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

Discrete-Time Adaptive Dynamic Programming using Wavelet Basis Function Neural Networks

Ning Jin, Derong Liu, Ting Huang, and Zhongyu Pang

***Abstract*— Dynamic programming for discrete time systems is difficult due to the "curse of dimensionality": one has to find a series of control actions that must be taken in sequence, hoping that this sequence will lead to the optimal performance cost, but the total cost of those actions will be unknown until the end of that sequence. In this paper, we present our work on adaptive dynamic programming (ADP) for nonlinear discrete time system using neural networks. The neural network we adopted here is the wavelet basis function (WBF) neural network. We will exam the performance of an ADP algorithm using WBF neural networks. The comparison shows that when WBF neural networks are employed, the ADP algorithm gives faster training speed than when RBF neural networks are employed.**

I. Introduction

Dynamic programming is a very useful tool in solving optimization and optimal control problems. However, it is often computationally untenable to run true dynamic programming due to the backward numerical process required for its solution, i.e., as a result of the well-known "curse of dimensionality" [3], [8]. One has to find a series of control actions that must be taken in sequence, hoping that this sequence will lead to the optimal performance cost, but the total cost of those actions will be unknown until the end of that sequence. If the optimal performance cost J^* is known, the optimal control law u^* can be obtained by applying J^* as a Lyapunov function for the system. Under some good analytic conditions for functions, the optimal cost function is the solution of the Hamilton-Jacobi-Bellman (HJB) equation [3]. However, the theoretical solution of the HJB equation is very difficult to obtain, except for systems satisfying some very good conditions, such as linear systems with quadratic utility and zero target.

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. [1], [14], [16], [18], [19], [24], [25], [27], [28]). The idea is to approximate dynamic programming solutions by using a function approximation structure to approximate the cost function.

In 1994, Saridis and Wang [18] studied non-linear stochastic systems described by

$$dx = F(x,t)dt + B(x,t)udt + G(x,t)dw, \qquad (1)$$

Ning Jin, Derong Liu, Ting Huang, and Zhongyu Pang are with the Department of Electrical and Computer Engineering, University of Illinois, Chicago, IL 60607-7053, USA (phone: 312-355-4455; fax: 312-966-6465; emails: njin@uic.edu, dliu@ece.uic.edu, zpang2@uic.edu, thuang20@uic.edu)

This work was supported in part by NSF grants ECCS-0621694, ECS-0529292, and ECS-0355364.

where $x \in \mathbb{R}^n$ is a vector of state of the stochastic system, $u \in \mathbb{R}^m$ is a control vector and $w \in \mathbb{R}^k$ is a separable Wiener process. $F(\cdot,\cdot)$, $B(\cdot,\cdot)$ and $G(\cdot,\cdot)$ are measurable functions. The performance cost of the system (1) is defined as

$$J(x_0;t_0,u) = E\left\{\int_{t_0}^{+\infty} \left[U(x,t) + \|u\|^2\ \right] dt \right. \\ \left. +\phi(x(T),T)\Big|x(t_0) = x_0\right\},$$

where $U(\cdot,\cdot)$ and $\phi(\cdot)$ are nonnegative functions. Instead of solving HJB equation to find the optimal performance cost J^*, Saridis and Wang [18] gave an iterative algorithm to approximate a bound V^* of J^* to achieve the so-called "suboptimal control" of the system.

In 2002, Cox, Lendaris, Murray and Seaks [10], [14] studied the (deterministic) continuous-time stabilizable systems

$$\frac{dx}{dt} = F(x) + B(x)u, \quad x(t_0) = x_0, \qquad (2)$$

with the cost function

$$J = \int_{t_0}^{+\infty} U(x,u)dt, \qquad (3)$$

where $U(x,u) = q(x) + u^T r(x)u$ is a nonnegative function and $r(x) > 0$. Similar to [18], an iterative process is proposed. Starting from any stable Lyapunov function J_0 [14] (or alternatively, starting from an arbitrary stable controller u_0 [10]), the algorithm will give two sequences $\{J_i\}$ and $\{u_i\}$ which will converge to the optimal cost function J^* and the optimal control u^*, respectively. By applying radial basis function approximation, no prior information of $F(x)$ and $B(x)$ are required in the algorithm. Thus their algorithm is an adaptive algorithm.

Meanwhile, a neural network approach for approximate dynamic programming has been developed in the literature. There are several synonyms used including "Adaptive Critic Designs" (ACD), "Approximate Dynamic Programming", "Neural Dynamic Programming", "Reinforcement Learning" (RL), and so on. In the early 1970's, Werbos set up the basic strategy of RL system for ACD (cf. [23], [28], [29] for details). A typical design of ACDs consists of three modules-Critic, Model, and Action. They are neural networks used to approximate the optimal cost function, the plant to be controlled, and the optimal controller, respectively. These three parts combined together form a "Reinforcement Learning System" (RLS) or an ACD. In ACDs, neural networks are designed to approximate the cost function J, to simulate the derivative of J, and to estimate the solution of Hamilton-Jacobi-Bellman equation. The principle of optimal control is used inside the neural networks to build the weight updating law. They can achieve good approximation most of the time.

In this paper, we study adaptive dynamic programming of discrete time-invariant nonlinear systems

$$x_{k+1} = F(x_k, u_k), \quad k = 0, 1, 2, \ldots . \tag{4}$$

There is no special requirement for the system functions F except that F is continuous. The target $\mathcal{T}$ is a neighborhood of zero and we will try to drive the system to reach the target in finite time (steps). There are no restrictions on the number of total control steps. The algorithm ADPDN [9] will be used in this paper. The algorithm ADPDN was developed in [9] after a step by step exploration of the system to find the relationship of optimal controls and optimal costs for different control steps. We have applied radial basis function (RBF) neural networks to approximate the associated functions of the system, the performance cost, the optimal controller, and the function to estimate the steps need to complete the trajectory. In this paper, instead of RBF neural networks, the new *wavelet basis function (WBF)* neural networks will be applied to approximate the functions. We will use wavelet basis function neural networks to approximate the nonlinear function F of the plant, the optimal cost function J^o, the optimal controller u^o, and the estimated control steps K^o. Wavelet basis function neural network is a new kind of neural networks designed to approximate function via wavelet basis functions. It is analogous to the radial basis function neural network with better generalization properties. We will examine the performance of the algorithm ADPDN [9] using WBF neural networks. Numerical experiments to illustrate the performance of our algorithm will be proposed. The comparison shows that when WBF neural networks are applied, the ADPDN algorithm gives faster training speed than when RBF neural networks are applied.

II. Problem Statement

Consider a discrete-time time-invariant plant

$$x_{k+1} = F(x_k, u_k), \; k = 0, 1, 2, \ldots,$$

with utility function $U(x, u)$ and $\phi(x)$, and performance measure

$$J = \phi(x_N) + \sum_{k=0}^{N-1} U(x_k, u_k), \tag{5}$$

where $x_k \in \mathbb{R}^n$ is the state and $u_k \in \mathbb{R}^m$ is the control. The control target $\mathcal{T} \subset \mathbb{R}^n$ is a simple connected closed region containing the origin. The initial state is x_0 and the final state is x_N.

Here, $F(x, u)$, $U(x, u)$ and $\phi(x)$ are continuous functions of state x and control u. N is the total number of control steps. They satisfy the following assumptions:

(C2.1) $F(0, 0) = 0$, and for any $x \in \mathcal{T}$, there exists a control $u \in \mathbb{R}^m$ such that $F(x, u) \in \mathcal{T}$. So if $x_0 \in \mathcal{T}$, then we can always find a control sequence $u_0, u_1, \ldots$ such that $x_k \in \mathcal{T}$ for all $k = 1, 2, \ldots$. In particular, if $x_0 = 0$, then $u_k \equiv 0$ gives $x_k \equiv 0$, $k = 0, 1, \ldots$.

(C2.2) $U(x, u) \geq 0$ for all (x, u). For each fixed x, $U(x, 0)$ will be the minimal value of $U(x, u)$. Hence if we do nothing at one step, then the cost at that step will be the smallest, although the whole cost may be bigger. The utility $x^T Rx + u^T Su$ and the utilities of the type $q(x) + u^T Su$ (with $q(x) > 0$ [14]) all satisfy this condition.

(C2.3) $\phi(x) \geq 0$ for all x, and $\phi(0) = 0$.

(C2.4) $x_N \in \mathcal{T}$, i.e., the trajectory must reach the target at the final control step. Notice that one can not always find a suitable control sequence to make the state reach the target in finite steps. In this case, we set $N = +\infty$ and omit the item $\phi(x_N)$ in (5).

Now the trajectory starting from x_0 under the control of $\underline{u}$ is $x_0 = x$, $x_1 = F(x_0, u_0)$, $x_2 = F(x_1, u_1)$, $\ldots$, $x_N = F(x_{N-1}, u_{N-1})$, and we have $x_N \in \mathcal{T}$. The total cost of this trajectory is $J(x, \underline{u}) = \phi(x_N) + \sum_{i=0}^{N-1} U(x_i, u_i)$, which depends on the values of x and $u_0, u_1, \ldots, u_{N-1}$. So it is a function of x and $u_0, u_1, \ldots, u_{N-1}$. Besides, it depends on N, too. We can express it as

$$J(x, \underline{u}) = J(x; u_0, \ldots, u_{N-1}) = \phi(x_N) + \sum_{i=0}^{N-1} U(x_i, u_i).$$

Define the cost from any step k to step N as follows:

$$\begin{aligned}
J_{N;N} &\triangleq \phi(x_N) \\
J_{N-1;N} &\triangleq J_{N-1}(x_{N-1}; u_{N-1}) \\
&= U(x_{N-1}, u_{N-1}) + J_N, \\
J_{N-2;N} &\triangleq J_{N-2}(x_{N-2}; u_{N-2}, u_{N-1}) \\
&= U(x_{N-2}, u_{N-2}) + J_{N-1}, \\
&\vdots \\
J_{1;N} &\triangleq J_1(x_1; u_1, u_2, \ldots, u_{N-1}) \\
&= U(x_1, u_1) + J_2, \\
J_{0;N} &\triangleq J_0(x_0; u_0, u_1, \ldots, u_{N-1}) \\
&= U(x_0, u_0) + J_1.
\end{aligned}$$

In the above, $J_{0;N}$ is just $J(x, \underline{u})$, i.e.,

$$J(x, \underline{u}) = J_0(x_0; u_0, u_1, \ldots, u_{N-1}).$$

To find the optimal control signal, we need to minimize the performance cost. We define the minimum cost from step k on to the end as follows. For $k = 0, \ldots, N$, let

$$\begin{aligned}
J^*_{k;N}(x_k) &= \min_{u_k, \ldots, u_{N-1}} \{J_k(x_k; u_k, \ldots, u_{N-1})\} \\
&= \min_{u_k, \ldots, u_{N-1}} \{\sum_{i=k}^{N-1} U(x_i, u_i) + \phi(x_N)\}.
\end{aligned}$$

Then $J^*_{k;N}(\cdot)$ is the optimal cost function from step k to the final step N. By Bellman's principle of optimality,

$$J^*_{k;N}(x_k) = U(x_k, u^*_k) + J^*_{k+1;N}(F(x_k, u^*_k)), \tag{6}$$

where $u_k^* = u_{k;N}^*(x_k)$ is the optimal control at step k, and

$$u_{k;N}^*(x_k) = \arg\min_u \{U(x_k, u) + J_{k+1;N}^*(F(x_k, u))\}. \quad (7)$$

According to (7), we can get $u_{k;N}^*(\cdot)$ from $J_{k+1;N}^*(\cdot)$. And by (6), we can get $J_{k;N}^*(\cdot)$ from $u_{k;N}^*(\cdot)$ and $J_{k+1;N}^*(\cdot)$. Thus, we can get all the optimal costs $J_{k;N}^*(\cdot)$ and optimal controls $u_{k;N}^*(\cdot)$, $k = 0, 1, \ldots, N-1$, backward from $J_{N;N}^*(\cdot)$. Our aim of this paper is to develop a neural network algorithm to approximate these functions.

According to (6) and (7), to determine the sequences of functions $J_{k;N}^*(\cdot)$ and $u_{k;N}^*(\cdot)$, $k = 0, \ldots, N-1$, we need to study $J_{N-1;N}^*(\cdot)$ first. (Notice that $J_{N;N}^*(\cdot) = \phi(\cdot)$ is known.)

Let x_0 be an arbitrary initial state. Choose a sequence of control signals $u_0, \ldots, u_{N-1}$, we will have a trajectory $x_0, x_1, \ldots, x_N$, where $x_{k+1} = F(x_k, u_k)$. First, look at the region of all possible values of state x_{N-1}. If $x \in \mathbb{R}^n$ is a possible value of state x_{N-1}, then there must exists at least one value $u \in \mathbb{R}^m$ of the control u_{N-1} such that

$$F(x, u) \in \mathcal{T}.$$

Hence, if the Nth step is the final step of control, then the equation $F(x, u) \in \mathcal{T}$ gives the region of all possible values of state x_{N-1} and the associated value of control u_{N-1} such that the final state (in step N) x_N reaches the target $\mathcal{T}$. Similarly, if x and u are possible values of state x_k and control u_k, respectively, in the kth step of the totally N control steps, then $F(x, u)$ must be a possible value of state in the $k+1$ step. For $N = 1, 2, 3, \ldots$, $k = 0, 1, \ldots, N-1$, define

$$\begin{cases} \mathcal{A}_{k;N} = \{(x, u) \in \mathbb{R}^{n+m} |\ F(x, u) \in \mathcal{T}_{k+1;N}\}, \\ \mathcal{T}_{k;N} = \{x \in \mathbb{R}^n |\ \exists u \in \mathbb{R}^m\ s.t.\ F(x, u) \in \mathcal{T}_{k+1;N}\}, \\ \mathcal{C}_{k;N} = \{u \in \mathbb{R}^m |\ \exists x \in \mathbb{R}^n\ s.t.\ F(x, u) \in \mathcal{T}_{k+1;N}\}, \end{cases}$$

where $\mathcal{T}_{N;N} = \mathcal{T}$ for all N. Then we have the following statements.

(S2.1) For $N = 1, 2, \ldots$, and $k = 0, 1, \ldots, N$, $\mathcal{T}_{k;N}$ is the set of all possible states x_k with $\mathcal{T}_{N:N} = \mathcal{T}$. When $k < N$, if and only if $x_k \in \mathcal{T}_{k;N}$, there exist controls u_k, u_{k+1}, $\ldots$, u_{N-1} such that $x_{k+1} = F(x_k, u_k) \in \mathcal{T}_{k+1;N}$, $x_{k+2} = F(x_{k+1}, u_{k+1}) \in \mathcal{T}_{k+2;N}$, $\ldots$, $x_N = F(x_{N-1}, u_{N-1}) \in \mathcal{T}_{N;N} = \mathcal{T}$. We have $\mathcal{A}_{k;N} \neq \emptyset$, $\mathcal{C}_{k;N} \neq \emptyset$ and $\mathcal{T}_{k;N} \neq \emptyset$.

(S2.2) We have $\mathcal{T}_{0;N} \supseteq \mathcal{T}_{1;N} \supseteq \cdots \supseteq \mathcal{T}_{N-1;N} \supseteq \mathcal{T}_{N;N} = \mathcal{T}$.

(S2.3) The set $\mathcal{A}_{k;N} = F^{-1}(\mathcal{T}_{k+1;N})$ is the inverse image of the set $\mathcal{T}_{k+1;N} \subset \mathbb{R}^n$ under F.

(S2.4) $\mathcal{T}_{k;N}$ is the orthogonal projection of $\mathcal{A}_{k;N} \subset \mathbb{R}^{n+m}$ in $\mathbb{R}^n$, i.e., $\mathcal{T}_{k;N} = p_x(\mathcal{A}_{k;N})$.

(S2.5) Similarly, $\mathcal{C}_{k;N} = p_u(\mathcal{A}_{k;N})$ is the orthogonal projection of $\mathcal{A}_{k;N} \subset \mathbb{R}^{n+m}$ in $\mathbb{R}^m$.

(S2.6) The optimal control $u_{N-1}^* = u_{N-1;N}^*(x_{N-1})$ at the final step for any possible state $x_{N-1} \in \mathcal{T}_{N-1;N}$ must satisfy the equation

$$F(x_{N-1}, u_{N-1}^*) \in \mathcal{T}.$$

If there is only one solution to this equation, the solution is just u_{N-1}^* since it is the only choice. If there are more than one solution, we will choose the one which minimizes the cost. Suppose $u = f^{(i)}(x)$, $i \in I$, are all the implicit functions defined by $F(x, u) = \xi$, $\xi \in \mathcal{T}$, where I is a suitable index set. Then

$$\begin{cases} J_{N-1;N}^*(x_{N-1}) = \min_{i \in I}\{\phi(x_N) + \\ \qquad\qquad + U(x_{N-1}, f^{(i)}(x_{N-1}))\} \\ \qquad = \phi(x_N) + U(x_{N-1}, u_{N-1;N}^*(x_{N-1})), \\ u_{N-1;N}^*(x_{N-1}) = f^{(i_0)}(x_{N-1}), \end{cases}$$

where i_0 is the index such that $f^{(i_0)}(x_{N-1})$ minimizes $\phi(x_N) + U(x_{N-1}, f^{(i)}(x_{N-1}))$.

(S2.7) For $k = 0, 1, \ldots, N-2$, let $x_k \in \mathcal{T}_{k;N}$ be any possible state, then

$$J_{k;N}^*(x_k) = \min_{u_k \in \mathcal{C}_{k;N}} \{U(x_k, u_k) + J_{k+1;N}^*(F(x_k, u_k))\}$$

and the minimum is reached when $u_k = u_{k;N}^*(x_k)$, i.e.,

$$u_{k;N}^*(x_k) = \arg\min_u \{U(x_k, u) + J_{k+1;N}^*(F(x_k, u))\}. \quad (8)$$

Statements (S2.1) and (S2.3)–(S2.6) all come directly from definitions. Statements (S2.2) and (S2.7) can easily be proved ([9]).

Now we have sequences:

$$\begin{array}{l} \{J_{k;N}^*(\cdot), k = 0, 2, \ldots, N\}, \\ \{u_{k;N}^*(\cdot), k = 0, 1, \ldots, N-1\}, \\ \{\mathcal{T}_{k;N}, k = 0, 1, \ldots, N-1\}, \\ \{\mathcal{C}_{k;N}, k = 0, 1, \ldots, N-1\}, \\ \{\mathcal{A}_{k;N}, k = 0, 1, \ldots, N-1\}, \end{array}$$

for $N = 1, 2, 3, \ldots$. For any initial state x_0, we need to use all of these sequences to find the optimal control for x_0. First, we need to find which one of $J_{0;N}^*(x_0)$, $N = 1, 2, \ldots$, is the smallest. If we can find N^* such that $J_{0;N^*}^*(x_0) = \min_N\{J_{0;N}^*(x_0)\}$, then $J_{0;N^*}^*(x_0)$ is the optimal cost and the associated control law $u_{k;N^*}^*(\cdot)$ will be used to determine the optimal control sequence.

Since the system considered here is time-invariant, we have the following relationships between items of these sequences for different N and k. We will state these relationships as Proposition 2.1. From these relationships, one can see that many items of $\mathcal{T}_{k;N}, \mathcal{C}_{k;N}, \mathcal{A}_{k;N}, J_{k;N}^*(\cdot)$, and $u_{k;N}^*(\cdot)$ are equivalent. Hence we do not need to handle so many items when we try to find the optimal control sequence.

Proposition 2.1: Let $N \geq 1$ and $k = 0, 1, \ldots, N-1$. Then

$$\begin{cases} \mathcal{T}_{k;N} = \mathcal{T}_{0;N-k}, \\ \mathcal{C}_{k;N} = \mathcal{C}_{0;N-k}, \\ \mathcal{A}_{k;N} = \mathcal{A}_{0;N-k}, \\ J_{k;N}^*(\cdot) = J_{0;N-k}^*(\cdot), \\ u_{k;N}^*(\cdot) = u_{0;N-k}^*(\cdot). \end{cases} \quad (9)$$

If the total number of steps of the system to reach the target is restricted to $N \leq \text{maxN}$, then the optimal cost and control are

$$\begin{aligned} J^o(x) &= J^*_{0;\text{maxN}}(x), \\ u^o(x) &= u^*_{0,\text{maxN}}(x). \end{aligned} \tag{10}$$

If there is no restriction on the total number of steps to reach the target, the optimal cost and control will be determined as follows. Let $\mathcal{T}_0 = \mathcal{T}$, $\mathcal{T}_k = \{x \in \mathbb{R}^n | \exists u \in \mathbb{R}^m$ such that $F(x,u) \in \mathcal{T}_{k-1}\}$ for $k = 1, 2, 3, \ldots$. If $x \in \mathcal{T}_N$, then

$$\begin{cases} J^o(x) = \inf\{J^*_{0;K}(x) | K \geq N\}, \\ u^o(x) = u^*_{0,K}(x), \quad \text{if } K \text{ is the smallest } K \\ \qquad\qquad \text{such that } J^o(x) = J^*_{0;K}(x). \end{cases} \tag{11}$$

III. Wavelet Basis Function Neural Network

Wavelet basis function (WBF) neural network is analogous to the well known radial basis function (RBF) neural network. The radial basis function, which is used in an RBF neural network, is replaced by a wavelet basis function.

In an RBF neural network, a function $f(x)$ is approximated as

$$f(x) \approx \alpha_0 + \sum_{k=1}^{K} \alpha_k \varphi\left(\frac{\|x - \mu_k\|}{\sigma_k}\right), \tag{12}$$

where α_k is the connecting weight of the kth neuron to the output neuron, μ_k is the center and σ_k is the width of the neuron, respectively, $k = 1, \ldots, K$. The radial basis function $\varphi(r) = \exp(-r^2)$ is the Gaussian function. During the training, the weights α_k and centers μ_k will be adjusted according to the input training pairs, while the width σ_k will decrease from a bigger number $\epsilon_{\max}$ to a smaller number $\epsilon_{\min}$.

In a WBF neural network, instead of the radial basis function, a target function will be approximated as a linear combination of wavelet basis. A *wavelet basis* is constructed with a *multiresolution approximation* of function space. Multiresolution approximation presents a way to approximate functions in multiple resolutions. Recall that the inner product of functions f and $g \in L^2$ is defined as $\langle f, g\rangle = \int f(x)g(x)dx$. A multiresolution approximation (MRA) of the function space L^2 is a doubly infinite nested sequence of subspaces of L^2

$$\cdots \subset V_{-1} \subset V_0 \subset V_1 \subset V_2 \subset \cdots$$

with properties [7]

(S3.1) $\cup_j V_j$ is dense in L^2, i.e., $\overline{\lim}_j V_j = L^2$.

(S3.2) $\cap_j V_j = \{0\}$.

(S3.3) $f(x) \in V_j \iff f(2x) \in V_{j+1}$ for all $j \in \mathbb{Z}$.

(S3.4) $f(x) \in V_j \iff f(x - 2^{-j}k) \in V_j$ for all $j, k \in \mathbb{Z}$.

(S3.5) There exists a function $\phi \in L^2$ so that $\{\phi(x - k) \colon k \in \mathbb{Z}\}$ forms an orthonormal basis of V_0. The function ϕ is called the *scaling function* of the MRA $\{V_j\}$.

(S3.6) Let $\phi_j^k(x) = 2^{j/2}\phi(2^j x - k)$ for $j, k \in \mathbb{Z}$. Then for any fixed integer $j \in \mathbb{Z}$, the set of functions $\{\phi_j^k | k \in \mathbb{Z}\}$ is an orthonormal basis of V_j.

(S3.7) There exists a function $\psi \in V_1$ so that $\langle \phi, \psi \rangle = 0$ and $\|\psi\| = 1$. Let $W_0 = \text{Span}\{\psi(x - k), k \in \mathbb{Z}\}$. Then the set of functions $\{\psi(x - k) | k \in \mathbb{Z}\}$ is an orthonormal basis of W_0, $W_0 \perp V_0$ and $V_1 = V_0 \oplus W_0$, where the symbol $\oplus$ denotes the orthogonal direct sum of function spaces [7]. ψ is called the *wavelet function* of the MRA $\{V_j\}$.

(S3.8) Let $\psi_j^k(x) = 2^{j/2}\psi(2^j x - k)$ for $j, k \in \mathbb{Z}$. Let $W_j = \text{Span}\{\psi_j^k : k \in \mathbb{Z}\}$. Then the set of functions $\{\psi_j^k | k \in \mathbb{Z}\}$ is an orthonormal basis of W_j, $W_j \perp V_j$ and $V_{j+1} = V_j \oplus W_j$.

(S3.9) For any $H_{\min}, H_{\max} \in \mathbb{Z}$, we have

$$V_{H_{\max}} = V_{H_{\min}} \oplus W_{H_{\min}} \oplus W_{H_{\min}+1} \oplus \cdots \oplus W_{H_{\max}-1}.$$

Furthermore,

$$L^2 = \overline{\lim_{H_{\max} \to \infty}} V_{H_{\max}} = V_{H_{\min}} \oplus (\oplus_{k=H_{\min}}^{\infty} W_k).$$

(S3.10) For any $f(x) \in L^2$ and any integer $j \in \mathbb{Z}$, there exists a unique function $P_j f \in V_j$ such that $f - P_j f \perp V_j$. $P_j f$ is called the approximation of f at resolution j. We have $\lim_{j \to \infty} P_j f = f$.

(S3.11) For any $f(x) \in L^2$ and any integer $j \in \mathbb{Z}$, there exists a unique function $Q_j f \in W_j$ such that $f - Q_j f \perp W_j$. $Q_j f$ is called the deviation of f at resolution j. We have $P_j f + Q_j f = P_{j+1} f$. Consequently, for any $H_{\min}, H_{\max} \in \mathbb{Z}$,

$$P_{H_{\max}} f = P_{H_{\min}} f + \sum_{H_{\min} \leq k < H_{\max}} Q_k f.$$

(S3.12) Let $f \in L^2$. Since $P_j f \in V_j$ and $\{\phi_j^k : k \in \mathbb{Z}\}$ is an orthonormal basis of V_j, we have

$$P_j f = \sum_k C_{j,k} \phi_j^k,$$

where $C_{j,k} = \langle f, \phi_j^k \rangle$.

(S3.13) Let $f \in L^2$. Since $Q_j f \in W_j$ and $\{\psi_j^k : k \in \mathbb{Z}\}$ is an orthonormal basis of W_j, we have

$$Q_j f = \sum_k D_{j,k} \psi_j^k,$$

where $D_{j,k} = \langle f, \psi_j^k \rangle$.

(S3.14) Let $f \in L^2$. For any given $H_{\min}, H_{\max} \in \mathbb{Z}$, we have the approximation

$$f(x) \approx \sum_{k \in \mathbb{Z}} C_{H_{\min},k} \phi_{H_{\min}}^k(x) + \sum_{s=H_{\min}}^{H_{\max}} \sum_{k \in \mathbb{Z}} D_{s,k} \psi_s^k(x). \tag{13}$$

(13) is called the wavelet approximation of f with resolutions from $H_{\min}$ to $H_{\max}$.

In a WBF neural network, a function is approximate by (13). The coefficients $C_{j,k}$ and $D_{j,k}$ are the weights of neurons. During the training, the values of the weights will be adjusted according to the training pairs. Comparing with the RBF neural network (12), we have the following differences:

(S3.15) A WBF neural network has two kind of basis functions, ϕ_j^k and ψ_j^k, coming from the scaling function ϕ and the wavelet function ψ, respectively, while an RBF neural network only has one kind of basis coming from one function φ.

(S3.16) During the training of a WBF neural network, only the weight coefficients $C_{j,k}$ and $D_{j,k}$ will be adjusted. But in the training of an RBF neural network, not only the coefficients α_k but also the centers μ_k and the widths σ_k will be modified.

For the scaling function ϕ and the associated wavelet function ψ of an MRA $\{V_j\}$, we know that $V_0 \subset V_1$ and $W_0 \subset V_1$. Hence ϕ and ψ can be written in terms of the basis of V_1. For $k \in \mathbb{Z}$, define

$$\begin{cases} h_k = \langle \phi^0_{-1}(x), \phi(x-k) \rangle, \\ g_k = \langle \psi^0_{-1}(x), \phi(x-k) \rangle. \end{cases}$$

Then (refer to [7])

$$\begin{cases} g_k = (-1)^{k-1} h_{1-k}, \\ \sum_k h_k h_{k-2s} = \delta_{0s}, \end{cases}$$

and

$$\begin{cases} \phi(x) = \sum_{s\in\mathbb{Z}} h_s \phi^s_1(x), \\ \psi(x) = \sum_{s\in\mathbb{Z}} g_s \phi^s_1(x). \end{cases} \tag{14}$$

For integers $j, k \in \mathbb{Z}$, replacing x by $2^j x - k$ in (14) gives

$$\begin{cases} \phi^k_j(x) = \sum_{s\in\mathbb{Z}} h_s \phi^{2k+s}_j(x), \\ \psi^k_j(x) = \sum_{s\in\mathbb{Z}} g_s \phi^{2k+s}_j(x). \end{cases} \tag{15}$$

The formulas (15) give the relationship between basis functions with different resolution levels. According to this relationship, it can be proved that the weights $C_{j,k}$ and $D_{j,k}$ of different resolution levels have the following relationship:

$$\begin{cases} C_{j-1,m} = \sum_{k\in\mathbb{Z}} h_{k-2m} C_{j,k}, \\ D_{j-1,m} = \sum_{k\in\mathbb{Z}} g_{k-2m} C_{j,k}. \end{cases} \tag{16}$$

Formulas (16) and statements (S3.12) and (S3.13) describe the way to adjust the weights $C_{j,k}$ and $D_{j,k}$ during the training. We will get $C_{j,k}$ and $D_{j,k}$ for the biggest $j = H_{\max}$ according to (S3.12) and (S3.13). Then we will calculate weights $C_{j,k}$ and $D_{j,k}$ backward from $j = H_{\max} - 1$ to $j = H_{\min}$ using (16). Thus, we get all the values of $C_{j,k}$ and $D_{j,k}$ ($j = H_{\min}, H_{\min}+1, \ldots, H_{\max}$) of the WBF neural network.

By [7], there exists wavelet basis which only has finite non-zero coefficients h_k and g_k. Such wavelets are called Daubechies' wavelets. It has the good property that both ϕ and ψ are functions with compact support, i.e., there exists a finite number R such that $\phi(x) = \psi(x) = 0$ when $|x| > R$. We will use Daubechies' wavelets in this paper for our WBF neural networks. Thus, all the summations in the formulas (13) and (16) are for finite terms.

Since ϕ and ψ are functions with compact support, all the ϕ^k_j and ψ^k_j are functions with compact support. Consequently, for any given state x, only finite number of basis functions will have non-vanish value at x. So, we only need to adjust the weights of these basis functions. In fact, it can be proved that there exists a limit N, no matter how many neurons the WBF neural network has, the number of weights which need to be adjusted are always smaller than N.

To modify the coeffiecients $C_{H_{\max},k}$ and $D_{H_{\max},k}$ for the highest resolution, we will calculate the error first. Suppose that we already have a WBF neural network

$$\hat{F} = \sum_{k\in\mathbb{Z}} C_{H_{\min},k} \phi^k_{H_{\min}}(x) + \sum_{s=H_{\min}}^{H_{\max}} \sum_{k\in\mathbb{Z}} D_{s,k} \psi^k_s(x).$$

Give a training pair (x_0, y_0). Then we have the error $\triangle = y_0 - \hat{F}(x)$. Define an error function

$$E(x) = \max\{1 - |x - x_0|, 0\}\triangle.$$

Let $\triangle C_{j,k} = \langle E(x), \phi^k_j(x) \rangle$, $\triangle D_{j,k} = \langle E(x), \psi^k_j(x) \rangle$. Then the weights can be adjusted by

$$\begin{cases} C^{new}_{j,k} = C_{j,k} + \triangle C_{j,k}, \\ D^{new}_{j,k} = D_{j,k} + \triangle D_{j,k}. \end{cases} \tag{17}$$

We describe the method of the training of WBF neural network as follows,

(S3.17) For a given pair of training data, update $C_{H_{\max},k}$ and $D_{H_{\max},k}$ for the highest resolution by (17).

(S3.18) For $H_{\min} \le j < H_{\max}$, update $C_{j,k}$ and $D_{j,k}$ by (16).

We can also update all the weights by (17), but that will need lots of integrations and require more training time. So we only calculate integrations for $j = H_{\max}$ and then obtain other weights with smaller j by (16).

IV. Discrete-Time Adaptive Dynamic Programming using WBF Neural Network

Now we recall the algorithm ADPDN ([9]), adaptive dynamic programming for discrete-time systems using neural networks. We will apply WBF neural networks to approximate the functions in the ADPDN algorithm.

(6) and (7) give a method for obtaining the optimal cost and optimal control for each individual k and N. The optimal control is always determined by the cost function. According to (10) and (11), the optimal control $u^o(x)$ of an arbitrary state x depends on two factors: which one of $\mathcal{T}_k$ contains x and the number of steps to reach the target. If $x \in \mathcal{T}_k$, then $x \in \mathcal{T}_i$ for any $i \ge k$. So, if the initial state is $x_0 = x$, then there exist control sequences which can make the system reach the target in k steps, $k+1$ steps, $k+2$ steps, $\ldots$, and so on. The optimal control will be chosen from all these controls by minimizing the cost values.

For every state x, we assign an integer $k = K(x)$ which indicate that the optimal cost will be realized in just k steps. Let $x \in \mathbb{R}^n$ be an arbitrary state. If there are some k's such that $J^*_{0;k} = \min_N J^*_{0,N}$, then $K(x)$ is the smallest k. If there is no k satisfying $J^*_{0;k} = \min_N J^*_{0,N}$, then $K(x)$ is infinite. In other words, if $K(x) = k_0$ then $J^*_{0;k_0}(x)$ is the optimal cost and if $K(x) = +\infty$, then the optimal cost cannot be reached in finite steps. It is clear that if $K(x) \le k$ then $x \in \mathcal{T}_k$ (but the converse conclusion does not hold). When $K(x) \le k$, the optimal control and optimal cost of x can be calculated by formula (10) or (11). Define $\mathcal{T}_k^* = \{x | K(x) = k\}$. Then $\mathcal{T}_k^*$ is the set of all such state x whose optimal control is $u^o(x) = u_{0;k}^*(x)$. We have that $\mathcal{T}_k^* \subseteq \mathcal{T}_k$ and $\mathcal{T} = \mathcal{T}_0^* \subseteq \mathcal{T}_1^* \subset \mathcal{T}_2^* \subseteq \cdots$. If $x \in \mathcal{T}_k^*$, then $F(x, u^o(x)) \in \mathcal{T}_{k-1}^*$, $J^o(x) = J_{0;k}^*(x) = \min_v(U(x,v) + J^o(F(x,v)))$.

The plant is deterministic but unknown. So we need to approximate it from the observation while it is running. We use neural networks to perform the approximation. We will use four neural networks, named $\hat{F}(x,u)$, $\hat{K}(x)$, $\hat{J}(x,k)$, and $\hat{u}(x)$, respectively, to approximate the functions $F(x,u)$, $K(x)$, $J_{0;k}^*(x)$, and $u^o(x)$ in our algorithm. The neural networks $\hat{F}(x,u)$, $\hat{K}(x)$, $\hat{J}(x,k)$ and $\hat{u}(x)$ will be trained whenever some observations are obtained. We will compare the performance of RBF neural networks and WBF neural networks for implementing the ADPDN algorithm.

As a computer algorithm, we only consider bounded regions and finite control steps. We use a number maxN to express the upper bound of the control steps. We will require that the total steps of the system to reach the target from any initial state be restricted to $N \leq$ maxN. In other words, for an initial state x, if one cannot control the system to reach the target within maxN steps starting from x, then we will consider x to be uncontrollable.

Now we recall the algorithm ADPDN, adaptive dynamic programming for discrete-time systems using neural networks.

Algorithm ADPDN (Adaptive Dynamic Programming for Discrete-time systems using Neural networks)

```
A01  Initialize neural networks $\hat{F}$, $\hat{K}$, $\hat{J}$ and $\hat{u}$;
A02  Choose an array of initial states $x_0$ randomly.
A03  Determine the number of control steps for $x_0$ by
     $k = \hat{K}(x_0)$;
A04  Determine the control signal by $u = \hat{u}(x_0, k)$;
A05  Run the plant to obtain $x_1 = F(x_0, u)$;
A06  Train neural network $\hat{F}$ by $\hat{F}(x_0, u) = x_1$;
A08  Train neural network $\hat{K}$ by $\hat{K}(x_0) = \hat{K}(x_1) + 1$;
A09  Adjust neural networks $\hat{J}$ and $\hat{u}$ according to (S2.6)
     and (S2.7).
A10  Let $x_0 = x_1$. Repeat A03–A09 for maxN times,
     where maxN is a prespecified positive number.
A11  Repeat A02–A10 until $\hat{J}$ is convergent.
```

The initial states x_0 are given in A02 randomly. When the iteration time is big enough, we can consider that x_0 will cover the entire state space, or the place in the state space where one expects to operate the system. Hence the algorithm will explore over the entire (interesting) state space. The update of $\hat{J}$ and $\hat{u}$ depend on the statements (S2.6) and (S2.7). (S2.6) and (S2.7) are obtained by applying the optimal principle to the sequences of functions $J_{k;N}^*$ and $u_{k;N}^*$. The limit of $\hat{J}$ and $\hat{u}$ will approximate the optimal cost $J_{k;N}^*$ and the optimal control $u_{k;N}^*$, respectively.

The final outputs of the algorithm ADPDN are four neural networks. $\hat{F}$ and $\hat{J}$ give the approximations to the plant F and the optimal cost $J_{0;N}^*$, and $\hat{u}$ is an approximation to the optimal controller u^o. Meanwhile, $\hat{K}$ gives the region where optimal control exists and how long it will last for controlling x to reach the target. For a state x, if $\hat{K}(x) =$ maxN then we can consider x to be uncontrollable. If $\hat{K}(x) <$ maxN, then $\hat{K}(x)$ is an estimate of the number of control steps to drive x to the target. Before applying the controller $\hat{u}$, use $\hat{K}$ to check whether the initial state is controllable. Then use $\hat{u}$ as controller to control it.

The algorithm ADPDN can apply to both linear and nonlinear systems. We just need that F and U are (uniformly) continuous functions and satisfy the conditions (C2.1)–(C2.4). In the next section, numerical experiments on an unstable nonlinear system will be provided. Our algorithm works well for this system.

V. Experiment

To evaluate the performance of the algorithm ADPDN when WBF neural networks are applied, and compare it with the case when RBF neural netwoks are applied, we select a simple system for numerical experiment. We consider a nonlinear system given by

$$x_{k+1} = F(x_k, u_k) \overset{\triangle}{=} x_k + \sin(x_u + u_k), \tag{18}$$

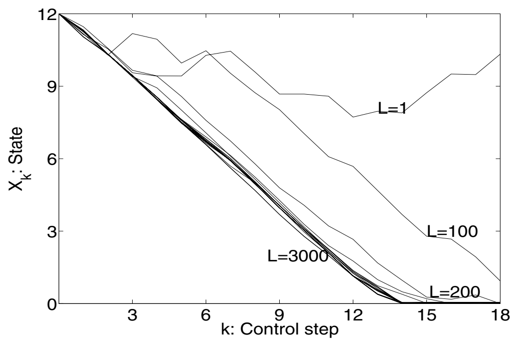
where $x_k, u_k \in \mathbb{R}$, and $k = 0, 1, 2, \ldots$. The control target is $\mathcal{T} = \{0\}$ and the utility functions are $U(x,u) = |x| + u^2$ and $\phi(x) \equiv 0$. Since $F(0,0) = 0$, 0 is an equilibrium state of system (18). But $\frac{dF}{dx}(0,0) = 2 > 1$, (18) is unstable at $x = 0$.

The region of states considered here is $|x| \leq 15$ and the number of control steps is restricted to 18. The size of the set of initial states x_0 (in the line A04 of the algorithm) will be 500 each time, i.e., at the beginning of the each inner loop iteration, we choose 500 initial states randomly.

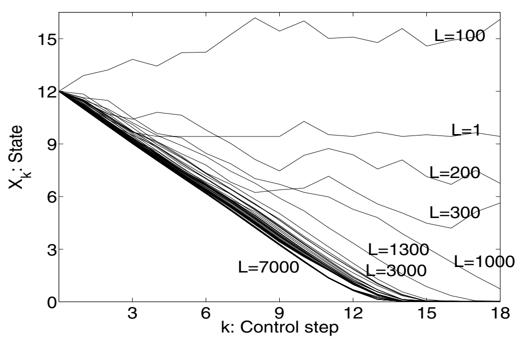
When WBF neural networks are applied to approximate the functions, we choose the Haar wavelet as the basis, which is the wavelet with the shortest support on all Daubechies' wavelets. We choose the minimal resolution as $H_{\min} = -4$ and set the maximal resolution to be $H_{\max} = 8$. Notice that $2^{-H_{\min}} = 2^4 = 16$ is close to the radius of the region $[-15, 15]$, and $2^{-H_{\max}} = 2^{-8} = 0.0039$ will limit the error of the approximation.

When RBF neural networks are applied, we choose upper bound of the width of the basis as $\epsilon_{\max} = 15$ and the lower bound of the width of the basis as $\epsilon_{\min} = 0.0025$. Notice that $\epsilon_{\min} < 2^{-H_{\max}}$, which indicates that in this experiment, the RBF neural networks may give more accurate approximations than WBF neural networks will give. But the simulation results show that WBF neural networks have better accuracy.

In both cases when RBF neural networks or WBF neural networks are applied, we perform our ADPDN algorithm and save $\hat{F}$, $\hat{J}$, $\hat{K}$ and $\hat{u}$ at the end of the outer loop of each outer loop iteration. Thus we obtain sequences of $\hat{F}$, $\hat{J}$, $\hat{K}$ and $\hat{u}$. They will approximate $F(x,u)$, $J^*(x,k)$, $K(x)$ and $u^o(x)$, respectively. Let $\hat{J}^o(x) = \lim_{k\to\infty} \hat{J}(x,k)$. Then $\hat{J}^o(x)$ will

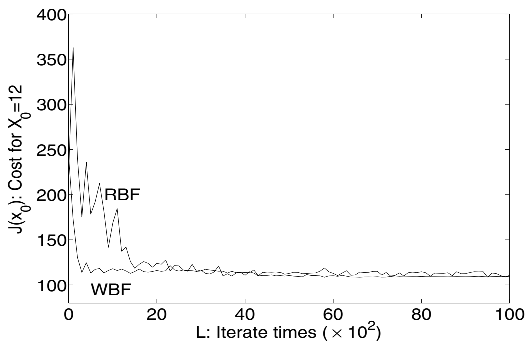


(a) Using WBFNN, $0 \le L \le 3000$

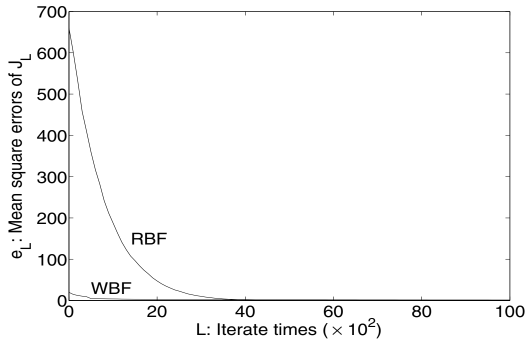


(b) Using RBFNN, $0 \le L \le 7000$

Fig. 1. The trajectories from initial state $x_0 = 12$.



(a) The cost for initial state $x_0 = 12$



(b) The mean square errors of $\hat{J}_L$

Fig. 2. Performance costs and its mean square errors

approximate J^o while L is increasing, where L is the number of outer loop iterations.

Figure 1 shows the trajectories starting from the initial state $x_0 = 12$ under the control law $\hat{u}$. Figure 1 (a) shows the trajectories when WBF neural networks are applied while Figure 1(b) shows those when RBF neural networks are applied. In Figure 1(a), we can see that the algorithm find the optimal trajectory at $L = 3000$ when WBF neural networks are applied. This optimal trajectory will go from 12 to the target 0 in 14 steps. But as shown in Figure 1(b), when RBF neural networks are applied, the algorithm has not found the way to make the trajectory go to the target at $L = 1000$. Even till $L = 3000$, the trajectory is not the optimal one: it can only reach the target in 17 steps instead of 14 steps. The trajectory tends close the optimal one when $L = 7000$.

Figure 2 shows the comparison of the performance costs. Figure 2(a) shows $\hat{J}_L(x_0)$ for $0 \le L \le 10000$. When WBF neural networks are applied, $\hat{J}_L(x_0)$ tends to reach stable after L go through 200. But when RBF neural networks are applied, $\hat{J}_L(x_0)$ tends to reach stable after L go through 2000. Figure 2(b) shows the mean square errors between $\hat{J}_L$ and J^o, i.e.,

$$e_L = \frac{1}{30}\int_{-15}^{15} (\hat{J}_L - J^o)^2 dx.$$

Similar with the case Figure 2(a), when WBF neural networks are applied, e_L tends to reach stable after L go through 500, but when RBF neural networks are applied, e_L tends to reach stable after L go through 4000.

VI. CONCLUSIONS

In this paper, instead of the RBF neural networks, the new WBF neural networks are applied in the algorithm ADPDN, for adaptive dynamic programming of discrete-time systems using Neural networks, to approximate the associated functions F, J, K and u. Numerical experiments are performed. The results indicate that WBF neural networks work better than RBF neural networks in dynamic programming. When WBF neural networks are applied, the time for training controller will be shorter. The algorithm ADPDN can find the optimal control more quickly when WBF neural networks are employed.

To get the optimal trajectory $\{x_i\}$ from an initial state x_0, one has to find a series of control actions which will act in sequence. In Section II, we indicate that all these different optimal control signals depend on the number of steps of the control. We introduce the symbols $u^*_{k;N}$ and $J^*_{k;N}$ to represent the optimal controllers and the optimal costs. The subscripts k and N means the kth step in the total of N steps.

We proved that $u^*_{k;N} = u^*_{0;N-k}$ and $J^*_{k;N} = J^*_{0;N-k}$ in Proposition 2.1. But in general, one cannot say that $u^*_{0;k} = u^*_{0;m}$ or $J^*_{0;k} = J^*_{0;m}$ when $k \neq m$. In fact, it is well known that the optimal control law of linear discrete-time

time-invariant system with quadric utility can be obtained by a backward iterated formula (cf., e. g., [12]). All the control laws for different steps are different in this case.

The algorithm ADPDN is based on the discussion of Section III. The control steps $\hat{K}(x)$ plays an important role in the algorithm. It will give the region where the states are controllable and will indicate how long it will need to control the state to reach the target. It also enables our algorithm to have the ability for exploring new values of control and to approximate the optimal costs $J^*_{0;k}$ for all $k = 0, 1, 2, \dots$.

Because of errors in the numerical algorithm, the target $\mathcal{T}$ in fact is just $\{x+r|x \in \mathcal{T}, |r| < \epsilon\}$ during the running of the program. Hence we can finally have $\hat{u} \approx u^o$ as the resulting optimal controller. It will serve as the unique controller which can be applied to all states in all the steps to get an optimal trajectory.

The algorithm ADPDN avoid to solve the HJB equation. Instead of solving differential equation, the ADPDN algorithm find the minimal cost by direct calculations from the associated neural networks. Thus, the ADPDN algorithm can deal with a wide range of systems. As we have stated in Section II, $F(x, u)$ is just a continuous function. We do not need to restrict F to be a differentiable function. We also do not have the restriction such as $F(x, u)$ to be the form $A(x) + B(x)u$ or $\theta_1 x_1^2 + \theta_2 x_1 x_2 + \theta_3 x_2 u$.

The learning behavior of algorithm ADPDN is quite similar to the learning behavior of a human being. At the beginning, the algorithm cannot find the optimal control. It even cannot drive the state along the right way to the target. But after some times of running, it will know how to reach the target. When it has enough trials, it will find a good approximation of the optimal controller.

References

[1] 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.

[2] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike elements that can solve difficult learning control problems," *IEEE Trans. on Sys., Man., Cybern.*, vol. 3, no. 5, pp. 455–465, 1983.

[3] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, 1957.

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

[5] C. Cox, S. Stepniewski, C. Jorgensen, R. Seaks, 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.

[6] 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.

[7] I. Daubechies, *Ten Lectures on Wavelets*, Society for Industrial and Applied Mathematics Press, vol. 61 of CBMS-NSF Regional Conference Series in Applied Mathematics, Philadelphia, 1992.

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

[9] Ning Jin and Derong Liu, "Finite Horizon Discrete-Time Approximate Dynamic Programming," *Porc. IEEE International Symposium on Intelligent Control*, Munich, Germany, Oct. 2006.

[10] G. G. Lendaris, C. Cox, R. Seaks, and J. Murray, "A radial basis function implementation of the adaptive dynamic programming algorithm," *Proc. 45th Midwest Symposium on Circuits and Systems*, vol. 2, pp. II338–II341, Aug. 2002.

[11] A. U. Levin and K. S. Narendra, "Control of nonlinear dynamical systems using neural networks: Controllability and Stabilization," *IEEE Trans. Neural Networks*, vol. 4, no. 2, pp. 192–206, 1993.

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

[13] D. Liu, Y. Zhang, and H. Zhang, "A self-learning call admission control scheme for CDMA cellular networks," *IEEE Trans. Neural Networks*, vol.16, no.5, pp. 1219-1228, 2005.

[14] Johb. J. Murray, Chadwick J. Cox, George G. Lendaris, and Richard Seaks, "Adaptive Dynamic Programming," *IEEE Trans. of Sys., Man, and Cyb.-Part C: App & Reviews*, vol. 32, no. 2, pp. 140–153, 2002.

[15] K. S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

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

[17] F. Rosenblatt, *Principles of Neurodynamics*, Spartan Books, Washington DC, 1962.

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

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

[20] R. S. Sutton, *Reinforcement Learning*, Kluwer Academic, Boston, 1996.

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

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

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

[24] P. J. Werbos, "Stable adaptive control using new critic designs," [Online]. Available: http://xxx.lanl.gov/abs/adap-org/9810001, Mar. 1998.

[25] P. J. Werbos, "Approximate dynamic programming for real-time control and neural modeling," *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.

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

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

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

[29] P. J. Werbos, "Beyond regression: New Tools for Prediction and Analysis in the Behavioral Sciences", *PhD. thesis, Committee on Applied Mathematics*, Harvard Univ., Cambridge, MA, 1974.

[30] B. Widrow, N. Gupta, and S. Maitra, "Punish/Reward: learning with a critic in adaptive threshold systems," *IEEE Trans. on Syst., Man, Cybern.*, vol. 3, no. 5, pp. 455–465, 1973.

[31] B. Widrow and M. Lehr, "30 Years of adaptive neural networks: perceptron, madaline, and backpropagation," *Proc. IEEE*, vol. 78, no. 9, pp. 1415–1442, Sept. 1990.

[32] W. M. Wonham, "Random differential equations in control theory," *Probabilistic Methods in Applied Mathematics*, Edited by A. T. Bharucha-Reid., N. Y., Academic Press, 1970.

[33] Y. Zhang and D. Liu, "Call admission control for CDMA cellular networks using adaptive critic designs," *Proceedings of the 18th IEEE International Symposium on Intelligent Control*, Houston, TX, pp. 511–516, Sept. 2003 (Invited paper).