye, and L. Zuo, *Kernel-Based Representation Policy Iteration with Applications to Optimal Path Tracking of Wheeled Mobile Robots*. Springer, 2013, pp. 722–730.

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

[10] F. L. L. Vrabie and D., "Reinforcement learning and adaptive dynamic programming for feedback control," *Circuits and Systems Magazine,IEEE*, vol. 9, pp. 32–50, 2009.

[11] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Transactions on Automatic Control*, vol. 42, no. 5, pp. 674–690, 1997.

[12] L. Baird, "Residual algorithms: Reinforcement learning with function approximation," in *ICML*, 1995, Conference Proceedings, pp. 30–37.

[13] H. R. Maei, C. Szepesvri, S. Bhatnagar, D. Precup, D. Silver, and R. S. Sutton, "Convergent temporal-difference learning with arbitrary smooth function approximation," in *Advances in Neural Information Processing Systems*, 2011, Conference Proceedings, pp. 1204–1212.

[14] J. Si and Y.-T. Wang, "Online learning control by association and reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 264–276, 2001.

[15] J. Fu, H. He, and X. Zhou, "Adaptive learning and control for mimo system based on adaptive dynamic programming," *Neural Networks, IEEE Transactions on*, vol. 22, no. 7, pp. 1133–1148, 2011.

[16] H. He, Z. Ni, and J. Fu, "A three-network architecture for on-line learning and optimization based on adaptive dynamic programming," *Neurocomputing*, vol. 78, no. 1, pp. 3–13, 2012.

[17] M. Abu-Khalaf and F. L. Lewis, "Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network hjb approach," *Automatica*, vol. 41, no. 5, pp. 779–791, 2005.

[18] R. A. Adams and J. J. Fournier, *Sobolev spaces*. Academic press, 2003, vol. 140.

[19] K. G. Vamvoudakis and F. L. Lewis, "Online actor-critic algorithm to solve the continuous-time infinite horizon optimal control problem," *Automatica*, vol. 46, no. 5, pp. 878–888, May 2010.