INTERNATIONAL JOURNAL OF INTELLIGENT CONTROL AND SYSTEMS
VOL. 10, NO. 1, MARCH 2005, 21-32

A Neural Dynamic Programming Approach For Learning Control Of Failure Avoidance Problems

Derong LIU and Huaguang ZHANG

Abstract— **In the present paper, we consider the implementation of adaptive critic designs using neural networks. We study a class of adaptive critic designs that can be classified as (model-free) action-dependent heuristic dynamic programming (ADHDP). The present ADHDP is equivalent to the conventional model-based heuristic dynamic programming (HDP) if we view the model network in the latter as completely embedded in the critic network. This is a valid viewpoint since a neural network connected to another simply forms a larger neural network. We will present three approaches for the training of neural networks in our ADHDP. In particular, for the critic network training, these include non-batch and batch learning with calculated target output values as well as batch learning with an analytically derived overall cost function as the target for learning. The application considered in the present paper is the learning control of failure avoidance problems for which we categorize using the choice of local cost function as zero throughout a trial except at the last time step when a failure occurs. For failure avoidance problems defined this way, we will derive an analytical form of its overall cost function which is defined as the infinite summation of the local cost function over time. We will use a benchmark problem of balancing the pole on a cart to demonstrate that the critic network learning achieved in both non-batch and batch learning with calculated target output values resemble well the learning achieved in the case with the analytically derived overall cost function.**

Index Terms— **Neural dynamic programming, neurodynamic programming, heuristic dynamic programming, approximate dynamic programming, adaptive dynamic programming, asymptotic dynamic programming, dynamic programming, reinforcement learning, adaptive critic designs**

1. INTRODUCTION

SINCE the introduction of adaptive critic designs (ACDs) in the 1970s [33], there have been growing interests in the subject [3], [8], [9], [11], [17], [21]–[26], [34]–[38]. ACDs are defined as designs that *approximate dynamic programming in the general case*, i.e., approximate optimal control over time in nonlinear environments. There are many problems in practice which can be formulated as cost maximization or minimization problems. Examples include error minimization, energy minimization, profit maximization, and the like. Dynamic programming is a very useful tool in solving these problems. However, it is often computationally untenable to run dynamic programming due to the backward numerical process required for its solutions, i.e., due to the "curse of dimensionality" [6], [14]. Over the years, progress has been made to circumvent the "curse of dimensionality" by building a system, called "critic," to approximate the cost function in dynamic programming (cf. [3], [21], [23], [26], [33], [36]–[38]). The idea is to approximate dynamic programming solutions by using a function approximation structure such as neural networks to approximate the cost function. There are three basic methods proposed in the literature for approximating dynamic programming solutions. They are collectively called ACDs, which include Heuristic Dynamic Programming (HDP), Dual Heuristic Programming (DHP), and Globalized Dual Heuristic Programming (GDHP) [21], [23], [36], [38]. A typical ACD consists of three modules that can be implemented by using neural networks. These three modules provide functions of decision, prediction, and evaluation, respectively. When in ACDs the critic network (i.e., the evaluation module) takes the action/control signal as part of its inputs, the designs are referred to as action dependent ACDs (ADACDs).

Reinforcement learning has also attracted considerable attention in the past [1], [5], [13], [28], [29], [31], [32]. Reinforcement learning is defined as learning what to do, i.e., how to map situations to actions, so as to maximize a numerical reward signal [29]. Such a numerical reward signal can be viewed as the local cost function in dynamic programming. In this case, the well-known methods in reinforcement learning, i.e., *temporal difference* method [5], [28], [29] and *Q-learning* method [29], [31], [32], become equivalent to ACDs. In particular, they become equivalent to HDP and ADHDP, respectively (cf. [4], [36], [38]). In other words, ACDs, temporal difference method, and Q-learning method are various approaches developed for approximating dynamic programming.

There are several types of artificial neural network structures proposed in the literature for function approximation [16]. Among them, the *multilayer feedforward neural networks* (FFNNs) are used most frequently in practice. It has been shown that two-layer FFNNs with sigmoidal type activation functions and with an arbitrarily large number of hidden units can approximate any continuous functions to any degree of accuracy, i.e., they are universal approximators [10], [15], [18]. Let us denote the inputs of an FFNN as vector x and the outputs as vector z. The input-output relationship of the FFNN can be written as

$$z = g(x, W) \tag{1}$$

where W represents the weight vector of the FFNN. In neural network training for function approximation, we are given a set of data samples composed of desired input-output pairs of

Manuscript received February 24, 2005; accepted May 12, 2005. This work was supported by the National Science Foundation under Grant ECS-9996428.

D. Liu is with the Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, IL 60607, USA (e-mail: dliu@ece.uic.edu); H. Zhang is with the School of Information Science and Engineering, Northeastern University, Shenyang, Liaoning 110004, P. R. China (e-mail: hgzhang@ieee.org).

the form

$$h\colon \{x^k\} \rightarrow \{d^k\},\ k = 1, 2, \cdots, P, \tag{2}$$

where h is the function to be learned and d^k is the (desired) output of the function h corresponding to input x^k. The training of an FFNN involves repeated presentations of the samples in (2) to the training algorithm and iterative adjustments of the weight vector W in (1). Many weight updating algorithms have been reported in the literature [16]. In the simulation studies of the present work, we use the gradient method [16] which has been implemented in MATLAB as function `traingdx`[1]. For any given function approximation problem, the choice of an appropriate neural network structure relies on experience and experiments. The present work will use FFNNs as a means for function approximation in the implementation of adaptive critic designs; we will assume that a neural network structure has been chosen for our task in function approximation.

The present paper is organized as follows. In Section **2**, we will introduce some necessary background materials including dynamic programming, approximate dynamic programming, and failure avoidance problems. In Section **3**, we will introduce a class of ACDs which is referred to as action-dependent heuristic dynamic programming (ADHDP). We will provide a viewpoint that combines the critic network and the model network in the conventional HDP to show the *equivalence of the present ADHDP to the conventional HDP*. The major advantage of the present ADHDP over the conventional HDP is its model-free nature which may lead to simpler implementations. The failure avoidance problems defined in the present paper are formulated such that we can derive an analytical form of the overall cost function. The overall cost function is expressed as a function of the time it takes for a failure to happen. Even though such a function may not be very useful in practice to guide a learning process, we will show through experiments that the cost function we have derived can in fact be used in learning control designs, i.e., for training the critic network in ACDs. In addition to deriving the overall cost function for the failure avoidance problems, we focus our attention in the present paper on the following two major contributions: (1) An ADHDP solution to the failure avoidance problems, and (2) an experimental proof to the fact that the critic network learning, whether using non-batch learning or batch learning, does indeed learn well the overall cost function derived herein. Practical examples of the class of failure avoidance problems formulated in the present paper include the cart-pole (inverted pendulum) problem, the autolander problem, and the ship steering problem (cf. [2]), among others. Through extensive experimental studies using the cart-pole problem, we are able to show that, without any knowledge about what the overall cost function look like, the critic network learning does indeed mimic the process of learning with the knowledge of the analytically derived cost function. In Section 4 of the paper, we will discuss two training approaches in detail including non-batch learning and batch learning. The non-batch learning approach can be applied in an on-line fashion for real-time learning control applications. In the batch learning approach, both the action network and the critic network are learned after each test/trial. In Section 5, the two approaches will be compared to the results of critic network learning directly from the derived cost function which is also a batch learning approach. Throughout our discussions, we will use the cart-pole problem as a benchmark in demonstrating our results. Finally, in Section **6**, we conclude the present paper with several pertinent remarks.

[1] `traingdx` stands for gradient descent with momentum and adaptive learning rate backpropagation [12].

2. HEURISTIC DYNAMIC PROGRAMMING AND FAILURE AVOIDANCE PROBLEMS

In this section, we first provide the dynamic programming formulation as is done in the book by Lewis and Syrmos [19]. We will then introduce the so-called heuristic dynamic programming as a way for approximating dynamic programming. Finally, we formulate the failure avoidance problems so that their solution using dynamic programming becomes evident.

Suppose that one is given a discrete-time nonlinear (time-varying) system

$$x(t+1) = F[x(t), u(t), t] \tag{3}$$

where $x \in R^n$ represents the state vector of the system and $u \in R^m$ denotes the control action. Suppose that one associates with this system the performance index (or cost)

$$J[x(i), i] = \sum_{k=i}^{\infty} \gamma^{k-i} U[x(k), u(k), k] \tag{4}$$

where U is called the utility function or local cost function and γ is the discount factor with $0 < \gamma \leq 1$. Note that J is dependent on the initial time i and the initial state $x(i)$, and it is referred to as the cost-to-go of state $x(i)$. The objective is to choose the control sequence $u(k)$, $k = i, i+1, \cdots$, so that the function J (i.e., the cost) in (4) is *minimized*. The cost in this case accumulates indefinitely; this kind of problems is referred to as *infinite horizon problems* in dynamic programming. On the other hand, in finite horizon problems, the cost accumulates over a finite number of steps. The present paper will consider infinite horizon problems. Dynamic programming is based on Bellman's *principle of optimality* [6], [7], [14], [19]: An optimal (control) policy has the property that no matter what previous decisions (i.e., controls) have been, the remaining decisions must constitute an optimal policy with regard to the state resulting from those previous decisions.

Suppose that one has computed the optimal cost $J^*[x(t+1), t+1]$ from time $t+1$ on for all possible states $x(t+1)$, and that one has also found the optimal control sequences from time $t+1$ on. The optimal cost results when the optimal control sequence $u^*(t+1), u^*(t+2), \cdots$, is applied to the system with initial state $x(t+1)$. Note that the optimal control sequence depends on $x(t+1)$. If one applies an arbitrary control $u(t)$ at time t and then uses the known optimal control sequence from $t+1$ on, the resulting cost will be

$$U[x(t), u(t), t] + \gamma J^*[x(t+1), t+1],$$

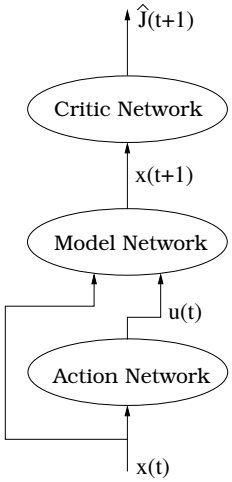

Fig. 1. The three modules in a typical adaptive critic design

where $x(t)$ is the state at time t and $x(t+1)$ is determined by (3). According to Bellman's principle of optimality, the optimal cost from time t on is equal to

$$J^*[x(t),t] = \min_{u(t)} \Big(U[x(t),u(t),t] + \gamma J^*[x(t+1),t+1] \Big).$$

The optimal control $u^*(t)$ at time t is the $u(t)$ that achieves this minimum, i.e.,

$$u^*(t) = \arg \min_{u(t)} \Big(U[x(t),u(t),t] + \gamma J^*[x(t+1),t+1] \Big). \quad (5)$$

Equation (5) is the principle of optimality for discrete-time systems. Its importance lies in the fact that it allows one to optimize over only one control vector at a time by working *backward* in time. In other words, any strategy of action that minimizes the function J in the short term will also minimize the sum of U over all future times.

In the computations in (5), whenever one knows the function J in (4) and the model F in (3), it is a simple problem in function minimization to pick the action $u^*(t)$ that achieves the minimum in (5). However, this procedure requires a backward numerical process and it is too computationally expensive to determine the solutions due to the so-called "curse of dimensionality" [6], [14]. Over the past years, progress has been made to circumvent the "curse of dimensionality" by building a system, called "critic," which approximates the function J [3], [21], [23], [26], [33], [36]–[38]. The idea is to approximate dynamic programming solutions by using a function approximation structure such as neural networks to approximate the function J.

A typical design of ACDs consists of three modules–Critic (for evaluation), Model (for prediction), and Action (for decision) [21], [23], [36], [38], as shown in Figure 1 for an HDP. In this case, the critic network outputs an estimate of the function J in equation (4). This is done by minimizing the following error measure over time,

$$\begin{aligned} \|E_h\| &= \sum_t E_h(t) \\ &= \tfrac{1}{2} \sum_t \left[\hat{J}(t) - U(t) - \gamma \hat{J}(t+1) \right]^2 \end{aligned} \quad (6)$$

where $\hat{J}(t) = \hat{J}[x(t),t,W_C]$ and W_C represents the parameters of the critic network. The function U is the same utility function as the one in (4) which indicates the performance of the overall system (see examples in [3], [21]–[23], [35], [38]). It is usually a function of $x(t)$, $u(t)$, and t, i.e., $U(t) = U[x(t),u(t),t]$. When $E_h(t) = 0$ for all t, (6) implies that

$$\begin{aligned} \hat{J}(t) &= U(t) + \gamma \hat{J}(t+1) \\ &= U(t) + \gamma [U(t+1) + \gamma \hat{J}(t+2)] \\ &= \cdots \\ &= \sum_{k=t}^{\infty} \gamma^{k-t} U(k) \end{aligned}$$

which is exactly the same as the cost function in (4). It is therefore clear that by minimizing the error function in (6), we will have a neural network trained so that its output becomes an estimate of the cost function J defined in (4). In (4), it is assumed that $J(t) < \infty$ which can usually be guaranteed by choosing the discount factor γ such that $0 < \gamma < 1$. The training of the critic network in this case is achieved by minimizing the error function defined in (6), for which many standard neural network training algorithms can be utilized. The model network in an ACD predicts $x(t+1)$ given $x(t)$ and $u(t)$, i.e., it learns the function F in (3). The model network can be trained previously off-line [23], [35], [38], or trained in parallel with the critic and action networks [24]. After the critic network's training is finished, the action network is trained with the objective of minimizing $\hat{J}(t+1)$, through adjusting the parameters W_A of the action network that outputs the action signal $u(t) = u[x(t),t,W_A]$. Once an action network is trained this way, i.e., trained by minimizing the output of the critic network, the action network will generate control action signals which are the optimal control actions or are very close to the optimal control actions, depending on how well the critic network has been trained. Recall that the goal of dynamic programming is to obtain an optimal control sequence as in (5), which minimizes the cost function J in (4). During the training of action network, the three networks will be connected as shown in Figure 1; the training of the action network is done through its parameter updates while keeping the parameters of the critic and the model networks fixed.

There are many problems in practice that have an objective of avoiding failures. The well-known example in this class of problems is the inverted pendulum or the cart-pole problem. In the cart-pole problem, the objective is to balance an upright pole, whose bottom is attached by a pivot to a cart that travels along a track. The state of this system is given by the pole's angle, angular velocity, the cart's horizontal position and velocity. The only available control actions are to exert forces of fixed magnitude on the cart that push it to the left or right. The event of the pole falling past a certain angle or the cart running into the bounds of its track is called a *failure*. A sequence of forces must be applied so that failures can be avoided by balancing the pole for as long as possible within the bounds of the track.

Throughout the rest of the present paper, we will use the cart-pole problem as a benchmark for demonstrating our results. In particular, in Sections 4 and 5, we will use the

cart-pole problem to show results from different variations in the implementation of our ADHDP to be introduced in the next section. The cart-pole problem has been used often in the literature as a benchmark problem for testing learning control algorithms [1], [5], [17], [20], [25], [26]. The cart-pole system is described by equations [2]

$$\ddot{\theta}(t) = \frac{mg\sin\theta(t) - \cos\theta(t)\left[f(t) + m_p l\dot{\theta}^2(t)\sin\theta(t)\right]}{(4/3)ml - m_p l\cos^2\theta(t)} \tag{7}$$

and

$$\ddot{x}(t) = \frac{f(t) + m_p l\left[\dot{\theta}^2(t)\sin\theta(t) - \ddot{\theta}(t)\cos\theta(t)\right]}{m} \tag{8}$$

where θ is the pole's angle from the vertical direction, $x(t)$ is the displacement of the cart from the center of the track, $f(t) = 10$ sign$[u(t)]$, and $u(t)$ is the action signal from the controller (i.e., the action network). Note that by choosing $f(t) = 10$ sign$[u(t)]$ in (7) and (8), we imply that the signals from the action network to the critic network [i.e., $u(t)$] is continuous and the control action that the pole-cart system received is a bang-bang type signal. The control force has a fixed magnitude of 10 N in this case. The parameters used in the above cart-pole system are given as in [2]: $g = 9.8$ m/s^2, $m = 1.1$ kg, $m_p = 0.1$ kg, $l = 0.5$ m. The step size suggested in [2] for numerical integration is 0.02 seconds. The constraints of the system are given by $-12^\circ < \theta < 12^\circ$ and $-2.4 < x < 2.4$ m. These constraints indicate that a failure occurs when either $|\theta| \geq 12^\circ$ or $|x| \geq 2.4$.

We will classify failure avoidance problems by the definition of the function U in ACDs given by

$$U(t) = \begin{cases} 0, & \text{before failure happens} \\ 1, & \text{when failure occurs.} \end{cases} \tag{9}$$

For the illustrating example considered in this paper, i.e., for the cart-pole problem, the function U is defined by

$$U(t) = \begin{cases} 0, & \text{if } |\theta(t)| < 12^\circ \text{ and } |x(t)| < 2.4 \text{ m} \\ 1, & \text{otherwise.} \end{cases} \tag{10}$$

Note that the definition given in (9) and (10) are for each time step t in a test/trial. The function U defined this way implies that a "reinforcement" signal [i.e., $U(t) \neq 0$] is only available when a failure occurs. In the cart-pole problem, the "reinforcement" signal is only available when the pole falls or when the cart runs into the bounds of its track [5]. Failure avoidance is indicated by minimizing the sum of the function U over time in (4). We will derive in Section 5 an analytical expression for the summation in (4) to show that it is a function of T, where T is the time when the pole falls or when the cart hits on the bound of its track. For the cart-pole problem, minimizing the summation in (4) is equivalent to maximizing T, which in turn implies keeping the pole from falling and the cart from running into the bounds of its track for as long as possible. Thus, the present failure avoidance problems can be equivalently expressed as minimizing the cost function in (4) and can be solved using approximate dynamic programming.

Other examples that can be classified as failure avoidance problems include the autolander and the ship steering problems

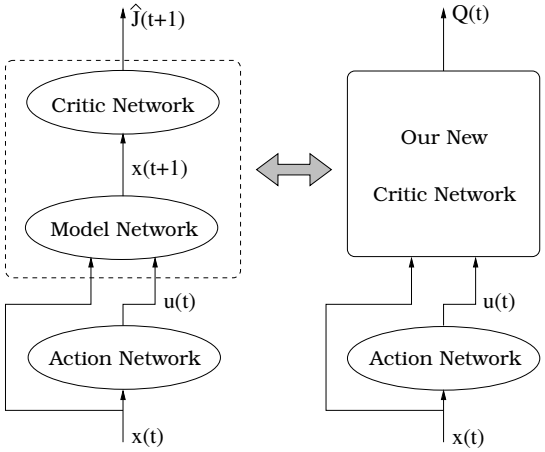

Fig. 2. A new critic network

[2]. We use the cart-pole problem in the present paper for purpose of illustration whenever needed. We do not intend to provide any new methods for solving the cart-pole problem. Instead, we use the cart-pole problem simply for demonstrating our results in Sections 4 and 5. We also note that the solutions for the cart-pole problem given in the present paper may not have been optimized and may not be comparable to other solutions in the vast literature on the cart-pole problem.

3. ACTION-DEPENDENT HEURISTIC DYNAMIC PROGRAMMING

In this section, we develop a scheme which is obtained by modifying the ACD/HDP introduced in the preceding section. We consider a new critic network defined in Figure 2. Note that each of the three modules in the HDP is built using a neural network and a neural network connecting to another simply forms a larger neural network. Our view point in Figure 2 is valid in the sense that the signal $x(t+1)$ can be totally hidden or internal. The new critic network in Figure 2 will include the explicit model network in Figure 1 as part of its internal states. We will now study the function and the training of a new critic network that takes $x(t)$ and $u(t)$ as inputs and uses $\hat{J}(t+1)$ as its output. We denote the output of our new critic network as $Q(t)$ [i.e., $Q(t) = \hat{J}(t+1)$] for notational convenience. The new structure of having only two modules gives us a few advantages including the simplification of the overall system design and the feasibility of the present approach to applications where a model network may be very difficult to obtain. When a neural network model of the process cannot be obtained, model-based ACDs cannot be applied anymore. For such cases, only model-free ACDs such as the present one can be considered. We also note that when there is enough information for building a good model network, ACDs with a model network in some cases lead to better learning control results than those without a model network [21].

In the following we will describe how the action network and the new critic network (therefore, *critic network* in the sequel) in Figure 2 are trained. Figure 3 shows a typical scheme where our new HDP is utilized. The plant in Figure 3 has the form of (3) where the function F is assumed to

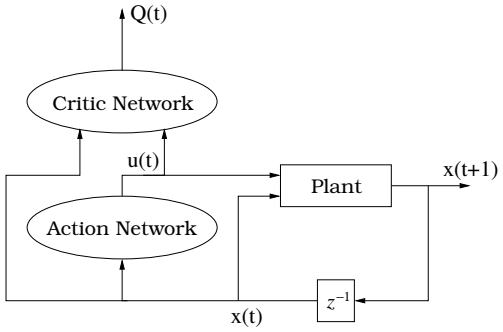

Fig. 3. A typical scheme of an action-dependent heuristic dynamic programming

be unknown. The goal of the control design is to obtain an action network that generates optimal control signals as its output to control the plant so that the cost function in (4) is minimized. The new design is actually an HDP with an embedded model network as analyzed above. Since the HDP in Figure 3 includes the control action signal as input to the critic network, it is therefore a model-free action-dependent HDP (ADHDP) as defined in the literature [22], [23], [36], [38]. Note that in the literature, ADHDP includes both model-free and model-based versions. In the present paper, we study a model-free ADHDP that has been shown to be equivalent to the conventional model-based HDP when all the modules in an ACD are implemented using neural networks.

Consider the ADHDP shown in Figure 3. The critic network in this case will be trained by minimizing the following error measure over time,

$$\begin{aligned} \|E_q\| &= \sum_t E_q(t) \\ &= \tfrac{1}{2}\sum_t \left[Q(t-1) - U(t) - \gamma Q(t)\right]^2 \end{aligned} \tag{11}$$

where $Q(t) = Q[x(t), u(t), t, W_C]$. When $E_q(t) = 0$ for all t, (11) implies that

$$\begin{aligned} Q(t-1) &= U(t) + \gamma Q(t) \\ &= U(t) + \gamma[U(t+1) + \gamma Q(t+1)] \\ &= \cdots \\ &= \sum_{k=t}^{\infty} \gamma^{k-t} U(k). \end{aligned} \tag{12}$$

Clearly, comparing (4) and (12), we have now that $Q(t) = J[x(t+1), t+1]$. Therefore, when minimizing the error function in (11), we have a neural network trained so that its output becomes an estimate of the cost function defined in (4) for $i = t+1$, i.e., the value of the cost function in the immediate future.

A. Critic Network Training

The training samples for the critic network are obtained over a trajectory starting from $x(0)$ at $t = 0$. Starting from $x(0)$, we can generate $u(0)$ from the action network and we can apply $\{x(0), u(0)\}$ to equation (3) (or the plant to be controlled for on-line applications) to obtain $x(1)$. We can then generate $u(1)$ from the action network using $x(1)$ and apply $\{x(1), u(1)\}$ to equation (3) to obtain $x(2)$, and so on. Initially, the action signals $u(t)$, $t = 0, 1, \cdots$, will be generated given $x(t)$, $t = 0, 1, \cdots$, from an action network that is initialized with random weights. We can collect training samples either over a fixed number of time steps (e.g., 300 consecutive points [23]) or from $t = 0$ until the final state is reached (e.g., the plane is crashed or landed in the autolander problem [22], or the pole has fallen in the present cart-pole problem). For each time instant t, the local cost function will be computed as $U(t) = U[x(t), u(t), t]$. Some or all of the data collected, $\{x(t), u(t), U(t)\}$ for $t = 0, 1, 2, \cdots$, are stored in the computer memory for purpose of the critic network and action network training to be described next.

The input-output relationship of the critic network in Figure 3 is given by

$$Q(t) = Q\left[x(t), u(t), t, W_C\right]$$

where W_C represents the weight vector of the critic network. There are two methods to train the critic network according to the error function (11) in the present case which are described next.

(1) *Backward-in-time:* We can train the critic network at time t, with the desired output target given by $[Q(t-1) - U(t)]/\gamma$. The training of the critic network is to realize the mapping given by

$$C_b\colon \left\{ \begin{array}{c} x(t) \\ u(t) \end{array} \right\} \rightarrow \left\{ \frac{1}{\gamma}\left[Q(t-1) - U(t)\right] \right\}. \tag{13}$$

In this case, we consider $Q(t)$ in (11) as the output from the *network to be trained* and the target output value for the critic network is calculated using its output at time $t-1$. Thus, we refer to this method as the *backward-in-time* method. Note that $Q(t-1) = Q[x(t-1), u(t-1), t-1, W_C]$ in (13) is obtained at time $t-1$ and stored in the memory.

(2) *Forward-in-time:* We can train the critic network at time $t-1$, with the desired output target given by $U(t) + \gamma Q(t)$. The training of the critic network is to realize the mapping given by

$$C_f\colon \left\{ \begin{array}{c} x(t-1) \\ u(t-1) \end{array} \right\} \rightarrow \{U(t) + \gamma Q(t)\}. \tag{14}$$

In this case, we consider $Q(t-1)$ in (11) as the output from the *network to be trained* and the target output value for the critic network is calculated using its output at time t. Thus, we refer to this method as the *forward-in-time* method. Note that the training in this case still happens at time t and $x(t-1)$ and $u(t-1)$ in (14) are obtained at time $t-1$ and stored in the memory.

In the following we will provide discussions concerning the forward-in-time method. Similar discussions are applicable to the backward-in-time method as well. Initially, the target for the critic network training given in (14) for time step t is calculated using $Q(t) = Q\left[x(t), u(t), t, W_C^{(0)}\right]$, where $W_C^{(0)}$ is the initial value of the critic network weights. After one step of weight update using either sequential mode training or batch mode training, the critic network weight vector will become $W_C^{(1)}$. This weight updating process will continue until no further improvements in the critic network training can be achieved. This is determined by either no more weight update

during training or no more improvement in the error function during training. A good critic network obtained after training will satisfy the following properties:

1) $|Q(t-1) - U(t) - \gamma Q(t)| < \varepsilon$, i.e.,

$$\Big|Q\big[x(t-1), u(t-1), t-1, W_C^{(p)}\big] - U[x(t), u(t), t] - \gamma Q\big[x(t), u(t), t, W_C^{(p)}\big]\Big| < \varepsilon$$

for each t and for some p, where ε is the maximum error tolerance, and

2) $\big\|W_C^{(p+1)} - W_C^{(p)}\big\| < \eta$ where η is very small and $\|\cdot\|$ denotes an appropriate norm.

The first condition implies that with the weight vector $W_C^{(p)}$ obtained after training, we are able to balance the equation

$$Q(t-1) \approx U(t) + \gamma Q(t)$$

with an error strictly below the error tolerance ε. We note that the key to approximate dynamic programming is to find a critic network so that its outputs will balance this equation for all t as is derived in (12). The second condition implies that the weight update of the critic network indicated by the training algorithm from the pth cycle to the $(p+1)$st cycle is so small that it can be practically ignored. This is because the training has reached the global minimum or a local minimum in the weight space of the error function.

Remark 3.1: The two methods described here for the critic network training are based on two different view points of the error function (11). For failure avoidance problems discussed in the present paper with the function U defined in (9), both backward-in-time and forward-in-time methods work well.

B. Action Network Training

After the critic network's training is finished, the action network's training starts with the objective of minimizing $Q(t)$. To collect training samples for the action network, we start with $x(0)$ and we apply $x(0)$ to the action network to obtain $u(0)$. We then use equation (3) (or from the plant to be controlled for on-line applications) to obtain $x(1)$. Using the action network, we obtain $u(1)$ by applying $x(1)$ and we obtain $x(2)$ from the plant at the next time step [or from the simulation of equation (3)]. Clearly, such a data collection process can be combined with the data collection process for the critic network training. This process continues until all the necessary training patterns are collected or a failure occurs. The goal of the action network training is to minimize the critic network output $Q(t)$. In this case, we can choose the target of the action network training as zero, i.e., we will train the action network so that the output of the critic network becomes as small as possible. In general, a good critic network should not output negative values if $U(t)$ is non-negative. This is particularly true when $U(t)$ is chosen as the square error function in tracking control problems [30]. The desired mapping which will be used for the training of the action network in the present ADHDP is given by

$$A\colon \{x(t)\} \to \{0(t)\} \tag{15}$$

where $0(t)$ indicates the target values of zero. We note that during the training of action network, it will be connected to the critic network as shown in Figure 3. The target in (15) is for the output of the *whole network*, i.e., the output of *the critic network after it is connected to the action network* as shown in Figure 3.

After the action network's training cycle is completed, one may check the system's performance, then stop or continue the training procedure by going back to the critic network's training cycle again, if the performance is not acceptable yet.

4. ILLUSTRATIONS USING NON-BATCH AND BATCH LEARNING APPROACHES

In general, there are two different ways that a learning controller can be obtained. The first approach is to perform learning at each time step of action, i.e., learning is performed in real-time. This type of learning is similar to non-batch or sequential training in neural network learning. The second approach is to perform learning after each trial typically in the form of a failure in failure avoidance problems. This type of learning is closely related to the so-called batch learning in the literature. In the following, we describe the two approaches for neural network learning in the present ADHDP. In the next section, we will describe yet another batch learning approach which uses an analytically derived cost function as the target outputs for the critic network training. We will use the cart-pole problem for illustrations whenever needed.

A. Non-Batch Learning

Training samples are obtained as described earlier for both the critic network and the action network. Non-batch learning is performed at each time step using the immediately obtained training sample. For example, at time t, we obtain $U(t)$ and $Q(t)$. Using the forward-in-time method, we then form the training sample as in (14) and train the critic network accordingly, where $x(t-1)$ and $u(t-1)$ are stored in the memory from the previous time step at $t-1$. In this case, the critic network will be trained for certain number of iterations as outlined earlier before the next data sample is collected. This approach is closely related to the non-batch or sequential training in the literature. Similar approach applies to the action network training as well.

We note that at every time step, the neural network training is performed with only one training pair as in (14) [or (13)] for the critic network and as in (15) for the action network. The difference of the non-batch learning approach described here and the sequential training approach in the literature is that we do not require repeated presentations of every training sample in the training data set to the training algorithm. Instead, at each time step, only one pair of training samples that is collected at the present time step will be used in the training. This pair will be presented repeatedly until it is learned or the learning gets stuck. A neural network trained this way will very likely learn well the last training sample presented while it may gradually forget some of the training samples presented and learned earlier. To illustrate this approach, we present in the following some data collected in our experiments for the

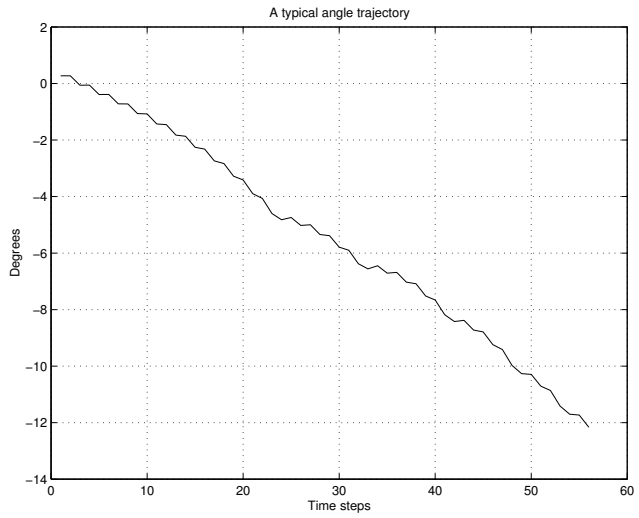

Fig. 4. Typical angle movements during a trial until the pole falls

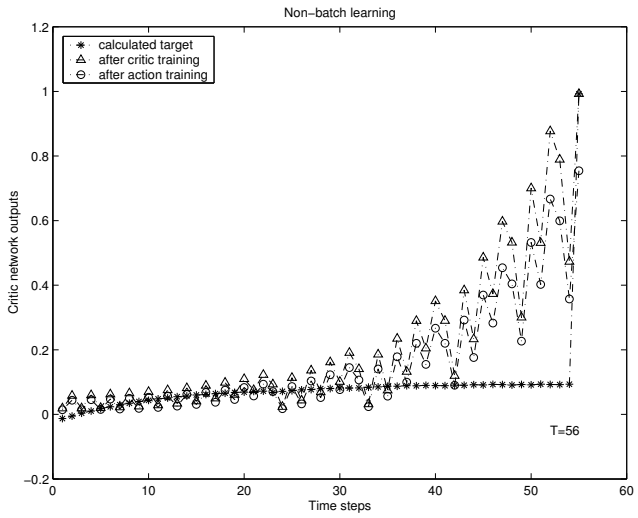

Fig. 5. The values of the critic network output in non-batch learning

cart-pole problem. We also note that if we present the *whole* training data set collected up to each time step repeatedly to the training algorithm, it is in a sense equivalent to the batch learning described in the next section. Also, non-batch learning with repeated presentation of the whole training data set to the training algorithm is much less efficient than batch learning and it *may not be suitable* for real-time learning control tasks since the training data set may be very large in some cases.

For the cart-pole problem, the critic network is chosen as a 5–7–1 structure with 5 input neurons and 7 hidden layer neurons. The 5 inputs are the 4 states $\theta(t)$, $\dot{\theta}(t)$, $x(t)$, and $\dot{x}(t)$, and the output of the action network $u(t)$. The hidden layer uses the sigmoidal function given by

$$y = \frac{1 - e^{-x}}{1 + e^{-x}},$$

i.e., the `tansig` function in MATLAB [12], and the output layer uses the linear function `purelin`. Utilizing the MATLAB Neural Network Toolbox, we have applied `traingdx` (gradient descent algorithm) for the critic network training. We have tried both the forward-in-time method and the backward-in-time method outlined earlier employing non-batch training for the critic network. We use $\gamma = 0.9$ in the present experiments for the cart-pole problem. The structure of the action network is chosen as 4–6–1 with 4 input neurons and 6 hidden layer neurons. The 4 inputs are $\theta(t)$, $\dot{\theta}(t)$, $x(t)$, and $\dot{x}(t)$. Both the hidden layer and the output layer use the sigmoidal function `tansig`. The training algorithm we choose to use is again the gradient descent algorithm.

Figure 4 illustrates the result of a typical trial in the experiments of balancing the pole in the cart-pole problem. The trial takes 56 steps (i.e., $T = 56$) and ends up with the pole fallen. Only the angle movement versus time is shown in Figure 4. We collect and calculate the training samples according to (14) for each time step with $U(t) = 0$ for $t = 0, 1, \cdots, T-1$ and $U(T) = 1$. The calculated target output values for the critic network training for the same example is displayed in Figure 5.

Since the critic network is initialized with small, random weights and $U(t) = 0$ for $t < T$, we can see that initially $U(t) + \gamma Q(t)$ in (14) will be very close to 0 for $t = 0, 1, \cdots, T-1$. Only at the last time step, i.e., when $t = T$, the value of $U(t) + \gamma Q(t)$ becomes close to 1 (cf. Figure 5). Intuitively, these calculated target values imply that the assessments for environments corresponding to time $t = 0, 1, \cdots, T-1$ are all good [since $Q(t) \approx 0$] and only one point in the data collected that corresponds to time $t = T$ has a bad assessment [$Q(T) \approx 1$]. Here the environment consists of the states of the cart-pole system $[\theta(t), \dot{\theta}(t), x(t), \dot{x}(t)]$ and the control action signal $u(t)$. Using the non-batch learning approach described here, we achieve the critic network outputs as shown in Figure 5 after the critic network training, which do not always match with the calculated targets. After the training is done at each time step, the critic network output will match the calculated target most of the time for the training sample at the time. At the last step when time $t = T$, the target value becomes $Q(T) \approx 1$. After training, the critic network output corresponding to $[\theta(T), \dot{\theta}(T), x(T), \dot{x}(T), u(T)]$ becomes very close to 1. This will create a sudden change in the output values from 0 to 1 among the environments surrounding the last point in a trial which has an effect of lifting up the surface of the critic network output. This is especially true for the environments that are close to the environment corresponding to time $t = T$, as shown in Figure 5. In the figure, the display is for the critic network output values versus time t. Two points close to each other in time may not translate into the closeness of their corresponding environments. For example, the environment that is closest to the last one at time $t = T$ $(= 56)$ seems to be the one at time $t = 53$ $(= T - 3)$ in Figure 5. When the surface of the critic network output is lifted up due to the training at $t = T$, the value of the critic network output corresponding to $t = T-3$ is lifted up the most among all the points in this trial. One of the implications of the results after the critic network training shown in Figure 5 is that environments close to time $t = T$ are not so good even though the calculated targets indicate the opposite. Only those

points which are much earlier than $t = T$ can be considered as good environments. We can view the effects shown in Figure 5 to the critic network output values after its training as the effect of "delayed reinforcement" [29].

After the critic network training, we start the action network training with the goal of minimizing the critic network output. The action network training can also be performed in a non-batch mode similarly as described above. In this case, at each time step, we collect training data samples as in (15) and then train the action network before the next time step of action. At each time step in a trial, we train the critic network first and then train the action network. We repeat the process until the pole falls or the cart runs into the bounds of its track, or until the pole has been balanced successfully. The training results achieved in the example illustrated above after the action network training are also shown in Figure 5. The results indicate that after modifying the weights of the action network during training, the action network will be able to generate new control action signals so that the critic network output values are reduced. This is particularly true for the environments close to time $t = T$ since they are considered to be worse than others and the controller should try to avoid them in the future. For the points (environments) that are much earlier in time than $t = T$, improvements are not as significant since they are already considered to be good environments.

It is well known in neural network training that network trained using sequential approach may require repeated presentations of training data samples to the learning algorithm in order to learn all the training samples. Using the non-batch learning approach described here, we frequently obtain training results for the critic network as in Figure 5 since we only present one training sample that is collected at each time step during the training. As mentioned earlier, the critic network trained using this approach will learn well the last training sample (at $t = T$) and may forget some of the training samples learned earlier. At a first glance, the discrepancies between the calculated critic network targets and the learned results may pose some problems. However, as analyzed above, the discrepancies in Figure 5 are in fact good for the critic network learning since the environments close to that at time $t = T$ should indeed be considered as "bad" ones. In addition, as will be shown in Section 5, the results learned in the critic network training are actually an approximation of the actual desired output values of the critic network that can be derived *analytically* in the present case of failure avoidance problems.

We point out that not every trial and training will achieve results similar to those in Figure 5. Sometimes the critic network training may show less desirable results, partly due to the local minimum problem (i.e., the training at some time steps during a trial may get stuck in a local minimum). Figure 6 shows the training results of the critic network outputs in a trial that lead to undesirable learning. Comparing Figures 5 and 6, we can see that the sudden change in the target and trained critic network output values has an effect that makes the critic network output values go up and down over time. Also, the training of the action network in Figure 6 does not seem to have enough effect to push down the values of the critic network outputs (cf. Figure 6 where almost every

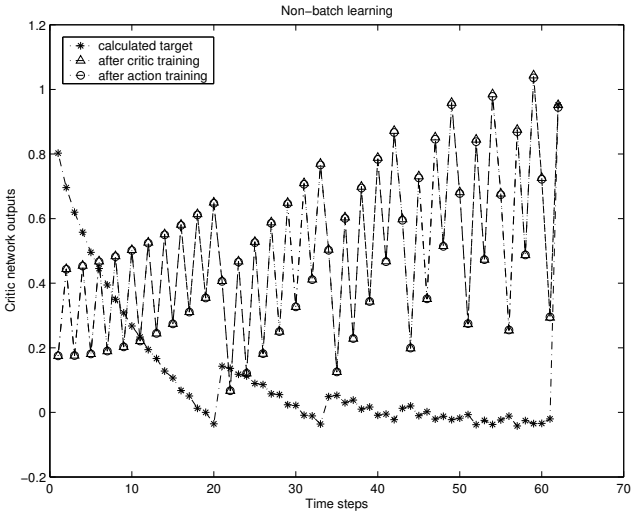

Fig. 6. The values of the critic network output in non-batch learning

"○" overlaps with a "△").

It is clear in the case of non-batch learning described here that the action network will be changing/updating from one time step to the next in a trial. This is in contrast to the result that can be obtained using the batch learning approach to be described next, where updates will only be done at the end of a *failed* trial.

B. Batch Learning

In contrast to the non-batch learning described above, we can also use batch learning for the critic network and the action network training. Training data will be collected as described earlier and will be formed as in (13) or (14) for the critic network and as in (15) for the action network. At the end of each (failed) trial, we start the critic network training. Results from a typical trial are illustrated in Figure 7. In this case, the critic network target values are close to zero before the time $t = T$ ($T = 73$ in Figure 7) when the pole falls, and the target value becomes close to 1 when $t = T$. Due to the sudden change in the critic network target values from 0 to 1, the calculated target function to be learned is not smooth anymore. It is known that to learn a non-smooth function may require a very large network [27]. The critic network outputs after the critic network training become a compromise between the target value at $t = T$ and those preceding it. We can see from Figure 7 that points that are close to the last sample at $t = T$ have their outputs lifted up and the last training sample is learned with some error. The learning process described here has a similar effect as in the case of non-batch learning to "lift up" the surface of the critic network output for points close to $t = T$. In this regard, both non-batch and batch learning in the present ADHDP perform similarly in the critic network learning. The difference between Figures 5 and 7 may be that non-batch learning "lift up" the learned critic network output more than in the case of batch learning. In the case of non-batch learning, the training sample at the time of failure is learned last and usually learned with very small error. That will entail a more significant "lift" for all the environments

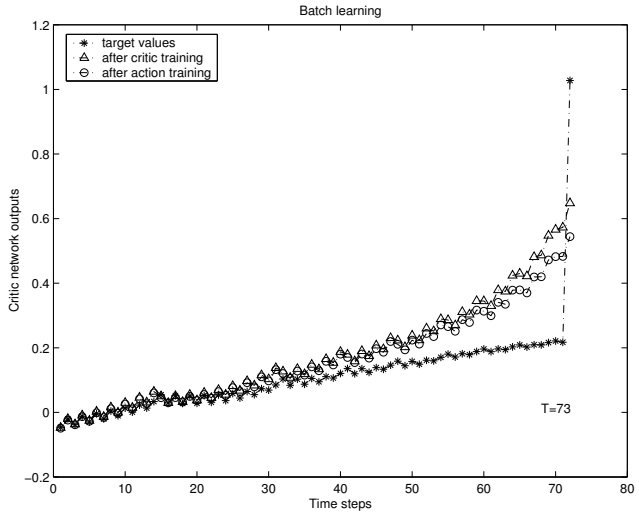

Fig. 7. The values of the critic network output in batch learning

TABLE I
DIFFERENT COMBINATIONS OF NEURAL NETWORK TRAINING IN THE PRESENT ADHDP

Critic network	Action network
non-batch	non-batch
non-batch	batch
batch	batch
batch	non-batch

close to $t = T$ (cf. Figure 5 where $T = 56$). On the other hand, in the case of batch learning, all the training samples are learned together. Typically, only one sample indicates a target close to 1 (i.e., the very last pair of the training data) and the rest of data samples indicate target values very close to 0. When all these data samples are learned together in a batch mode, it is clear that the last training sample will be very difficult to learn due to its sudden increase/change from 0 to 1 in the target values corresponding to the environment that may not be far away from those points preceding it in time.

The action network training can also be trained similarly in a batch mode. For the same example, the critic network output after the action network training using the batch mode is shown in Figure 7. We see that the training of the action network to minimize the outputs of the critic network will typically lower the surface of the critic network output. The closer the point is to the last point at $t = T$, the more reduction to the critic network output value will usually be.

We can also mix and match the training approaches described for the critic and the action networks. We may then have a total of four possible combinations as shown in Table 1. Another approach is a compromise between batch and non-batch learning. For example, we can train the critic and the action networks with at most the last M (e.g., $M = 10$) training data samples collected. Still, this is a kind of non-batching learning, but with a group of training data samples learned together. A compromise like this may be more desirable in certain applications than both batch and non-batch learning described here.

We note that results shown in Figure 7 are very typical in the case of batch learning, i.e., most of the trials achieve similar training results to those shown in Figure 7. Also, we point out that the results shown in Figure 7 for outputs of the critic network after its training is a good approximation of the analytically derived cost function to be introduced next.

5. LEARNING WITH ANALYTICALLY DERIVED COST FUNCTION AND COMPARISON RESULTS OF THE THREE LEARNING APPROACHES

In this section, we will first derive the analytical form of the actual function $J(t)$ that is true for *all* failure avoidance problems with the function $U(t)$ defined in (9). Such a derivation is possible with the present choice of local cost function in the failure avoidance problems formulated in the present paper. We will then show some results in comparison with the approaches introduced in the preceding section.

Assume that the task of balancing the pole in a cart-pole system is on the infinite time horizon, i.e., each trial or test is consider for time $t \in [0, \infty)$. In this case, according to (4), the function $J(t)$ is given by

$$J(t) = \sum_{k=t}^{\infty} \gamma^{k-t} U(k). \tag{16}$$

Assume that the pole falls at time $t = T$. From (9), the function $U(t)$ in this case can be expressed as

$$U(t) = \begin{cases} 0, & \text{when } t < T \\ 1, & \text{when } t \geq T. \end{cases} \tag{17}$$

Combining (16) and (17), we can derive

$$J(t) = \begin{cases} \gamma^{T-t}/(1-\gamma), & t < T \\ 1/(1-\gamma), & t \geq T. \end{cases} \tag{18}$$

From (18), we can see that minimizing the cost function $J(t)$ is equivalent to maximizing the time T. A successful balancing of the pole is indicated by $T = \infty$ which corresponds to the minimum possible value that the cost function $J(t)$ in (18) can attain. We note that $Q(t)$ is used in the present ADHDP and it is related to $J(t)$ by [cf. (12)]

$$Q(t) = J(t+1) = \begin{cases} \gamma^{T-t-1}/(1-\gamma), & t \leq T \\ 1/(1-\gamma), & t > T. \end{cases} \tag{19}$$

Our third approach for the training of neural networks in the present ADHDP is to train the critic network after each trial (i.e., in a batch mode) with the function $Q(t)$ given by (19). With the known cost function derived above as the target in the critic network training, we are now able to judge how well a learning approach performs. The function in (19) depends on how long the pole is balanced (T) before failure happens and the discount factor γ. The parameter γ determines the shape/curve of the function $Q(t)$ for $t \leq T$ and T determines the point after which the function becomes flat (constant). The action network can be trained as in the previous section by minimizing the output of the critic network after it is trained. A plot of the function $Q(t)$ in (19) and the results of the critic network training are shown in Figure 8. We have scaled the function in (19) by $(1 - \gamma)$ in our computer programs. We

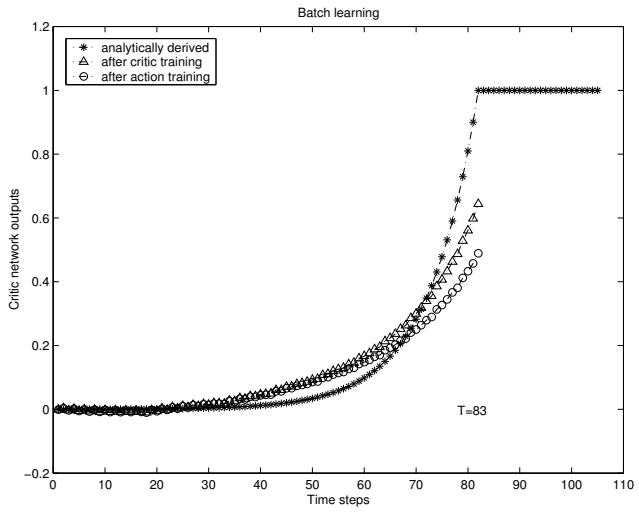

Fig. 8. The critic network learning with the analytically derived function Q

note that this scaling is equivalent to replacing the value of $U(t)$ for $t \geq T$ by $1 - \gamma$. It is also equivalent to using a scaled neural network as is done extensively in [22]. In the present study, when using a normalized/scaled neural network, we have all the input and output signals of the network in the range from -1 to $+1$. It is a convenient choice when using MATLAB that we employ scaled neural networks in implementations. Clearly, the results from the trained critic networks in Figures 5 and 7 *both resemble well* the plot of the true, analytically derived $Q(t)$ in Figure 8 (for $t \leq T$). That is to say, even though there are discrepancies in both the non-batch and batch learning described in the preceding section, what we have achieved after the critic network training in both cases actually approximates well the analytically derived cost function. Furthermore, the two approaches in the previous section assume no knowledge at all about the form of the overall cost function derived herein.

Results illustrated in Figure 8 also show discrepancies between the target values of the critic network outputs determined by (19) and the output values after the critic network training. We note that these discrepancies are caused by the small network size and sometimes insufficient number of iterations in training. For the cart-pole problem illustrated here, if we choose a larger network size, e.g., if we use 14 hidden layer neurons instead of 7 for the critic network, we will be able to train the critic network using sufficiently large number of iterations so that its output values after training match with the analytically determined values in most trials.

Finally, we emphasize that the analytical form of the cost function $Q(t)$ derived in (19) is true for the whole class of failure avoidance problems considered in the present paper with the local cost function $U(t)$ defined as in (9). Even though such an analytical form of the overall cost function may not be very useful in guiding a learning process in practice, we have shown that it is very helpful in understanding the learning process of the critic network and the action network in ACDs.

To compare the three learning approaches described in the present paper, we choose to use batch learning for the action network in the cart-pole problem. Using the three approaches for the critic network training, in each experiment, we initialize randomly the critic network and the action network. We repeat the experiments for a total of 100 times with different initial weights for each of the three learning approaches. Our goal is to compare the performance of the three learning approaches for critic network training in terms of the number of trials that is needed to balance the pole. It turns out after many experiments that the learning in batch mode with calculated critic network output target values and with analytically derived cost function perform very close to each other, i.e., they usually require similar number of trials to balance the pole. The learning in non-batch mode for the critic network usually requires more trials to balance the pole than the two batch learning approaches. Table 2 lists comparison results from 100 successful trials for each of the three learning approaches for the critic network training (shown in the table as non-batch, batch/calculated, and batch/derived, respectively). We display in the table the average number of trails needed to balance the pole and the standard deviation of these numbers from a total of 100 trials. We can see from the table that the results are very close to each other for batch learning with calculated target values and batch learning with derived cost function as the target. Due to the undesirable behavior shown in Figure 6 which happens sometimes in non-batch learning, it usually takes more trials to balance the pole than the other two approaches. We note that results shown in Table 2 depend on the parameters that can be tuned in the training process such as the learning rate, number of training epochs, and the like. In the present simulation studies, we use the default learning rate in MATLAB which is 0.01 in each case. For critic network training, we use 100 epochs for batch learning and we use 5 epochs for non-batch learning. For action network training, we use 80 epochs (batch learning only). We note that these numbers are chosen such that over-training can be avoided in each case. We also note that the training algorithm used here is again `traingdx` in MATLAB.

TABLE II

AVERAGE NUMBER OF TRIALS NEEDED TO BALANCE THE POLE

Mode	Average number of trials	Standard deviation
non-batch	62	41.49
batch/calculated	25.64	33.99
batch/derived	24.04	32.68

We can therefore conclude that all three approaches presented in the present paper work well for the neural network learning in the present ADHDP. Non-batch learning approach can be employed on-line for real-time learning control applications. We can also conclude that for the critic network learning both batch and non-batch learning approaches with calculated target functions resemble well the learning approach with analytically derived cost function. The resemblance comes as a result of the sudden change in the calculated target values and from the inability in learning such a non-smooth function using neural networks. One implication of the present results is that the inability of learning exactly the calculated target values

is often useful in critic network learning. The discrepancies between the calculated target values and the output values after the critic network learning actually indicate that the environments near the end of a failed trial are not so good and should be avoided in future trials.

6. CONCLUDING REMARKS

We developed in the present paper a class of adaptive critic designs that can be classified as (model-free) action-dependent heuristic dynamic programming (ADHDP). The present ADHDP is equivalent to the conventional HDP if we view the model network in the latter as completely hidden and internal. We argue that this is a valid viewpoint since a neural network connected to another simply forms a larger neural network. With this viewpoint, we consider a new critic network which is formed of the critic and the model networks in the conventional HDP. This new critic network takes the action network's outputs as part of its inputs. We presented three approaches for the training of neural networks in the present ADHDP. Non-batch learning and batch learning have similar features to the sequential and batch approaches, respectively, for neural network training in the literature.

We consider in the present paper the learning control of failure avoidance problems which are categorized by a special choice of the local cost function U. The function U has zero value at every time step before a failure occurs and it becomes non-zero only at the time of failure. Failure avoidance problems considered in the present paper have many potential applications in practice. Examples include the cart-pole problem, the autolander problem, the ship steering problem, and the like. For the failure avoidance problems, we derived an analytical form of the overall cost function J which is a function of the discount factor γ and the time T when failure happens in a trial. We use the cart-pole problem as the benchmark example to illustrate the process of the critic network learning. In non-batch learning, after the learning of the last training data sample collected at the time of failure, the critic network will tend to forget some of the samples learned earlier. This effect is equivalent to a lift to the surface of the critic network output, especially at those points that are close to the last data sample. It is demonstrated through experiments that the outputs of the critic network after training become an approximation to the analytically derived cost function due to the "lift" during the training at the last time step in a failed trial. We also demonstrated through experiments that batch learning with calculated target values also has a similar lifting effect to the surface of the critic network output. In this case, the lift is due to a compromise between the target value of close to 1 at the last time step in a failed trial and the target values of close to zero at all other time steps. The sudden jump in the target values from zero to 1 makes it difficult and sometimes impossible for a neural network to learn the desired target function. Thus, after the critic network training, points that are close to that of the last time step will have their output values lifted up while the last training sample will usually be learned with some error. Simulation results using the cart-pole problem show that the learning in non-batch and batch learning with calculated target values, without the knowledge of the true overall cost function, both resemble well the learning with the analytically derived cost function. We emphasize that the analytical form of the overall cost function is derived for the whole class of failure avoidance problems studied in this paper. The derivation of such a function is critical to understanding the learning process of neural networks when performing approximate dynamic programming.

REFERENCES

[1] C. W. Anderson, "Learning to control an inverted pendulum using neural networks," *IEEE Control Systems Magazine*, vol. 9, pp. 31–37, Apr. 1989.

[2] C. W. Anderson and W. T. Miller III, "Challenging control problems," in *Neural Networks for Control* (Appendix A), Edited by W. T. Miller III, R. S. Sutton, and P. J. Werbos, Cambridge, MA: The MIT Press, 1990.

[3] S. N. Balakrishnan and V. Biega, "Adaptive-critic-based neural networks for aircraft optimal control," *Journal of Guidance*, *Control*, *and Dynamics*, vol. 19, pp. 893–898, July-Aug. 1996.

[4] A. G. Barto, "Reinforcement learning and adaptive critic methods," in *Handbook of Intelligent Control: Neural*, *Fuzzy*, *and Adaptive Approaches* (Chapter 12), Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

[5] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, vol. SMC-13, pp. 834–846, Sept./Oct. 1983.

[6] R. E. Bellman, *Dynamic Programming*, Princeton, NJ: Princeton University Press, 1957.

[7] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, Belmont, MA: Athena Scientific, 1995.

[8] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Belmont, MA: Athena Scientific, 1996.

[9] C. Cox, S. Stepniewski, C. Jorgensen, R. Saeks, and C. Lewis, "On the design of a neural network autolander," *International Journal of Robust and Nonlinear Control*, vol. 9, pp. 1071–1096, Dec. 1999.

[10] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control*, *Signals*, *and Systems*, vol. 2, pp. 303–314, 1989.

[11] J. Dalton and S. N. Balakrishnan, "A neighboring optimal adaptive critic for missile guidance," *Mathematical and Computer Modeling*, vol. 23, pp. 175–188, Jan. 1996.

[12] H. Demuth and M. Beale, *Neural Network Toolbox User's Guide*, Natick, MA: MathWorks, 1998 (Version 3).

[13] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *Journal of Artificial Intelligence Research*, vol. 13, 227-303, 2000.

[14] S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming*, New York, NY: Academic Press, 1977.

[15] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–192, 1989.

[16] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Upper Saddle River, NJ: Prentice Hall, 1999.

[17] G. G. Lendaris and C. Paintz, "Training strategies for critic and action neural networks in dual heuristic programming method," *Proceedings of the 1997 IEEE International Conference on Neural Networks*, Houston, TX, June 1997, pp. 712–717.

[18] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[19] F. L. Lewis and V. L. Syrmos, *Optimal Control*, New York, NY: John Wiley, 1995.

[20] D. Liu, X. Xiong, and Y. Zhang, "Action-dependent adaptive critic designs," *Proceedings of the INNS-IEEE International Joint Conference on Neural Networks*, Washington, DC, July 2001, pp. 990–995.

[21] D. V. Prokhorov, *Adaptive Critic Designs and Their Applications*, Ph.D. Dissertation, Texas Tech University, Lubbock, TX, Oct. 1997.

[22] D. V. Prokhorov, R. A. Santiago, and D. C. Wunsch, "Adaptive critic designs: A case study for neurocontrol," *Neural Networks*, vol. 8, pp. 1367–1372, 1995.

[23] D. V. Prokhorov and D. C. Wunsch, "Adaptive critic designs," *IEEE Transactions on Neural Networks*, vol. 8, pp. 997–1007, Sept. 1997.

[24] R. E. Saeks, C. J. Cox, K. Mathia, and A. J. Maren, "Asymptotic dynamic programming: Preliminary concepts and results," *Proceedings of the 1997 IEEE International Conference on Neural Networks*, Houston, TX, June 1997, pp. 2273–2278.

[25] R. A. Santiago and P. J. Werbos, "New progress towards truly brain-like intelligent control," *Proceedings of the World Congress on Neural Networks*, San Diego, CA, June 1994, vol. I, pp. 27–33.

[26] J. Si and Y.-T. Wang, "On-line learning control by association and reinforcement," *IEEE Transactions on Neural Networks*, vol. 12, pp. 264–276, Mar. 2001.

[27] E. D. Sontag, "Feedforward nets for interpolation and classification," *Journal of Computer and System Sciences*, vol. 45, pp. 20–48, 1992.

[28] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, 1988.

[29] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Cambridge, MA: The MIT Press, 1998.

[30] N. A. Visnevski and D. V. Prokhorov, "Control of a nonlinear multivariable system with adaptive critic desings," *Proceedings of the Artificial Neural Networks in Engineering Conference*, St. Louis, MO, Nov. 1996, pp. 559–565.

[31] C. J. C. H. Watkins, *Learning from Delayed Awards*, Ph.D. Thesis, Cambridge University, Cambridge, England, 1989.

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

[33] P. J. Werbos, "Advanced forecasting methods for global crisis warning and models of intelligence," *General Systems Yearbook*, vol. 22, pp. 25–38, 1977.

[34] P. J. Werbos, "Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research," *IEEE Transactions on Systems*, *Man*, *and Cybernetics*, vol. SMC-17, pp. 7–20, Jan./Feb. 1987.

[35] P. J. Werbos, "Consistency of HDP applied to a simple reinforcement learning problem," *Neural Networks*, vol. 3, pp. 179–189, 1990.

[36] P. J. Werbos, "A menu of designs for reinforcement learning over time," in *Neural Networks for Control* (Chapter 3), Edited by W. T. Miller, R. S. Sutton, and P. J. Werbos, Cambridge, MA: The MIT Press, 1990.

[37] P. J. Werbos, "Neurocontrol and supervised learning: An overview and evaluation," in *Handbook of Intelligent Control: Neural*, *Fuzzy*, *and Adaptive Approaches* (Chapter 3), Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

[38] P. J. Werbos, "Approximate dynamic programming for real-time control and neural modeling," in *Handbook of Intelligent Control: Neural*, *Fuzzy*, *and Adaptive Approaches* (Chapter 13), Edited by D. A. White and D. A. Sofge, New York, NY: Van Nostrand Reinhold, 1992.

Derong Liu received the Ph.D. degree in electrical engineering from the University of Notre Dame, Notre Dame, Indiana, in 1994, the M.S. degree in electrical engineering from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 1987, and the B.S. degree in mechanical engineering from the East China Institute of Technology (now Nanjing University of Science and Technology), Nanjing, China, in 1982. From 1982 to 1984, he was a product design engineer at China North Industries Corporation, Jilin, China. From 1987 to 1990, he was an instructor at the Graduate School of the Chinese Academy of Sciences, Beijing, China. From 1993 to 1995, he was a staff fellow at General Motors Research and Development Center, Warren, Michigan. From 1995 to 1999, he was an Assistant Professor in the Department of Electrical and Computer Engineering, Stevens Institute of Technology, Hoboken, New Jersey. He joined the University of Illinois at Chicago in 1999 as an Assistant Professor of Electrical Engineering and Computer Science, where he is now an Associate Professor of Electrical and Computer Engineering, of Bioengineering, and of Computer Science. He is coauthor (with A. N. Michel) of the books Dynamical Systems with Saturation Nonlinearities: Analysis and Design (New York: Springer-Verlag, 1994) and Qualitative Analysis and Synthesis of Recurrent Neural Networks (New York: Marcel Dekker, 2002). He is coeditor (with P. J. Antsaklis) of the book Stability and Control of Dynamical Systems with Applications (Boston, MA: Birkhauser, 2003).

Dr. Liu was a member of the Conference Editorial Board of the IEEE Control Systems Society (1995-2000), served as an Associate Editor for IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications (1997-1999), and served as an Associate Editor for IEEE Transactions on Signal Processing (2001-2003). Since 2004, he has been an Associate Editor for IEEE Transactions on Neural Networks. He is the Program Chair for the 21st IEEE International Symposium on Intelligent Control (2006) and 2006 International Conference on Networking, Sensing and Control, and he has served and is serving as a member of the organizing committee and the program committee of several international conferences. He was recipient of the Michael J. Birck Fellowship from the University of Notre Dame (1990), the Harvey N. Davis Distinguished Teaching Award from Stevens Institute of Technology (1997), and the Faculty Early Career Development (CAREER) award from the National Science Foundation (1999). He is a Fellow of the IEEE and a member of Eta Kappa Nu.

Huaguang Zhang was born in Jilin, China, in 1959. He received the B.S degree and the M.S degree in control engineering from Northeastern Electric Power University of China in 1982 and 1985, respectively. He received the Ph. D degree in thermal power engineering and automation from Southeast University of China in 1991. He entered the automatic control department, Northeastern University, in Jan. 1992, as a postdoctoral fellow for two years. Since 1994, he has been a professor and head of the electric automation institute, Northeastern University, Shenyang, China. His main research interests are fuzzy control, chaos control, neural networks based control, nonlinear control, signal processing, and their applications.

Dr. Zhang received the "Distinguished Young Scientist Award" from the National Natural Science Foundation of China in 2003.