2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

Heuristic Dynamic Programming Strategy with Eligibility Traces

Tao Li[1], Dongbin Zhao[1,2], *Member, IEEE*, and Jianqiang Yi[1], *Member, IEEE*

***Abstract*—In traditional Adaptive Dynamic Programming (ADP), only one step estimate is considered for training process， Thus, learning efficiency is lower. If more steps estimates are included, learning process will be speed up. Eligibility traces record the past and current gradients of estimation. It can be used to work with ADP for speeding up learning. In this paper, Heuristic Dynamic Programming (HDP) which is a typical structure of ADP is considered. An algorithm, HDP(λ), integrating HDP with eligibility traces is presented. The algorithm is illustrated from both forward view and back view for clear comprehension. Equivalency of two views is analyzed. Furthermore, differences between HDP and HDP(λ) are considered from both aspects of theoretic analysis and simulation results. The problem of balancing a pendulum robot (pendubot) is adopted as a benchmark. The results indicate that compared to HDP, HDP(λ) shows higher convergence rate and training efficiency.**

***Index Terms*—Heuristic dynamic programming, Adaptive dynamic programming, Eligibility trace, Pendulum robot**

I. Introduction

Adaptive Dynamic Programming (ADP) [1] is an optimization technique combining concepts of reinforcement learning and approximate dynamic programming. An optimal control policy for the entire range of initial conditions can be obtained by ADP with small computational cost. Thus, ADP has received more and more research and application attentions.

In optimal control area, dynamic programming as a useful tool has been applied to many different fields such as engineering, economics, and so on [2-3]. But it connects with very high computational cost which is called “curse of dimensionality” [4]. This disadvantage limits application of the dynamic programming to low dimensional problems.

A key step of ADP is to estimate the cost function in the dynamic programming. Artificial neutral network has universal approximation capacity for nonlinear functions. Thus, it is adopted to estimate the cost-to-go function in the dynamic programming for solving the above defects. ADP can achieve optimal control result from random initialization. During training, ADP only needs to know the desired cost. Many contributions have been achieved in this area from different aspects [5-9].

The existing ADP algorithms are often classified into three categories: 1) Heuristic dynamic programming (HDP); 2) Dual heuristic dynamic programming (DHP); 3) Globalized dual heuristic dynamic programming (GDHP). Learning strategies are different for three categories. For the first category, the approximate cost function J is calculated as training signal; the second selects the derivative of J; and the third selects both J and its derivative. The action dependent (AD) versions of the above architectures are also presented. AD refers to the design that the action output is also taken as one input of the critic network. The structure of ADP is usually composed of three modules: model network, action network, and critic network. However, considering different learning strategy, the model is needed for DHP and GDHP algorithms. However, this need makes their learning process more complicated.

The traditional ADP is a case with only one step estimate, changing an earlier estimate based on how it differs from a later estimate. If more information is considered, updating of the control police will be more effectively. Eligibility traces can record the past and current gradients, and its adoption could speed up the learning process. It offers significantly fast learning, particularly when rewards are delayed many steps. Thus it often makes sense to use eligibility traces when data are scarce and cannot be repeatedly processed, as is often the case in online applications. Eligibility traces have been used in reinforcement learning such as Q-learning and Sarsa [10-13]. But for ADP, initial exploration about combination between eligibility traces and training of ADP was only introduced in [14]. This paper focuses on designing an ADP algorithm with eligibility traces, and investigating the performance of considering more than one step estimate. Heuristic dynamic programming with eligibility traces, HDP(λ) will be proposed and described in detail with the following sections.

This paper is organized as follows: Section II describes the structure of the traditional HDP algorithm adopted in this paper. Section III presents the combination of eligibility traces and HDP. Section IV provides a case study of pendulum robot

Manuscript received September 21, 2007. This work was supported in part by the NSFC Projects under Grant No. 60621001, the National 973 Project No. 2006CB705500, China.

1. Tao Li, Dongbin Zhao, and Jianqiang Yi are with the Laboratory of Complex Systems and Intelligence Science, Institute of Automation, Chinese Academy of Sciences. 95 Zhongguancun East Road, Haidian District, Beijing 100080, China (phone: +86-10-82615422; fax: +86-10-62658815; e-mail: xpnslt@eyou.com, xpnslt@hotmail.com).

2. Dongbin Zhao is also with the University of Arizona as a visiting scholar funded by China Scholarship Council.

benchmark plant. Some conclusions are given in the last section.

II. Heuristic Dynamic Programming

A schematic diagram of the action dependent heuristic dynamic programming (ADHDP) is shown in Fig. 1.

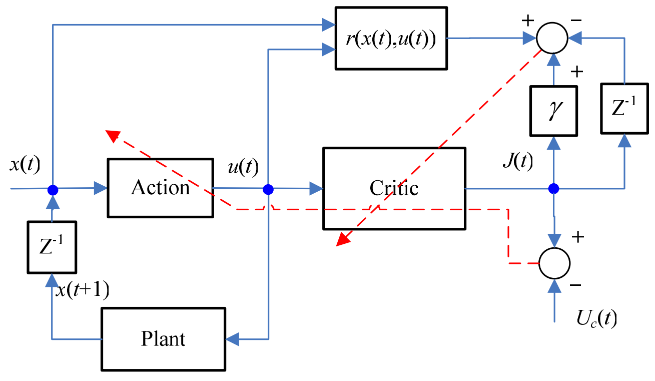

Fig. 1. Schematic diagram of the action dependent heuristic dynamic programming. The solid lines represent signal flow, while the dashed lines are the paths for parameter tuning

The inputs of the action network are the system states $x(t)$. The output of the action network is the control variable $u(t)$. The system states $x(t)$ and the control variable $u(t)$ are chosen as the inputs of the critic network. The output $J(t)$ of the critic network is defined as estimating the discounted total cost-to-go.

In traditional HDP, the error $e_c(t)$ and the objective function $E_c(t)$ for training the critic network are defined as follows:

$$e_c(t) = r(t) + \gamma J(t) - J(t-1) = \delta(t)$$
$$E_c(t) = \tfrac{1}{2}e_c^2(t)\,, \tag{1}$$

where γ $(0<\gamma<1)$ is a discount factor for infinite-horizon problems, and $\gamma = 0.85$ is used in the case studies. $\delta(t)$ is one-step error.

The purpose of training the critic network is to adjust the error $e_c(t)$ close to zero. Thus, the following equation could be derived

$$J(t) = r(t+1) + \gamma r(t+2) + \cdots = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} r(k) = R(t)\,. \tag{2}$$

Equation (2) accords with the form defined in the Bellman equation. $R(t)$ is the sum of the sequenced rewards.

For the training of the action network, the error $e_a(t)$ and the objective function $E_a(t)$ are defined as follows:

$$e_a(t) = J(t) - U_c(t)$$
$$E_a(t) = \tfrac{1}{2}e_a^2(t) \tag{3}$$

where $U_c(t)$ is the desired ultimate objective.

For training the action network and the critic network gradient descent method is often adopted. The whole training process is described as follows. Beginning with a set of random inputs for the network weights $w_a(t)$ and $w_c(t)$ of the action and critic networks respectively, the system state $x(t)$ is sampled. Subsequently, the output of the action network $u(t)$ is calculated. Furthermore, the cost-to-go $J(t)$ and the reinforcement signal $r(t)$ could be received. The following update algorithms are adopted to adjust the weights of these two networks.

$$\begin{aligned} W_c(t+1) &= W_c(t) + \Delta W_c(t) \\ \Delta W_c(t) &= l_c(t)[-\frac{\partial E_c(t)}{\partial W_c(t)}] = -l_c(t)\frac{\partial E_c(t)}{\partial J(t)}\frac{\partial J(t)}{\partial u(t)} \\ &= -l_c(t)e_c(t)\frac{\partial J(t)}{\partial u(t)} \end{aligned} \tag{4}$$

$$\begin{aligned} W_a(t+1) &= W_a(t) + \Delta W_a(t) \\ \Delta W_a(t) &= l_a(t)[-\frac{\partial E_a(t)}{\partial W_a(t)}] = -l_a(t)\frac{\partial E_a(t)}{\partial J(t)}\frac{\partial J(t)}{\partial u(t)} \\ &= -l_a(t)e_a(t)\frac{\partial J(t)}{\partial u(t)} \end{aligned} \tag{5}$$

where $l_c(t)>0$ and $l_a(t)>0$ are the learning rates of the critic and the action networks at time t, respectively.

HDP adjusts an earlier estimate based on how it differs from a later estimate. Only one step estimate is considered in the traditional HDP algorithm. When a state is visited, its activity becomes higher and then gradually decays until the same state is revisited. If more information of future states is included, then it will have much more opportunities to be inspired for "truth" in future time.

If more information is considered, updating of the control police will be more effective. But calculation will need much time and memories if all infinite states are considered. One kind of compromise would base on consideration of some further states from current state. More than one state, but less than all states until termination will be considered. The following section presents an algorithm combining eligibility traces with the traditional HDP scheme.

III. HDP with Eligibility Traces

In this section, a new adaptive dynamic programming algorithm, HDP(λ), is put forward and analyzed. This algorithm combines HDP and eligibility traces.

A. Eligibility Traces

The concept of eligibility traces is first introduced into the Temporal Difference (TD) learning process to form an efficient reinforcement learning algorithm named as TD(λ) [10]. Following the work of [10], some notions of eligibility traces for the application in HDP from forward view, backward view and equivalence analysis are discussed. Due to the direction of the trace, the aliases are forward view and backward view respectively. The forward view is most useful for understanding what is computed by methods using eligibility traces, whereas the backward view is more appropriate for developing practical process about the algorithms themselves.

1) The forward view

The cost-to-go function $R(t)$ in (2) can be rewritten as

$$R_n(t) = r(t+1) + \gamma r(t+2) + \cdots + \gamma^{n-1} r(t+n) + \gamma^n J(t+n) \tag{6}$$

which consists of the reward truncated after n steps and an approximately corrected term for the truncation $\gamma^n J(t+n)$, estimating $\gamma^n r(t+n+1) + \gamma^{n+1} r(t+n+2) + \cdots$, the next nth state value. When n=1, $R_1(t) = r(t+1) + \gamma J(t+1)$. $R_n(t)$ is called the corrected n-step truncated reward, which will be used to form a new cost function with more than one steps rewards, representing the eligibility traces of $J(t)$. A weight λ^{n-1} is introduced to each n-step truncated reward, then a λ-return cost function $R^\lambda(t)$ is defined by

$$\begin{aligned} R^\lambda(t) &= (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} R_n(t) \\ &= (1-\lambda)\sum_{n=1}^{T-t-1} J^{n-1} R_n(t) + \lambda^{T-t-1} R(t) \end{aligned} \tag{7}$$

where the factor $1-\lambda$ is applied to normalize the weights sum to 1. The second row is derived with that the n-step return after a terminal state is $R(t)$. So, if λ=1, $R^\lambda(t) = R(t)$, else if λ=0, $R^\lambda(t) = R_1(t)$, the 1-step return. λ determines the influence of n-step return on the total cost function in a exponential rate, which plays same roles here as in TD(λ).

The adopted algorithm which performs backups based on the λ-return is defined as the algorithm. The algorithm computes an increment to the value of the state on each step as following:

$$\Delta J^F(t) = -\alpha[J(t) - R^\lambda(t)], \tag{8}$$

where α is a positive step-size parameter. $\Delta J^F(t)$ denote the update at time t of $J(t)$. If $x(t) \neq x(t-1)$, eligibility trace will be changed. Thus, it can be deduced that the increments for all $x(t) \neq x(t-1)$ are $\Delta J^F(t) = 0$.

Overview of the forward algorithm is summarized as follows. When a state is visited, all further rewards are reviewed in time, and the algorithm decides to combine them or give up. After a state is updated, this state needs not to be considered. At the same time, we view and process further states on each arrived state.

2) The backward view

A causal and incremental mechanism is defined as the backward view which is adopted to approximate the forward view. It is useful because it is simple conceptually and computationally. In the backward view, each state associated with its eligibility trace can be taken as an additional memory variable. The eligibility trace for state x at time t is denoted as $e(t) \in \Re^+$.

$$e(t) = \begin{cases} \gamma\lambda e(t-1) & \text{if } x(t) \neq x(t-1) \\ \gamma\lambda e(t-1) + 1 & \text{if } x(t) = x(t-1) \end{cases} \tag{9}$$

where $e(0) = 0$. λ, as introduced above, is referred to as the trace-decay parameter. The weights of future are determined by λ exponentially states based on their temporal distance-smoothly interpolating between λ=0 and λ=1.

From the backward view, the increment occurring on that step is computed:

$$\Delta J^B(t) = -\alpha\delta(t)e(t). \tag{10}$$

here $\delta(t)$ in defined in (1). $\Delta J^B(t)$ denote the update at time t.

The algorithm adopts the eligibility trace to record which states have recently been visited. When a state is visited, the eligibility trace will be accumulated. Otherwise, it will be faded away gradually until the state is revisited. The "recently" is defined by $\gamma\lambda$. The degree to which each state is eligible for undergoing learning changes is indicated by the trace.

3) Equivalence of forward and backward views

The aim of this section is to show that the sum of all the updates is the same for the two views:

$$\sum_{t=0}^{T} \Delta J^B(t) = \sum_{t=0}^{T} \Delta J^F(t) I_s, \tag{11}$$

where

$$I_s = \begin{cases} 1 & x(t) = x(t-1) \\ 0 & x(t) \neq x(t-1) \end{cases}. \tag{12}$$

An eligibility trace can be written as:

$$\begin{aligned} e(t) &= \gamma\lambda e(t-1) + I_s = (\gamma\lambda)^2 e(t-2) + \gamma\lambda I_s + I_s \\ &= \cdots = \sum_{k=0}^{t} (\gamma\lambda)^{t-k} I_s \end{aligned} \tag{13}$$

Thus, the left side of (11) can be written

$$\begin{aligned}\sum_{t=0}^{T}\Delta J^{B}(t) &= \sum_{t=0}^{T}-\alpha\delta(t)\sum_{k=0}^{t}(\gamma\lambda)^{t-k}I_s \\ &= \sum_{t=0}^{T}-\alpha\sum_{k=t}^{\infty}(\gamma\lambda)^{t-k}I_s\delta(k) \\ &= \sum_{t=0}^{T}-\alpha I_s\sum_{k=t}^{\infty}(\gamma\lambda)^{t-k}\delta(k)\end{aligned} \tag{14}$$

For the right side of (11), consider an individual update:

$$\begin{aligned}-\tfrac{1}{\alpha}\Delta J^{F}(t) &= J_t^{\lambda}-J(t) \\ &= -J(t)+(1-\lambda)\lambda^{0}[r(t+1)+\gamma J(t+1)] \\ &\quad +(1-\lambda)\lambda^{1}[r(t+1)+\gamma r(t+2)+\gamma^{2}J(t+1)] \\ &\qquad \vdots \qquad\qquad \vdots \qquad\qquad \ddots\end{aligned} \tag{15}$$

From above equation, it can be shown that all the weighting factors of the $r(t+i)$ are geometric series, $i \in 1,2,\cdots$. For example, all the $r(t+i)$'s with their weighting factors of $1-\lambda$ times powers of λ. It turns out that all the weighting factors sum to 1. Combining $r(t+i)$ and summing up coefficient, respectively, we can get:

$$\begin{aligned}-\tfrac{1}{\alpha}\Delta J^{F}(t) &= -J(t) \\ &\quad +(\gamma\lambda)^{0}[r(t+1)+\gamma J(t+1)-\gamma\lambda J(t+1)] \\ &\quad +(\gamma\lambda)^{1}[r(t+2)+\gamma J(t+2)-\gamma\lambda J(t+2)] \\ &\qquad \vdots \\ &= (\gamma\lambda)^{0}[r(t+1)+\gamma J(t+1)-J(t)] \\ &\quad +(\gamma\lambda)^{1}[r(t+2)+\gamma J(t+2)-J(t+1)] \\ &\qquad \vdots \\ &\approx \sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta(k)\end{aligned}$$

Thus, the right side of (11) can be written:

$$\sum_{t=0}^{T}\Delta J^{F}(t)I_s = \sum_{t=0}^{T}-\alpha I_s\sum_{k=t}^{\infty}(\gamma\lambda)^{k-t}\delta(k), \tag{16}$$

Above function is the same as (14). Equation (11) is proved.

B. *HDP* (λ) *and ADHDP* (λ)

In this section, the eligibility trace is adopted for training the critic network.

The reinforcing events concerned with are the moment-by-moment one-step critic network errors. In the traditional HDP algorithm, the critic network error is defined as $e_c(t)$ in (3).

In the forward view of HDP(λ), the error between $\lambda-return$ and the output of the critic network is adopted as training target:

$$e_c^{\lambda}(t) = J(t) - R^{\lambda}. \tag{17}$$

The objective function $E_c(t)$ is defined as that in (1). Weight update law ΔW_c of the critic network in (4) is calculated as:

$$\Delta W_c^{\lambda}(t) = -l_c(t)[J(t)-R^{\lambda}]\frac{\partial J(t)}{\partial W_c(t)}. \tag{18}$$

According to the analysis in above section, the forward view mentioned above is equivalent to the following backward view, which is more appropriate for developing practical process about the algorithms themselves. The backward view is provided by

$$\Delta W_c^{\lambda}(t) = -l_c(t)\delta(t)e(t) \tag{19}$$

where $\delta(t)$ is the usual critic network error defined in (10), and $e(t)$ is the eligibility trace, and its update law is adopted by

$$e(t) = \gamma\lambda e(t-1) + \frac{\partial J(t)}{\partial W_c(t)}. \tag{20}$$

with $e(0)=0$.

HDP(λ) and ADHDP(λ) have the same update algorithm for the weights of the critic network. In HDP(λ), the item $\frac{\partial E_a(t)}{\partial u(t)}$ in the update algorithm (5) is achieved through a model network. However in ADHDP (λ), the item $\frac{\partial E_a(t)}{\partial u(t)}$ in the update algorithm (5) is achieved by back-propagating error from the critic network.

In the traditional HDP and ADHDP algorithms, only one state preceding the current one is changed by the critic network error. The backward view of HDP(λ) and ADHDP(λ) is oriented backward in time. At each moment we look at the current critic network error and assign it backward to each prior state according to the state's eligibility trace at that time. When $\lambda=0$, HDP(λ) and ADHDP(λ) are same as the traditional HDP and ADHDP algorithms, respectively. If $\lambda=1$, HDP(λ) learns something until the end of training. This is same as a Monte Carlo method. Using $\lambda>0$ allows one to incorporate prediction differences from multiple steps, to hopefully speed up learning. For larger values of λ, but still $\lambda<1$, the preceding states are changed more greatly. At the same time, recently visited states are changed less because its eligibility trace is smaller. Pseudo code of the ADHDP(λ) control algorithm is described as follows:

```
Step 1: Initialize $W_a(t)$, $W_c(t)$, $x(t)$ arbitrarily and $e(t)=0$.
Step 2: Calculate action $u(t)$ in state $x(t)$.
Step 3: Take action $u(t)$, observe $r(t)$ and next state $x(t+1)$.
Step 4: Calculate $J(t)$, $e(t)$, and train the critic network.
Step 5: Train the action network.
Repeat step 2 to step 5 until final state or error criteria is
```

attained.

IV. Case Study – Pendubot Benchmark

Under-actuated mechanical systems are often adopted as the benchmark for test performance of different control strategies. Inverted pendulum [5] and pendubot (pendulum robot) [16] are such systems. The pendubot is a typical structure of two link under-actuated robotic system, which is featured as simple structure but complex system dynamics, and is widely adopted to test performance of different control algorithms [17-19]. For its complexity, this paper takes this problem as an example. In this section, ADHDP(λ) is adopted to deal with this case study.

A. The Pendubot Balancing Problem

The schematic diagram of the pendubot system is shown in Fig. 2. The pendubot system is with only one external torque actuated on the first joint, while another joint is passive.

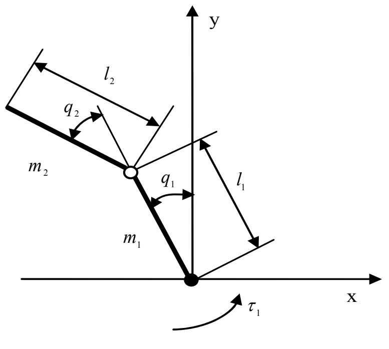

Fig. 2. Scheme of pendubot

Suppose that there is no friction, the system dynamics equations are depicted by

$$\tau = D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q). \tag{21}$$

where $\tau = [\tau_1 \ 0]^T$ is the external torque, $q = [q_1, q_2]$ represents the angles of the two links. D, C and G represent the inertial, coriolis, and gravity terms of the system respectively. Above five variables can be described by five parameters $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$ as

$$D(q) = \begin{bmatrix} \theta_1 + \theta_2 + 2\theta_3 \cos q_2 & \theta_2 + \theta_3 \cos q_2 \\ \theta_2 + \theta_3 \cos q_3 & \theta_2 \end{bmatrix},$$

$$C(q,\dot{q}) = \begin{bmatrix} -\theta_3 \dot{q}_2 \sin q_2 & -\theta_3(\dot{q}_2 + \dot{q}_1)\sin q_2 \\ \theta_3 \dot{q}_1 \sin q_2 & 0 \end{bmatrix},$$

$$G(q) = \begin{bmatrix} -\theta_4 g \sin q_1 - \theta_5 g \sin(q_1 + q_2) \\ -\theta_5 g \sin(q_1 + q_2) \end{bmatrix},$$

where $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5\}$ are denoted by

$$\theta_1 = m_1 l_{c_1}^2 + m_2 l_1^2 + I_1,$$
$$\theta_2 = m_2 l_{c_2}^2 + I_2,$$
$$\theta_3 = m_2 l_1 l_{c_2},$$
$$\theta_4 = m_1 l_{c_1} + m_2 l_1,$$
$$\theta_5 = m_2 l_{c_2}.$$

The system parameters are set same as that in [19]:

$$\theta_1 = 0.0308, \quad \theta_2 = 0.0306, \quad \theta_3 = 0.0095,$$
$$\theta_4 = 0.2087, \quad \theta_5 = 0.0630.$$

The top position $q = [0, 0]$ is an unstable equilibrium. The control object is to balance the pendubot system around its top equilibrium position from a random position around top equilibrium position in this case.

B. Simulation Results

For comparison, the performance of ADHDP [5] is also tested, and the simulation environments are set the same for both algorithms.

The reinforcement signal will be generated as follow:

$$r = \begin{cases} 0 & \text{if } -40^\circ < q_1 < 40^\circ \text{ and } -12^\circ < q_2 < 12^\circ \\ -1 & \text{otherwise} \end{cases}. \tag{22}$$

The bang-bang control strategy is applied to the system, with a constant torque of 0.5 Nm in clockwise or counter-clockwise direction on the first joint. The states are defined as the angles and the angular velocities of the two links $x = \{q_1, \dot{q}_1, q_2, \dot{q}_2\}$.

In this simulation, a run consists of a maximum of 300 consecutive trials. The task is considered successful if a trial has lasted 6000 time steps, where the step time is 0.01 seconds.

The structure of the critic and action networks uses feed-forward network with one hidden layer. The number of the hidden neurons is 6. The states of the system are taken as inputs of the action network. The output $u(t)$ of the action network is continuous, served as the inputs of the critic network together with the states. The control variable to pendubot plant will be 0.5 Nm if $u(t) > 0$, -0.5 Nm otherwise.

For ADHDP(λ) algorithm, $\lambda = 0.85$ is used. The convergence criteria for the action and critic network are chosen as $|e_a| < 0.005$ and $|e_c| < 0.05$, respectively. The max training step is 100 and 50, respectively.

100 runs are performed to calculate the success rate and the average trials which are the average number of step times

before the algorithm converges in the 100 runs. The results are listed in Table I.

TABLE I
COMPARISON OF TWO LEARNING ALGORITHMS FOR THE BALANCING PROBLEM OF THE PENDUBOT

	Success Rate	Number of Trials
ADHDP	24%	168.7
ADHDP(λ)	89%	100.3

Each run is initialized with random normalized weights of the critic and action networks. For ADHDP, the success rate to balance the pendubot system is 24%. Comparatively, the success rate with ADHDP(λ) is 89%. Number of trails reflects the necessary time before the algorithm converges. Litter number of trials means quicker convergence rate. It can be derived that ADHDP(λ) speeds up the convergence rate and has higher successful percentage. It is affected less by the randomness of the initial parameters. It is important to reduce decision time in real-time control problems. A typical result during a successful learning trial by ADHDP(λ) is shown in Fig. 3.

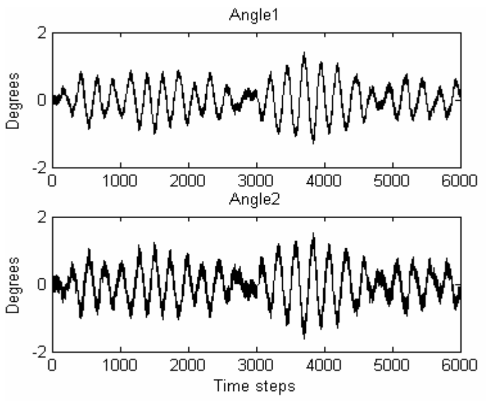

Fig. 3. A typical control result during a successful learning trial for the pendubot system: angle1 and angle2.

V. CONCLUSIONS

In this paper, an new ADP algorithm, ADHDP(λ), combining HDP with eligibility traces is presented. The eligibility trace is adopted for the training of the heuristic dynamic programming. ADHDP(λ) achieves higher convergence rate and training efficiency. This is especially an important feature for on-line control of real-time.

Instead of adjusting a value approximation based solely on one further state, ADHDP(λ) updates control policy based on an exponential weighting of values of future states. More information is considered in ADHDP(λ) training. Using $0 < \lambda < 1$ allows one to incorporate prediction differences from multiple steps. Thus the estimate for updating control policy is more creditable.

Different with $Q(\lambda)$ and Sarsa(λ) derived in [10], the eligibility trace is not applied to update the choice of actor, but adopted for training the critic network. Thus, a more appropriate reinforcement signal will be achieved early which is used to direct the action network to a more optimized one.

ACKNOWLEDGMENT

The authors would like to thank Professor Fei-Yue Wang of Chinese Academy of Sciences and the University of Arizona for his constructive suggestions on this research.

The authors also gratefully acknowledge the support of K. C. Wong Education Foundation, Hong Kong.

REFERENCES

[1] P. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General System Yearbook*, vol. 22. pp. 25-38, 1977.
[2] T. Borgers and R. Sarin, "Learning through reinforcement and replicator dynamics," *Journal of Economic Theory*, vol. 77, no. 1, pp. 1-17, 1997.
[3] J. Dalton and S. N. Balakrishnan, "A neighboring optimal adaptive critic for missile guidance," *Mathematical and Computer Modelling*, vol.23, no.1, pp. 175-188, 1996.
[4] D. Kirk, *Optimal Control Theory: An Introduction.* Englewood Cliffs, NJ: Prentice-Hall, 1970.
[5] J. Si and Y. T. Wang, "On-line learning control by association and reinforcement," *IEEE Transactions on Neural Networks*, vol.12, no.2, pp. 264-276, 2001.
[6] D. B. Prokhorov and D. C. Wunsch, "Adaptive critic design," *IEEE Transactions on Neural Networks*, vol.8, no.5, pp. 997-1007, 1997.
[7] J. J. Murray, C. J. Cox, G. G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Transactions on Systems, Man, Cybernetics Part C*, vol. 32, no.2, pp. 140-152, 2002.
[8] D. Liu, X. Xiong, and Y. Zhang, "Action-dependent adaptive critic designs," *in Proceedings of the 2001 IEEE International Joint Conference on Neural Networks*, vol. 2, 2001, pp. 990-995.
[9] J. Si, A. Barto, W. Powell, and D. Wunsch, *Handbook of Learning and Approximate Dynamic Programming.* IEEE Press, John Wiley & Sons, Inc. 2004.
[10] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Induction*. MIT Press, Cambridge, MA, 1998.
[11] P. Cichosz, "Truncating temporal differences: on the efficient implementation of TD(λ) for reinforcement learning," *Journal of Artificial Intelligence Research*, vol .2, pp. 287-318, 1995.
[12] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: a survey," *Journal of Artificial Intelligence Research*, vol. 4, 1996, pp. 237-285.
[13] P. Dayan, "The converge of TD(λ) for general λ," *Machine Learning*, vol. 8, no. 3, 1992, pp. 341-365.
[14] J. Xu, F. M. Liang, and W. S. Yu, "Learning with eligibility traces in adaptive critic designs," in *Proceedings of the 2006 IEEE International Conference on Vehicular Electronics and safety*, 2006, pp. 309-313.
[15] H. J. Kushner and G. G. Yin, *Stochastic Approximation Algorithms and Applications,* New York: Springer-Verlag, 1997.
[16] M. W. Spong and D. J. Block, *Pendubot Installation and User Guide.* Mechatronic System Inc. 1997.
[17] M. J. Zhang and T. J. Tarn, "Hybrid control of the pendubot," *IEEE/ASME Transactions on Mechatronics*. vol. 7, pp.79-86, 2002.
[18] D. B. Zhao, J. Q. Yi, and D. R. Liu, "Particle swarm optimized adaptive dynamic programming," in *Proceedings of the 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007, pp. 32-37.
[19] M. A. Perez-Cisneros, R. Leal-Ascencio, and P. A. Cook, "Reinforcement learning neurocontroller applied to a 2-dof manipulator," in *Proceedings of the 2001 IEEE International Symposium on Intelligent Control*. Mexico, 2001, pp.56-61.