Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)

Particle Swarm Optimized Adaptive Dynamic Programming

Dongbin Zhao, *Member, IEEE*, Jianqiang Yi, *Member, IEEE*, Derong Liu, *Fellow, IEEE*

***Abstract*—Particle swarm optimization is used for the training of the action network and critic network of the adaptive dynamic programming approach. The typical structures of the adaptive dynamic programming and particle swarm optimization are adopted for comparison to other learning algorithms such as gradient descent method. Besides simulation on the balancing of a cart pole plant, a more complex plant pendulum robot (pendubot) is tested for the learning performance. Compared to traditional adaptive dynamic programming approaches, the proposed evolutionary learning strategy is verified as faster convergence and higher efficiency. Furthermore, the structure becomes simple because the plant model does not need to be identified beforehand.**

***Index Terms*—adaptive dynamic programming, particle swarm optimization, pendubot, pole balancing**

I. Introduction

Adaptive dynamic programming designed from the combination of dynamic programming and neural network has appeared in recent years and has received more and more research and application attention.

Dynamic programming with Bellman equation provides an optimal control strategy for nonlinear systems. It has been widely applied in many application fields of engineering, economics, and so on. But it also suffers from a crucial problem of computational cost called "curse of dimensionality". Therefore, the application of the approach is often limited to low dimensional problems.

Artificial neural networks are verified to approximate any nonlinear function with a high accuracy. It is adopted to estimate the cost-to-go in the dynamic programming to solve the above-mentioned problem. Extensive applications of the dynamic programming could become possible and a new field of adaptive dynamic programming is constructed. Early contributions are from [1-3], and more attention comes from different aspects [4-9].

Based on the learning or training of the neural networks with gradient descent method, the adaptive dynamic programming is often classified into three categories: 1) heuristic dynamic programming (HDP); 2) dual heuristic dynamic programming (DHP); 3) globalized heuristic dynamic programming (GDHP). The first category calculates training signal with the approximate cost function J, the second with the derivative of J, and the third with both J and its derivative. The infrastructure of the adaptive dynamic programming is often composed of model network, action network, and critic network. If the action output is also considered as one input to the critic network, the approach turns to an action dependent (AD) version. It can also be divided into two classes: model free or model dependent without or with neural network model for the plant. However, to make the error backpropagation learning process feasible, the model is necessary for DHP and GDHP approaches.

Evolutionary computation is inspired from the evolution of natural species, and is designed to mimic the natural intelligence. Different kinds of species are considered in the study and many famous algorithms or fields are formed, such as genetic algorithm, ant colony optimization, artificial immune system, particle swarm optimization, and so on. The algorithms usually depend on the heuristic searching principle in the solution process. They have gained successful applications in the fields of function regression, scheduling, robotics, and so on. Some of the algorithms are verified as efficient approaches in the network training compared to gradient descent method. On the other hand, without error backpropagation in the evolutionary learning process, the differential computation difficulties in DHP and GDHP disappear. Therefore, the plant model is not necessary to be identified beforehand.

This paper emphasizes on the application of the evolutionary algorithm, especially the particle swarm optimization, to the learning of the neural networks of the adaptive dynamic programming approach. Section II introduces the structure of the adaptive dynamic programming adopted in this paper. Section III describes the particle swarm optimization used for the training of the neural networks. Section IV and Section V provide the case studies of balancing the cart-pole plant, and a more difficult pendulum robot benchmark plant. Section VI presents some discussions of the proposed algorithm. Conclusions are derived in the end.

This work was supported in part by NSFC Projects (No. 60475030, No. 60575047 and No. 60621001), National 973 Project (No. 2006CB705500), and the International Cooperative Project on Intelligence and Security Informatics by Chinese Academy of Sciences, China.

Dongbin Zhao and Jianqiang Yi are with the Key Lab of Complex Systems and Intelligence Science, Institute of Automation, Chinese Academy of Sciences, Beijing 100080, P.R. China. (phone/fax: 8610 6265-8815; e-mail: dongbin.zhao@ia.ac.cn, jianqiang.yi@mail.ia.ac.cn).

Derong Liu is with the Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL 60607. (phone: 312-355-4475, fax: 312-996-6465; e-mail: dliu@ece.uic.edu.)

II. Adaptive Dynamic Programming

An action dependent heuristic dynamic programming (ADHDP) is adopted here. The scheme is shown in Fig. 1.

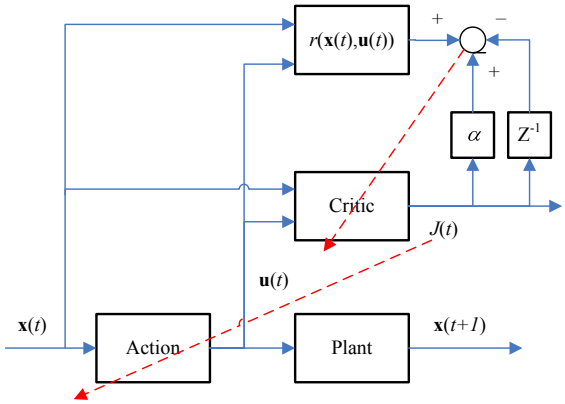

Fig. 1. A typical scheme of an action dependent adaptive dynamic programming.

The inputs to the critic network are the system states $\mathbf{x}(t)$ and the control action $\mathbf{u}(t)$. The output is defined to estimate the cost-to-go $J(t)$ in the Bellman equation. To train the network, an error $e_c(t)$ and the object function $E_c(t)$ to be minimized in the critic network are defined as

$$e_c(t) = \alpha J(t) - J(t-1) + r(t)$$
$$E_c(t) = \frac{1}{2} e_c^2(t) \tag{1}$$

where α ($0 < \alpha < 1$) is a discount factor for finite horizon problems, and α=0.95 is implemented in the following case studies. r is the reinforcement signal or utility function.

The objective of training the critic network is to make the error $e_c(t)$ close to zero, then following equation could be derived

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

The above equation is exactly the form as that defined in the Bellman equation. Meanwhile, the action network is trained by the definition of the error $e_a(t)$ and the object function $E_a(t)$ as

$$e_a(t) = J(t)$$
$$E_a(t) = \frac{1}{2} e_a^2(t) . \tag{3}$$

The training is often conducted with gradient descent method. The whole training process is described as follows. Given random numbers for the weights $\mathbf{w}_a(t)$ and $\mathbf{w}_c(t)$ of both action and critic networks, an action signal $\mathbf{u}(t)$ will be derived with the action network. Together with system states $\mathbf{x}(t)$, the cost-to-go $J(t)$ is calculated by the critic network. With the association or memory of $J(t\text{-}1)$ and the reinforcement signal $r(t)$ evaluating the action, $e_c(t)$ could be calculated.

Then $\mathbf{w}_c(t+1)$ is modified with $E_c(t)$ by the backpropagation algorithm. The critic network training stops when some criterion is met. Afterwards, $\mathbf{w}_a(t+1)$ is modified with $E_a(t)$ and the new updated $\mathbf{w}_c(t+1)$ through the critic and action network backpropagation. The states $\mathbf{x}(t+1)$ at the next time is also generated from the plant with the action $\mathbf{u}(t)$ at the current time. Then, the states and weights of the action and critic neural networks at the next time are obtained, the iteration process proceeds for the next time cycle.

The above action dependent heuristic dynamic programming is also called direct neural dynamic programming (**DNDP**) [5]. Instead of gradient descent method to train the neural networks, other learning algorithms, such as evolutionary computation algorithms, could be also implemented. The following section presents a particle swarm optimized direct neural dynamic programming (**PSDNDP**).

III. Particle Swarm Optimization

Particle swarm optimization (PSO) is a population based stochastic optimization technique developed by Eberhart and Kennedy in 1995, inspired by social behavior of bird flocking or fish schooling [10]. PSO is characterized as few parameters to adjust and easy to implement, compared to genetic algorithm (GA). So, it has been successfully applied in many areas: function optimization, artificial neural network training, fuzzy system control, and so on.

The principle for simulating the bird flocking behavior of PSO is illustrated as follows. A group of birds are randomly positioned in an area where there is only one piece of food to be searched. The exact position of the food is not known as prior to all birds, but they are clear how far the food is. Then an effective way to search the food is to follow the bird, that is the nearest to the food. In PSO, birds, called particles representing potential solutions, fly through the search space by following the current optimum particles. Each particle is characterized by two parameters: position and velocity. As shown in the pseudo code of the procedure, the position is evaluated by a fitness

---

PSEUDO CODE OF PSO

Initialize particles
Do
    Calculate fitness value for each particle
    Set the best fitness value p_{id} of each particle
    Set the global best fitness value p_{gd} of all particles
    Calculate particle velocity according to (4)
    Update particle position according to (5)
    New generation of particles is updated
While maximum iterations or error criteria is not attained

---

value to be optimized.

By comparison, two "best" values are derived. The first one, called p_{id}, is *the best solution* each particle has achieved so far. The other "best" value, called p_{gd}, is a global best solution all particles of the population have achieved so far. If some local particles are used to arrive at the best solution of the population, then p_{gd} is replaced with the local best value p_{ld}. After finding the two best values, the particle updates its velocity v_{id} and positions x_{id} according to (4) and (5):

$$v_{id} = \omega v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}) \tag{4}$$

$$x_{id} = x_{id} + v_{id} \tag{5}$$

where v_{id} is the particle velocity, x_{id} is the position of the current particle (solution). Each particle is treated as a point in a D dimensional space. p_{id} and p_{gd} are defined as before. r_1 and r_2 are random numbers between (0,1). c_1 and c_2 are learning factors.

The influences of parameters such as $v_{\max}$, w, c_1 and c_2, on the performance of PSO have been clearly clarified in [11-12].

A particle's velocity is an important parameter because it determines the resolution about the solution regions. If $v_{\max}$ is too high, particles might fly past good solutions. If $v_{\max}$ is too small, on the other hand, particles may not explore sufficiently beyond locally good regions. In fact, they could become trapped in local optima, unable to move far enough to reach a better position in the problem space. The acceleration constants c_1 and c_2 represent the weighting of the stochastic acceleration terms that pull each particle toward p_{id} and p_{gb} positions. The use of the inertia weight w has provided improved performance in a number of applications.

Experiences with particle swarm optimization arrive at a conclusion that the acceleration constants c_1 and c_2 could be set equal to 2.0 for almost all applications. $v_{\max}$ is thus the only parameter to be adjusted, and it is usually set at about 10% of the dynamic range of the variable in each dimension. The initial weight ω often is decreased linearly from about 0.9 to 0.4 during a whole iteration process as follows:

$$\omega = \omega_{\min} + iter / iter_{\max} (\omega_{\max} - \omega_{\min}) \tag{6}$$

where *iter* represents the step of the iteration process.

As to the training of the neural networks, the weights to be determined are taken as the particle positions, and the error signal is considered in the definition of the fitness for the weights. Because the error is always positive, to prevent a large value, the following transformation is adopted

$$fitness = \exp(-e). \tag{7}$$

Then the fitness value is limited within the range of (0,1). The training problem is also transferred into a fitness maximum optimization problem. In the following simulation sections, the PSDNDP is adopted to learn the weights of the neural networks and apply optimal control of the following case studies.

IV. Performance Evaluation of Case Study One

A. The Cart Pole Balancing Problem

To verify the feasibility and performance of the particle swarm optimized adaptive dynamic programming, a widely adopted benchmark plant of cart-pole system is taken in our first simulation experiment. The system description could be also found in [2], [5], repeated as follows:

$$\frac{d^2\theta}{d^2t} = \frac{g\sin\theta + \cos\theta[-F - ml\dot{\theta}^2 \sin\theta + \mu_c \operatorname{sgn}(\dot{x})] - \dfrac{\mu_p \dot{\theta}}{ml}}{l\left(\dfrac{4}{3} - \dfrac{m\cos^2\theta}{m_c + m}\right)} \tag{8}$$

$$\frac{d^2x}{d^2t} = \frac{F + ml[\dot{\theta}^2 \sin\theta - \ddot{\theta}\cos\theta] - \mu_c \operatorname{sgn}(\dot{x})}{m_c + m} \tag{9}$$

where

g 9.8 m/s^2, acceleration due to gravity;
m_c 1.0 kg, mass of the cart;
m 0.1 kg, mass of the pole;
l 0.5 m, half-pole length;
μ_c 0.0005, coefficient of friction of the cart on track;
μ_p 0.000 002, coefficient of friction of the pole on the cart;
F ±10 N, force applied to the cart's center of mass;

$$\operatorname{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Fourth-order Runge-Kutta method is applied to solve the above nonlinear functions. The position of the cart on the track x and the angle θ of the pole with respect to the vertical position, and their derivatives are taken as the states $\{\theta, \dot{\theta}, x, \dot{x}\}$.

The control objective is to balance the pole about the vertical equilibrium position for as long as possible. If the system goes beyond the link angle range or the cart position range, it is thought as fallen and the learning process stops. The reinforcement signal is defined by:

$$r = \begin{cases} 0 & \text{If } -12° < \theta < 12° \text{ and } -2.4° < x < 2.4°, \\ -1 & \text{Otherwise.} \end{cases} \tag{10}$$

B. Simulation Results

The adaptive dynamic programming is verified as an efficient algorithm for the cart-pole balancing problem with the trained gradient descent method [5]. For comparison of the learning efficiency between DNDP with gradient descent method and PSDNDP, the same parameters are adopted.

The structure of the action and critic neural networks uses feedforward network with one hidden layer. The number of the hidden neurons N_h is 6. There are 4 states taken as the input to the action network, further normalized by the scope $\{12*\pi/180, 120*\pi/180, 2.4, 1.5\}$. The output of the action

TABLE I
COMPARISON OF TWO LEARNING ALGOTIHMS FOR ADPATIVE DYNAMIC PROGRAMMING FOR THE CART-POLE BALANCING PROBRLEM

Noise type	DNDP [5]		PSDNDP	
	Success rate	# of trials	Success rate	# of trials
Uniform 5% sensor	100%	32	100%	**3.5**
Uniform 10% sensor	100%	54	100%	**3.2**

network is continuous, served as the input to the critic network together with the states. However, a bang-bang control strategy is adopted to transfer the continuous action output to a signed constant force F on the plant system.

The internal cycle of critic network N_c and action network N_a are 50 and 100, respectively. The internal training error threshold for the critic network T_c and action network T_a are 0.05 and 0.005, respectively.

A total 100 runs have been carried out to calculate the success rate. Each run consists of 1000 trials. If the last trial of the run has lasted 6000 time steps, the control process is thought of as successful. The sampling step is 0.02s.

For gradient descent method, the learning rates of the critic network and action network are the same, initialized as 0.3, and decreased by 0.05 every five time steps until they reach 0.005 and stay there. The parameters of particle swarm optimization are as described in the above section. Besides, the population size is 20.

The comparison results are tabulated in Table I. The simulation results with uniform state sensor noise implemented through $\theta = \theta$ *(1+random(-noise percentage, noise percentage)) are recorded in the table. Each run is initialized with random normalized weights of the neural networks. All the runs are successful for the cart-pole balancing problem. Moreover, the average trial numbers with particle swarm optimization are reduced to a rather small value. This assures the PSDNDP a faster convergence rate. Another phenomenon shows that the average trial number does not increase with the problem complexity, but maintains a small value. More experiments with uniform 20% sensor noise are also conducted, and the result shows that the average trial number is 3.4.

V. PERFORMANCE EVALUATION OF CASE STUDY TWO

A. The Pendubot Balancing Problem

The pendubot (pendulum robot) is a typical structure of two link under-actuated robotic system, which is featured as simple structure but complex system dynamics. About ten years ago, the Mechatronic Systems Incorporation developed a laboratory experimental system penbuot™[16], which is widely adopted to test performance of different controllers [13-15].

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

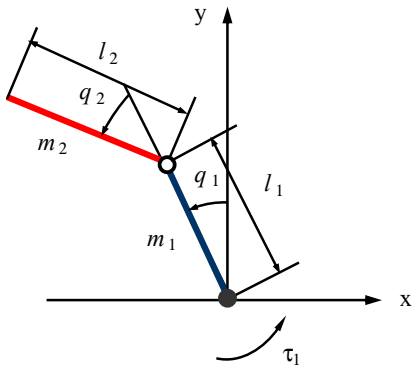

Fig. 2. Scheme of pendubot.

$$\tau = D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) \tag{11}$$

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, which could be described by five parameters $\{\theta_1,\theta_2,\theta_3,\theta_4,\theta_5\}$ as

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

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

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

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 solutions $(q_1\ q_2)$ of (11) represent four equilibriums of the system, of which the commonly reachable two are as follows

(0, 0): Top position for both links, $E = E_{top} = \theta_4 g + \theta_5 g$

(π, 0): Low position for both links, $E = E_{low} = -\theta_4 g - \theta_5 g$

The top position is an unstable equilibrium. The balance objective is to maintain the system about the equilibrium.

The lower position is a stable equilibrium. Another upswing objective is to control the system from the stable equilibrium to the unstable top equilibrium, which is a more difficult task.

For comparison, the system parameters are adopted the same

as that in [15]:

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

The control objective is to balance the pendubot system about its vertical equilibrium position. A reinforcement signal will be generated while the system is out of the pre-defined region.

$$r = \begin{cases} 0 & \text{If } -40° < q_1 < 40° \text{ and } -12° < q_2 < 12°, \\ -1 & \text{Otherwise.} \end{cases} \quad (12)$$

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 input to the critic network is also continuous. The states are defined as the angles and angular velocities of the two links $x = \{q_1, \dot{q}_1, q_2, \dot{q}_2\}$.

B. Simulation Results

In the pendubot balancing problem, a run consists of a maximum of 1000 consecutive trials. Similar to the cart-pole balancing problem, it is considered successful if the trial has lasted 6000 time steps. However, the sampling step time is 0.01 seconds instead of 0.02 seconds as in the cart-pole balancing problem.

The network structure, internal training error threshold, and internal cycle are the same as those defined in the case study of the cart-pole balancing problem. The states input to the action network are normalized by the scope $\{40,360,12,360\} * \pi / 180$.

10 runs are performed to calculate the success rate and the average trials, and the results are listed in Table II. Each run is initialized with random normalized weights of the neural networks. For the DNDP with gradient descent method, the success rate to balance the pendubot system is only 30%. However, the success rate with PSDNDP is 90%. The number of trials is also small, noting that the failure trials of DNDP with gradient descent method are not used in the statistical results of the average trials. It is no doubt PSDNDP outperforms DNDP with gradient descent method as faster convergence rate and higher successful percentage. Compared to [15] with adaptive heuristic critic, it cost about 60-70 seconds to balance the system, while it take about 2 seconds with PSDNDP. A typical control result with the achieved action network is also shown in Fig. 3.

TABLE II
COMPARISON OF TWO LEARNING ALGOTIHMS FOR ADPATIVE DYNAMIC PROGRAMMING FOR THE BALANCING PROBRLEM OF THE PENDUBOT

	success rate	# of trials
DNDP with gradient descent method	30%	240.3
PSDNDP	**90%**	**203.6**

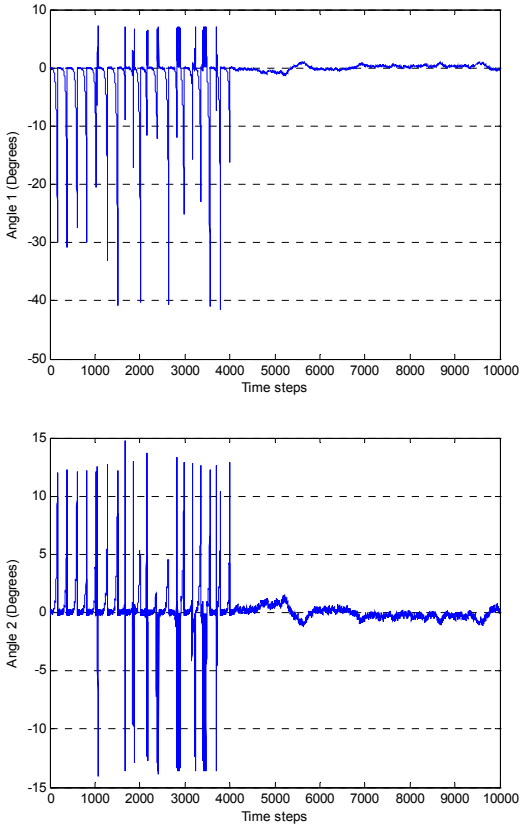

Fig. 3. A typical control result with the learned action network for the balancing problem of the pendubot system: angle 1 and angle 2.

VI. DISCUSSIONS

Through experiments, the PSDNDP is verified as an effective algorithm for the learning of the adaptive dynamic programming. It will guarantee faster convergence rate of the neural networks than DNDP with gradient descent method. This is crucial especially for a difficult control problem or a plant with complex dynamics. The PSDNDP usually has a high success rate in achieving an optimal or sub-optimal solution, while the DNDP with gradient descent method may often fail.

As mentioned above, to find the optimal solution of the weights of the neural networks with evolutionary algorithms, the calculation of the error differential is no longer needed. An important change to the adaptive dynamic programming is that modeling for the plant is not required for any categories. This merit will facilitate the use of all the three branches of adaptive dynamic programming described in the introduction section, and benefit extensive engineering applications.

There are lots of evolutionary computation branches, such as genetic algorithm, ant colony optimization, artificial immune system, and so on. They could be also thought as the neural

network training methods. Particle swarm optimization is adopted in this paper, because it is simple, involved with only two equations, but quite efficient. There have been many validations on the faster convergence rate with particle swarm optimization than that with gradient descent method for neural network training. The convergence proof of particle swarm optimization is under investigation, and will be utilized for the optimal solution achievement of the adaptive dynamic programming.

VII. Conclusions

This paper provides the feasibility of particle swarm optimization applied in the training of the adaptive dynamic programming. A typical case study of the cart-pole balancing problem is first investigated, to validate the particle swarm optimization algorithm outperform gradient descent method with faster convergence rate to train the critic and action networks. A pendulum robot (pendubot) system is characterized as structure simple but dynamics complex. The system is taken as a more complex case study. Simulation results on balancing the pendubot indicate that the particle swarm optimization has both merits on higher success percentage and faster convergence rate than traditional gradient descent method for the learning of adaptive dynamic programming.

As to the limit of our knowledge, the upswing problem has not been resolved successfully with the reinforcement learning approach, leaving much blank to fill.

Moreover, the proposed approach possesses other merits, including model free for the adaptive dynamic programming, and so on.

Acknowledgment

The authors would like to thank Jennie Si for the Matlab code of the cart-pole balancing problem trained with gradient descent method.

References

[1] P. J. Werbos, *Neural Networks for Control, Chapter A Menu of Designs for Reinforcement Learning over Time*. MIT Press, Cambridge, 1990.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 835-846, 1983.

[3] C. Watkins and P.Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279-292, 1992.

[4] G. N. Saridis and F.Y. Wang, "Suboptimal control of nonlinear stochastic systems," *Control Theory and Advanced Technology*, vol.10, no. 4, pp. 847-871, 1994.

[5] J. Si and Y. T. Wang, "On-line learning control by association and reinforcement," *IEEE Trans. on Neural Networks*, vol.12, no.2, pp. 264-276, 2001.

[6] D. B. Prokhorov and D. C. Wunsch, "Adaptive critic design," *IEEE Trans. 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 Trans. Syst., Man, Cybern. part C*, vol. 32, no.2, pp. 140-152, 2002.

[8] D. Liu, X. Xiong, and Y. Zhang, "Action-dependent adaptive critic designs," in *Proc. of the 2001 IEEE Int. Joint Conf. 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] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proc. of the 1995 IEEE Int. Conf. on Neural Networks.* Australia, 1995, pp.1942 -1948.

[11] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *IEEE World Congress on Computational Intelligence*, 1998. pp.69-73

[12] ------, "Parameter selection in particle swarm optimization," *Evolutionary Programming VII: Proc. EP 98*, 1998, pp.591-600.

[13] M. J. Zhang and T. J. Tarn, "Hybrid control of the pendubot," *IEEE/ASME Trans. on Mechatronics*. vol. 7, pp.79-86, 2002.

[14] I. Fantoni, R. Lozano and M. W. Spong, "Energy based control of the Pendubot," *IEEE Trans. on Automatic Control*, vol.45, pp.725-729, 2000.

[15] M. A. Perez-Cisneros, R. Leal-Ascencio, and P. A. Cook, "Reinforcement learning neurocontroller applied to a 2-dof manipulator," in *Proc. of the IEEE Int. Sym. On Intelligent Control*. Mexico, 2001, pp.56-61.

[16] M. W. Spong and D. J. Block, *Pendubot Installation and User Guide*. Mechatronic System Inc. 1997.