On-Line Learning Control by Association and Reinforcement

Jennie Si, *Senior Member, IEEE,* and Yu-Tsung Wang, *Member, IEEE*

***Abstract*—This paper focuses on a systematic treatment for developing a generic on-line learning control system based on the fundamental principle of reinforcement learning or more specifically neural dynamic programming. This on-line learning system improves its performance over time in two aspects. First, it learns from its own mistakes through the reinforcement signal from the external environment and tries to reinforce its action to improve future performance. Second, system states associated with the positive reinforcement is memorized through a network learning process where in the future, similar states will be more positively associated with a control action leading to a positive reinforcement. A successful candidate of on-line learning control design will be introduced. Real-time learning algorithms will be derived for individual components in the learning system. Some analytical insight will be provided to give guidelines on the learning process took place in each module of the on-line learning control system. The performance of the on-line learning controller is measured by its learning speed, success rate of learning, and the degree to meet the learning control objective. The overall learning control system performance will be tested on a single cart-pole balancing problem, a pendulum swing up and balancing task, and a more complex problem of balancing a triple-link inverted pendulum.**

***Index Terms*—Neural dynamic programming (NDP), on-line learning, reinforcement learning.**

I. Introduction

WE ARE considering a class of learning decision and control problems in terms of optimizing a performance measure over time with the following constraints. First, a model of the environment or the system that interacts with the learner is not available *a priori*. The environment/system can be stochastic, nonlinear, and subject to change. Second, learning takes place "on-the-fly" while interacting with the environment. Third, even though measurements from the environment are available from one decision and control step to the next, a final outcome of the learning process from a generated sequence of decisions and controls may come as a delayed signal in only an indicative "win" or "loose" format.

Dynamic programming has been applied in different fields of engineering, operations research, economics, and so on for many years [2], [5], [6], [22]. It provides truly optimal solutions to nonlinear stochastic dynamic systems. However, it is well understood that for many important problems the computation costs of dynamic programming are very high, as a result of the "curse of dimensionality" [8]. Other complications in applications include a user supplied explicit performance measure and a stochastic model of the system [2]. Incremental optimization methods come in handy to approximate the optimal cost and control policies [3], [11].

Reinforcement learning has held great intuitive appeal and has attracted considerable attention in the past. But only recently it has made major advancements by implementing the temporal difference (TD) learning method [1], [16], [21]. The most noteworthy result is a TD-Gammon program that has learned to play Backgammon at a grandmaster level [17]–[19]. Interestingly enough, the development history of Gammon programs also reflects the potentials and limitations of various neural networks learning paradigms. With the success of TD-Gammon, the TD algorithm is no doubt a powerful learning method in Markovian environments such as game playing.

How does one ensure successful learning in a more generic environment? Heuristic dynamic programming (HDP) was proposed in the 1970s [22] and the ideas were firmed up in the early 1990s [23]–[25] under the names of adaptive critic designs. The original proposition for HDP was essentially the same as the formulation of reinforcement learning (RL) using TD methods. Specifically a critic network "critiques" the generated action value in order to optimize a future "reward-to-go" by propagating a temporal difference between two consecutive estimates from the critic/prediction network. This formulation falls exactly into the Bellman equation. Even with the same intention at the beginning, the two approaches started to differentiate by the way the actions were generated. HDP and the adaptive critics in general train a network to associate input states with action values. On the other hand, TD-based Gammon programs, as well as Q-learning, opted for search algorithms to determine the optimal moves and, hence, avoid additional error during the initial action network training, with a price paid for search speed.

Existing adaptive critic designs [26] can be categorized as: 1) HDP; 2) dual heuristic dynamic programming (DHP); and 3) globalized dual heuristic dynamic programming (GDHP). Variations from these three basic design paradigms are also available, such as action dependent (AD) versions of the above architectures. AD refers to the fact that the action value is an additional input to the critic network. Action dependent variants from the original three paradigms will be denoted with an abbreviation of "AD" in front of their specific architecture. For example, ADHDP and ADDHP denote "action dependent heuristic dynamic programming" and "action dependent dual heuristic dynamic programming," respectively. Our proposed

Manuscript received October 7, 1999; revised March 20, 2000 and November 20, 2000. This work was supported by NSF under Grants ECS-9553202 and ECS-0002098 and in part by EPRI-DOD under Grant WO8333-01, by DARPA under Grant MDA 972-00-1-0027, and by Motorola.

The authors are with Department of Electrical Engineering, Arizona State University, Tempe, AZ 85287-7606 USA (e-mail: si@asu.edu).

Publisher Item Identifier S 1045-9227(01)01404-7.

on-line learning system is most relevant to ADHDP. Once again, the basic idea in adaptive critic design is to adapt the weights of the critic network to make the approximating function, J, satisfy the modified Bellman equation. In this framework, instead of finding the exact minimum, an approximate solution is provided for solving the following equation:

$$J^*(X(t)) = \min_{u(t)}\{J^*(X(t+1)) + g(X(t), X(t+1)) - U_0\} \tag{1}$$

where $g(X(t), X(t+1))$ is the immediate cost incurred by $u(t)$ at time t, and U_0 is a heuristic term used to balance [24]. To adapt $J(X(t))$ in the critic network, the target on the right-hand side of (1) must be known *a priori*. To do so, one may wait for a time step until the next input becomes available. Consequently, $J(X(t+1))$ can be calculated by using the critic network at time $t+1$. Another approach is to use a *model network*, which is a pretrained network to approximate the system dynamics. In principle, such a network can be trained on-line.

One major difference between HDP and DHP is within the objective of the critic network. In HDP, the critic network outputs J directly, while DHP estimates the derivative of J with respect to its input vector. Since DHP builds derivative terms over time directly, it reduces the probability of error introduced by backpropagation. GDHP is a combination of DHP and HDP, approximating both $J(X(t))$ and $\partial J(X(t))/\partial X(t)$ simultaneously with the critic network. Therefore, the performance of GDHP is expected to be superior to both DHP and HDP. However, the complexities of computation and implementation are high for GDHP. The second derivative terms, $\partial^2 J(X(t))/\partial X(t)\partial w_c(t)$, need to be calculated at every time step. Analysis and simulation results in [12] and [13] are consistent with this observation.

Adaptive critic designs such as HDP, DHP, and GDHP, as well as their action dependent versions have been applied to an autolanding problem [12]. In implementations, the critic networks of HDP and ADHDP are used to approximate J. To obtain the value of J at time $t+1$, the states or/and actions are predicted by using a model network. The model network approximates plant dynamics for a given state $X(t)$ and action $u(t)$, and the model network outputs $X(t+1)$. In [12], the model network was trained off-line. Results from [12] show that GDHP and DHP are better designs than the HDP and ADHDP for the autolanding problem. The auto-landers trained with wind shear for GDHP and DHP successfully landed 73% of all 600 trials while those for HDP and ADHDP were below 50%.

From the previous discussions, we can also categorize adaptive critic designs by whether or not a model was used in the learner, as shown in [26]. Note that in adaptive critic designs, there are two partial derivative terms in the backpropagation path from the Bellman equation. They are $\partial J(t)/\partial W_c(t)$ and $\partial J(t+1)/\partial W_c(t)$. When adaptive critic designs were implemented without a model network (i.e., two-network design), the second partial derivative term was simply ignored. The price paid for omitting this term can be high. Results in [12] and [13], seem to agree with this observation. In later implementations such as DHP and GDHP, a model network was employed to take into account the $\partial J(t+1)/\partial W_c(t)$ term.

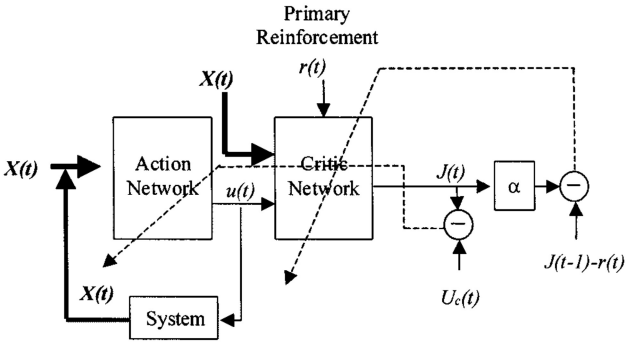

Fig. 1. Schematic diagram for implementations of neural dynamic programming. The solid lines represent signal flow, while the dashed lines are the paths for parameter tuning.

Our proposed approach in this paper is closely related to ADHDP. One major difference is that we do not use a system model to predict the future system state value and consequently the cost-to-go for the next time step. Rather, we store the previous J value. Together with the current J value, we can obtain the temporal difference used in training. We have thus resolved the dilemma of either ignoring the $\partial J(t+1)/\partial W_c(t)$ term by sacrificing learning accuracy or including an additional system model network by introducing more computation burden. In this paper, we present a systematic examination on our proposed neural dynamic programming (NDP) design that includes two networks, the action and the critic, as building blocks. In the next two sections, we first introduce the building blocks of the proposed NDP implementations and then the associated on-line learning algorithms. In Section III, we provide evaluations on the on-line NDP designs for a single cart-pole balancing problem. Section IV gives evaluations of NDP designs in a pendulum swing up and balancing task. Section V includes simulation results of a more difficult on-line learning control problem, namely the triple-link inverted pendulum balancing task. After the presentation on NDP designs, algorithms, and performance evaluations, we try to provide some initial results on analytical insight of our on-line NDP designs using stochastic approximation argument. Finally, a section on conclusions and discussions is provided where we also provide some preliminary findings on improving the scalability of our proposed NDP designs.

II. A General Framework for Learning Through Association and Reinforcement

Fig. 1 is a schematic diagram of our proposed on-line learning control scheme. The binary reinforcement signal $r(t)$ is provided from the external environment and may be as simple as either a "0" or a "-1" corresponding to "success" or "failure," respectively.

In our on-line learning control design, the controller is "naive" when it just starts to control, namely the action network and the critic network are both randomly initialized in their weights/parameters. Once a system state is observed, an action will be subsequently produced based on the parameters in the action network. A "better" control value under the specific system state will lead to a more balanced equation of the

principle of optimality. This set of system operations will be reinforced through memory or association between states and control output in the action network. Otherwise, the control value will be adjusted through tuning the weights in the action network in order to make the equation of the principle of optimality more balanced.

To be more quantitative, consider the critic network as depicted in Fig. 1. The output of the critic element, the J function, approximates the discounted total reward-to-go. Specifically, it approximates $R(t)$ at time t given by

$$R(t) = r(t+1) + \alpha r(t+2) + \cdots \tag{2}$$

where $R(t)$ is the future accumulative reward-to-go value at time t, α is a discount factor for the infinite-horizon problem ($0 < \alpha < 1$). We have used $\alpha = 0.95$ in our implementations. $r(t+1)$ is the external reinforcement value at time $t+1$.

A. The Critic Network

The critic network is used to provide $J(t)$ as an approximate of $R(t)$ in (2). We define the prediction error for the critic element as

$$e_c(t) = \alpha J(t) - [J(t-1) - r(t)] \tag{3}$$

and the objective function to be minimized in the critic network is

$$E_c(t) = \tfrac{1}{2}\, e_c^2(t). \tag{4}$$

The weight update rule for the critic network is a gradient-based adaptation given by

$$\mathbf{w}_c(t+1) = \mathbf{w}_c(t) + \Delta \mathbf{w}_c(t) \tag{5}$$

$$\Delta \mathbf{w}_c(t) = l_c(t) \left[-\frac{\partial E_c(t)}{\partial \mathbf{w}_c(t)} \right] \tag{6}$$

$$\frac{\partial E_c(t)}{\partial \mathbf{w}_c(t)} = \left[-\frac{\partial E_c(t)}{\partial J(t)} \frac{\partial J(t)}{\partial \mathbf{w}_c(t)} \right] \tag{7}$$

where $l_c(t) > 0$ is the learning rate of the critic network at time t, which usually decreases with time to a small value, and $\mathbf{w}_c$ is the weight vector in the critic network.

B. The Action Network

The principle in adapting the action network is to indirectly backpropagate the error between the desired ultimate objective, denoted by U_c, and the approximate J function from the critic network. Since we have defined "0" as the reinforcement signal for "success," U_c is set to "0" in our design paradigm and in our following case studies. In the action network, the state measurements are used as inputs to create a control as the output of the network. In turn, the action network can be implemented by either a linear or a nonlinear network, depending on the complexity of the problem. The weight updating in the action network can be formulated as follows. Let

$$e_a(t) = J(t) - U_c(t). \tag{8}$$

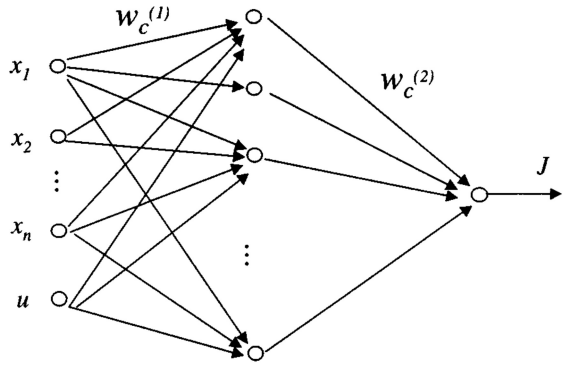

Fig. 2. Schematic diagram for the implementation of a nonlinear critic network using a feedforward network with one hidden layer.

The weights in the action network are updated to minimize the following performance error measure:

$$E_a(t) = \tfrac{1}{2}\, e_a^2(t). \tag{9}$$

The update algorithm is then similar to the one in the critic network. By a gradient descent rule

$$\mathbf{w}_a(t+1) = \mathbf{w}_a(t) + \Delta \mathbf{w}_a(t) \tag{10}$$

$$\Delta \mathbf{w}_a(t) = l_a(t) \left[-\frac{\partial E_a(t)}{\partial \mathbf{w}_a(t)} \right] \tag{11}$$

$$\frac{\partial E_a(t)}{\partial \mathbf{w}_a(t)} = \frac{\partial E_a(t)}{\partial J(t)} \frac{\partial J(t)}{\partial u(t)} \frac{\partial u(t)}{\partial \mathbf{w}_a(t)} \tag{12}$$

where $l_a(t) > 0$ is the learning rate of the action network at time t, which usually decreases with time to a small value, and $\mathbf{w}_a$ is the weight vector in the action network.

C. On-Line Learning Algorithms

Our on-line learning configuration introduced above involves two major components in the learning system, namely the action network and the critic network. In the following, we devise learning algorithms and elaborate on how learning takes place in each of the two modules.

In our NDP design, both the action network and the critic network are nonlinear multilayer feedforward networks. In our designs, one hidden layer is used in each network. The neural network structure for the nonlinear multilayer critic network is shown in Fig. 2.

In the critic network, the output $J(t)$ will be of the form

$$J(t) = \sum_{i=1}^{N_h} w_{c_i}^{(2)}(t) p_i(t) \tag{13}$$

$$p_i(t) = \frac{1 - \exp^{-q_i(t)}}{1 + \exp^{-q_i(t)}}, \qquad i = 1, \cdots, N_h \tag{14}$$

$$q_i(t) = \sum_{j=1}^{n+1} w_{c_{ij}}^{(1)}(t) x_j(t), \quad i = 1, \cdots, N_h \tag{15}$$

where

q_i	ith hidden node input of the critic network;
p_i	corresponding output of the hidden node;
N_h	total number of hidden nodes in the critic network;

$n+1$ total number of inputs into the critic network including the analog action value $u(t)$ from the action network.

By applying the chain rule, the adaptation of the critic network is summarized as follows.

1) $\Delta \mathbf{w}_c^{(2)}$ (hidden to output layer)

$$\Delta w_{c_i}^{(2)}(t) = l_c(t)\left[-\frac{\partial E_c(t)}{\partial w_{c_i}^{(2)}(t)}\right] \tag{16}$$

$$\frac{\partial E_c(t)}{\partial w_{c_i}^{(2)}(t)} = \frac{\partial E_c(t)}{\partial J(t)}\frac{\partial J(t)}{\partial w_{c_i}^{(2)}(t)} = \alpha e_c(t) p_i(t). \tag{17}$$

2) $\Delta \mathbf{w}_c^{(1)}$ (input to hidden layer)

$$\Delta w_{c_{ij}}^{(1)}(t) = l_c(t)\left[-\frac{\partial E_c(t)}{\partial w_{c_{ij}}^{(1)}(t)}\right] \tag{18}$$

$$\frac{\partial E_c(t)}{\partial w_{c_{ij}}^{(1)}(t)} = \frac{\partial E_c(t)}{\partial J(t)}\frac{\partial J(t)}{\partial p_i(t)}\frac{\partial p_i(t)}{\partial q_i(t)}\frac{\partial q_i(t)}{\partial w_{c_{ij}}^{(1)}(t)} \tag{19}$$

$$= \alpha e_c(t) w_{c_i}^{(2)}(t)\left[\tfrac{1}{2}\left(1-p_i^2(t)\right)\right] x_j(t). \tag{20}$$

Now, let us investigate the adaptation in the action network, which is implemented by a feedforward network similar to the one in Fig. 2 except that the inputs are the n measured states and the output is the action $u(t)$. The associated equations for the action network are

$$u(t) = \frac{1-\exp^{-v(t)}}{1+\exp^{-v(t)}} \tag{21}$$

$$v(t) = \sum_{i=1}^{N_h} w_{a_i}^{(2)}(t) g_i(t) \tag{22}$$

$$g_i(t) = \frac{1-\exp^{-h_i(t)}}{1+\exp^{-h_i(t)}}, \qquad i = 1, \ldots, N_h \tag{23}$$

$$h_i(t) = \sum_{j=1}^{n} w_{a_{ij}}^{(1)}(t) x_j(t), \qquad i = 1, \ldots, N_h \tag{24}$$

where v is the input to the action node, and g_i and h_i are the output and the input of the hidden nodes of the action network, respectively. Since the action network inputs the state measurements only, there is no $(n+1)$th term in (24) as in the critic network [see (15) for comparison]. The update rule for the nonlinear multilayer action network also contains two sets of equations.

1) $\Delta \mathbf{w}_a^{(2)}$ (hidden to output layer)

$$\Delta w_{a_i}^{(2)}(t) = l_a(t)\left[-\frac{\partial E_a(t)}{\partial w_{a_i}^{(2)}(t)}\right] \tag{25}$$

$$\frac{\partial E_a(t)}{\partial w_{a_i}^{(2)}(t)} = \frac{\partial E_a(t)}{\partial J(t)}\frac{\partial J(t)}{\partial u(t)}\frac{\partial u(t)}{\partial v(t)}\frac{\partial v(t)}{\partial w_{a_i}^{(2)}(t)} \tag{26}$$

$$= e_a(t)\left[\tfrac{1}{2}\left(1-u^2(t)\right)\right] g_i(t) \sum_{i=1}^{N_h} \cdot \left[w_{c_i}^{(2)}(t)\,\tfrac{1}{2}\left(1-p_i^2(t)\right) w_{c_{i,\,n+1}}^{(1)}(t)\right]. \tag{27}$$

In the above equations, $\partial J(t)/\partial u(t)$ is obtained by changing variables and by chain rule. The result is the summation term. $w_{c_{i,\,n+1}}^{(1)}$ is the weight associated with the input element from the action network.

2) $\Delta \mathbf{w}_a^{(1)}$ (input to hidden layer)

$$\Delta w_{a_{ij}}^{(1)}(t) = l_a(t)\left[-\frac{\partial E_a(t)}{\partial w_{a_{ij}}^{(1)}(t)}\right], \tag{28}$$

$$\frac{\partial E_a(t)}{\partial w_{a_{ij}}^{(1)}(t)} = \frac{\partial E_a(t)}{\partial J(t)}\frac{\partial J(t)}{\partial u(t)}\frac{\partial u(t)}{\partial v(t)}\frac{\partial v(t)}{\partial g_i(t)}\frac{\partial g_i(t)}{\partial h_i(t)}\frac{\partial h_i(t)}{\partial w_{a_{ij}}^{(1)}(t)} \tag{29}$$

$$= e_a(t)\left[\tfrac{1}{2}\left(1-u^2(t)\right)\right] w_{a_i}^{(2)}(t)\left[\tfrac{1}{2}\left(1-g_i^2(t)\right)\right] x_j(t) \cdot \sum_{i=1}^{N_h}\left[w_{c_i}^{(2)}(t)\,\tfrac{1}{2}\left(1-p_i^2(t)\right) w_{c_{i,\,n+1}}^{(1)}(t)\right]. \tag{30}$$

Normalization is performed in both networks to confine the values of the weights into some appropriate range by

$$\mathbf{w}_c(t+1) = \frac{\mathbf{w}_c(t)+\Delta \mathbf{w}_c(t)}{\|\mathbf{w}_c(t)+\Delta \mathbf{w}_c(t)\|_1} \tag{31}$$

$$\mathbf{w}_a(t+1) = \frac{\mathbf{w}_a(t)+\Delta \mathbf{w}_a(t)}{\|\mathbf{w}_a(t)+\Delta \mathbf{w}_a(t)\|_1}. \tag{32}$$

In implementation, (17) and (20) are used to update the weights in the critic network and (27) and (30) are used to update the weights in the action network.

III. Performance Evaluation for Case Study One

The proposed NDP design has been implemented on a single cart-pole problem. To begin with, the self-learning controller has no prior knowledge about the plant but only on-line measurements. 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 learning controller is to provide a force (applied to the cart) of a fixed magnitude in either the right or the left direction so that the pole stands balanced and avoids hitting the track boundaries. The controller receives reinforcement only after the pole has fallen.

In order to provide the learning controller measured states as inputs to the action and the critic networks, the cart-pole system was simulated on a digital computer using a detailed model that includes all of the nonlinearities and reactive forces of the physical system such as frictions. Note that these simulated states would be the measured ones in real-time applications.

A. The Cart-Pole Balancing Problem

The cart-pole system used in the current study is the same as the one in [1].

$$\frac{d^2\theta}{dt^2} = \frac{g\sin\theta + \cos\theta[-F - ml\dot{\theta}^2\sin\theta + \mu_c\,\mathrm{sgn}(\dot{x})] - \dfrac{\mu_p\dot{\theta}}{ml}}{l\left(\dfrac{4}{3} - \dfrac{m\cos^2\theta}{m_c+m}\right)} \tag{33}$$

$$\frac{d^2x}{dt^2} = \frac{F + ml[\dot{\theta}^2\sin\theta - \ddot{\theta}\cos\theta] - \mu_c\mathrm{sgn}(\dot{x})}{m_c+m} \tag{34}$$

where

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

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

The nonlinear differential equations (33) and (34) are numerically solved by a fourth-order Runge–Kutta method. This model provides four state variables: 1) $x(t)$, position of the cart on the track; 2) $\theta(t)$, angle of the pole with respect to the vertical position; 3) $\dot{x}(t)$, cart velocity; 4) $\dot{\theta}(t)$, 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 600 000 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 $[-12^\circ, 12^\circ]$ and/or the cart is beyond the range of $[-2.4, 2.4]$ m in reference to the central position on the track. Note that although the force F applied to the cart is binary, the control $u(t)$ fed into the critic network as shown in Fig. 1 is continuous.

B. Simulation Results

Several experiments were conducted to evaluate the effectiveness of our learning control designs. The parameters used in the simulations are summarized in Table I with the proper notations defined in the following:

$l_c(0)$ initial learning rate of the critic network;
$l_a(0)$ initial learning rate of the action network;
$l_c(t)$ learning rate of the critic network at time t which is decreased by 0.05 every five time steps until it reaches 0.005 and it stays at $l_c(f) = 0.005$ thereafter;
$l_a(t)$ learning rate of the action network at time t which is decreased by 0.05 every five time steps until it reaches 0.005 and it stays at $l_a(f) = 0.005$ thereafter;
N_c internal cycle of the critic network;
N_a internal cycle of the action network;
T_c internal training error threshold for the critic network;
T_a internal training error threshold for the action network;
N_h number of hidden nodes.

Note that the weights in the action and the critic networks were trained using their internal cycles, N_a and N_c, respectively. That is, within each time step the weights of the two networks were updated for at most N_a and N_c times, respectively, or stopped once the internal training error threshold T_a and T_c have been met.

TABLE I
SUMMARY OF PARAMETERS USED IN OBTAINING THE RESULTS GIVEN IN TABLE II

Parameter	$l_c(0)$	$l_a(0)$	$l_c(f)$	$l_a(f)$	*
Value	0.3	0.3	0.005	0.005	*
Parameter	N_c	N_a	T_c	T_a	N_h
Value	50	100	0.05	0.005	6

TABLE II
PERFORMANCE EVALUATION OF NDP LEARNING CONTROLLER WHEN BALANCING A CART-POLE SYSTEM. THE SECOND COLUMN REPRESENTS THE PERCENTAGE OF SUCCESSFUL RUNS OUT OF 100. THE THIRD COLUMN DEPICTS THE AVERAGE NUMBER OF TRIALS IT TOOK TO LEARN TO BALANCE THE CART-POLE. THE AVERAGE IS TAKEN OVER THE SUCCESSFUL RUNS

Noise type	success rate	# of trials
Noise free	100%	6
Uniform 5% actuator	100%	8
Uniform 10% actuator	100%	14
Uniform 5% sensor	100%	32
Uniform 10% sensor	100%	54
Gaussian $\sigma^2 = 0.1$ sensor	100%	164
Gaussian $\sigma^2 = 0.2$ sensor	100%	193

To be more realistic, we have added both sensor and actuator noise to the state measurements and the action network output. Specifically, we implemented the actuator noise through $u(t) = u(t)+\rho$, where ρ is a uniformly distributed random variable. For the sensor noise, we experimented with adding both uniform and Gaussian random variables to the angle measurements θ. The uniform state sensor noise was implemented through $\theta = (1 +$ noise percentage$) \times \theta$. Gaussian sensor noise was zero mean with specified variance.

Our proposed configuration of neural dynamic programming has been evaluated and the results are summarized in Table II. The simulation results summarized in Table II were obtained through averaged runs. Specifically, 100 runs were performed to obtain the results reported here. Each run was initialized to random conditions in terms of network weights. If a run is successful, the number of trials it took to balance the cart-pole is then recorded. The number of trials listed in the table corresponds to the one averaged over all of the successful runs. Therefore there is a need to record the percentage of successful runs out of 100. This number is also recorded in the table. A good configuration is the one with a high percentage of successful runs as well as a low average number of trials needed to learn to perform the balancing task.

Fig. 3 shows a typical movement or trajectory of the pendulum angle under NDP controller for a successful learning trial. The system under consideration is not subject to any noise.

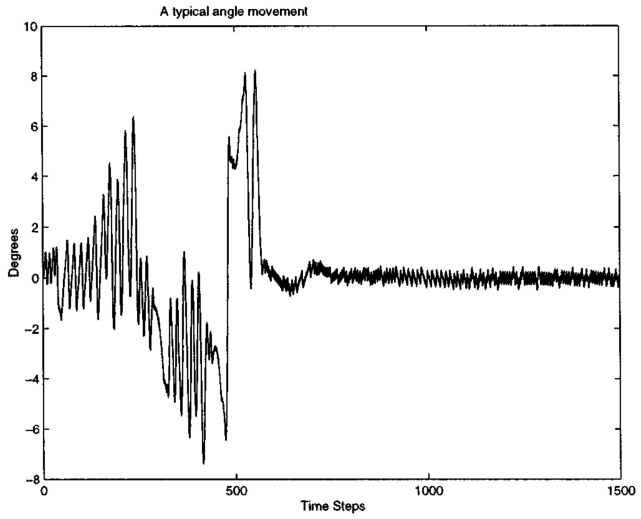

Fig. 3. A typical angle trajectory during a successful learning trial for the NDP controller when the system is free of noise.

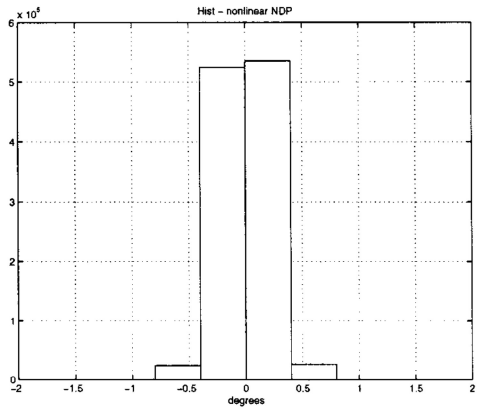

Fig. 4. Histogram of angle variations under the control of NDP on-linear learning mechanism in the single cart-pole problem. The system is free of noise in this case.

Fig. 4 represents a summary of typical statistics of the learning process in histograms. It contains vertical angle histograms when the system learns to balance the cart-pole using ideal state measurements without noise corruption.

IV. Performance Evaluation for Case Study Two

We now examine the performance of the proposed NDP design in a pendulum swing up and balancing task. The case under study is identical to the one in [15].

The pendulum is held by one end and can swing in a vertical plane. The pendulum is actuated by a motor that applied a torque at the hanging point. The dynamics of the pendulum is as follows:

$$\frac{d\omega}{dt} = \frac{3}{4ml^2}(F + mlg\sin(\theta)) \tag{35}$$

$$\frac{d\theta}{dt} = \omega \tag{36}$$

TABLE III
Performance Evaluation of NDP Learning Controller to Swing up and then Balance a Pendulum. The Second Column Represents the Percentage of Successful Runs out of 60. The Third Column Depicts the Average Number of Trials it Took to Learning to Successfully Perform the Task. The Average is Taken over the Successful Runs

reinforcement implementation	success rate	# of trials
Setting 1	100%	4.2
Setting 2	96%	3.5

where $m = 1/3$ and $l = 3/2$ are the mass and length of the pendulum bar, respectively, and $g = 9.8$ is the gravity. The action is the angular acceleration F and it is bounded between -3 and 3, namely, $F_{\min} = -3$, and $F_{\max} = 3$. A control action is applied every four time steps. The system states are the current angle θ and the angular velocity ω. This task requires the controller to not only swing up the bar but also to balance it at the top position. The pendulum initially sits still at $\theta = \pi$. This task is considered difficult in the sense that 1) no closed-form analytical solution exists for the optimal solution and complex numerical methods are required to compute it and 2) the maximum and minimum angular acceleration values are not strong enough to move the pendulum straight up from the starting state without first creating angular momentum [15].

In this study, a run consists of a maximum of 100 consecutive trials. It is considered successful if the last trial (trial number less than 100) of the run has lasted 800 time steps (with a step size of 0.05 s). Otherwise, if the NDP controller is unable to swing up and keep the pendulum balanced at the top within 100 trials (i.e., none of the 100 trails has lasted over 800 time steps), then the run is considered unsuccessful. In our simulations, a trial is either terminated at the end of the 800 time steps or when the angular velocity of the pendulum is greater than 2π, i.e., $\omega > 2\pi$.

In the following, we studied two implementation scenarios with different settings in reinforcement signal r. In **Setting 1**, $r = 0$ when the angle displacement is within 90° from the position of $\theta = 0$; $r = -0.4$ when the angle is in the rest half of the plane; and $r = -1$ when the angular velocity $\omega > 2\pi$. In **Setting 2**, $r = 0$ when the angle displacement is within 10° from the position of $\theta = 0$; $r = -0.4$ when the angle is in the remaining area of the plane; and $r = -1$ when the angular velocity $\omega > 2\pi$.

Our proposed NDP configuration is then used to perform the above described task. We have used the same configuration and the same learning parameters as those in the first case study. NDP controller performance is summarized in Table III. The simulation results summarized in the table were obtained through averaged runs. Specifically, 60 runs were performed to obtain the results reported here. Note that we have used more runs than that in [15] (which was 36) to generate the final result statistics. But we have kept every other simulation condition the same as that in the paper [15]. Each run was initialized to $\theta = \pi$ and $\omega = 0$. The number of trials listed in the table corresponds to the one averaged over all of the successful runs. The percentage of successful runs out of 60 was also recorded

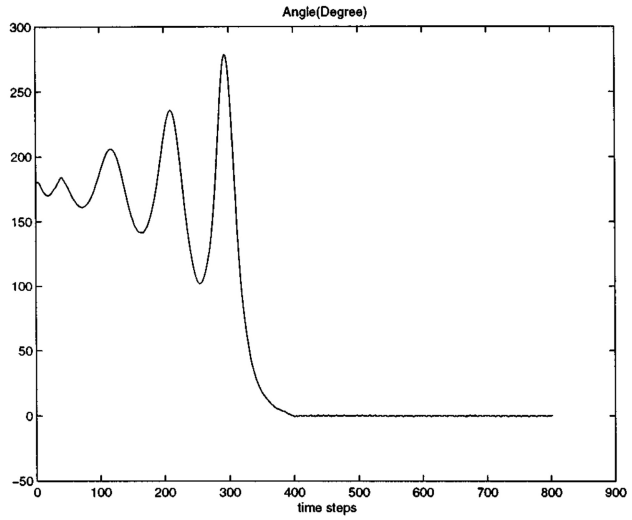

Fig. 5. A typical angle trajectory during a successful learning trial for the NDP controller in the pendulum swing up and balancing task. *Left*: the entire trial. *Right*: portion of the entire trajectory to show the stabilization process in detail.

in the table. Fig. 5 shows a typical trajectory of the pendulum angle under NDP controller for a successful learning trial. This trajectory is characteristic for both Setting 1 and Setting 2.

V. Performance Evaluation for Case Study Three

The NDP design introduced in the previous sections is now applied to a more complex on-line learning control problem than the single cart-pole balancing task, namely, the triple-link inverted pendulum problem with single control input. We have successfully implemented our proposed NDP configuration on this problem. The details of the implementation and results will be given in the following subsections.

A. Triple-Link Inverted Pendulum with Single Control Input

The system model for the triple-link problem is the same as that in [7]. Fig. 6 depicts the notation used in the state equations that govern the system.

The equation governing the system is

$$F(q)\frac{d^2q}{dt^2} = -G\left(q, \frac{dq}{dt}\right)\frac{dq}{dt} - H(q) + L(q, u) \qquad (37)$$

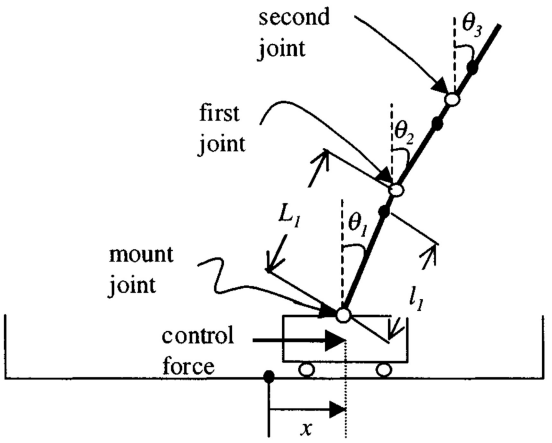

Fig. 6. Definition of notation used in the system equations for the triple-link inverted pendulum problem.

where the components are shown in the equation at the bottom of the page. Note that the μ's in $L(q, u)$ are Coulomb friction coefficients for links and they are not linearizable [7]. In our simulations, $\mu_x = 0.07$, and $\mu_1 = \mu_2 = \mu_3 = 0.003$. The A_i coefficients are system constants and are given in Table IV.

The parameters used in the system are defined as follows:

g	9.8 m/s^2, acceleration due to gravity;
M	1.014 kg, mass of the cart;
m_1	0.4506 kg, mass of the first link;
m_2	0.219 kg, mass of the second link;
m_3	0.0568 kg, mass of the third link;
l_1	0.37 m, the length from the mount joint to the center of gravity of the first link;
l_2	0.3 m, the length from the first joint to the center of gravity of the second link;
l_3	0.05 m, the length from the second joint to the center of gravity of the third link;
L_1	0.43 m, total length of the first link;
L_2	0.33 m, total length of the second link;
L_3	0.13 m, total length of the third link;
I_1	0.0042 kgm^2, mass moment of inertia of the first link about its center of gravity;
I_2	0.0012 kgm^2, mass moment of inertia of the second link about its center of gravity;

$$F(q) = \begin{bmatrix} A_1 & A_2\cos(\theta_1) & A_3\cos(\theta_2) & A_4\cos(\theta_3) \\ A_9\cos(\theta_1) & A_{10} & A_{11}\cos(\theta_1-\theta_2) & A_{12}\cos(\theta_1-\theta_3) \\ A_{18}\cos(\theta_2) & A_{19}\cos(\theta_1-\theta_2) & A_{20} & A_{21}\cos(\theta_2-\theta_3) \\ A_{28}\cos(\theta_3) & A_{29}\cos(\theta_1-\theta_3) & A_{30}\cos(\theta_2-\theta_3) & A_{31} \end{bmatrix}$$

$$G\left(q, \frac{dq}{dt}\right) = \begin{bmatrix} A_5 & A_6\sin(\theta_1)\dot{\theta}_1 & A_7\sin(\theta_2)\dot{\theta}_2 & A_8\sin(\theta_3)\dot{\theta}_3 \\ 0 & A_{13} & A_{14}\sin(\theta_1-\theta_2)\dot{\theta}_2 + A_{15} & A_{16}\sin(\theta_1-\theta_3)\dot{\theta}_3 \\ 0 & A_{22}\sin(\theta_1-\theta_2)\dot{\theta}_1 + A_{23} & A_{24} & A_{25}\sin(\theta_2-\theta_3)\dot{\theta}_3 + A_{26} \\ 0 & A_{33}\sin(\theta_1-\theta_3)\dot{\theta}_1 & A_{35}\sin(\theta_2-\theta_3)\dot{\theta}_2 + A_{36} & A_{32} \end{bmatrix}$$

$$q = \begin{bmatrix} x \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}, \quad H(q) = \begin{bmatrix} 0 \\ A_{17}\sin(\theta_1) \\ A_{27}\sin(\theta_2) \\ A_{34}\sin(\theta_3) \end{bmatrix}, \quad L(q, u) = \begin{bmatrix} K_s u - \operatorname{sgn}(x)\mu_x A_{37} \\ -\operatorname{sgn}(\theta_1)\mu_1 A_{38} \\ -\operatorname{sgn}(\theta_2)\mu_2 A_{39} \\ -\operatorname{sgn}(\theta_3)\mu_3 A_{40} \end{bmatrix}.$$

TABLE IV
SYSTEM CONSTANTS FOR THE TRIPLE LINK INVERTED PENDULUM PROBLEM

Constant	Value	Constant	Value
A_1	$M+m_1+m_2+m_3$	A_{21}	$m_3l_3L_2$
A_2	$m_1l_1+(m_2+m_3)L_1$	A_{22}	$-(m_2l_2+m_3L_2)L_1$
A_3	$m_2l_2+m_3L_2$	A_{23}	$-C_2$
A_4	m_3l_3	A_{24}	C_2+C_3
A_5	C_c	A_{25}	$m_3l_3L_2$
A_6	$-m_1l_1-(m_2+m_3)L_1$	A_{26}	$-C_3$
A_7	$-(m_2l_2+m_3L_2)$	A_{27}	$-g(m_2l_2+m_3L_2)$
A_8	$-m_3l_3$	A_{28}	m_3l_3
A_9	$m_1l_1+(m_2+m_3)L_1$	A_{29}	$m_3l_3L_1$
A_{10}	$I_1+m_2l_1^2+(m_2+m_3)L_1^2$	A_{30}	$m_3l_3L_2$
A_{11}	$(m_2l_2+m_3L_2)L_1$	A_{31}	$I_3+m_3l_3^2$
A_{12}	$m_3l_3L_1$	A_{32}	C_3
A_{13}	C_1+C_2	A_{33}	$-m_3l_3L_1$
A_{14}	$(m_2l_2+m_3L_2)L_1$	A_{34}	$-gm_3l_3$
A_{15}	$-C_2$	A_{35}	$-m_3l_3L_2$
A_{16}	$m_3l_3L_1$	A_{36}	$-C_3$
A_{17}	$-g(m_1l_1+m_2L_1+m_3L_1)$	A_{37}	1.3
A_{18}	$m_2l_2+m_3L_2$	A_{38}	0.506
A_{19}	$(m_2l_2+m_3L_2)L_1$	A_{39}	0.219
A_{20}	$I_2+m_3L_2^2+m_2l_2^2$	A_{40}	0.568

I_3 0.000 106 09 kgm^2, mass moment of inertia of the third link about its center of gravity;
C_c 5.5 Nms, dynamic friction coefficient between the cart and the track;
C_1 0.000 268 75 Nms, dynamic friction coefficient for the first link;
C_2 0.000 268 75 Nms, dynamic friction coefficient for the second link;
C_3 0.000 268 75 Nms, dynamic friction coefficient for the third link.

The only control u (in volts) generated by the action network is converted into force by an analog amplifier through a conversion gain K_s (in Newtons/volt). In simulations, $K_s = 24.7125$ N/V. Each link can only rotate in the vertical plane about the axis of a position sensor fixed to the top of each link. The sampling time interval is chosen to be 5 ms. From the nonlinear dynamical equation in (37), the state-space model can be described as follows:

$$\dot{Q}(t) = f(Q(t), u(t)) \tag{38}$$

with

$$f(Q(t), u(t)) = \begin{bmatrix} \mathbf{0}_{4\times4} & \mathbf{I}_{4\times4} \\ \mathbf{0}_{4\times4} & -F^{-1}(Q(t))G(Q(t)) \end{bmatrix} Q(t) + \begin{bmatrix} \mathbf{0}_{4\times4} \\ -F^{-1}(Q(t))[H(Q(t))-L(Q(t), u(t))] \end{bmatrix}$$

and

$$Q(t) = \left[x(t)\ \theta_1(t)\ \theta_2(t)\ \theta_3(t)\ \dot{x}(t)\ \dot{\theta}_1(t)\ \dot{\theta}_2(t)\ \dot{\theta}_3(t)\right]^T.$$

TABLE V
SUMMARY OF PARAMETERS USED IN OBTAINING THE RESULTS GIVEN IN TABLE VI FOR THE TRIPLE-LINK INVERTED PENDULUM PROBLEM

Parameter	$l_c(0)$	$l_a(0)$	$l_c(f)$	$l_a(f)$	*
Value	0.8	0.8	0.001	0.001	*
Parameter	N_c	N_a	T_c	T_a	N_h
Value	10	200	0.01	0.001	14

There are eight state variables in this model: 1) $x(t)$, position of the cart on the track; 2) $\theta_1(t)$, vertical angle of the first link joint to the cart; 3) $\theta_2(t)$, vertical angle of the second link joint to the first link; 4) $\theta_3(t)$, vertical angle of the third link joint to the second link; 5) $\dot{x}(t)$, cart velocity; 6) $\dot{\theta}_1(t)$, angular velocity of $\theta_1(t)$; 7) $\dot{\theta}_2(t)$, angular velocity of $\theta_2(t)$; and 8) $\dot{\theta}_3(t)$, angular velocity of $\theta_3(t)$.

In the triple-link inverted pendulum problem, a run consists of a maximum of 3000 consecutive trials. Similar to the single cart-pole balancing problem, a run is considered successful if the last trial of the run lasts 600 000 time steps. However, now a unit time step is 5 ms instead of 20 ms as in the single cart-pole problem. The constraints for the reinforcement learning are: 1) the cart track extends 1.0 m to both ends from the center position; 2) the voltage applied to the motor is within [−30, 30] V range; and 3) each link angle should be within the range of [−20°, 20°] with respect to the vertical axis. In our simulations, condition 2 is assured to be satisfied by using a sigmoid function at the output of the action node. For conditions 1) and 3), if either one fails or both fail, the system provides an indicative signal $r = -1$ at the moment of failure, otherwise $r = 0$ all the time. Several experiments were conducted to evaluate the effectiveness of the proposed learning control designs. The results are reported in the following section.

B. Simulation Results

Note that the triple-link system is highly unstable. To see this, the positive eigenvalues of the linearized system model are far away from zero (the largest is around 10.0). In obtaining the linearized system model, the Coulomb friction coefficients are assumed to be negligible. Besides, the system dynamics changes fast. It requires a sampling time below 10 ms.

Since the analog output from the action network is directly fed into the system this time, the controller is more sensitive to the actuator noise than the one in the single cart-pole where a binary control is applied. Experiments conducted in this paper include evaluations of NDP controller performance under uniform actuator noise, uniform or Gaussian sensor noise, and the case without noise.

Before the presentation of our results, the learning parameters are summarized in Table V. Simulation results are tabulated in Table VI. Conventions such as noise type and how they are included in the simulations are described in Section III-B.

Fig. 7 shows typical angle trajectories of the triple-link angles under NDP control for a successful learning trial. The system under consideration is not subject to any noise. The corresponding control force trajectory is also given in Fig. 8.

TABLE VI
PERFORMANCE EVALUATION OF NDP LEARNING CONTROLLER WHEN BALANCING A TRIPLE-LINK INVERTED PENDULUM. THE SECOND COLUMN REPRESENTS THE PERCENTAGE OF SUCCESSFUL RUNS OUT OF 100. THE THIRD COLUMN DEPICTS THE AVERAGE NUMBER OF TRIALS IT TOOK TO LEARNING TO BALANCE THE CART POLE. THE AVERAGE IS TAKEN OVER THE SUCCESSFUL RUNS

Noise type	success rate	# of trials
None	97%	1194
Uniform 5% actuator	92%	1239
Uniform 10% actuator	84%	1852
Uniform 5% sensor on θ_1	89%	1317
Uniform 10% sensor on θ_1	80%	1712
Gaussian sensor on θ_1 variance = 0.1	85%	1508
Gaussian sensor on θ_1 variance = 0.2	76%	1993

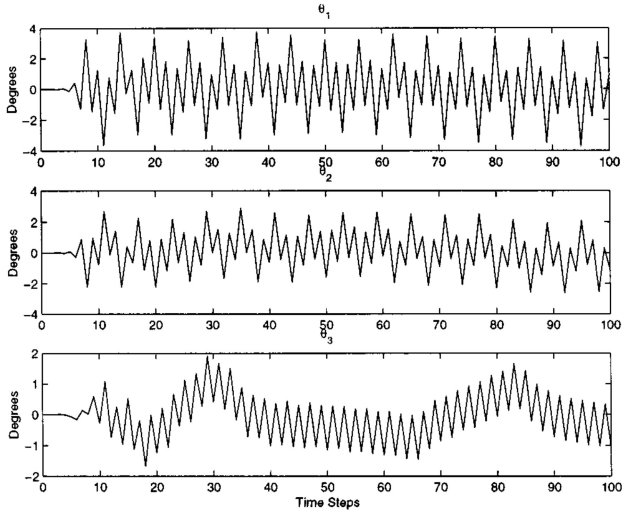

Fig. 7. Typical angle trajectories of the triple-link angles during a successful learning trial using on-line NDP control when the system is free of noise.

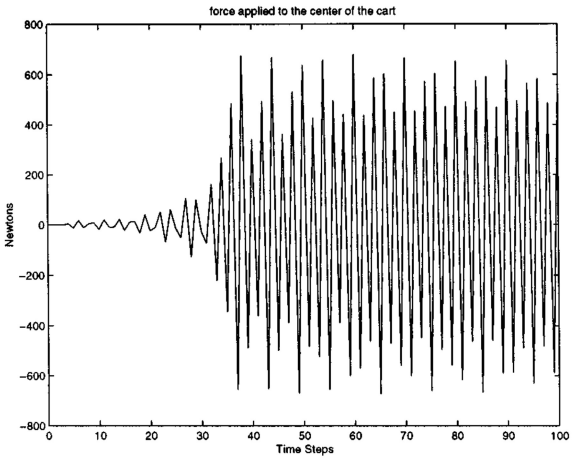

Fig. 8. The force (in Newton) trajectory, which is converted from control u into voltage, applied to the center of the cart for balancing the triple-link inverted pendulum, corresponding to the angle trajectory in Fig. 7.

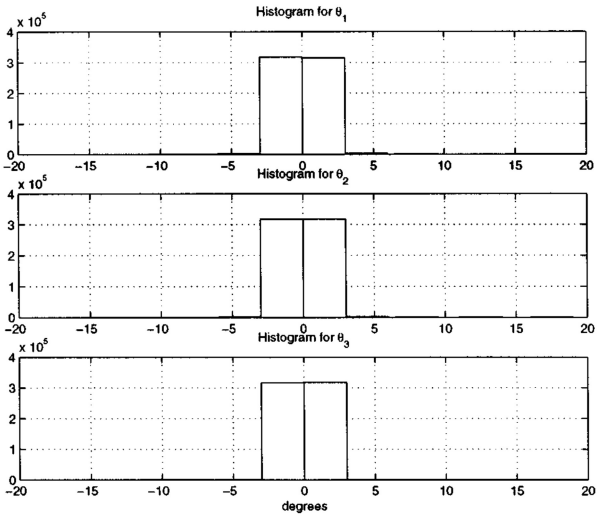

Fig. 9. Histogram of the triple-link angle variations when the system is free of noise.

Fig. 9 represents a summary of statistics of the learning process in histograms.

The results presented in this case study have again demonstrated the validity of the proposed NDP designs. The major characteristics of the learning process for this more complex system are still similar to the single cart-pole problem in many ways. It is worth mentioning that the NDP controlled angle variations are significantly smaller than those using nonlinear control system design as in [7].

VI. ANALYTICAL CHARACTERISTICS OF ON-LINE NDP LEARNING PROCESS

This section is dedicated to expositions of analytical properties of the on-line learning algorithms in the context of NDP. It is important to note that in contrast to usual neural-network applications, there is no readily available training sets of input–output pairs to be used for approximating J^* in the sense of least squares fit in NDP applications. Both the control action u and the approximated J function are updated according to an error function that changes from one time step to the next. Therefore, the convergence argument for the steepest descent algorithm does not hold valid for any of the two networks, action or critic. This results in a simulation approach to evaluate the cost-to-go function J for a given control action u. The on-line learning takes place aiming at iteratively improving the control policies based on simulation outcomes. This creates analytical and computational difficulties that do not arise in a more typical neural-network training context.

Some analytical results in terms of approximating J function was obtained by Tsitsiklis [20] where a linear in parameter function approximator was used to approximate the J function. The limit of convergence was characterized as the solution to a set of interpretable linear equations, and a bound is placed on the resulting approximation error.

It is worth pointing out that the existing implementations of NDP are usually computationally very intensive [4], and often require a considerable amount of trial and error. Most of the computations and experimentations with different approaches were conducted off-line.

In the following, we try to provide some analytical insight on the on-line learning process for our proposed NDP designs. Specifically we will use the stochastic approximation argument to reveal the asymptotic performance of our on-line NDP learning algorithms in an averaged sense for the action and the critic networks under certain conditions.

A. Stochastic Approximation Algorithms

The original work in recursive stochastic algorithms was introduced by Robbins and Monro, who developed and analyzed a recursive procedure for finding the root of a real-valued function $g(w)$ of a real variable w [14]. The function is not known, but noise-corrupted observations could be taken at values of w selected by the experimenter.

A function $g(w)$ with the form $g(w) = Ex[f(w)]$ ($Ex[\,]$ is the expectation operator) is called a regression function of $f(w)$, and conversely, $f(w)$ is called a sample function of $g(w)$. The following conditions are needed to obtain the Robbins–Monro algorithm [14].

(C1) $g(w)$ has a single root w^*, $g(w^*) = 0$, and

$$g(w) < 0 \quad \text{if } w < w^*$$
$$g(w) > 0 \quad \text{if } w > w^*.$$

This is assumed with little loss of generality since most functions of a single root not satisfying this condition can be made to do so by multiplying the function by -1.

(C2) The variance of $f(w)$ from $g(w)$ is finite

$$\sigma^2(w) = Ex[g(w) - f(w)]^2 < \infty. \tag{39}$$

(C3)

$$|g(w)| < B_1|w - w^*| + B_0 < \infty. \tag{40}$$

(C3) is a very mild condition. The values of B_1 and B_0 need not be known to prove the validity of the algorithm. As long as the root lies in some finite interval, the existence of B_1 and B_0 can always be assumed.

If the conditions (C1) through (C3) are satisfied, the algorithm due to Robbins and Monro can be used to iteratively seek the root w^* of the function $g(w)$:

$$w(t+1) = w(t) - l(t)f[w(t)] \tag{41}$$

where $l(t)$ is a sequence of positive numbers which satisfy the following conditions:

$$\begin{aligned} &1) \quad \lim_{t\to\infty} l(t) = 0 \\ &2) \quad \sum_{t=0}^{\infty} l(t) = \infty \\ &3) \quad \sum_{t=0}^{\infty} l^2(t) < \infty. \end{aligned} \tag{42}$$

Furthermore, $w(t)$ will converge toward w^* in the mean square error sense and with probability one, i.e.,

$$\lim_{t\to\infty} Ex\left[\|w(t) - w^*\|^2\right] = 0 \tag{43}$$

$$\text{Prob}\left\{\lim_{t\to\infty} w(t) = w^*\right\} = 1. \tag{44}$$

The convergence with probability one in (44) is also called convergence almost truly.

In this paper, the Robbins–Monro algorithm is applied to optimization problems [10]. In that setting, $g(w) = \partial E/\partial w$, where E is an objective function to be optimized. If E has a local optimum at w^*, $g(w)$ will satisfy the condition (C1) locally at w^*. If E has a quadratic form, $g(w)$ will satisfy the condition (C1) globally.

B. Convergence in Statistical Average for the Action and the Critic Networks

Neural dynamic programming is still in its early stage of development. The problem is not trivial due to several consecutive learning segments being updated simultaneously. A practically effective on-line learning mechanism and a step by step analytical guide for the learning process do not co-exist at this time. This paper is dedicated to reliable implementations of NDP algorithms for solving a general class of on-line learning control problem. As demonstrated in previous sections, experimental results in this direction are very encouraging. In the present section, we try to provide some asymptotic convergence results for each component of the NDP system. The Robbins–Monro algorithm provided in the previous section is the main tool to obtain results in this regard. Throughout this paper, we have implied that the state measurements are samples of a continuous state space. Specifically we will assume without loss of generality that the input $X_j \in \mathcal{X} \subset \mathcal{R}^n$ has discrete probability density $p(X) = \sum_{j=1}^{N} p_j \delta(X - X_j)$, where $\delta(\,)$ is the delta function.

In the following, we analyze one component of the NDP system at a time. When one component (e.g., the action network) is under consideration, the other component (e.g., the critic network) is considered to have completed learning, namely their weights do not change any more.

To examine the learning process taking place in the action network, we define the following objective function for the action network:

$$\begin{aligned} \tilde{E}_a &= \tfrac{1}{2} \sum_i p_i [J(X_i) - U_c]^2 \\ &= \tfrac{1}{2} Ex[(J - U_c)^2]. \end{aligned} \tag{45}$$

It can be seen that (45) is an "averaged" error square between the estimated J and a final desired value U_c. To contrast this notion, (9) is an "instantaneous" error square between the two. To obtain a (local) minimum for the "averaged" error measure in (45), we can apply the Robbins–Monro algorithm by first taking a derivative of this error with respect to the parameters, which are the weights in the action network in this case. Let

$$\tilde{e}_a = J - U_c. \tag{46}$$

Since J is smooth in $\tilde{\mathbf{w}}_a$, and $\tilde{\mathbf{w}}_a$ belongs to a bounded set, the derivative of $\tilde{E}_a$ with respect to the weights of the action network is then of the form:

$$\frac{\partial \tilde{E}_a}{\partial \tilde{\mathbf{w}}_a} = Ex\left[\tilde{e}_a \frac{\partial \tilde{e}_a}{\partial \tilde{\mathbf{w}}_a}\right]. \tag{47}$$

According to the Robbins–Monro algorithm, the root (can be a local root) of $\partial \tilde{E}_a/\partial \tilde{\mathbf{w}}_a$ as a function of $\tilde{\mathbf{w}}_a$ can be obtained by

the following recursive procedure, if the root exists and if the step size $l_a(t)$ meets all the requirements described in (42):

$$\tilde{\mathbf{w}}_a(t+1) = \tilde{\mathbf{w}}_a(t) - l_a(t)\left[\tilde{e}_a \frac{\partial \tilde{e}_a}{\partial \tilde{\mathbf{w}}_a}\right]. \tag{48}$$

Equation (46) may be considered as an instantaneous error between a sample of the J function and the desired value U_c. Therefore, (48) is equivalent to the update equation for the action network given in (10)–(12). From this viewpoint, the on-line action network updating rule of (10)–(12) is actually converging to a (local) minimum of the error square between the J function and the desired value U_c in a statistical average sense. Or in other words, even though (10)–(12) represent a reduction in instantaneous error square at each iterative time step, the action network updating rule asymptotically reaches a (local) minimum of the statistical average of $(J - U_c)^2$.

By the same token, we can construct a similar framework to describe the convergence of the critic network. Recall that the residual of the principle of optimality equation to be balanced by the critic network is of the following form:

$$e_c(t) = \alpha J(t) - J(t-1) + r(t). \tag{49}$$

And the "instantaneous" error square of this residual is given as

$$E_c(t) = \tfrac{1}{2}\, e_c^2(t). \tag{50}$$

Instead of the "instantaneous" error square, let

$$\tilde{E}_c = Ex[E_c] \tag{51}$$

and assume that the expectation is well defined over the discrete state measurements. The derivative of $\tilde{E}_c$ with respect to the weights of the critic network is then of the form

$$\frac{\partial \tilde{E}_c}{\partial \tilde{\mathbf{w}}_c} = Ex\left[e_c \frac{\partial e_c}{\partial \tilde{\mathbf{w}}_c}\right]. \tag{52}$$

According to the Robbins–Monro algorithm, the root (can be a local root) of $\partial \tilde{E}_c / \partial \tilde{\mathbf{w}}_c$ as a function of $\tilde{\mathbf{w}}_c$ can be obtained by the following recursive procedure, if the root exists and if the step size $l_c(t)$ meets all the requirements described in (42):

$$\tilde{\mathbf{w}}_c(t+1) = \tilde{\mathbf{w}}_c(t) - l_c(t)\left[e_c \frac{\partial e_c}{\partial \tilde{\mathbf{w}}_c}\right]. \tag{53}$$

Therefore, (53) is equivalent to the update rule for the critic network given in (5)–(7). From this viewpoint, the on-line critic network update rule of (5)–(7) is actually converging to a (local) minimum of the residual square of the equation of the principle of optimality in a statistical average sense.

VII. Discussion and Conclusion

This paper focuses on providing a systematic treatment of an NDP design paradigm, from architecture, to algorithm, analytical insights, and case studies. The results presented in this paper represent an effort toward generic and robust implementations of on-line NDP designs. Our design presented in the paper has, in principle, advanced in several aspects from the existing results. First, our proposed configuration is simpler than adaptive critic designs. Even though it is very similar to ADHDP, we have provided a mechanism that does not require a prediction model. The ADHDP design in adaptive critics either ignores the predictive model that results in nontrivial training errors or includes an additional prediction network that results in additional complexities in learning. Also, our design is robust in the sense that it is insensitive to parameters such as initial weights in the action and/or critic network, the values for the ultimate objective, U_c, etc. Key learning parameters are listed clearly in the paper for reproduction of our NDP design. Second, our NDP design has been tested on cases and has shown robustness in different tasks. The triple-link inverted pendulum balancing is a difficult nonlinear control problem in many ways. Our results measured by tightness of the vertical angles to the upright position is much improved over the traditional nonlinear control system designs used in [7]. When compared to the original RL designs by Barto [1], our on-line learning mechanism is more robust when subject to various noise, faster in learning to perform the task, and requires less number of trials to learn the task. The designs in [15] generally require more free parameters than our NDP design. Plus, our reinforcement signal is simpler than that in [15]. However our simulation results have demonstrated very robust performance by the proposed NDP configuration. Third, the fundamental guideline for the on-line learning algorithms is the stochastic gradient. We therefore have an estimate of the asymptotic convergence property for our proposed on-line NDP designs under some conditions in statistical sense. This provides more insight to the on-line learning NDP designs.

Next, we would like to share some of our experience on ways to handle large scale problems or the issue of scalability in NDP design. We developed and tested several designs that included a self-ganizing map (SOM) prior to the action network. The SOM takes the system states as inputs and produces an organized or reduced dimension expression of these input states to be passed onto the action network. In our experiments, we have used standard SOM algorithm developed by Kohonen [9]. The SOM as a state classifier can compress state measurements into a smaller set of vectors represented by the weights of the SOM network. On the other hand, it introduces quantization error that degrades the overall system performance accordingly. In terms of learning efficiency with the added SOM component, on one hand it reduces the feature space that the action network is exposed to and therefore it contributes toward a reduction in learning complexity in the action network. But on the other hand, adding a new network such as SOM (using the standard Kohonen training) can introduce tremendous computation burden on the overall learning system.

To have a more quantitative understanding of the effect of SOM on the overall learning system performance, we performed several learning tasks. First, we studied the system performance on the single cart-pole problem. We added an SOM network right in front of the action network. Refer to Fig. 1, the input to the SOM is the state $X(t)$ and the weight vectors of the SOM become the inputs to the action network. We then performed the same set of experiments as those documented in Table II. When compared to the results in the table without the SOM element in the learning system, we have observed much degraded performance by adding the SOM element. Specifically, we saw a 7.1% decreases in the success rate through the seven different cases (with different noise type), and a 142% increase in the number

of trials needed to learn to balance the single cart-pole. Apparently the quantization error and the increased learning burden from SOM are contributing to this performance degradation.

We then performed a similar set of experiments on the triple-link inverted pendulum. Again, significant performance degradation was observed. With a closer examination at the quantization error, we realized that it was as high as 0.2 rad which corresponds to about 11°! It is soon realized that this high quantization error was due to the fast dynamics inherent in the triple-link inverted pendulum, which results in significant variances in the derivatives of the state measurements. The imbalance between the original state measurements (such as the angles) and the derivatives of the angles (the angular velocity) has created high demand in SOM resolution, or if not, significant quantization errors will result. To circumvent this, we divided the action network inputs into two sections. First, an SOM is employed to compress the state measurements, x, θ_1, θ_2, and θ_3, into a finite set of four-dimensional weight vectors, which forms one set of the inputs to the action network. The other set of inputs to the action network comes directly from the derivatives of the state measurements, namely, $\dot{x}$, $\dot{\theta}_1$, $\dot{\theta}_2$, $\dot{\theta}_3$. With such an implementation, the quantization error from the SOM is only about 0.04 rad (or, equivalently, 2°) on average. The overall learning performance was improved also. Specifically, we only see 3.8% decrease in success rate through the seven cases when compared to those in Table VI, and only 1.6% increase in the number of trials needed to learn to balance the triple-link inverted pendulum.

Note that, we believe there is still room for improving the quantization error and the learning speed inherent with the SOM network. One interesting observation from our experiments with SOM is that the learning speed for a controller with the SOM is not that much slower than the one without the SOM, especially in a complex task. The effect of reducing learning complexity has surfaced by using SOM as a compressor. It is quite convincing from our experiments on the single cart-pole that the SOM has added a tremendous computation overhead for the overall learning speed, especially for a relatively simple and low-dimensional problem. However, when one deals with a more complex system with more state variables, although the SOM still introduces computation overhead, its advantage of reducing the number of training patterns for the action network becomes apparent, which as a matter of fact may have reduced the overall learning complexity.

As discussed earlier, a good learning controller should be the one that learns to perform a task quickly, and also, learns almost all the time with a high percentage of success rate. Another factor that may not be explicitly present in the reinforcement signal is the degree of meeting the performance requirement. In all case studies, however, it is quite intriguing to see that the learning controllers are not only trying to balance the poles, but also trying to maintain the poles as centered as possible.

In summary, our results are very encouraging toward designing truly automatic and adaptive learning systems. But this paper only represents a start of this highly challenging task. Many more issues are still open such as how to characterize the overall learning process, instead of isolated and asymptotic guidelines, and how to interpret the controller output and system performance in a more systematic manner. As foreseeable goals, for one thing, the learning speed of the SOM can be improved potentially, and the overall system design should be tested on many more complex systems.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful comments. They also would like to thank L. Yang for carrying out the simulation study of Case II.

REFERENCES

[1] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuron like adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 13, pp. 834–847, 1983.
[2] R. Bellman and S. Dreyfus, *Applied Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1962.
[3] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*. Englewood Cliffs, NJ: Prentice-Hall, 1987.
[4] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Belmont, MA: Athena, 1996.
[5] T. Borgers and R. Sarin, "Learning through reinforcement and replicator dynamics," *J. Economic Theory*, vol. 77, no. 1, pp. 1–17, 1997.
[6] J. Dalton and S. N. Balakrishnan, "A neighboring optimal adaptive critic for missile guidance," *Math. Comput. Modeling*, vol. 23, no. 1, pp. 175–188, 1996.
[7] K. D. Eltohamy and C.-Y. Kuo, "Nonlinear optimal control of a triple link inverted pendulum with single control input," *Int. J. Contr.*, vol. 69, no. 2, pp. 239–256, 1998.
[8] D. Kirk, *Optimal Control Theory: An Introduction*. Englewood Cliffs, NJ: Prentice-Hall, 1970.
[9] T. Kohonen, *Self-Organizing Map*. Heidelberg, Germany: Springer-Verlag, 1995.
[10] H. J. Kushner and G. G. Yin, *Stochastic Approximation Algorithms and Applications*. New York: Springer-Verlag, 1997.
[11] R. E. Larson, "A survey of dynamic programming computational procedures," *IEEE Trans. Automat. Contr.*, vol. AC-12, pp. 767–774, 1967.
[12] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch II, "Adaptive critic designs: A case study for neuro-control," *Neural Networks*, vol. 8, pp. 1367–1372, 1995.
[13] D. V. Prokhorov and D. C. Wunsch II, "Adaptive critic designs," *IEEE Trans. Neural Networks*, vol. 8, pp. 997–1007, Sept. 1997.
[14] H. Robbins and S. Monro, "A stochastic approximation method," *Ann. Math. Statist.*, vol. 22, pp. 400–407, 1951.
[15] "COINS Tech. Rep.," Univ. Mass., Amherst, 96–88, Dec. 1996.
[16] R. S. Sutton, "Learning to predict by the methods of temporal difference," *Machine Learning*, vol. 3, pp. 9–44, 1988.
[17] G. Tesauro, "Neurogammon: A neural-network backgammon program," in *Proc. Int. Joint Conf. Neural Networks*, San Diego, CA, 1990, pp. 33–40.
[18] ——, "Practical issues in temporal difference learning," *Machine Learning*, vol. 8, pp. 257–277, 1992.
[19] ——, "TD-Gammon, a self-teaching backgammon program achieves master-level play," *Neural Comput.*, vol. 6, 1994.
[20] J. N. Tsitsiklis and B. Van Roy, "An analysis of temporal-difference learning with function approximation," *IEEE Trans. Automat. Contr.*, vol. 42, pp. 674–690, May 1997.
[21] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 257–277, 1992.
[22] P. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General System Yearbook*, vol. 22, pp. 25–38, 1977.
[23] ——, "A menu of design for reinforcement learning over time," in *Neural Networks for Control*, W. T. Miller III, R. S. Sutton, and P. J. Werbos, Eds. Cambridge, MA: MIT Press, 1990, ch. 3.
[24] ——, "Neuro-control and supervised learning: An overview and valuation," in *Handbook of Intelligent Control*, D. White and D. Sofge, Eds. New York: Van Nostrand, 1992.
[25] ——, "Approximate dynamic programming for real-time control and neural modeling," in *Handbook of Intelligent Control*, D. White and D. Sofge, Eds. New York: Van Nostrand, 1992.

[26] ——, "Tutorial on neurocontrol, control theory and related techniques: from back propagation to brain-like intelligent systems," in *Proc. 12th Int. Conf. Math Comput Modeling Sci. Comput.*, 1999, www.iamcm.org/pwerbos/.

Jennie Si (S'90–M'91–SM'99) received the B.S. and M.S. degrees from Tsinghua University, Beijing, China, in 1985 and 1988, respectively, and the Ph.D. degree from University of Notre Dame, Notre Dame, IN, in 1991.

She has been with Arizona State University, Tempe, since 1991 and is now Professor in the Department of Electrical Engineering. Her current research interest includes theory and application of artificial neural learning systems, specifically learning algorithms for statistical signal modeling and data analysis. The objective is to provide adaptive human–machine interface to solve large-scale signal and data analysis problems. Applications include semiconductor process optimization and yield management at both equipment and factory floor levels; spatial-temporal biological signal modeling in motor cortical and auditory systems, 2-D visual information processing through region of interest and robust feature selection; mining and interpretation of large data sets.

Dr. Si is a recipient of the 1995 NSF/White House Presidential Faculty Fellow Award and a recipient of the Motorola Excellence Award in 1995. She was Associate Editor of the IEEE TRANSACTIONS ON AUTOMATIC CONTROL in 1998 and 1999, has been an Associate Editor of the IEEE TRANSACTIONS ON SEMICONDUCTOR MANUFACTURING since 1998.

Yu-Tsung Wang (M'00) was born in 1971. He received the Master of Science degree in engineering and the Ph.D. degree in electrical engineering from Arizona State University (ASU), Tempe, in 1994 and 1999, respectively.

He was a Research Assistant in the Department of Electrical Engineering at ASU from 1995 to 1999. The main research was in the field of dynamic programming using neural network structures and mechanisms. In 1999, he joined Scientific Monitoring, Inc., as a Senior Engineer. His major work is in data trending algorithm, data analysis of aircraft engines, and software development. He is also involved in developing a turbine engine health monitoring system for the United States Air Force.