Neural Network-based ϵ-Adaptive Dynamic Programming and ϵ-Optimal Control for Nonlinear Systems

BY

NING JIN
B.S., East China Normal University, China, 1984
M.S., East China Normal University, China, 1987
Ph.D (in Math), East China Normal University, China, 1990

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2011

Chicago, Illinois

Defense Committee:

Derong Liu, Chair and Advisor
Natasha Devroye
Sudip K. Mazumder
Dan Schonfeld
Houshang Darabi, Mechanical and Industrial Engineering

ACKNOWLEDGMENTS

I would like to express my sincere and profound thanks to my adviser, Professor Derong Liu, for his patient guidance and constant encouragement during my years of study at the University of Illinois. It is his consistent support and rigorous supervision which help me make steady progress in the Ph.D. study. Working with Professor Liu has been a highly rewarding and extremely enjoyable experience.

I am very grateful to Dr. Houshang Darabi, Dr. Natasha Devroye, Dr. Sudip K. Mazumder and Dr. Dan Schonfeld for serving on my Ph.D. dissertation committee and for their valuable time, support, comments and suggestions.

Finally, I would like to thank all my officemates Mr. Ting Huang, Mr. Zhuo Wang, Ms. Shu Wang, Ms. Yingying Ma, Mr. Nan Xu, Dr. Sanqing Hu, Dr. Xiaoxu Xiong, Dr. Ying Cai, Dr. Chonghui Song, Dr. Dan Meng, Dr. Qi Kang, Dr. Zhigang Liu, Mr. Le Liu, Ms. Na Dong, Ms. Yang Zhang, Professor Xiaofeng Lin, Professor Guowei Yang, and Professor Peifeng Niu for helpful discussions.

NJ

TABLE OF CONTENTS

TABLE OF CONTANTS (continued)

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF ABBREVIATIONS

ACD	Adaptive Critic Design
ADP	Adaptive Dynamic Programming
ADPDN(ϵ)	ϵ-ADP for Discrete-time Systems using Neural Network
GDHP	Globalized Dual Heuristic Programming
HDP	Heuristic Dynamic Programming
HJB	Hamilton-Jacobi-Bellman
MRA	Multiresolution Approximation
MRAN	Minimal Resource Allocation Networks
NN	Neural Network
RAN	Resource Allocation Network
RANEKF	RAN via Extended Kalman Filter
RBFNN	Radial Basis Function Neural Network
RL	Reinforcement Learning
RLS	Reinforcement Learning System
SLWBF	Sequential Learning of WBFNN
WBFNN	Wavelet Basis Function Neural Network
WNN	Wavelet Neural Network

SUMMARY

Optimal control and dynamic programming for complex dynamical systems is difficult due to two reasons. The first is the so called "curse of dimensionality": one has to find a series of control actions that must be taken in sequence. This sequence will lead to the optimal performance index, but the total cost of these actions will be unknown until the end of the sequence. Thus, the time expense and the space expense will be huge during the dynamic programming process. The second is the accuracy of the calculation. When an approximation of the optimal performance index is obtained, it is possible that the approximation is even "more optimal" than the optimal performance index. And then, unfortunately, this "over optimal" approximation will provide a control sequence which will possibly drive the system to run unstably.

The aim of our work is to overcome these two difficulties. We start by considering the "over optimal" problem. We introduce a novel ϵ-optimal performance index function $V_\epsilon^*(\cdot)$ as an approximation of the optimal performance index fuction. The associated ϵ-optimal controller $\mu_\epsilon^*(\cdot)$ can always control the system state to approach the equilibrium state stably, while the performance index is close to the optmal performance index within an error bound according to ϵ. An numerical algorithm to find the ϵ-optimal controller is suggested and simulated experiments are performed to explore the behavior of the algorithm. The experiment results show that the algorithm works well. The algorithm can find a good approximation of the ϵ-optimal controller.

In the first stage of our work, we study the nonlinear discrete-time systems while there is no discount in the performance index, i.e., the discount

factor in the performance cost is 1. We define a novel ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$ based on the optimal performance cost functions $J_k^*(\cdot)$, $k = 1, 2, \cdots$. According to the Bellman's principle of optimality, the optimal performance cost functions $J_k^*(\cdot)$ satisfy the Bellman equation. The optimal control sequence $v_k^*(x)$ will be obtained by solving the Bellman equation backforward. The process to solve $v_k^*(x)$ from the Bellman equation is called dynamic programming. In our work, we show that our ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$ also satisfis the Bellman equation. A method similar to dynamic programming can be applied on $V_\epsilon^*(\cdot)$ so that a ϵ-optimal controller μ_ϵ^* can be obtained from the Bellman equation of $V_\epsilon^*(\cdot)$. The associated ϵ-optimal controller $\mu_\epsilon^*(\cdot)$ can always control the state to approach the equilibrium state, while the performance cost is close to the greatest lower bound of all performance cost within an error bound according to ϵ. We call this method the ϵ-adaptive dynamic programming method. Since only one performance cost function is used in ϵ-adaptive dynamic programming, ϵ-adaptive dynamic programming is helpful to overcome the curse of dimensionality.

After the theory of ϵ-adaptive dynamic programming is established, a numerical algorithm for the ϵ-adaptive dynamic programming is designed. Neural networks are applied to be the approximate structures of the associated system model, performance cost function, the controller, et al. These neural networks are trained according to the running of the plant and the Bellman equation. The algorithm will provide a sequence of approximations of the ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$. The associated controllers can always control the state to approach the equilibrium state, while the performance cost is approaching the ϵ-optimal performance cost. Examples of simulation are studied. The results of simulation show that our ϵ-adaptive dynamic programming can provide the stability when the performance cost

function is close to the optimal performance cost function.

After we established the ϵ-adaptive dynamic programming theory for system without discount factor in the performance cost function, we began to study the more general case. We study the nonlinear discrete-time systems that have a discount factor $0 < \gamma \leq 1$ in their performance index functions. This case seems more complex than the case without discount factor (i.e., the discount factor is 1). However, it is fortunate that the concepts and results of ϵ-optimal control can be generalize to this more general case. The ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$ is defined which will approach the least upper bound $J_\infty^*(\cdot)$ of the optimal performance cost functions $J_k^*(\cdot)$, $k = 1, 2, \cdots$, when $\epsilon \to 0$. $V_\epsilon^*(\cdot)$ also satisfies the Bellman equation in this case. Consequenctly, we obtain a generalized ϵ-optimal control and dynamic programming theory. An admissible controller can be obtained by solving the Bellman equation of $V_\epsilon^*(\cdot)$ so that the system state will attend to the equilibrium state under the control of this controller. Similar to the case without discount factor, we provide an implementation of the adaptive dynamic programming method for the case with discount factor. We design a numerical algorithm for the ϵ-adaptive dynamic programming. The algorithm is similar to that in case without discount factor. It will provide a sequence of approximations of the ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$. A simulated experiment is provided and it showes the algorithm runs well. A stable controller is obtained which provides an ϵ-optimal performance cost.

The previous results on ϵ-optimal dynamic programming provide stable controllers in the sense of ϵ-optimality. By using a single performance cost function in ϵ-adaptive dynamic programming, ϵ-adaptive dynamic programming is also helpful to overcome the curse of dimensionality. However, the

time expense of the ϵ-optimal dynamic programming algorithm is still big. In the third part of this thesis, certain restriction is set on the system so that fast algorithms can be found. It is assumed that the utility function of system is a positive definite quadratic function. Under this assumption, the iterative adaptive dynamic programming (ADP) algorithm using globalized dual heuristic programming (GDHP) technique is introduced to obtain the optimal controller with convergence analysis in terms of cost function and control law. In order to implement the iterative algorithm, a neural network is constructed first to identify the unknown nonlinear system. Then, based on the learned system model, two other neural networks are used as parametric structures to facilitate the implementation of the iterative algorithm, which aims at approximating at each iteration the cost function and the control law, respectively. A simulation example is provided to verify the effectiveness of the presented ADP algorithm using GDHP technique.

The last part of this thesis is about wavelet neural networks. In the numerical simulations in our research on dynamic programming, we use neural networks to approximate the functions, such as the performance cost function and the oprimal controller. The neural networks will be trained from sequences of input-output data. So the generalization ability of the neural networks is important for our purpose. Radial basis function neural networks (RBFNNs) and wavelet neural networks (WNNs) are well known neural networks that have good generalization ability. In a RBFNN, a function $f(x)$ is approximated as $\hat{f}(x) = \sum_i w_i \phi\left(\frac{\|x-a_i\|}{b_i}\right)$, where $\phi(r)$ is the basis function. In a WNN, a function $f(x)$ is approximated as $\hat{f}(x) = \sum_i w_i \phi\left(\frac{x-a_i}{b_i}\right)$, where $\phi(x)$ is the basis function coming from wavelet theory – the scaling function, the wavelet function, or the basis function of continuous wavelet transform. WNNs can approximate functions more accurately and they have better gen-

eralization property than RBFNNs. But all the existing training algorithms of WNNs are not especially for sequential learning and the orthogonal properties of wavelets have not been used in these algorithms. Hence we designed a wavelet basis function neural networks (WBFNNs) for sequential learning. Both the scaling function ϕ and the wavelet function ψ are used as basis functions in WBFNN. Functions ϕ and ψ are orthogonal to each other. In a wavelet decomposition of a function, they will give approximations in different level of details, i.e., coarse and fine approximations, respectively. A sequential learning algorithm is produced from the orthogonal properties of multiresolution approximation and Mallat's formula of wavelet decomposition.

1 INTRODUCTION

1.1 Dynamic Programming and the Curse of Dimensionality

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" [5, 11]. One has to find a series of control actions that must be taken in sequence. This sequence will give the optimal performance index, but the total cost of those actions is unknown until the end of that sequence. For continuous-time systems, if the optimal performance index 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 on system function and utility function, the optimal cost function is the solution of the Hamilton-Jacobi-Bellman (HJB) equation [5]. 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 when the target is zero. For discrete-time systems, this problem becomes even more difficult. In the discrete-time case, the optimal performance index function and optimal controller are variant in each step. We have to consider a sequence of functions for dynamic programming.

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, 27, 32, 35, 36, 42, 43, 44, 45]). The idea is to approximate the dynamic programming solutions by using function approximation structures to approximate the optimal cost function and the

optimal controller.

In 1994, Saridis and Wang [35] studied the nonlinear stochastic systems described by

$$dx = F(x,t)dt + B(x,t)udt + G(x,t)dw, \tag{1.1.1}$$

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 index of the system (1.1.1) is given by

$$\begin{aligned} J(x_0;t_0,u) =& E\Big\{ \int_{t_0}^{+\infty} \big[U(x,t) + \|u\|^2 \ \big]\, dt \\ & + \phi(x(T),T) \colon x(t_0) = x_0 \Big\}, \end{aligned}$$

where $U(\cdot,\cdot)$ and $\phi(\cdot)$ are nonnegative functions. For a given initial condition $x(t_0) = x_0$, the optimal control law u^* and the associated optimal cost J^* satisfy

$$J^*(x_0;t_0) \equiv J(x_0;t_0,u^*) = \inf_u J(x_0;t_0,u).$$

If it is assumed that the optimal control law u^* exists and if the corresponding cost function J^* is sufficiently smooth, then u^* and J^* may be found by solving Hamilton-Jacobi-Bellman (HJB) equation

$$\begin{cases} \dfrac{\partial J^*}{\partial t} + \min_u \left\{ \dfrac{1}{2}\mathrm{tr}\left[GG^T \dfrac{\partial}{\partial x}\left(\dfrac{\partial J^*}{\partial x}\right)^T \right] + \right. \\ \qquad \left. \left(\dfrac{\partial J^*}{\partial x}\right)^T [F + Bu] + U(x,t) + \|u\|^2 \right\} = 0, \\ J^*(x(T),T) = \phi(x(T),T). \end{cases} \tag{1.1.2}$$

Except in the case of linear quadratic Gaussian control (see [47]), an analytical solution of the Hamilton-Jacobi-Bellman for solving the optimal stochastic control problem cannot be obtained in general when the system (1.1.1) is

nonlinear. Instead of solving the Hamilton-Jacobi-Bellman equation (1.1.2), Saridis and Wang [35] introduced the following equation

$$\frac{\partial V}{\partial t}+\frac{1}{2}\mathrm{tr}\left[GG^T\frac{\partial}{\partial x}\left(\frac{\partial V}{\partial x}\right)^T\right]+\left(\frac{\partial V}{\partial x}\right)^T(F+Bu)$$
$$+U(x,t)+\|u\|^2=\nabla V. \tag{1.1.3}$$

An upper bound V^* and a lower bound V_* of the optimal cost J^* are found by solving equation (1.1.3). Then a control law $u(x)$ can be obtained by applying V^* (or V_*) as Lyapunov function for the system. This leads to the so-called "suboptimal control" of the system. It was proved that such controls are stable for infinite-time approximative optimal control problems. The benefit of the suboptimal control is that the bound V of the optimal cost J^* can be approximated by an iterative process. Starting from certain chosen stable controller u_0 and the associated performance index function V_0, let

$$u_i=-\frac{1}{2}B^TV_{i-1},\quad i=1,2,\ldots, \tag{1.1.4}$$

put $u=u_i$ in (1.1.3) and get the solution V_i from

$$\frac{\partial V_i}{\partial t}+\frac{1}{2}\mathrm{tr}\left[GG^T\frac{\partial}{\partial x}\left(\frac{\partial V_i}{\partial x}\right)^T\right]+\left(\frac{\partial V_i}{\partial x}\right)^T(F+Bu_i)$$
$$+U(x,t)+\|u_i\|^2=\nabla V_i. \tag{1.1.5}$$

Then repeatedly applying (1.1.4) and (1.1.5), one will get sequences of functions $\{u_i\}$ and $\{V_i\}$. The sequence $\{V_i\}$ will converges to the bound V^* (or V_*) of the cost function J^*. Consequently, $\{u_i\}$ will approximate the suboptimal control sequence. Suboptimal control works well and gives good control results. The sequences $\{V_i\}$ and $\{u_i\}$ are obtainable by computation

and they approximate the suboptimal cost and the suboptimal control law, respectively.

Meanwhile, a neural network-based approach for approximate dynamic programming has been developed in the literature. There are several synonyms used including "Adaptive Critic Designs" (ACD), "Approximative Dynamic Programming", "Neural Dynamic Programming", "Reinforcement Learning" (RL), and so on. In the early 1970's, Werbos set up the basic strategy for ACD (cf. [41, 45, 46] 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. They can achieve good approximation most of the time.

In 2002, Murray et al. [18, 27] studied the (deterministic) continuous-time stabilizable systems

$$\frac{dx}{dt} = F(x) + B(x)u, \quad x(t_0) = x_0, \tag{1.1.6}$$

with the cost function

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

where $U(x,u) = q(x) + u^T r(x)u$ is a nonnegative function and $r(x) > 0$. Similar to [35], an iterative process is proposed to find the control law. But this time the optimal cost and optimal control law are approximated. In this case, the Hamilton-Jacobi-Bellman equation can be simplified to

$$u^*(x) = -\frac{1}{2} r^{-1}(x) B^T(x) \left[\frac{dJ^*(x)}{dx}\right]^T. \tag{1.1.7}$$

Starting from any stable Lyapunov function J_0 (or alternatively, starting from an arbitrary stable controller u_0, cf. [18]) and replacing J^* by J_i, (1.1.7) becomes

$$u_i(x) = -\frac{1}{2}r^{-1}(x)B^T(x)\left[\frac{dJ_i(x)}{dx}\right]^T, \tag{1.1.8}$$

where $J_i = \int_{t_0}^{+\infty} U(x_{i-1}, u_{i-1})dt$ is the cost of the trajectory $x_{i-1}(t)$ of plant (1.1.6) under the input $u(t) = u_{i-1}(t)$. The sequences $\{J_i\}$ and $\{u_i\}$ will converge to the optimal cost function J^* and the optimal control u^*, respectively. By applying radial basis function approximation, no prior information on $F(x)$ and $B(x)$ are required in the algorithm. Thus their algorithm is an adaptive algorithm.

In 2007, Al-Tamimi and Lewis [2] studied the (deterministic) discrete-time stabilizable systems

$$x_{k+1} = f(x_k) + g(x_k)u_k, \tag{1.1.9}$$

with the cost function

$$J(x_k) = \sum_{i=k}^{+\infty}\left(x_i^T Q x_i + u_i^T R u_i\right), \tag{1.1.10}$$

where $Q \in \mathbb{R}^{n\times n}$ and $R \in \mathbb{R}^{m\times m}$ are positive definite matrices. Similar to [35], an iterative process, which is referred as *Heuristic Dynamic Programming* (HDP, cf. [43]), is proposed to find the optimal control law. Starting from $V_0(x) \equiv 0$, define

$$\begin{cases} u_i(x_k) = -\frac{1}{2}R^{-1}g(x_k)^T\frac{\partial V_i(x_{k+1})}{\partial x_{k+1}}, \\ V_{i+1}(x_k) = x_k^T Q x_k + u_i^T(x_k)Ru_i(x_k) + V_i(x_{k+1}), \end{cases} \tag{1.1.11}$$

where $x_{k+1} = f(x_k) + f(x_k)u_i(x_k)$. In [2], it was proved that $\{V_i\}$ is a nondecreasing sequence and bounded, and hence converges to V^*. It was claimed that the limit V^* is the optimal cost and satisfies the HJB equation

$$V^*(x) = \min_u \left[x^T Qx + u^T Ru + V^*(f(x) + g(x)u)\right], \tag{1.1.12}$$

and the sequence $\{u_i\}$ converges to the optimal controller u^*, which is given by

$$u^*(x) = \arg\min_u \left[x^T Qx + u^T Ru + V^*(f(x) + g(x)u)\right]. \tag{1.1.13}$$

Under the control of u^*, the system will converge to the equilibrium state as a limit when time $k \to \infty$, with the optimal cost V^*.

1.2 The ϵ-Adaptive Dynamic Programming

In this thesis, the optimal control of nonlinear discrete-time systems is considered. While there is no discount factor in the performance cost, i.e., the discount factor in the performance cost is just 1, a novel ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$ was introduced to approximate the optimal cost fuction. The associated ϵ-optimal controller $\mu_\epsilon^*(\cdot)$ can always control the state to approach the equilibrium state, while the performance cost is close to the greatest lower bound of all performance cost within an error bound according to ϵ. An algorithm to find the ϵ-optimal controller was suggested and numerical experiments were performed to explore the behavior of the algorithm. The experiment results shown that the algorithm works well. It can find a good approximation of the ϵ-optimal controller.

The method by using ϵ-optimal performance cost and ϵ-optimal controller to approximate the optimal control is refered as ϵ-adaptive dynamic programming method. Since only one performance cost function is used in ϵ-adaptive

dynamic programming, ϵ-adaptive dynamic programming is helpful to overcome the curse of dimensionality.

Before the establish of ϵ-adaptive dynamic progrmming, the behavior of the greatest lower bound of all performance cost of the system has been studied. For a general nonlinear discrete-time system with any positive utility function, it is proved that $\lim_{k\to\infty} J_k^*(x) = J_\infty^*(x)$, where $J_k^*(x)$ is the greatest lower bound of all performance costs of the system starting from x and reaching the target in k steps, and $J_\infty^*(x)$ is greatest lower bound of all performance cost of the system starting from x and reaching the target in (any) finite steps. Similar results have been proved by A. Al-Tamimi and F. Lewis (2007[2]) when the system is affine and the unitlity is quadratic. The result proved in this thesis is for more general system. Forthermore, it was found that $J_\infty^*(x)$ cannot guarantee an admissible controller to drive the system to approach the equilibrium state. This is another reason why an ϵ-optimal performance cost is introduced.

After we established the ϵ-adaptive dynamic programming theory for system without discount factor in the performance cost function, we began to study the more general case. We study the nonlinear discrete-time systems that have a discount factor $0 < \gamma \leq 1$ in their performance index functions. This case seems more complex than the case with out discount factor (i.e., the discount factor is 1). However, it is fortunate that the concepts and results of ϵ-optimal control can be generalize to this more general case. The ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$ is defined which will approach to the least upper bound $J_\infty^*(\cdot)$ of the optimal performance cost functions $J_k^*(\cdot)$, $k = 1, 2, \cdots$ when $\epsilon \to 0$. $V_\epsilon^*(\cdot)$ also satisfies the Bellman equation in this case. Consequenctly, we obtain a generalized ϵ-optimal control and dynamic

programming theory. An admissible controller can be obtained by solving the Bellman equation of $V_\epsilon^*(\cdot)$ so that the system state will attend to the equilibrium state under the control of this controller. Similar with the case without discount factor, we introduce the adaptive dynamic programming method for the case with discount factor. We design a numerical algorithm for the ϵ-dynamic programming. The algorithm is similar to that in case without discount factor. It will provide a sequence of approximations of the ϵ-optimal performance cost function $V_\epsilon^*(\cdot)$. A simulated experiment is provided and it showes the algorithm runs well. A stable controller is obtained which provides an ϵ-optimal performance cost.

1.3 Iterative Algorithm

The previous results on ϵ-optimal dynamic programming provide stable controllers in the sence of ϵ-optimal. By using one performance cost function in ϵ-adaptive dynamic programming, ϵ-adaptive dynamic programming is also helpful to overcome the curse of dimensionality. However, the time expense of the ϵ-optimal dynamic programming algorithm is still big. In the third part of this thesis, certain restriction is set on the system so that some fast algorithm can be found. It is assumed that the utility function of system is a positive definite quadratic function. Under this assumption, the iterative adaptive dynamic programming (ADP) algorithm using globalized dual heuristic programming (GDHP) technique is introduced to obtain the optimal controller with convergence analysis in terms of cost function and control law. This method can be considered as a geralization of the work of A. Al-Tamimi and F. Lewis (2007[2]) on affine systems with positive definite quadratic utility. In order to implement the iterative algorithm, a neural net-

work is constructed first to identify the unknown nonlinear system. Then, based on the learned system model, two other neural networks are used as parametric structures to facilitate the implementation of the iterative algorithm, which aims at approximating at each iteration the cost function and the control law, respectively. A simulation example is provided to verify the effectiveness of the presented optimal control scheme dynamic programming (ADP) algorithm using globalized dual heuristic programming (GDHP) technique is introduced.

1.4 A New Class of Wavelet Neural Networks

The last part of this thesis is about wavelet neural networks. In the numerical simulations in our research on dynamic programming, we use neural network to approximate the functions, such as the performance cost function and the oprimal controller. The neural networks will be trained from sequences of input-output data. So the generalization ability of the neural networks is important for our purpose. Radial basis function neural networks (RBFNNs) and wavelet neural networks (WNNs) are well known neural networks that have good generalization ability. In a RBFNN, a function $f(x)$ is approximated as $\hat{f}(x) = \sum_i w_i \phi\left(\frac{\|x-a_i\|}{b_i}\right)$, where $\phi(r)$ is the basis function. In a WNN, a function $f(x)$ is approximated as $\hat{f}(x) = \sum_i w_i \phi\left(\frac{x-a_i}{b_i}\right)$, where $\phi(x)$ is the basis function coming from wavelet theory ([10, 26]) – the scaling function, the wavelet function, or the basis function of continuous wavelet transform. WNNs can approximate functions more accurately and they have better generalization property than RBFNNs. But all the existing training algorithms of WNNs are not especially for sequential learning and the orthogonal properties of wavelets have not been used in these algorithms.

Hence we designed a wavelet basis function neural networks (WBFNNs) for sequential learning. Both the scaling function ϕ and the wavelet function ψ are used as basis functions in WBFNN. Functions ϕ and ψ are orthogonal to each other. In a wavelet decomposition of a function, they will give approximations in different level of details, i.e., coarse and fine approximations, respectively. A sequential learning algorithm is produced from the orthogonal properties of multiresolution approximation and Mallat's formula of wavelet decomposition ([10, 26]).

1.5 Summary

The Thesis has 6 chapters. In Chapter 2, the results of ϵ-adaptive dynamic programming for discrete-time system when the performance cost has no discount factor will be discussed. In Chapter 3, we will generalize the results in Chapter 2 to descrete-time system when the performance cost has discount factor $0 < \gamma \leq 1$. In Chapter 4, an iterative algorithm is designed for adaptive dynamic programming under the restriction that the utility of the system is a positive definite quadratic form. Chapter 5 is refer to the wavelet neural networks. Finally, Chapter 6 is the conclution.

2 THE ϵ-ADAPTIVE DYNAMIC PROGRAMMING FOR DISCRETE-TIME SYSTEM WITHOUT DISCOUNT FACTOR IN ITS PERFORMANCE COST

In this Chapter, we give a brief introduction of the ϵ-optimal control and ϵ-adaptive dynamic programming. All the results in this Chapter have been published in the paper "Adaptive dynamic programming for finite-horizon optimal control of discrete-time nonlinear systems with ϵ-error bound" [38].

Dynamic programming for complex dynamical systems is difficult due to the "curse of dimensionality": one has to find a series of control actions that must be taken in sequence. This sequence will lead to the optimal performance index, but the total cost of these actions will be unknown until the end of that sequence. In this chapter, we present our work on dynamic programming for nonlinear discrete-time systems using neural networks, which is referred as adaptive dynamic programming with an error bound according to ϵ or ϵ-adaptive dynamic programming. A single controller, called the ϵ-optimal controller $\mu_\epsilon^*(\cdot)$, which is determined by an ϵ-optimal performance index function $V_\epsilon^*(\cdot)$, is utilized to approximate the optimal controller. The ϵ-optimal controller $\mu_\epsilon^*(\cdot)$ can always control the state to approach the target state, while the performance index is close to the greatest lower bound of all performance indices within an error according to ϵ. An algorithm for finding the ϵ-optimal controller is presented and numerical experiments are also given to illustrate the applicability of the algorithm.

2.1 Introduction

In this chapter, we study dynamic programming of deterministic nonlinear, discrete-time, time-invariant systems given by

$$x_{k+1} = F(x_k, u_k), \quad k = 0, 1, 2, \ldots . \tag{2.1.1}$$

There is no special requirement for the system function F except that F is continuous, the origin is an equilibrium point of the system, and the system is controllable near the origin. The control target is the origin and we will try to drive the system to reach the target in finite but unknown time steps, i.e., we will study optimal control problems with finite-horizon and unspecified terminal time.

The optimal controller of a discrete-time system may be different at different control steps. To obtain 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 and each obeys a different control law. In this chapter, after some discussion about the limiting behavior of optimal performance index, we introduce an algorithm based on the idea of ϵ-*optimal control* (see Section 2.4 for details). Only one control law $\mu_\epsilon^*(\cdot)$ will be required to obtain an ϵ-optimal control sequence. The system state will reach the target under this control law.

In Section 2.2, we give some statements of the optimal control problem for discrete-time systems. We recall some details about dynamic programming which will be useful in this chapter. Simple systems are studied as illustrations in remarks and examples of Sections 2.2 and 2.4. In Section 2.3, we prove that the optimal performance indexes J_k^* for different control steps k form a nonincreasing sequence (see Theorem 2.1). The limit of J_k^* as $k \to \infty$ equals the infimum J_∞^* of all performance indexes and satisfies

Bellman's principle of optimality (see (2.3.3) of Theorem 2.2). However, one cannot guarantee that J_∞^* will give an admissible controller. When the optimal cost $J_\infty^*(x)$ is applied, the associated control law $v_\infty^*(x)$ may not be admissible (see Example 2.3). In Section 2.4, we introduce the functions $K_\epsilon(x)$ and V_ϵ^*. V_ϵ^* is an approximation of J_∞^* with error less than ϵ such that $V_\epsilon^* \geq J_\infty^* \geq V_\epsilon^* - \epsilon$. The value of $K_\epsilon(x)$ gives the length of optimal control sequence starting from x with cost V_ϵ^*. We prove that V_ϵ^*, associated with K_ϵ, satisfies some relationships similar to Bellman's principle (see Corollary 2.1 and formulas (2.4.11)–(2.4.16)). The associated controller $\mu_\epsilon^*(x)$ is admissible and will be used as the only controller in all steps of an ϵ-optimal control sequence. The difference between the performance index under μ_ϵ^* and the infimum cost J_∞^* is given in Theorem 2.6. According to the results in Section 2.4, we establish our algorithm of adaptive dynamic programming with an error bound according to ϵ, i.e., the *ϵ-adaptive dynamic programming* for discrete-time systems using neural networks (ADPDN(ϵ)) in Section 2.5. In the algorithm ADPDN(ϵ), three neural networks $\hat{J}$, $\hat{\mu}$ and $\hat{K}$ are trained to approximate V_ϵ^*, μ_ϵ^* and K_ϵ, respectively. Numerical experiments are given in Section 2.6 to evaluate the performance of the algorithm ADPDN(ϵ). We apply our algorithm ADPDN(ϵ) to a nonlinear unstable system $x_{k+1} = x_k + \sin(x_k + u_k)$ with utility function $U(x, u) = |x| + u^2$. We also apply our algorithm ADPDN(ϵ) to the ball and beam experiment. Our algorithm performs well in the experiments. We conclude the present chapter in Section 2.7.

2.2 Problem Statement

In this chapter, we will study deterministic discrete-time systems

$$x_{k+1} = F(x_k, u_k), \ k = 0, 1, 2, \ldots, \tag{2.2.1}$$

where $x_k \in \mathbb{R}^n$ is the state and $u_k \in \mathbb{R}^m$ is the control. The system function $F(x, u)$ is continuous and $F(0, 0) = 0$. Hence $x = 0$ is an equilibrium state of the system (2.2.1) under control $u = 0$. The control target is the origin $x = 0$. We will study optimal control problems for (2.2.1) with finite-horizon and unspecified terminal time.

Suppose that system (2.2.1) has a utility function $U(x, u)$, $U(x, u) \geq 0$ for any (x, u) and $U(0, 0) = 0$. Then the performance index for a state x under the control sequence $\underline{u} = (u_0, u_1, \ldots, u_{N-1})$ is defined as

$$J(x, \underline{u}) = \sum_{k=0}^{N-1} U(x_k, u_k), \tag{2.2.2}$$

where $x_0 = x$, and $x_k = F(x_{k-1}, u_{k-1})$ for $k = 1, 2, \ldots, N$.

Let x be an initial state and $\underline{u} = (u_0, u_1, \ldots, u_{N-1})$ be a finite sequence of controls. Then system (2.2.1) gives a trajectory starting from x: $x_0 = x$, $x_1 = F(x_0, u_0)$, $x_2 = F(x_1, u_1)$, $\ldots$, $x_N = F(x_{N-1}, u_{N-1})$. We call the number of elements in the control sequence $\underline{u}$ the length of $\underline{u}$ and denote it as $|\underline{u}|$. Then $|\underline{u}| = N$. For convenience, we say that the length of the associated trajectory $x_0, x_1, \ldots, x_N$ is also N. We denote the final state of the trajectory as $x_N(x, \underline{u})$, i.e., $x_N(x, \underline{u}) = x_N$.

An initial state $x_0 = x$ is called controllable if there exists a control sequence $\underline{u}$ such that $x_N(x, \underline{u}) = 0$, while the control sequence $\underline{u}$ is called an

admissible control sequence of x. Let

$$\mathfrak{A}_x = \{\underline{u}\colon x_N(x, \underline{u}) = 0\} \tag{2.2.3}$$

be the set of all admissible control sequences of x. Let

$$\mathfrak{A}_x^{(k)} = \{\underline{u}\colon x_N(x, \underline{u}) = 0, |\underline{u}| = k\} \tag{2.2.4}$$

be the set of all admissible control sequences of x with length k. Then $\mathfrak{A}_x = \cup_{1 \le k < \infty} \mathfrak{A}_x^{(k)}$. By this notation, a state x is controllable if and only if $\mathfrak{A}_x \neq \emptyset$.

We assume that $F(x, u)$ satisfies the following condition.

(C2.1) The system (2.2.1) is controllable in an open neighborhood of $x = 0$.

According to condition (C2.1), there exists a positive number $M > 0$, whenever $\|x\| \le M$, one can always find a control sequence $\underline{u}$ such that $x_N(x, \underline{u}) = 0$, where $\|\cdot\|$ is a vector norm. Hence, if $\|x\| \le M$, then $\mathfrak{A}_x \neq \emptyset$.

For any given initial state x, the objective of optimal control is to find an admissible control sequence $\underline{u} \in \mathfrak{A}_x$ to minimize the performance index $J(x, \underline{u})$. The optimal control sequence $\underline{u} \in \mathfrak{A}_x$ has finite length. However, before it is determined, we do not know its length. This kind of optimal problems is called finite-horizon problems with unspecified terminal time [8]. Before studying this kind of problems, we start by considering the finite-horizon problem with fixed terminal time k, for every $k = 1, 2, \ldots$. Let

$$J_k^*(x) = \min\left\{J(x, \underline{u})\colon \underline{u} \in \mathfrak{A}_x^{(k)}\right\}, \quad k = 1, 2, \ldots. \tag{2.2.5}$$

Then $J_k^*(x)$ is the optimal cost among all trajectories starting from x with length k and ending at the origin. Bellman's principle of optimality can be

described by,

$$J_k^*(x) = \min_u \left\{ U(x,u) + J_{k-1}^*(F(x,u)) \right\}. \tag{2.2.6}$$

Now, define the law of control sequence of length k by

$$\underline{v}_k^*(x) = \arg\min \left\{ J(x,\underline{u}) \colon \underline{u} \in \mathfrak{A}_x^{(k)} \right\}, \tag{2.2.7}$$

and define the law of single control vector by

$$v_k^*(x) = \arg\min_u \left\{ U(x,u) + J_{k-1}^*(F(x,u)) \right\}. \tag{2.2.8}$$

Then we have

$$\begin{cases} J_k^*(x) = J(x, \underline{v}_k^*(x)), \\ J_k^*(x) = U(x, v_k^*(x)) + J_{k-1}^*(F(x, v_k^*(x))). \end{cases} \tag{2.2.9}$$

It is easy to see that Bellman's principle of optimality (2.2.6) is equivalent to

$$\underline{v}_k^*(x) = (v_k^*(x), \underline{v}_{k-1}^*(F(x, v_k^*(x))). \tag{2.2.10}$$

Consequently, we have

$$\underline{v}_k^*(x) = (v_k^*(x_0), v_{k-1}^*(x_1), \ldots, v_1^*(x_{k-1})), \tag{2.2.11}$$

where $x_0 = x$ and $x_{j+1} = F(x_j, v_{k-j}^*(x_j))$ for $j = 0, 1, \ldots, k-1$. In particular, $v_k^*(x)$ is the first component of the control sequence $\underline{v}_k^*(x)$.

The first step of dynamic programming is to determine the function $v_1^*(\cdot)$. For any given state vector x, the control vector $v_1^*(x)$ is the solution of the following minimization problem

$$\min_u U(x,u) \ \text{ subject to } F(x,u) = 0. \tag{2.2.12}$$

The associated performance index is

$$J_1^*(x) = U(x, v_1^*(x)).$$

After the functions $v_1^*(\cdot)$ and $J_1^*(\cdot)$ have been determined, we can determine $v_j^*(\cdot)$ and $J_j^*(\cdot)$, for $j = 2, 3, \ldots, k$, by applying formulas (2.2.6) and (2.2.8). Then we can determine $\underline{v}_k^*(x_0) = (v_k^*(x_0), v_{k-1}^*(x_1), \ldots, v_1^*(x_{k-1}))$ as follows. Applying $v_k^*(\cdot)$ to x_0 gets $u_0^* = v_k^*(x_0)$. Applying u_0^* to the system (2.2.1) gets $x_1 = F(x_0, u_0^*)$. Then applying $v_{k-1}^*(\cdot)$ to x_1 gets $u_1^* = v_{k-1}^*(x_1)$, and applying u_1^* to the system gets $x_2 = F(x_1, u_1^*)$. Repeat this process, we get the optimal control sequence $\underline{v}_k^*(x_0) = \{u_0^*, u_1^*, \ldots, u_{k-1}^*\}$.

But then we have to face the problem of "curse of dimensionality". We have to calculate and record all the functions $J_j^*(\cdot)$ and $v_j^*(\cdot)$, $j = 1, 2, \ldots, k$, even though only one control trajectory is desired. In general, $J_j^*(\cdot)$ and $J_l^*(\cdot)$ are different functions when $j \neq l$. Similarly, $v_j^*(\cdot)$ and $v_l^*(\cdot)$ are different functions when $j \neq l$. In most real world applications, large k is necessary. But for large k, the computation and storage requirements are huge, especially when the dimension n of the problem is large.

Example 2.1

Consider the linear system described by

$$x_{k+1} = x_k + u_k,$$

where $x_k \in \mathbb{R}$ and $u_k \in \mathbb{R}$, with quadratic utility function $U(x, u) = u^2$. To find $v_1^*(x)$, look at (2.2.12). Now it becomes

$$\min_u u^2 \quad \text{subject to } x + u = 0.$$

So $v_1^*(x) = -x$ and $J_1^*(x) = x^2$. Assume that we have already obtained $J_{k-1}^*(x) = Q_{k-1}x^2$. Then by (2.2.8),

$$v_k^*(x) = \arg\min_u \{u^2 + Q_{k-1}(x+u)^2\}. \tag{2.2.13}$$

To find the minimum, take derivative of the function on the right-hand side of (2.2.13). Then, we obtain $2u + 2Q_{k-1}(x+u) = 0$, which leads to

$$v_k^*(x) = -\frac{Q_{k-1}}{1+Q_{k-1}}x.$$

Consequently, from (2.2.9)

$$\begin{aligned}
J_k^*(x) &= \left(-\frac{Q_{k-1}x}{1+Q_{k-1}}\right)^2 + Q_{k-1}\left(x - \frac{Q_{k-1}x}{1+Q_{k-1}}\right)^2 \\
&= \left[\left(-\frac{Q_{k-1}}{1+Q_{k-1}}\right)^2 + Q_{k-1}\left(1 - \frac{Q_{k-1}}{1+Q_{k-1}}\right)^2\right]x^2 \\
&= \left[\frac{Q_{k-1}^2}{(1+Q_{k-1})^2} + \frac{Q_{k-1}}{(1+Q_{k-1})^2}\right]x^2 \\
&= \frac{Q_{k-1}}{1+Q_{k-1}}x^2.
\end{aligned}$$

Thus, we know that the optimal control and optimal performance index have the form $v_k^*(x) = -\alpha_k x$ and $J_k^*(x) = Q_k x^2$. The coefficients α_k and Q_k satisfy

$$\begin{cases} Q_1 = \alpha_1 = 1, \\ Q_k = \alpha_k = \dfrac{Q_{k-1}}{1+Q_{k-1}}, \quad \text{for } k = 2, 3, 4, \ldots . \end{cases}$$

It is easy to verify that for $k = 1, 2, 3, \ldots$, $Q_k = \alpha_k = 1/k$, and we have

$$v_k^*(x) = -\frac{1}{k}x \quad \text{and} \quad J_k^*(x) = \frac{1}{k}x^2. \tag{2.2.14}$$

To determine the optimal control sequence for a given initial state x_0, with the assumption that the length of control sequence is K, one needs to calculate

the value of control signals by formula (2.2.14). Applying $u_0^* = v_K^*(x_0) = -x_0/K$ in the first step of control, we get $x_1 = x_0 + u_0^* = (K-1)x_0/K$. Subsequently, we get $x_{k+1} = x_k + u_k^* = (K-k-1)x_0/K$, where $u_k^* = v_{K-k}^*(x_k) = -x_k/(K-k) = -x_0/K$ for $k = 1, 2, \ldots, K-1$. To store the controller, we can either save all the coefficients Q_k and α_k, $k = 1, 2, \ldots, K$, in storage, or calculate them via formulas $Q_k = \alpha_k = 1/k$ every time when they are needed. ■

In Example 2.1, the system is linear, the utility is quadratic, and the functions in (2.2.14) are simple. But nonetheless, (2.2.14) indicates that formulas for functions $J_k^*(\cdot)$ and $v_k^*(\cdot)$ will change when k changes its value. In general, the calculation and the form of the functions $J_k^*(\cdot)$ and $v_k^*(\cdot)$ are much more complex. It will be very difficult to find similar formulas like (2.2.14). To avoid calculating and recording the sequences of functions $v_k^*(\cdot)$ and $J_k^*(\cdot)$, one can use the method for *infinite-horizon* problems (cf. [19, 21, 22, 45] and their references). In infinite-horizon problems, instead of control sequences with finite length, control sequences with infinite length are studied. The definition of admissible control sequence will be changed so that it will drive the state asymptotically to the target with finite performance index (cf. [1, 4]). One hopes that the state will approach the target in some "infinite future". For an infinite-horizon problem, there is only one optimal performance index. It does not depend on the length of the control sequence, since all control sequences have the same length: ∞. But the performance index will be a summation of infinite number of terms. It is the limit of an infinite series. As for limits of infinite series, we have to consider whether the series $J(x, \underline{u}) = \sum_{k=0}^{\infty} U(x_i, u_i)$ converges. If the limit exists, whether the greatest lower bound $\inf J(x, \underline{u})$ is reachable, and whether $\inf J(x, \underline{u})$ satisfies the HJB equation.

In [2], for system (1.1.9) with infinite-horizon performance index (1.1.10), a nondecreasing sequence of performance index $\{V_i\}$ is introduced to approximate a function V^*, which satisfies the HJB equation (1.1.12). A controller u^* can be obtained from V^* by solving the HJB equation (see (1.1.12)). u^* can be considered as the optimal controller of the system, it will drive the state to the target when time $k \to \infty$. However, if one wants to control the state to reach the target in some finite time, the controller u^* may not be enough. In many cases, u^* can only make the state approach the target asymptotically. The state may never reach the target in any finite time. Moreover, when the positive-definite condition on the matrices Q and R of the performance index (1.1.10) does not hold, u^* may even be a non-admissible controller. The following example is an illustration.

Example 2.2

Consider Example 2.1 again. But this time, we consider the infinite-horizon problem.

Let x_0 be any non-zero initial state and $\underline{u} = (u_0, u_1, \dots)$ be any sequence of controls with infinite length. The trajectory starting from x_0 under the control of $\underline{u}$ is: x_0, $x_1 = x_0 + u_0$, $\dots$, $x_k = x_0 + \sum_{i=0}^{k-1} u_i$, $\dots$. The associated performance index is $J(x_0, \underline{u}) = \sum_{i=0}^{\infty} u_i^2$. In this case, $\underline{u}$ is admissible if and only if $\sum_{i=0}^{\infty} u_i = -x_0$ and $\sum_{i=0}^{\infty} u_i^2 < \infty$.

For any positive integer $k = 1, 2, \dots$, let $\hat{\underline{u}}^{(k)}$ be the control sequence such that the first k-entries are all $-x_0/k$ and the others are all zero, i.e.,

$$\hat{\underline{u}}^{(k)} = \Big(\underbrace{-\frac{x_0}{k}, \dots, -\frac{x_0}{k}}_{k}, 0, 0, \dots \Big).$$

Then $\hat{\underline{u}}^{(k)}$ is admissible and

$$J(x_0, \hat{\underline{u}}^{(k)}) = \frac{1}{k} x_0^2.$$

Thus, for $k = 1, 2, 3, \ldots,$

$$\inf_{\underline{u}} J(x_0, \underline{u}) \leq J(x_0, \hat{\underline{u}}^{(k)}) = \frac{1}{k} x_0^2.$$

Let $k \to \infty$, we have

$$\inf_{\underline{u}} J(x_0, \underline{u}) = 0. \tag{2.2.15}$$

Now we know that the greatest lower bound of performance indexes is $\inf_{\underline{u}} J(x_0, \underline{u}) = 0$. But for any admissible control sequence $\underline{u}$, $J(x_0, \underline{u}) > 0$, i.e., the greatest lower bound $\inf J(x, \underline{u})$ cannot be reached by any admissible control sequence. On the other hand, the control sequence associated with $\inf_{\underline{u}} J(x_0, \underline{u}) = 0$ is just $\tilde{\underline{u}} = (0, 0, \ldots)$ which cannot make x_0 move to the target 0 at all. ■

The system in Example 2.2 is a very simple one. For many systems, especially for nonlinear systems, it may not always be true that there exists an admissible control sequence $\hat{\underline{u}}$ such that the performance index $J(x_0, \hat{\underline{u}})$ equals the greatest lower bound $\inf_{\underline{u}} J(x_0, \underline{u})$. On the other hand, if $\underline{u}^*$ is the control sequence determined using $\inf_{\underline{u}} J(x_0, \underline{u})$ as the Lyapunov function, it may not always be true that $\underline{u}^*$ is an admissible control sequence. Therefore, when we try to find a single optimal cost and a single optimal controller for infinite horizon problems, we have to face the problem whether this single optimal cost can be reached by an admissible control sequence and whether this single optimal controller is an admissible controller. Unfortunately, the infimum $\inf_{\underline{u}} J(x_0, \underline{u})$ is not a feasible one, although it is the greatest lower bound of all

performance indexes. When one tries to control a system by approximating $\inf_{\underline{u}} J(x_0, \underline{u})$, a non-admissible controller may be obtained sometimes. In the rest of this chapter, we will introduce an "ϵ-optimal cost function" $V_\epsilon^*(\cdot)$ associated with an "ϵ-optimal controller" $\mu_\epsilon^*(\cdot)$ which will always give an admissible control sequence $\underline{\mu}_\epsilon^*(x_0)$ for any controllable initial state x_0, with a performance index within an error bound according to ϵ from the infimum $\inf_{\underline{u}} J(x_0, \underline{u})$.

2.3 The Limit of J_k^*

In this section, we will consider finite-horizon problems with unspecified terminal time. We go back to the system (2.2.1) with performance index (2.2.2). Recall that the law of optimal control sequence $\underline{v}_k^*(\cdot)$ of length k, the law of optimal control $v_k^*(\cdot)$, and the associated optimal performance index J_k^* satisfy (2.2.6)–(2.2.11).

For any state vector x, define

$$J_\infty^*(x) = \inf\{J(x, \underline{u}) \colon \underline{u} \in \mathfrak{A}_x\}. \tag{2.3.1}$$

Then we can consider $J_\infty^*(x)$ to be the "optimal" performance index starting from x under all admissible control sequences of any length, i.e., $J_\infty^*(x)$ will be the "optimal" performance index starting from x for the finite-horizon problem with unspecified terminal time. It is possible that there is no admissible control sequence $\underline{u} \in \mathfrak{A}_x$ such that the equality $J_\infty^*(x) = J(x, \underline{u})$ holds. However, we can prove that $J_\infty^*(x)$ is the limit of $J_k^*(x)$.

Theorem 2.1 *Let x be an arbitrary state vector. Suppose that there is a positive integer p such that $\mathfrak{A}_x^{(p)} \neq \emptyset$. Then for any $k > p$, $\mathfrak{A}_x^{(k)} \neq \emptyset$, $J_k^*(x) \leq$*

$J_p^*(x)$, *and*

$$\lim_{k\to\infty} J_k^*(x) = J_\infty^*(x). \tag{2.3.2}$$

Proof. First, let us show that for any admissible control sequence $\underline{u}$ with length p, we can assign an admissible control sequence $\hat{\underline{u}}$ with length k such that $J(x,\underline{u}) = J(x,\hat{\underline{u}})$, if $k > p$. Suppose that $\underline{u} = (u_0, u_1, ..., u_{p-1}) \in \mathfrak{A}_x^{(p)}$, $|\underline{u}| = p$ and $x_N(x,\underline{u}) = 0$. The trajectory starting from x under the control of $\underline{u}$ is $x_0 = x$, $x_1 = F(x_0, u_0)$, ..., $x_p = F(x_{p-1}, u_{p-1}) = 0$. By adding $k-p$ 0's to the end of sequence $\underline{u}$, we define a control sequence $\hat{\underline{u}} = (\underline{u}, 0, 0, ..., 0)$. Obviously $|\hat{\underline{u}}| = k$. The trajectory starting from x under the control of $\hat{\underline{u}}$ is $x_0 = x$, $x_1 = F(x_0, u_0)$, ..., $x_p = F(x_{p-1}, u_{p-1}) = 0$, $x_{p+1} = ... = x_k = F(0,0) = 0$. So $\hat{\underline{u}}$ is admissible. Furthermore, $J(x,\hat{\underline{u}}) = U(x_0,u_0)+\cdots+U(x_{p-1},u_{p-1})+U(x_p,u_p)+\cdots+U(x_{k-1},u_{k-1}) = U(x_0,u_0)+\cdots+U(x_{p-1},u_{p-1})+U(0,0)+\cdots+U(0,0) = U(x_0,u_0)+\cdots+U(x_{p-1},u_{p-1}) = J(x,\underline{u})$.

Now we can prove that $\mathfrak{A}_x^{(k)} \neq \emptyset$ if $k > p$. By the assumption of this theorem, $\mathfrak{A}_x^{(p)} \neq \emptyset$. So there exists at least one $\underline{u} \in \mathfrak{A}_x^{(p)}$. Then we can have $\hat{u}$ as constructed above such that $\hat{\underline{u}} \in \mathfrak{A}_x^{(k)}$. Thus $\mathfrak{A}_x^{(k)} \neq \emptyset$ for $k > p$.

Fix k to be an arbitrary integer larger than p. We will prove that $J_k^*(x) \leq J_p^*(x)$. As shown above, for every control sequence $\underline{u} \in \mathfrak{A}_x^{(p)}$, $\hat{u}$ is a control sequence in $\mathfrak{A}_x^{(k)}$. Hence $\{\hat{\underline{u}} \colon \underline{u} \in \mathfrak{A}_x^{(p)}\} \subseteq \mathfrak{A}_x^{(k)}$. Therefore, we have

$$\begin{aligned} J_k^*(x) &= \min\{J(x,\underline{u}) \colon \underline{u} \in \mathfrak{A}_x^{(k)}\} \\ &\leq \min\{J(x,\hat{\underline{u}}) \colon \underline{u} \in \mathfrak{A}_x^{(p)}\} \\ &= \min\{J(x,\underline{u}) \colon \underline{u} \in \mathfrak{A}_x^{(p)}\} = J_p^*(x). \end{aligned}$$

We have shown that the sequence $\{J_k^*(x)\colon k \geq p\}$ is nonincreasing. Besides, $J_k^*(x) \geq 0$ for each k. So $\lim_{k\to\infty} J_k^*(x)$ exists and it is a finite nonnegative number.

Finally, we will prove that $\lim_{k\to\infty} J_k^*(x)$ equals $J_\infty^*(x)$. Since $J_\infty^*(x) = \inf_{\underline{u}}\{J(x,\underline{u})\} \leq J_k^*(x)$ for any k, we have $J_\infty^*(x) \leq \lim_{k\to\infty} J_k^*(x)$. For any $\epsilon > 0$, there exists $\underline{\nu}$ such that $J(x,\underline{\nu}) \leq J_\infty^*(x) + \epsilon$. Suppose that $|\underline{\nu}| = q$. Then $\lim_{k\to\infty} J_k^*(x) \leq J_q^*(x) \leq J(x,\underline{\nu}) \leq J_\infty^*(x) + \epsilon$. Since ϵ is chosen arbitrarily, we have $J_\infty^*(x) = \lim_{k\to\infty} J_k^*(x)$. ∎

Now let us consider what will happen when we make $k \to \infty$ in (2.2.6). The left-hand side is simply $J_\infty^*(x)$. But for the right-hand side, it is not obvious to see since the minimum will be reached at different u for different k. However, the following result can be proved.

Theorem 2.2 *Let x be an arbitrary state vector. Suppose that there is a positive integer p such that $\mathfrak{A}_x^{(p)} \neq \emptyset$. Then*

$$J_\infty^*(x) = \inf_u\{U(x,u) + J_\infty^*(F(x,u))\}. \tag{2.3.3}$$

Proof. For any u and $k > p$, by Bellman's principle of optimality (2.2.6),

$$J_k^*(x) \leq U(x,u) + J_{k-1}^*(F(x,u)).$$

By the definition of $J_\infty^*(x)$, $J_\infty^*(x) \leq J_k^*(x)$. So for any $k > p$,

$$J_\infty^*(x) \leq U(x,u) + J_{k-1}^*(F(x,u)). \tag{2.3.4}$$

Let $k \to \infty$ in (2.3.4), then

$$J_\infty^*(x) \leq U(x,u) + J_\infty^*(F(x,u)).$$

So

$$J_\infty^*(x) \le \inf_u \{U(x,u) + J_\infty^*(F(x,u))\}. \tag{2.3.5}$$

Let $\epsilon > 0$ be an arbitrary positive number. Since $\{J_k^*\}$ is nonincreasing and $\lim_{k\to\infty} J_k^*(x) = J_\infty^*(x)$, there exists a positive integer K such that

$$J_K^*(x) - \epsilon \le J_\infty^*(x) \le J_K^*(x).$$

Let $u_0^* = v_K^*(x) = \arg\min_u \{U(x,u) + J_{K-1}^*(F(x,u))\}$. Then

$$J_K^*(x) = U(x,u_0^*) + J_{K-1}^*(F(x,u_0^*)).$$

Hence

$$\begin{aligned} J_\infty^*(x) &\ge U(x,u_0^*) + J_{K-1}^*(F(x,u_0^*)) - \epsilon \\ &\ge U(x,u_0^*) + J_\infty^*(F(x,u_0^*)) - \epsilon. \\ &\ge \inf_u \{U(x,u) + J_\infty^*(F(x,u))\} - \epsilon. \end{aligned}$$

Since ϵ is arbitrary, we have

$$J_\infty^*(x) \ge \inf_u \{U(x,u) + J_\infty^*(F(x,u))\}. \tag{2.3.6}$$

Combining (2.3.5) and (2.3.6) we have

$$J_\infty^*(x) = \inf_u \{U(x,u) + J_\infty^*(F(x,u))\},$$

which proves the theorem. ■

Equation (2.3.3) is similar to (2.2.6). But in (2.3.3), there is only one performance index function $J_\infty^*(x)$, while (2.2.6) holds for a sequence of functions $J_k^*(x)$, $k = 1, 2, \ldots$.

In Theorems 2.1 and 2.2, we have the assumption that $\mathfrak{A}_x^{(p)} \ne \emptyset$. If $\mathfrak{A}_x^{(p)} \ne \emptyset$ for some p is not satisfied for any x, these two theorems will be

meaningless. Now let us look at this condition. We will study the case when $\mathfrak{A}_x^{(p)} \neq \emptyset$ for some p and the function $J_\infty^*(\cdot)$ can be defined at x.

By Section II, $\mathfrak{A}_x = \cup_{1\leq k<\infty}\mathfrak{A}_x^{(k)}$. So $\mathfrak{A}_x^{(p)} \neq \emptyset$ for some p if and only if $\mathfrak{A}_x \neq \emptyset$. If $\mathfrak{A}_x \neq \emptyset$, then there exists at least one control sequence $\underline{u}$ with finite length such that the trajectory starting from x under the control of $\underline{u}$ will reach the target in final step, i.e., $x_N(x, \underline{u}) = 0$. On the other hand, if $\mathfrak{A}_x = \emptyset$, then the trajectory starting from x under any arbitrary control sequence of finite length will never reach the target. So $\{x\colon \mathfrak{A}_x \neq \emptyset\}$ is the set of all controllable states and $\{x\colon \mathfrak{A}_x = \emptyset\}$ is the set of all uncontrollable states. Moreover, $J_\infty^*(x)$ is well defined if and only if x is controllable.

Let $\mathcal{T}_0 = \{0\}$. For $k = 1, 2, \ldots,$ define

$$\mathcal{T}_k = \{x \in \mathbb{R}^n |\ \exists u \in \mathbb{R}^m \ s.t.\ F(x, u) \in \mathcal{T}_{k-1}\}. \tag{2.3.7}$$

Since $\mathcal{T}_0 = \{0\}$ and $F(0, 0) = 0$, we know that $0 \in \mathcal{T}_1$. Hence $\mathcal{T}_0 \subseteq \mathcal{T}_1$. Assume that $\mathcal{T}_{k-1} \subseteq \mathcal{T}_k$. If $x \in \mathcal{T}_k$, then $F(x, u) \in \mathcal{T}_{k-1}$ for some u. Hence $F(x, u) \in \mathcal{T}_k$. So $x \in \mathcal{T}_{k+1}$ by (2.3.7). Thus $\mathcal{T}_k \subseteq \mathcal{T}_{k+1}$. By the principle of induction, we have

$$\{0\} = \mathcal{T}_0 \subseteq \mathcal{T}_1 \subseteq \cdots \subseteq \mathcal{T}_{k-1} \subseteq \mathcal{T}_k \subseteq \cdots .$$

By condition (C2.1), the system is controllable in a neighborhood of the origin. So there exists a non-zero controllable state $x_0 \neq 0$. There always exists a control sequence $\underline{u} = (u_0, u_1, \ldots, u_{N-1})$ such that $x_{N-1} \neq 0$ and $x_N = 0$, where $x_{i+1} = F(x_i, u_i)$ for $i = 0, \ldots, N-1$. Obviously, $x_{N-1} \in \mathcal{T}_1$. Thus, $\mathcal{T}_1 \supseteq \{0\}$ and $\mathcal{T}_1 \neq \{0\}$, i.e., $\mathcal{T}_1 \supset \{0\}$. So $\mathcal{T}_k \supseteq \mathcal{T}_1 \supset \{0\}$ when $k > 1$.

Now, $\mathcal{T}_0 = \{0\}$ is the target. Let $x \in \mathcal{T}_1$. Then there exists a control

vector u such that $F(x,u)=0$, which implies that $u \in \mathfrak{A}_x^{(1)}$. Thus, $\mathfrak{A}_x^{(1)} \neq \emptyset$ for any $x \in \mathcal{T}_1$. Assume that $\mathfrak{A}_x^{(k-1)} \neq \emptyset$ for any $x \in \mathcal{T}_{k-1}$. Let $x \in \mathcal{T}_k$. Then there exists a control vector u such that $F(x,u) \in \mathcal{T}_{k-1}$. Hence $\mathfrak{A}_{F(x,u)}^{(k-1)} \neq \emptyset$ by assumption. Let $\underline{u}_1 \in \mathfrak{A}_{F(x,u)}^{(k-1)}$. Then $(u, \underline{u}_1)$ will control the state x to 0 and has length k, i.e., $(u, \underline{u}_1) \in \mathfrak{A}_x^{(k)}$. Therefore, for any k, $\mathfrak{A}_x^{(k)} \neq \emptyset$ if $x \in \mathcal{T}_k$.

On the other hand, suppose that x is a state such that $\mathfrak{A}_x^{(k)} \neq \emptyset$. Let $\underline{u} = (u_0, \ldots, u_{k-1}) \in \mathfrak{A}_x^{(k)}$. Then we have the trajectory starting from x under the control of $\underline{u}$ given by $x_0 = x$, $x_1 = F(x_0, u_0), \ldots, x_k = F(x_{k-1}, u_{k-1})$. Since $\underline{u}$ is admissible, $x_k = 0$. Then it is easy to see that $x_k \in \mathcal{T}_0$, $x_{k-1} \in \mathcal{T}_1$, $\ldots$, $x = x_0 \in \mathcal{T}_k$.

Therefore, we have proved the following results.

Theorem 2.3

(i) $\{0\} = \mathcal{T}_0 \subset \mathcal{T}_1 \subseteq \cdots \subseteq \mathcal{T}_{k-1} \subseteq \mathcal{T}_k \subseteq \cdots$, *i.e., if* $k < k'$ *then* $\mathcal{T}_k$ *is a subset of* $\mathcal{T}_{k'}$, *and* $\mathcal{T}_k$ *contains at least one non-trivial state vector when* $k > 0$.

(ii) *For any* k, $x \in \mathcal{T}_k \Leftrightarrow \mathfrak{A}_x^{(k)} \neq \emptyset \Leftrightarrow J_k^*(\cdot)$ *can be defined at* x.

(iii) *Let* $\mathcal{T}_\infty = \cup_{k=1}^\infty \mathcal{T}_k$. *Then* $x \in \mathcal{T}_\infty \Leftrightarrow \mathfrak{A}_x \neq \emptyset \Leftrightarrow J_\infty^*(\cdot)$ *can be defined at* $x \Leftrightarrow x$ *is controllable.*

(iv) *If* $J_k^*(\cdot)$ *can be defined at* x, *then* $J_p^*(\cdot)$ *can be defined at* x *for every* $p > k$.

(v) $J_\infty^*(\cdot)$ *can be defined at* x *if and only there exists* k *such that* $J_k^*(\cdot)$ *can be defined at* x. ■

2.4 The ϵ-optimal Cost V_ϵ^* and the Function $K_\epsilon(x)$

In the previous section, we proved that the infimum

$$J_\infty^*(x) = \inf_{\underline{u}}\{J(x, \underline{u}),\ \underline{u} \in \mathfrak{A}_x\}$$

is the limit of $J_k^*(x)$ when $k \to \infty$. We proved that $J_\infty^*(x)$ satisfies Bellman's equation (2.3.3) and $J_\infty^*(x)$ has definition for every controllable state $x \in \mathcal{T}_\infty$. One can consider $J_\infty^*(x)$ as the optimal performance index because it is really the infimum of all possible performance indexes under admissible control sequences.

One possible natural strategy for optimal control is to find a control sequence such that the associated performance index is just $J_\infty^*(x)$. But unfortunately, $J_\infty^*(x)$ may not be achievable. In many cases, one cannot find the equality $J_\infty^*(x) = J_k^*(x)$ for any k. That is, for any control sequence $\underline{u}$ with finite length, the performance index starting from x under the control of $\underline{u}$ is larger than, not equal to, $J_\infty^*(x)$. On the other hand, by solving (2.3.3), one can assign a control vector $v_\infty^*(x)$ to x and then give a control sequence $\underline{v}_\infty^*(x) = (v_\infty^*(x_0), v_\infty^*(x_1), \ldots, v_\infty^*(x_k), \ldots)$, where $x_0 = x$, $x_1 = F(x_0, v_\infty^*(x_0))$, $\ldots$, $x_k = F(x_{k-1}, v_\infty^*(x_{k-1}))$, $\ldots$. In general, $\underline{v}_\infty^*(x)$ has infinite length. That is, the controller by solving (2.3.3) cannot control the state to reach the target in finite steps.

Furthermore, $\underline{v}_\infty^*(x)$ may not even be admissible in some cases (see Example 2.3 for an illustration of this situation). Even if $\underline{v}_\infty^*(x)$ is admissible, it may also be abused in real world applications since one often has to obtain $\underline{v}_\infty^*(x)$ from an approximation $\hat{J}$ of J_∞^*, not from J_∞^* itself. If it happens to have the inequality $\hat{J}(x) < J_\infty^*(x)$ which is caused by computational errors, the control obtained from $\hat{J}$ will not be admissible, since J_∞^* is the greatest

lower bound of all performance indexes with respect to all admissible control sequences and thus $\hat{J}$ is not a performance index function any more.

Example 2.3

Look at Examples 2.1 and 2.2 again. We will examine whether the control sequence $\underline{v}^*_\infty(\cdot)$ is admissible. The system is $x_{k+1} = x_k + u_k$ and the utility is $U(x, u) = u^2$. We have obtained that $J^*_k(x) = x^2/k$ and $v^*_k(x) = -x/k$ (see (2.2.14)). In this simple case, we have $\mathcal{T}_1 = \mathcal{T}_2 = \ldots = \mathcal{T}_\infty = \mathbb{R}$. We have $\inf_{\underline{u}} J(x, \underline{u}) = 0$ in (2.2.15). So we obtain

$$J^*_\infty(x) = \inf_{\underline{u}} J(x, \underline{u}) = 0 \text{ for all } x \in \mathbb{R}.$$

We can also get $J^*_\infty(x)$ according to Theorem 2.1 by

$$J^*_\infty(x) = \lim_{k\to\infty} J^*_k(x) = 0.$$

Now, (2.3.3) becomes

$$0 = \min_u \{u^2 + 0\}.$$

This gives $v^*_\infty(x) = 0$ and

$$\underline{v}^*_\infty(x) = (0, 0, \ldots). \tag{2.4.1}$$

This is just $\underline{\tilde{u}}$ in Example 2.2. Starting from an initial state x, the trajectory under the control of (2.4.1) is

$$x_0 = x,\ x_1 = x,\ \ldots,\ x_k = x,\ \ldots. \tag{2.4.2}$$

In sequence (2.4.2), $x_k = x$ for every k. So $\lim_{k\to\infty} x_k = x \neq 0$ if $x \neq 0$. The trajectory does not converge to the target. Thus we can claim that in this example, the control sequence obtained by solving Bellman's equation (2.3.3)

according to $J^*_\infty(x)$ is not an admissible control sequence. In this example, the greatest lower bound of all performance indexes is 0. Therefore, in this case, the optimal control sequence is not admissible according to the definition in (2.2.3) and (2.2.4) since no control sequence can drive the system state to the origin with zero cost. Thus, the optimal performance index $J^*_\infty(x)$ in this example cannot be realized by any admissible control sequence. ■

We will consider an alternative for the optimal control of discrete-time systems. The simplest way is to fix the length of control sequence. Preset a positive integer K and then use $J^*_K(x)$ as an approximation of $J^*_\infty(x)$. When K is large enough, $J^*_K(x)$ will be a good approximation of $J^*_\infty(x)$. But the convergence of $\lim_{k\to\infty} J^*_k(x) = J^*_\infty(x)$ may not be uniformly. For a fixed K, the error of the approximation $J^*_K(x) - J^*_\infty(x)$ maybe very large when x is large. For example, look at Examples 2.1 and 2.2. We have that $J^*_K(x) = x^2/K$, $J^*_\infty(x) = 0$ and the error is $J^*_K(x) - J^*_\infty(x) = x^2/K$. It is obvious that $J^*_K(x) - J^*_\infty(x) = O(x^2) \to \infty$ when $|x| \to \infty$. The error will be very large when x is far away from the origin.

Hence, we will introduce our method for dynamic programming with the consideration of the length of control sequences. For different x, we will use different K for the length of optimal control sequence. For a given error bound according to $\epsilon > 0$, the number K will be chosen so that the error between $J^*_\infty(x)$ and $J^*_K(x)$ is bounded with a bound according to ϵ.

Definition 2.1 *Let $x \in \mathcal{T}_\infty$ be a controllable state vector. Let $\epsilon > 0$ be a positive number. The approximate length of optimal control with respect to ϵ is defined as*

$$K_\epsilon(x) = \min\{k\colon |J^*_k(x) - J^*_\infty(x)| \le \epsilon\}.$$ ■

Given a positive number ϵ, for any state vector x, the number $K_\epsilon(x)$ gives a suitable length of control sequence for optimal control starting from x. For $x \in \mathcal{T}_\infty$, since $\lim_{k\to\infty} J_k^*(x) = J_\infty^*(x)$, we can alway find k such that $|J_k^*(x) - J_\infty^*(x)| \leq \epsilon$. So $\{k\colon |J_k^*(x) - J_\infty^*(x)| \leq \epsilon\} \neq \emptyset$ and $K_\epsilon(x)$ is well defined. The optimal control sequence with this length will give a performance index which is close enough to the "theoretical optimal cost" $J_\infty^*(x)$. The performance index will be slightly larger than $J_\infty^*(x)$, but less than $J_\infty^*(x) + \epsilon$, which can be achieved by a control sequence with length K_ϵ. A control sequence with shorter length cannot lead to a performance index close enough to $J_\infty^*(x)$. Meanwhile, a control sequence with longer length will not be necessary if one considers $K_\epsilon(x)$ to be enough.

Definition 2.2 *Let $x \in \mathcal{T}_\infty$ be a controllable state vector and ϵ be a positive number. For $k = 1, 2, \ldots,$ define the set*

$$\mathcal{T}_k^{(\epsilon)} = \{x \in \mathcal{T}_\infty \colon K_\epsilon(x) \leq k\}. \qquad \blacksquare$$

When $x \in \mathcal{T}_k^{(\epsilon)}$, to find the optimal control sequence which has performance index less than or equal to $J_\infty^*(x) + \epsilon$, one only needs to consider the control sequences $\underline{u}$ with length $|\underline{u}| \leq k$. The sets $\mathcal{T}_k^{(\epsilon)}$ has the following properties.

Theorem 2.4 *Let $\epsilon > 0$ and $k = 1, 2, \ldots$. Then,*

(i) *$x \in \mathcal{T}_k^{(\epsilon)}$ if and only if $J_k^*(x) \leq J_\infty^*(x) + \epsilon$;*

(ii) $\mathcal{T}_k^{(\epsilon)} \subseteq \mathcal{T}_k$;

(iii) $\mathcal{T}_k^{(\epsilon)} \subseteq \mathcal{T}_{k+1}^{(\epsilon)}$;

(iv) $\cup_k \mathcal{T}_k^{(\epsilon)} = \mathcal{T}_\infty$;

(v) *If* $\epsilon > \delta > 0$, *then* $\mathcal{T}_k^{(\epsilon)} \supseteq \mathcal{T}_k^{(\delta)}$.

Proof. (i) Let $x \in \mathcal{T}_k^{(\epsilon)}$. By Definition 2.2, $K_\epsilon(x) \leq k$. Let $p = K_\epsilon(x)$. Then $p \leq k$ and by Definition 2.1, $|J_p^*(x) - J_\infty^*(x)| \leq \epsilon$. So $J_p^*(x) \leq J_\infty^*(x) + \epsilon$. By Theorem 2.1 $J_k^*(x) \leq J_p^*(x) \leq J_\infty^*(x) + \epsilon$. On the other hand, if $J_k^*(x) \leq J_\infty^*(x) + \epsilon$, then $|J_k^*(x) - J_\infty^*(x)| \leq \epsilon$. So $K_\epsilon(x) = \min\{p\colon |J_p^*(x) - J_\infty^*(x)| \leq \epsilon\} \leq k$. Therefore, $x \in \mathcal{T}_k^{(\epsilon)}$.

(ii) When $x \in \mathcal{T}_k^{(\epsilon)}$, $K_\epsilon(x) \leq k$. So $J_k^*(\cdot)$ has definition at x. Thus $x \in \mathcal{T}_k$. Hence $\mathcal{T}_k^{(\epsilon)} \subseteq \mathcal{T}_k$.

(iii) When $x \in \mathcal{T}_k^{(\epsilon)}$, $K_\epsilon(x) \leq k < k+1$. So $x \in \mathcal{T}_{k+1}^{(\epsilon)}$. Thus $\mathcal{T}_k^{(\epsilon)} \subseteq \mathcal{T}_{k+1}^{(\epsilon)}$.

(iv) Obviously $\cup_k \mathcal{T}_k^{(\epsilon)} \subseteq \mathcal{T}_\infty$ since $\mathcal{T}_k^{(\epsilon)}$ are subsets of $\mathcal{T}_\infty$. For any $x \in \mathcal{T}_\infty$, let $p = K_\epsilon(x)$. Then $x \in \mathcal{T}_p^{(\epsilon)}$. So $x \in \cup_k \mathcal{T}_k^{(\epsilon)}$. Hence $\mathcal{T}_\infty \subseteq \cup_k \mathcal{T}_k^{(\epsilon)} \subseteq \mathcal{T}_\infty$, and thus $\cup_k \mathcal{T}_k^{(\epsilon)} = \mathcal{T}_\infty$.

(v) When $x \in \mathcal{T}_k^{(\delta)}$, $J_k^*(x) \leq J_\infty^*(x) + \delta$ by part (i) of this theorem. Clearly, $J_k^*(x) \leq J_\infty^*(x) + \epsilon$ since $\delta < \epsilon$. This implies that $x \in \mathcal{T}_k^{(\epsilon)}$. Therefore $\mathcal{T}_k^{(\epsilon)} \supseteq \mathcal{T}_k^{(\delta)}$. ∎

According to Theorem 2.4(i), $\mathcal{T}_k^{(\epsilon)}$ is just the region where J_k^* is close to J_∞^* with error less than ϵ. This region is a subset of $\mathcal{T}_k$, see Theorem 2.4(ii). As stated in Theorem 2.4(iii), when k is large, the set $\mathcal{T}_k^{(\epsilon)}$ is also large. That means, when k is large, we have large region where we can

use $J_k^*(x)$ as the approximation of J_∞^* under certain error. On the other hand, we claim that for large x, we have to choose long control sequence to approximate the optimal control. Theorem 2.4(iv) means that for every controllable state $x \in \mathcal{T}_\infty$, we can always find a suitable length k of control sequence to approximate the optimal control. The size of the set $\mathcal{T}_k^{(\epsilon)}$ depends on the value of ϵ. Smaller value of ϵ gives smaller set $\mathcal{T}_k^{(\epsilon)}$; see Theorem 2.4(v).

Theorem 2.5 *Let $x \in \mathcal{T}_k^{(\epsilon)}$ and $u^* = v_k^*(x)$. Then $F(x, u^*) \in \mathcal{T}_{k-1}^{(\epsilon)}$. In other words, if $K_\epsilon(x) = k$, then $K_\epsilon(F(x, u^*)) \leq k - 1$.*

Proof. Since $x \in \mathcal{T}_k^{(\epsilon)}$, by Theorem 2.4(i) we know that

$$J_k^*(x) \leq J_\infty^*(x) + \epsilon. \tag{2.4.3}$$

By (2.2.6) and (2.2.7),

$$J_k^*(x) = U(x, u^*) + J_{k-1}^*(F(x, u^*)). \tag{2.4.4}$$

From (2.4.3) and (2.4.4), we have

$$\begin{aligned} J_{k-1}^*(F(x, u^*)) &= J_k^*(x) - U(x, u^*) \\ &\leq J_\infty^*(x) + \epsilon - U(x, u^*). \end{aligned} \tag{2.4.5}$$

By Theorem 2.2,

$$J_\infty^*(x) \leq U(x, u^*) + J_\infty^*(F(x, u^*)). \tag{2.4.6}$$

Putting (2.4.6) into (2.4.5) we obtain

$$J_{k-1}^*(F(x, u^*)) \leq J_\infty^*(F(x, u^*)) + \epsilon.$$

By Theorem 2.4(i) again, $F(x, u^*) \in \mathcal{T}_{k-1}^{(\epsilon)}$.

When $K_\epsilon(x) = k$, we know that $x \in \mathcal{T}_k^{(\epsilon)}$. Hence $F(x, u^*) \in \mathcal{T}_{k-1}^{(\epsilon)}$. Thus $K_\epsilon(F(x, u^*)) \leq k - 1$. The theorem is proved. ■

According to Theorem 2.5, the optimal controller $v_k^*(\cdot)$ has a property that it will drive a state vector from $\mathcal{T}_k^{(\epsilon)}$ to $\mathcal{T}_{k-1}^{(\epsilon)}$. So, when we want to find the optimal control for state $x \in \mathcal{T}_k^{(\epsilon)}$ via Bellman's principle of optimality (2.2.6), the minimum on the right-hand side will only need to be chosen from those costs with u such that $F(x, u) \in \mathcal{T}_{k-1}^{(\epsilon)}$. In Section II, we have defined admissible control sequences to be those which can drive a state to the origin. Now, we can define an admissible control vector, i.e., a single control vector, not a control sequence. For a controllable state $x \in \mathcal{T}_k^{(\epsilon)}$, a control vector u is called an admissible control vector of x if $F(x, u) \in \mathcal{T}_{k-1}^{(\epsilon)}$. Let

$$\Pi_x^{\epsilon,k} = \{u \colon F(x, u) \in \mathcal{T}_{k-1}^{(\epsilon)}\}, \quad x \in \mathcal{T}_k^{(\epsilon)}, \tag{2.4.7}$$

be the set of all admissible control vector of x. Then we have the following result:

Corollary 2.1 *If $x \in \mathcal{T}_k^{(\epsilon)}$, then*

$$J_k^*(x) = \min_{u \in \Pi_x^{\epsilon,k}} \{U(x, u) + J_{k-1}^*(F(x, u))\}, \tag{2.4.8}$$

and

$$v_k^*(x) = arg \min_{u \in \Pi_x^{\epsilon,k}} \{U(x, u) + J_{k-1}^*(F(x, u))\}.$$ ■

Equation (2.4.8) looks quite similar to (2.2.6). But there are some important differences. First, the state x belongs to the set $\mathcal{T}_k^{(\epsilon)}$ in (2.4.8), while in (2.2.6), x is any controllable state, i.e., $x \in \mathcal{T}_\infty$. Because $\mathcal{T}_k^{(\epsilon)}$ is smaller than

$\mathcal{T}_\infty$, less computation are needed in solving (2.4.8). Second, when we take the minimum, we choose u from $\Pi_x^{\epsilon,k}$ in (2.4.8). But in (2.2.6), we have to choose u from the whole space $\mathbb{R}^m$. Again, the set $\Pi_x^{\epsilon,k}$ is smaller and will lead to less computation. Of course, as clarified earlier, the control vector u in (2.2.6) has to be chosen so that u will be part of an admissible control sequence, i.e., there must be some $u_1, u_2, \ldots, u_{k-1}$ such that $(u, u_1, \ldots, u_{k-1}) \in \mathfrak{A}_x^k$. Unfortunately, this is very difficult to do in real applications. However, in (2.4.8), $u \in \Pi_x^{\epsilon,k}$. This can be done by choosing u such that $F(x,u) \in \mathcal{T}_{k-1}^{(\epsilon)}$, i.e., $K_\epsilon(F(x,u)) \leq k-1$. This is also not very easy, but at least it can be operated. Besides, when we try to find the optimal control, we only need to solve (2.4.8) for those $x \in \mathcal{T}_k^{(\epsilon)} \setminus \mathcal{T}_{k-1}^{(\epsilon)}$, i.e., those x such that $K_\epsilon(x) = k$, which will reduce the computation further more.

For any $x \in \mathcal{T}_\infty$, suppose that $K_\epsilon(x) = k$. Then $J_k^*(x)$ is an approximation of $J_\infty^*(x)$ with error less than ϵ. Thus we define the ϵ-optimal cost V_ϵ^* as

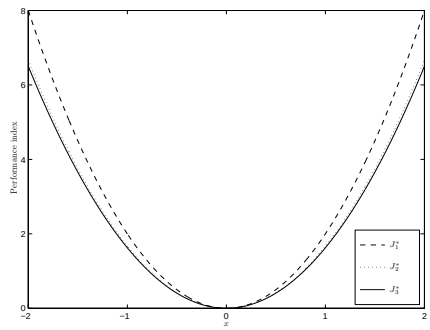
$$V_\epsilon^*(x) = \begin{cases} J_k^*(x), & \text{if } x \neq 0 \text{ and } K_\epsilon(x) = k, \\ & k = 1, 2, \ldots ; \\ 0, & \text{if } x = 0. \end{cases} \tag{2.4.9}$$

According to $V_\epsilon^*(x)$, we denote the associated controller by $\mu_\epsilon^*(\cdot)$ and call it the ϵ-optimal control vector:

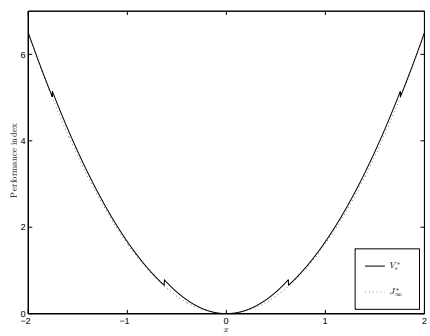
$$\mu_\epsilon^*(x) = \begin{cases} v_k^*(x), & \text{if } x \neq 0 \text{ and } K_\epsilon(x) = k, \\ & k = 1, 2, \ldots ; \\ 0, & \text{if } x = 0. \end{cases} \tag{2.4.10}$$

Both $V_\epsilon^*(x)$ and $\mu_\epsilon^*(x)$ are piecewise functions. When $x \in \mathcal{T}_k^{(\epsilon)} \setminus \mathcal{T}_{k-1}^{(\epsilon)}$, $V_\epsilon^*(x) = J_k^*(x)$ and $\mu_\epsilon^*(x) = v_k^*(x)$. See Fig. 2.1 for an illustration.

By using $k = K_\epsilon(x)$ in Corollary 2.1, we obtain that for $x \in \mathcal{T}_\infty$, if



(a) J_k^* for $k = 1, 2, 3$.



(b) V_ϵ^* and J_∞^*.

Figure 2.1: V_ϵ^* for system $x_{k+1} = x_k + u_k$ with utility $U(x, u) = x^2 + u^2$ and $\epsilon = 0.15$.

It is easy to verify that $J_1^*(x) = 2x^2$, $J_2^*(x) = 5x^2/3$, $J_3^*(x) = 13x^2/8$, and $J_\infty^*(x) = (1 + \sqrt{5})x^2/2$. The function $V_\epsilon^*(x)$ is a piecewise function. $V_\epsilon^*(x) = 2x^2$ if $|x| \leq 0.6267$, $V_\epsilon^*(x) = 5x^2/3$ if $0.6267 < |x| \leq 1.7562$, and $V_\epsilon^*(x) = 13x^2/8$ if $1.7562 < |x| \leq 4.6404$.

$k = K_\epsilon(x) > 0$, then

$$V_\epsilon^*(x) = \min_{u \in \Pi_x^{\epsilon,k}} \{U(x, u) + V_\epsilon^*(F(x, u))\}, \tag{2.4.11}$$

and

$$\mu_\epsilon^*(x) = \arg \min_{u \in \Pi_x^{\epsilon,k}} \{U(x, u) + V_\epsilon^*(F(x, u))\}. \tag{2.4.12}$$

The minimum is taken according to all $u \in \Pi_x^{\epsilon,k}$, where $k = K_\epsilon(x) > 0$, i.e.,

$x \in \mathcal{T}_k^{(\epsilon)}$,

$$\begin{aligned}\Pi_x^{\epsilon,k} &= \{u\colon F(x,u) \in \mathcal{T}_{k-1}^{(\epsilon)}\} \\ &= \{u\colon K_\epsilon(F(x,u)) \le k-1\}. \end{aligned} \tag{2.4.13}$$

In particular, if $K_\epsilon(x) = 1$,

$$\Pi_x^{\epsilon,1} = \{u\colon F(x,u) = 0\}, \tag{2.4.14}$$

$$V_\epsilon^*(x) = \min_{u \in \Pi_x^{\epsilon,1}} \{U(x,u)\} = J_1^*(x), \tag{2.4.15}$$

and

$$\mu_\epsilon^*(x) = \arg\min_{u \in \Pi_x^{\epsilon,1}} \{U(x,u)\} = v_1^*(x). \tag{2.4.16}$$

Equation (2.4.11) is quite similar to (2.3.3). But in (2.3.3), the right-hand side is the infimum, while in (2.4.11), the right-hand side is minimum. The infimum cannot always be reached. On one hand, for any admissible control sequence $\underline{u}$, the equality $J(x,\underline{u}) = J_\infty^*(x)$ may not hold. On the other hand, if $\underline{\tilde{u}}$ is a control sequence such that $J(x,\underline{\tilde{u}}) = J_\infty^*(x)$, then $\underline{\tilde{u}}$ may not be admissible. The trajectory under the control of $\underline{\tilde{u}}$ may not converge to the target at all, which has been illustrated in Examples 2.2 and 4.1. However, the minimum means that it can be reached. $V_\epsilon^*(x)$ is just $J_k^*(x)$ for some k: $k = K_\epsilon(x)$. The control sequence $\underline{v}_k^*(x)$ is admissible. It will drive the associated trajectory to the target $\{0\}$ in k steps and $J(x,\underline{v}_k^*(x)) = V_\epsilon^*(x)$.

Now we know that the ϵ-optimal cost $V_\epsilon^*(x)$ satisfies (2.4.11) (if $K_\epsilon(x) > 1$) and (2.4.15) (if $K_\epsilon(x) = 1$). We also know that the ϵ-optimal controller $\mu_\epsilon^*(x)$ satisfies (2.4.12). Let us consider what will happen if we use $\mu_\epsilon^*(x)$ to control the system. We have the following result.

Theorem 2.6 *Let $x \in \mathcal{T}_\infty$ be an arbitrary controllable state. Define*

$$\underline{\mu}^*_\epsilon(x) = (\mu^*_\epsilon(x_0), \mu^*_\epsilon(x_1), \ldots, \mu^*_\epsilon(x_j), \ldots), \tag{2.4.17}$$

where $x_0 = x$ and $x_{j+1} = F(x_j, \mu^*_\epsilon(x_j))$ for $j = 0, 1, \ldots$. If $K_\epsilon(x) = k > 0$, then $x_j = 0$ and $\mu^*_\epsilon(x_j) = 0$ for all $j \geq k$, and the performance index satisfies

$$J(x, \underline{\mu}^*_\epsilon(x)) \leq J^*_\infty(x) + \max(k-1, 1)\epsilon.$$

Proof. If $K_\epsilon(x) = 1$, then $x \in \mathcal{T}_1^{(\epsilon)}$ and $\mu^*_\epsilon(x) = v^*_1(x)$. So $x_1 = F(x, v^*_1(x)) = 0$. Hence the trajectory starting from $x_0 = x$ under the control of $\underline{\mu}^*_\epsilon(x)$ is $x_0 = x, 0, 0, \ldots$, while $\underline{\mu}^*_\epsilon(x) = (v^*_1(x), 0, 0, \ldots)$. The performance index is

$$J(x, \underline{\mu}^*_\epsilon(x)) = U(x, v^*_1(x)) + 0 + \ldots = J^*_1(x).$$

Since $x \in \mathcal{T}_1^{(\epsilon)}$, $J^*_1(x) \leq J^*_\infty(x) + \epsilon$. Thus $J(x, \underline{\mu}^*_\epsilon(x)) \leq J^*_\infty(x) + \epsilon$. The result of the theorem is proved in this case.

If $K_\epsilon(x) = 2$, then $x \in \mathcal{T}_2^{(\epsilon)}$ and $\mu^*_\epsilon(x) = v^*_2(x)$. So $x_1 = F(x, v^*_2(x)) \in \mathcal{T}_1^{(\epsilon)}$ (by Theorem 2.5) and $\mu^*_\epsilon(x_1) = v^*_1(x_1)$. Consequently, the trajectory starting from $x_0 = x$ under the control of $\underline{\mu}^*_\epsilon(x)$ is $x_0 = x, x_1, 0, 0, \ldots$, while $\underline{\mu}^*_\epsilon(x) = (v^*_2(x), v^*_1(x_1), 0, 0, \ldots)$. The performance index is

$$\begin{aligned} J(x, \underline{\mu}^*_\epsilon(x)) &= U(x, v^*_2(x)) + U(x_1, v^*_1(x_1)) + 0 + \ldots \\ &= J^*_2(x). \end{aligned}$$

Since $x \in \mathcal{T}_2^{(\epsilon)}$, $J^*_2(x) \leq J^*_\infty(x) + \epsilon$. Thus $J(x, \underline{\mu}^*_\epsilon(x)) \leq J^*_\infty(x) + \epsilon$. The result of the theorem is proved in this case.

Let $L > 2$. Assume that the result of the theorem is true when $K_\epsilon(x) < L$. Now let us consider the case $K_\epsilon(x) = L$. In this case, $x \in \mathcal{T}_L^{(\epsilon)}$ and

$\mu_\epsilon^*(x) = v_L^*(x)$ (see (2.4.10)). So $x_1 = F(x, v_L^*(x)) \in \mathcal{T}_{L-1}^{(\epsilon)}$ (by Theorem 2.5), and hence $K_\epsilon(x_1) \leq L-1$. Now

$$\begin{aligned}\underline{\mu}_\epsilon^*(x) &= (\mu_\epsilon^*(x_0), \mu_\epsilon^*(x_1), \dots, \mu_\epsilon^*(x_j), \dots),\\ &= (v_L^*(x_0), \underline{\mu}_\epsilon^*(x_1)),\end{aligned}$$

where $x_0 = x$, $x_{j+1} = F(x_j, \mu_\epsilon^*(x_j))$ for $j = 0, 1, \dots,$ and

$$\underline{\mu}_\epsilon^*(x_1) = (\mu_\epsilon^*(x_1), \mu_\epsilon^*(x_2), \dots, \mu_\epsilon^*(x_j), \dots).$$

Obviously $(x_1, x_2, \dots, x_j, \dots)$ is the trajectory starting from x_1 under the control of $\underline{\mu}_\epsilon^*(x_1)$. Since $K_\epsilon(x_1) \leq L-1 < L$, by the induction assumption, the result of the theorem holds for x_1. So $x_j = 0$ and $\mu_\epsilon^*(x_j) = 0$ for all $j-1 \geq L-1$, and the performance index

$$J(x_1, \underline{\mu}_\epsilon^*(x_1)) \leq J_\infty^*(x_1) + (L-2)\epsilon.$$

Now we have $x_j = 0$ and $\mu_\epsilon^*(x_j) = 0$ for all $j \geq L$, and the performance index

$$\begin{aligned}J(x_0, \underline{\mu}_\epsilon^*(x_0)) &= U(x_0, v_K^*(x_0)) + J(x_1, \underline{\mu}_\epsilon^*(x_1))\\ &\leq U(x_0, v_K^*(x_0)) + J_\infty^*(x_1) + (L-2)\epsilon\\ &\leq U(x_0, v_K^*(x_0)) + J_{K-1}^*(x_1) + (L-2)\epsilon\\ &= J_K^*(x_0) + (L-2)\epsilon\\ &\leq J_\infty^*(x_0) + (L-1)\epsilon.\end{aligned}$$

Hence the result of the theorem holds when $K_\epsilon(x) = L$. Therefore, by principle of induction, the result of the theorem holds for any $x \in \mathcal{T}_\infty$. ■

According to Theorem 2.6, the control sequence $\underline{\mu}_\epsilon^*(x)$ is an admissible control sequence. If $K_\epsilon(x) = k$, the system will go to the target at most in k steps under the control of $\underline{\mu}_\epsilon^*(\cdot) = \underline{v}_k^*(\cdot)$. In this case, the performance index

$J(x, \underline{\mu}_\epsilon^*(x))$ is an approximation of $J_\infty^*(x)$, with error less than $\max(k-1, 1)\epsilon$. Besides, according to (2.4.17), the control sequence $\underline{\mu}_\epsilon^*(x)$ can be calculated using the single control law $\mu_\epsilon^*(\cdot)$. In the next section, we will introduce our new method for dynamic programming of discrete-time systems. The ϵ-optimal cost $V_\epsilon^*(\cdot)$ and ϵ-optimal controller $\underline{\mu}_\epsilon^*(\cdot)$ will be employed as alternatives for the optimal cost and optimal controller.

2.5 The ϵ-Adaptive Dynamic Programming for Discrete-Time System using Neural Networks

In this section, we will introduce a numerical algorithm of ϵ-adaptive dynamic programming for discrete-time systems. A neural network $\hat{J}$ will be used to approximate the ϵ-optimal cost V_ϵ^*, i.e., it will approximate $J_k^*(x)$ if $K_\epsilon(x) = k$, where $\epsilon > 0$ determines an error bound. At the same time, a neural network $\hat{\mu}$ will be used to approximate the ϵ-optimal controller μ_ϵ^*. A third network in our algorithm is $\hat{K}$, which is used to approximate K_ϵ.

For initialization, consider the problem

$$J_1^*(x) = \min_u U(x, u) \text{ subject to } F(x, u) = 0. \tag{2.5.1}$$

The set of x such that (2.5.1) has solutions is just $\mathcal{T}_1$. When $x \in \mathcal{T}_1$, consider the equation $F(x, u) = 0$. If there is only one solution to this equation, the solution is just $v_1^*(x)$ since it is the only choice. If there are more than one solutions, we will choose the one which minimizes the cost. Suppose that $u = f^{(i)}(x)$, $i \in I$, are all the implicit functions defined by $F(x, u) = 0$, where I is a suitable index set. Then

$$\begin{cases} J_1^*(x) = \min_{i \in I}\{U(x, f^{(i)}(x))\} = U(x, v_1^*(x)), \\ v_1^*(x) = f^{(i_0)}(x), \end{cases} \tag{2.5.2}$$

where i_0 is the index such that $f^{(i_0)}(x)$ minimizes $U(x, f^{(i)}(x))$. Then we will initialize the networks according to

$$\begin{aligned} \hat{K}(x) &= \begin{cases} 1, \text{ if } x \in \mathcal{T}_1, \ x \neq 0, \\ 0, \text{ otherwise,} \end{cases} \\ \hat{J}(x) &= \begin{cases} J_1^*(x), \text{ if } x \in \mathcal{T}_1, \\ 0, \text{ otherwise,} \end{cases} \\ \hat{\mu}(x) &= \begin{cases} v_1^*(x), \text{ if } x \in \mathcal{T}_1, \\ 0, \text{ otherwise.} \end{cases} \end{aligned} \tag{2.5.3}$$

After that, the networks $\hat{J}$ and $\hat{\mu}$ will be updated from measured data according to the formulas (2.4.11) and (2.4.12). According to (2.4.13), if $K_\epsilon(F(x,u)) = k$, then $K_\epsilon(x) \geq k+1$. So we will update $\hat{K}$ by

$$\hat{K}(x) = \hat{K}(F(x, \hat{\mu}(x)) + 1. \tag{2.5.4}$$

For every state $x \in \mathbb{R}^n$, the integer $k = K_\epsilon(x)$ indicates that the optimal performance index will be realized (with an error bound determined by ϵ) in just k steps. If $K_\epsilon(x)$ is infinite, then x is uncontrollable. In other words, if $K_\epsilon(x) = k$ then $J_k^*(x)$ is an approximation to the optimal performance index $J_\infty^*(x)$, and if $K_\epsilon(x) = +\infty$, then the optimal cost cannot be reached in finite steps. As defined earlier, if $K_\epsilon(x) \leq k$ then $x \in \mathcal{T}_k^{(\epsilon)}$.

As a numerical algorithm, we only consider bounded regions and finite control steps. We use a number maxK to represent the upper bound of control steps. The total number of steps of the system to reach the target from any initial state is restricted to $k \leq$ maxK. In other words, for an initial state x, if one cannot control the system to reach the target within maxK steps starting from x, then x will be considered to be uncontrollable.

The following gives the algorithm ADPDN(ϵ) for ϵ-*adaptive dynamic programming.*

Algorithm ADPDN(ϵ) (ϵ-adaptive dynamic programming for discrete time systems using neural networks)

A00 Initialization. Solve problem (2.5.1) to determine the set $\mathcal{T}_1$ and the functions $J_1^*(x)$ and $v_1^*(x)$. Then train $\hat{K}(x)$, $\hat{J}(x)$ and $\hat{\mu}(x)$ according to (2.5.3)

A01 Choose randomly an array of initial states $x_0 = (x_0^{(1)}, x_0^{(2)}, \ldots, x_0^{(p)})$ from the entire state space.

A02 For $k = 1, 2, \ldots, \text{maxK}$, do A03–A06.

A03 Run the system from the array of states x_0 under the control of $\hat{\mu}$. Record the resultant array of states $x_1 = (x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(p)})$. For $i = 1, 2, \ldots, p$, calculate the associated costs $C_1^{(i)} = U(x_0^{(i)}, \hat{\mu}(x_0^{(i)})) + \hat{J}(x_1^{(i)})$. Record each $k_0^{(i)} = \hat{K}(x_0^{(i)})$ and $k_1^{(i)} = \hat{K}(x_1^{(i)})$.

A04 Update $\hat{J}$ and $\hat{K}$. For each $i = 1, \ldots, p$,

if $x_0^{(i)} \in \mathcal{T}_1$ and $J_1^*(x_0^{(i)}) \leq C_1^{(i)} + \epsilon$ then

$$\begin{cases} \hat{K}(x_0^{(i)}) = \begin{cases} 0, & \text{if } x_0^{(i)} = 0, \\ 1, & \text{if } x_0^{(i)} \neq 0, \end{cases} \\ \hat{J}(x_0^{(i)}) = J_1^*(x_0^{(i)}), \end{cases}$$

elseif $k_1^{(i)} \leq \max(k_0^{(i)} - 1, 1)$ then

$$\begin{cases} \hat{K}(x_0^{(i)}) = k_1^{(i)} + 1, \\ \hat{J}(x_0^{(i)}) = C_1^{(i)}. \end{cases}$$

endif

A05 Update $\hat{\mu}$. For each $i = 1, \ldots, p$,

if $x_0^{(i)} \in \mathcal{T}_1$ and $J_1^*(x_0^{(i)}) \leq \hat{J}(x_0^{(i)}) + \epsilon$ then

$$\hat{\mu}(x_0^{(i)}) = v_1^*(x_0^{(i)}),$$

else

$$\hat{\mu}(x_0^{(i)}) = \arg\min_u \{U(x_0^{(i)}, u) + \hat{J}(F(x_0^{(i)}, u)) : \\ \hat{K}(F(x_0^{(i)}, u)) \leq \max(\hat{K}(x_0^{(i)}) - 1, 1)\},$$

endif

A06 Let $x_0 = x_1$.

A07 Go to A02 until the process converges.

At the beginning of the algorithm ADPDN(ϵ), in A00, the problem (2.5.1) is solved according to (2.5.2). The functions J_1^*, v_1^* and the set $\mathcal{T}_1$ are determined. Then, the neural networks $\hat{K}$, $\hat{J}$ and $\hat{\mu}$ are initialized according to (2.5.3). Thus, for $x \in \mathcal{T}_1^{(\epsilon)}$, we have $\hat{K}(x) = K_\epsilon(x)$, $\hat{J}(x) = V_\epsilon^*(x)$, and $\hat{\mu}(x) = \mu_\epsilon^*(x)$. Furthermore, according to A04, the values of $\hat{K}$, $\hat{J}$ and $\hat{\mu}$ for $x \in \mathcal{T}_1^{(\epsilon)}$ will not change during the training of the network. The final outputs of the ADPDN(ϵ) algorithm are the three neural networks, $\hat{K}$, $\hat{J}$ and $\hat{\mu}$,. They will approximate K_ϵ, V_ϵ^* and μ_ϵ^*, respectively.

From A01 to A07, there are two levels of loops. The outer loop, starting from A01 and ending at A07, will give initial states x_0 and then go into the inner loop. The inner loop runs the system maxK steps. As we have mentioned before, for an initial state x, if one cannot control the system to reach the target within maxK steps starting from x, then we will consider x to be uncontrollable.

The body of the inner loop, from A03 to A05, will run the system from x_0 under the control of $\hat{\mu}$ and update the networks according to the results. There are two different ways to adjust the networks. When $x_0^{(i)} \in \mathcal{T}_1$ and $J_1^*(x_0^{(i)}) \leq C_1^{(i)} + \epsilon$, one will consider $x_0^{(i)} \in \mathcal{T}_1^{(\epsilon)}$ and update the networks by (2.5.3). Otherwise, the condition $k_1^{(i)} \leq \max(k_0^{(i)} - 1, 1)$ implies that $x_1^{(i)}$ is one step closer to the target $x = 0$, and one will update networks by (2.4.11), (2.4.12) and (2.5.4).

The initial states x_0 are chosen randomly in A01. Suppose that the associated random probability density function is non-vanished everywhere. Then we can assume that all the states will be explored. So we know that the resulting networks tend to satisfy the formulas (2.4.11)–(2.4.16) for all state vectors x. Since $\hat{K}(x) = K_\epsilon(x)$, $\hat{J}(x) = V_\epsilon^*(x)$, and $\hat{\mu}(x) = \mu_\epsilon^*(x)$ for $x \in \mathcal{T}_1^{(\epsilon)}$, we conclude that the limit of $\hat{K}$, $\hat{J}$, and $\hat{\mu}$ will approximate the values K_ϵ, V_ϵ^*, and μ_ϵ^*, respectively.

The network $\hat{K}$, which is an approximation of K_ϵ, is very important in the algorithm ADPDN(ϵ). It is applied to check whether a state $x_0^{(i)}$ belongs to $\mathcal{T}_1^{(\epsilon)}$ or $\mathcal{T}_k^{(\epsilon)}$ for $k > 1$. It also indicates the region where optimal control exists, and how long a state will spend to get to the target under the control of an optimal controller $v_k^*(\cdot)$. For a state x, if $\hat{K}(x) > \text{maxK}$ then we can consider x to be uncontrollable. If $\hat{K}(x) \leq \text{maxK}$, then $\hat{K}(x)$ is an estimate of the number of control steps to drive x to the target.

Our ADPDN(ϵ) algorithm can also allow the use of a neural network $\hat{F}$ for approximating the system function F. $\hat{F}$ is usually trained off-line. Starting from an initial state, it will try to reach the target. If it cannot find the way in maxK steps, it will go back to the starting position and explore again. At the beginning, the algorithm may not find the optimal controller. It may not

even drive the state along a right path to the target. But after some times of running, it will know how to reach the target. Then it will try to find a way with smaller cost. When it has enough experience, it will find a good approximation to the optimal controller.

The algorithm ADPDN(ϵ) can be applied to both linear and nonlinear systems. We only need that F and U are continuous functions and satisfy the conditions that $F(0,0) = 0$, $U(x,u) \geq 0$, $U(0,0) = 0$, and (C2.1) holds. But some details of the algorithm ADPDN(ϵ) will be adjusted according to different F and U. In A00, we have to solve the problem (2.5.1). We need to find all the implicit functions of $F(x,u) = 0$. This can be done by either analytic method or numerical method, depending on the function $F(x,u)$. In A05, we have to find $\arg\min_u[U(x,u) + \hat{J}(F(x,u))]$. For different F and U, and different kind of network $\hat{J}$, one can use different methods to solve this minimization problem. For example, if $F(x,u) = a(x) + b(x)u$, $U(x,u) = x^2 + u^2$, and a radial basis function neural network (RBFNN) is used with $\hat{J}(x) = \sum_i \omega_i \phi(\|x - b_i\|^2/a_i)$, then one can find the minimum by solving equation

$$\frac{\partial}{\partial u}[U(x,u) + \hat{J}(w)] = 2u + \hat{J}'(w)b(x) = 0,$$

where $w = F(x,u)$. Therefore

$$u = -\frac{1}{2}b(x)\sum_i \omega_i \phi'(\|w - b_i\|^2/a_i),$$

which is similar to the iterative formula in [2, 18, 27, 35] (see (1.1.4), (1.1.8) and (1.1.11)). In the next section, numerical experiments using RBFNNs will be provided. Our algorithm works well in all these experiments. We note that other kinds of neural network structures such as backpropagation neural networks as well as recurrent neural networks can also be employed for our numerical experiments.

2.6 Numerical Experiments

To evaluate the performance of algorithm ADPDN(ϵ), we select two nonlinear unstable systems with non-quadratic utility functions for numerical experiments. The program is written in MATLAB and running on a Dell Dimension 8300 PC with Windows XP and Pentium IV processor.

Example 2.4

We consider the system

$$x_{k+1} = F(x_k, u_k) \triangleq x_k + \sin(x_k + u_k), \tag{2.6.1}$$

where $x_k, u_k \in \mathbb{R}$, and $k = 0, 1, 2, \ldots$. The utility function is $U(x, u) = |x| + u^2$. Since $F(0, 0) = 0$, $x = 0$ is an equilibrium state of system (2.6.1). But since $(\partial F/\partial x)(0, 0) = 2 > 1$, (2.6.1) is unstable at $x = 0$.

We choose the value of $\epsilon = 0.01$. The region of states considered here is $|x| \leq 5$ and the number of control steps is restricted to 10. The size of the set of initial states x_0 (A01 of the algorithm) will be 50 each time, i.e., at the beginning of the each inner loop iteration, we choose randomly 50 initial states.

The neural networks $\hat{K}$, $\hat{J}$ and $\hat{\mu}$ will be trained from the observations while the algorithm is running. We expect that they have the ability to keep the old feature when they are trained by some new data. For this purpose, radial basis function neural networks are adopted. Each network will be a linear combination of functions $\sum_i w_i \phi(\|x - c_i\|/\rho_i)$, where w_i are the weights of neurons, $\phi(r)$ is the basis function, c_i is the center and ρ_i is the width of each neuron. For $\hat{K}$, $\phi(r) = 1$ if $-1 < r \leq 1$ and $\phi(r) = 0$ elsewhere. For

$\hat{J}$ and $\hat{\mu}$, $\phi(r) = \exp(-r^2)$. The number of neurons, and the centers and widths of all neurons will be obtained through training. The value of the width ρ_i will go from 2 down to 0.001 after training. The updating rule is the resource allocation network via extended Kalman filter (RANEKF) algorithm [15], [37] for RBFNNs.

Since $F(x,u) = x+\sin(x+u)$, the implicit functions according to $F(x,u) = 0$ are

$$u = f^{(i)}(x) = 2i\pi + \sin^{-1}(-x) - x,$$

and

$$u = g^{(i)}(x) = (2i+1)\pi - \sin^{-1}(-x) - x,$$

where $-1 \le x \le 1$, and $i = 0, \pm 1, \pm 2, \ldots$. It is easy to see that $\mathcal{T}_1 = [-1, 1]$ and

$$\begin{cases} J_1^*(x) = \min\{U(x,u)\colon F(x,u) = 0\} \\ \qquad\quad = |x| + (-x + \sin^{-1}(-x))^2, \\ v_1^*(x) = -x + \sin^{-1}(-x), \end{cases}$$

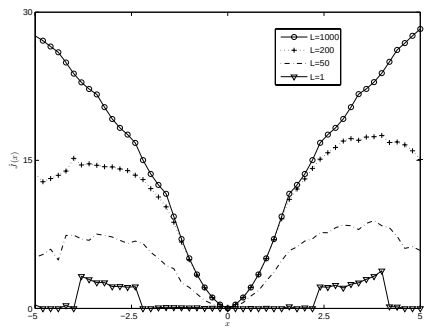
for $x \in \mathcal{T}_1$. Since the value of $\sin(x+u)$ is between -1 an 1, one can easily determine that $\mathcal{T}_k = [-k, k]$ for any $k = 1, 2, \ldots$.

In A05 of the algorithm ADPDN(ϵ), we need to solve $\arg\min_u[U(x,u) + \hat{J}(F(x,u))]$. Now $U(x,u) = |x| + u^2$, $F(x,u) = x + \sin(x+u)$. So the minimization problem becomes

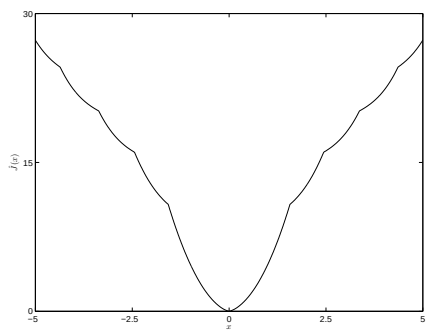
$$\min_u\{|x| + u^2 + \hat{J}(w)\}, \tag{2.6.2}$$

where $w = x + \sin(x+u)$. Since $\sin(\cdot)$ is a periodic function with period 2π, we know that the minimum of (2.6.2) will be reached when $u \in (-\pi.\pi]$.

We perform our ADPDN(ϵ) algorithm and save $\hat{K}$, $\hat{J}$, and $\hat{\mu}$ at the end of each outer loop iteration, i.e., save them in A06 of the algorithm. Then



(a) $L = 1, 50, 200, 1000$

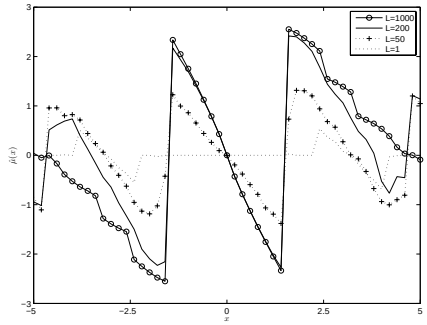


(b) $L = 1500, \ldots, 2000$

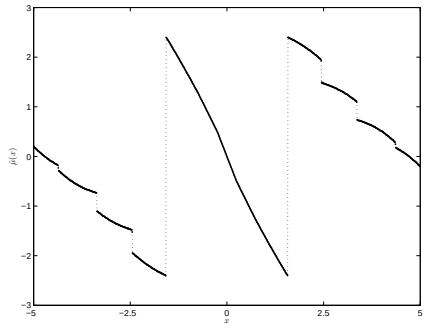
Figure 2.2: $\hat{J}(x)$ for $x \in [-5, 5]$

we obtain sequences of $\hat{K}$, $\hat{J}$, and $\hat{\mu}$. They will approximate $K_\epsilon(x)$, $V_\epsilon^*(x)$, and $\mu_\epsilon^*(x)$, respectively. Denote the number of iterations of the outer loop by L. We run the algorithm till $L = 2000$ and it took about 20 minutes. The resulting networks $\hat{K}$, $\hat{J}$ and $\hat{\mu}$ have 93, 3539 and 6210 neurons, respectively.

Figs. 2.2 and 2.3 are graphs of $\hat{J}(x)$ and $\hat{\mu}(x)$, respectively. In Figs. 2.2(a) and 2.3(a), $L = 1, 50, 200, 1000$. Figs. 2.2(b) and 2.3(b) are the graphs of $\hat{J}(x)$ and $\hat{\mu}(x)$ when $1500 \leq L \leq 2000$, which are almost the same for different L. They are approximations of $V_\epsilon^*(x)$ and $\mu_\epsilon^*(x)$, respectively. Obviously, both $V_\epsilon^*(x)$ and $\mu_\epsilon^*(x)$ are piecewise functions.



(a) $L = 1, 50, 200, 1000$

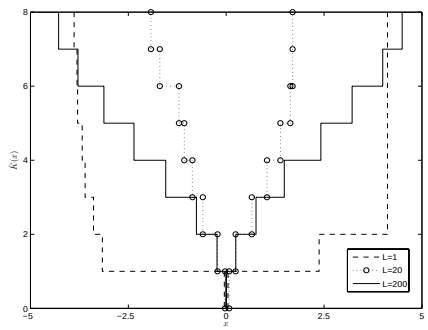


(b) $L = 1500, \ldots, 2000$

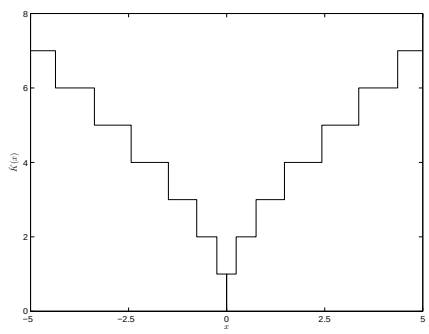
Figure 2.3: $\hat{\mu}(x)$ for $x \in [-5, 5]$

Fig. 2.4 is the graph of $\hat{K}$. In Fig. 2.4(a), $L = 1, 20, 200$. Fig. 2.4(b) is the final result of $\hat{K}$ when $L = 2000$, which is just K_ϵ or very close to it. When $|x|$ is small, $\hat{K}(x)$ is small, i.e., fewer number of steps of controls is needed to drive x to the target, and when $|x|$ is large, more steps are needed to control the system to the target.

Fig. 2.5 is the trajectory starting from initial state $x_0 = 4$ under the control law $\hat{\mu}$. In Fig. 2.5(a), $L = 10, 20, \ldots, 200$. When $L \leq 70$, the trajectory cannot reach the target in 10 steps. When $L \geq 80$, the trajectory can reach the target in 10 steps. Fig. 2.5(b) is the trajectory starting from $x_0 = 4$ for $1500 \leq L \leq 2000$. They are all so close to each other that they



(a) $L = 1, 20, 200$



(b) $L = 2000$

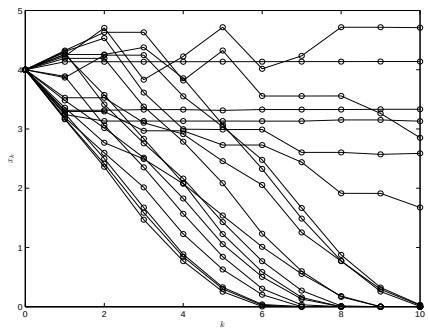
Figure 2.4: $\hat{K}(x)$ for $x \in [-5, 5]$

seem to be just one trajectory. Hence, when $L \geq 1500$, the trajectory will arrive at the target in 6 steps and will be quite close to the optimal trajectory.

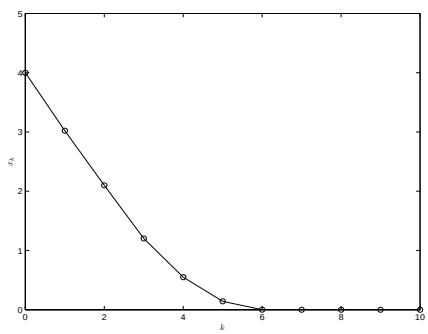
■

Example 2.5

Now we consider the ball and beam experiment. A ball is placed on a beam; see Fig. 2.6. The beam is allowed to roll with about 2 degrees of freedom along the length of the beam. A lever arm is attached to the beam at one end. A servo gear is attached to the other end of the arm. When the servo gear turns by an angle θ, the lever changes the angle of the beam by α.



(a) $L = 1, 20, \ldots, 200$



(b) $L = 1500, \ldots, 2000$

Figure 2.5: Trajectories starting from $x_0 = 4$

When the angle α is changed, gravity causes the ball to roll along the beam. A controller will be designed by controlling the gear of this system so that the ball goes to the middle of the beam.

Here, we assume that the ball rolls without slipping, and friction between the beam and ball is negligible. The second derivative of the input θ will affect the second derivative of r. However, we will ignore this contribution since it is very small. The equation of motion for the ball is then given by the following equation

$$\left(\frac{M}{R^2} + m\right)\ddot{r} + mg\sin\alpha - mr(\dot{\alpha})^2 = 0, \tag{2.6.3}$$

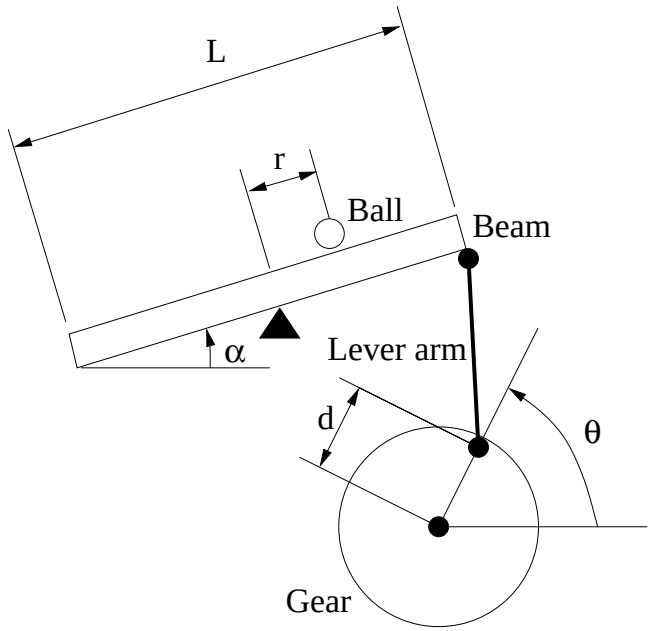


Figure 2.6: Ball and beam experiment

where $m = 0.1$ kg is the mass of the ball, $R = 0.015$ m is the radius of the ball, $d = 0.03$ m is the radius of the lever gear, $g = 9.8$ m/s^2 is the gravitational acceleration, $L = 1.0$ m is the length of the beam, $M = 10^{-5}$ kgm^2 is the ball's moment of inertia, and r is the ball position coordinate, $|r| \leq 0.5$, and $r = 0$ is at the middle of the beam. The beam angle α can be expressed in terms of the servo gear angle θ as

$$\alpha \approx \frac{2d}{L}\theta.$$

We assume that $|\theta| \leq \pi/4 \approx 0.7854$ (consequently, the range of values of α is $|\alpha| \leq 0.06\pi/4 \approx 2.7°$). We will use $u = \theta$ as the control signal of this system.

Given the time step $\triangle t$, let $r_i = r(i\triangle t)$, $\dot{r}_i = \dot{r}(i\triangle t)$, $\ddot{r}_i = \ddot{r}(i\triangle t)$, $\alpha_i = \alpha(i\triangle t)$, and $\theta_i = \theta(i\triangle t)$. Discretize (2.6.3) by

$$\begin{cases} \dot{r}_i = \dfrac{r_i - r_{i-1}}{\triangle t}, \\ \ddot{r}_i = \dfrac{r_{i+1} - 2r_i + r_{i-1}}{\triangle t^2}, \\ \dot{\alpha}_i = \dfrac{\alpha_i - \alpha_{i-1}}{\triangle t}. \end{cases}$$

Then (2.6.3) becomes

$$\begin{aligned} r_{i+1} = 2r_i - r_{i-1} - \frac{mg\triangle t^2 R^2}{M + mR^2}\sin\alpha_i + \\ \frac{mR^2}{M + mR^2} r_i(\alpha_i - \alpha_{i-1})^2. \end{aligned} \tag{2.6.4}$$

Now define $x_i = r_i$, $y_i = r_i - r_{i-1}$, and $z_i = \theta_{i-1}$. Let

$$P = \frac{mg\triangle t^2 R^2}{M + mR^2}$$

and

$$Q = \frac{4d^2 mR^2}{L^2(M + mR^2)}.$$

Then

$$\begin{cases} x_{i+1} = x_i + y_i - P\sin\left(\dfrac{2d}{L}\theta_i\right) + Qx_i(\theta_i - z_i)^2, \\ y_{i+1} = y_i - P\sin\left(\dfrac{2d}{L}\theta_i\right) + Qx_i(\theta_i - z_i)^2, \\ z_{i+1} = \theta_i. \end{cases} \tag{2.6.5}$$

(2.6.5) gives a nonlinear system. The state vector is $(x_i, y_i, z_i)^T \in \mathbb{R}^3$ and the control signal is $u_i = \theta_i \in \mathbb{R}$.

The control target here is $x_i = y_i = z_i = 0$. The utility $U(0,0,0,0) = 0$ and $U(x,y,z,\theta) = 1$ for $(x,y,z,\theta) \neq 0$. Thus the optimal control is the control which can make the ball go to the middle of the beam the fastest.

The region of states we considered here is $|x| \leq 0.5$, $|y| \leq 0.15$, and $|z| \leq 0.8$. The maximal time of control is restricted to 4 seconds. Since the state-control space has dimension 4 in this experiment, the size of the radial basis function neural network will be very large if $\triangle t$ is small. Considered both the accuracy and the time expense, we choose $\triangle t = 0.1$ and we set $\epsilon = 0.002$. Then the maximal control steps maxK will be 40. The size of x_0 (in A01 of the algorithm) is chosen as 500 each time, i.e., at the beginning of each inner loop iteration, we choose randomly 500 initial states.

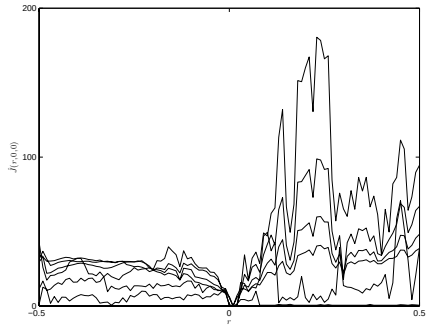
With the values of the given parameters, we obtain $P = 0.067846$ and $Q = 0.0024923$. The ADPDN(ϵ) algorithm runs until it reaches $L = 550000$ for outer loops, which takes about 27 hours of running time. It is important to note that after a short while of running the program (when L reaches 40000), we were able to achieve a solution which moves the ball to the center of the beam. At that time, the controller is not optimized yet in the sense that it can only drive the state to the target but it may not be the fastest. If we continue to run our algorithm, after a long time (27 hours, when $L = 550000$), our algorithm seems to converge to the *real* optimal controller of the problem which can drive the state to the target with minimum amount of time. It is also important to note that our goal in these experiments is to show that controllers obtained using our ADPDN algorithm do show gradual improvement through learning and they do converge to a controller that can do the job very well in the end. Further reduction in the total amount of time needed to obtain such a controller is under way by optimizing the choice of neural network structures and training algorithms. We save $\hat{K}$, $\hat{J}$, and $\hat{\mu}$ at the end of each outer loop iteration. Then we obtain sequences of $\hat{K}$, $\hat{J}$, and $\hat{\mu}$ for $L = 1, 2, \ldots, 550000$. They will approximate $K_\epsilon(x, y, z)$, $V_\epsilon^*(x, y, z)$, and $\mu_\epsilon^*(x, y, z)$, respectively. When $L = 550000$, the sizes of $\hat{J}$, $\hat{K}$ and $\hat{\mu}$ are

about 1.2×10^7, 3.46×10^8, and 3.24×10^8, respectively. During the running of our program, the peak memory is about 2G.

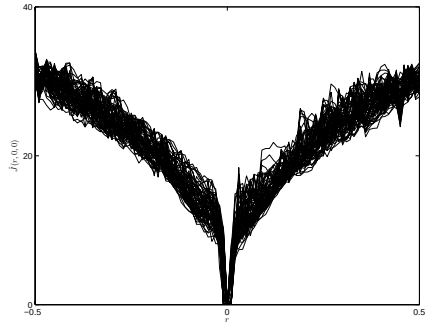
Figs. 2.7 and 2.8 show $\hat{J}(x, y, z)$ for $(x, y, z) = (r, 0, 0)$ and $(x, y, z) = (r, 0, 0.4)$, where $-0.5 \leq r \leq 0.5$. The states $(r, 0, 0)$ and $(r, 0, 0.4)$ mean the ball is at position r with velocity zero while the beam is in horizontal position or has an angle $0.4(2d/L) \approx 0.024$ ($\approx 1.375°$), respectively. Figs. 2.9 and 2.10 show $\hat{\mu}(x, y, z)$ for $(x, y, z) = (r, 0, 0)$ and $(x, y, z) = (r, 0, 0.4)$, where $-0.5 \leq r \leq 0.5$. When $L = 550000$, both $\hat{J}(x, y, z)$ and $\hat{\mu}(x, y, z)$ are very close to their limits, $V_\epsilon^*(x, y, z)$ and $\mu_\epsilon^*(x, y, z)$, respectively. The curves of $\hat{\mu}$ seem quite smooth, although one can believe there must be some small variations, which is caused by $\epsilon = 0.02$. As for $\hat{J}$, by Theorem 4.3, the error between $\hat{J}$ and J_∞^* will be less than $(k-1)\epsilon \leq 32 * 0.02 = 0.64$. The errors can be observed from Figs. 2.7(c) and 2.8(c).

Fig. 2.11 shows the graph of $\hat{K}(x, y, z)$ when $L = 550000$ for $(x, y, z) = (r, 0, 0)$ and $(x, y, z) = (r, 0, 0.4)$, where $-0.5 \leq r \leq 0.5$. From Fig. 2.11(a) we can see that $\hat{K}(x, y, z) \leq 32$ for all $(x, y, z) = (r, 0, 0)$, $-0.5 \leq r \leq 0.5$. So the controller $\hat{\mu}$ will move the ball to the middle of the beam in a maximum of 32 steps, i.e., in 3.2 seconds, if the ball is placed on the beam initially at rest while the beam is in horizontal state, i.e., the initial state is $(r, 0, 0)$. If the beam has an angle 0.4 at the beginning, the time to drive the ball to the middle will be slightly different, as shown in Fig. 2.11(b).

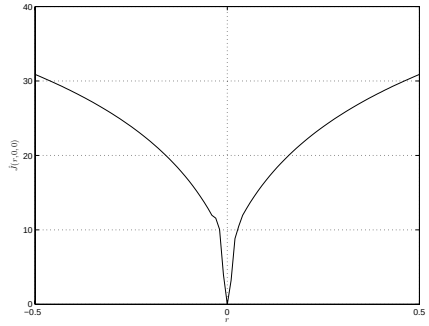
Consider the initial state $(x_0, y_0, z_0) = (0.45, 0, 0)$. From the value of $\hat{K}(0.45, 0, 0)$ we know that the ϵ-optimal control will drive the ball to the middle in less than 30 steps, i.e., 3 seconds. Fig. 2.12 is the graph of the trajectory starting from the initial state $(0.45, 0, 0)$. When $L < 30000$, the trajectory is totally out of order. The ball rolls on the beam left and right and



(a) $\hat{J}(r, 0, 0)$ for $L = 1, 2, \ldots, 7$

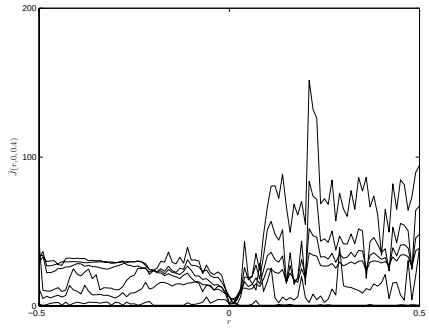


(b) $\hat{J}(r, 0, 0)$ for $L = 1000, 2000, \ldots, 550000$

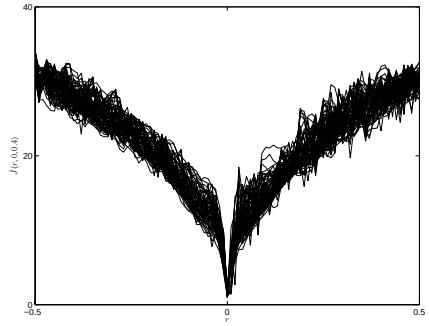


(c) $\hat{J}(r, 0, 0)$ for $L = 550000$

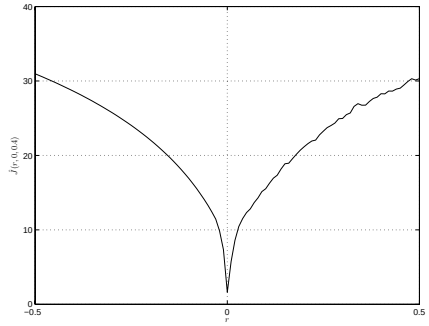
Figure 2.7: $\hat{J}(r, 0, 0)$ for $L = 0, \ldots, 550000$



(a) $\hat{J}(r, 0, 0.4)$ for $L = 1, 2, \ldots, 7$

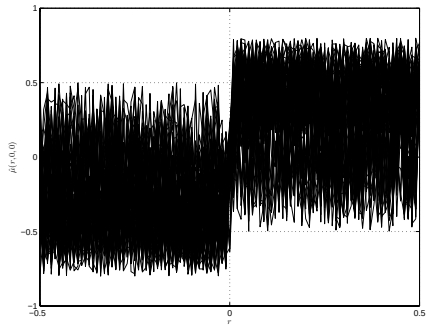


(b) $\hat{J}(r, 0, 0.4)$ for $L = 1000, 2000, \ldots, 550000$

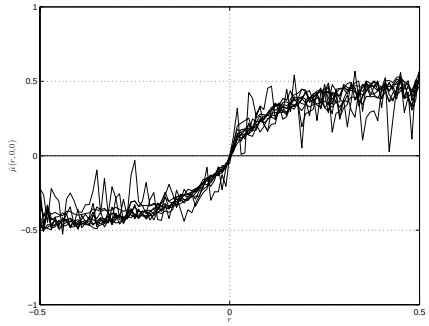


(c) $\hat{J}(r, 0, 0.4)$ for $L = 550000$

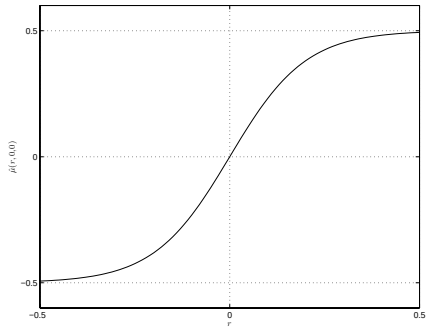
Figure 2.8: $\hat{J}(r, 0, 0.4)$ for $L = 0, \ldots, 550000$



(a) $\hat{\mu}(r, 0, 0)$ for $L = 5000, 10000, \ldots, 300000$

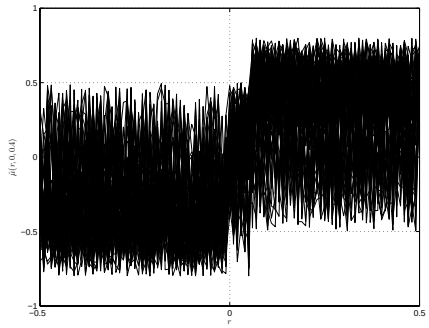


(b) $\hat{\mu}(r, 0, 0)$ for $L = 350000, 400000, \ldots, 550000$

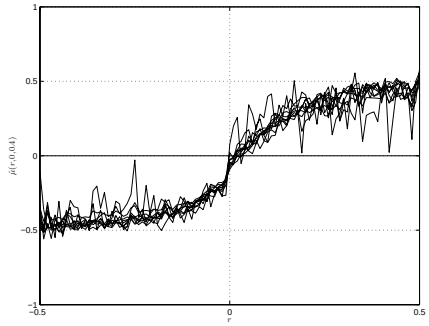


(c) $\hat{\mu}(r, 0, 0)$ for $L = 550000$

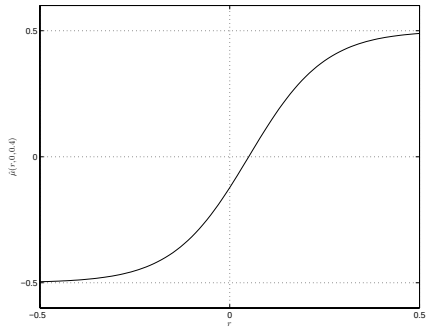
Figure 2.9: $\hat{\mu}(r, 0, 0)$ for $L = 5000, 10000, \ldots, 550000$



(a) $\hat{\mu}(r, 0, 0.4)$ for $L = 5000, 10000, \ldots, 300000$

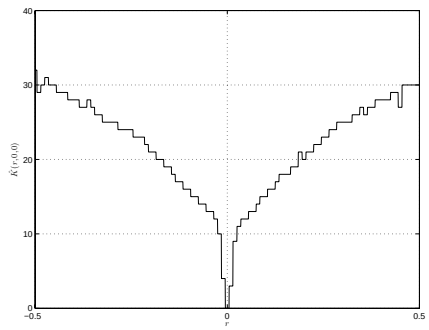


(b) $\hat{\mu}(r, 0, 0.4)$ for $L = 350000, 400000, \ldots, 550000$

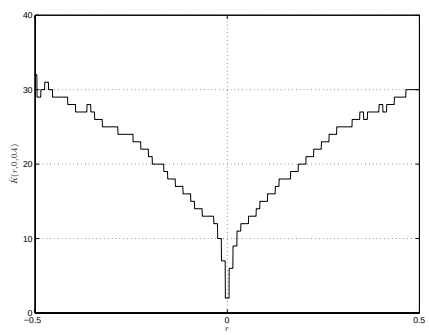


(c) $\hat{\mu}(r, 0, 0.4)$ for $L = 550000$

Figure 2.10: $\hat{\mu}(r, 0, 0.4)$ for $L = 5000, 10000, \ldots, 550000$

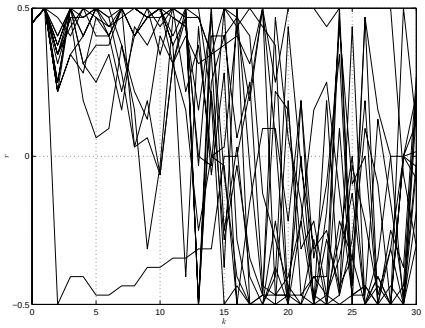


(a) $\hat{K}(r, 0, 0)$

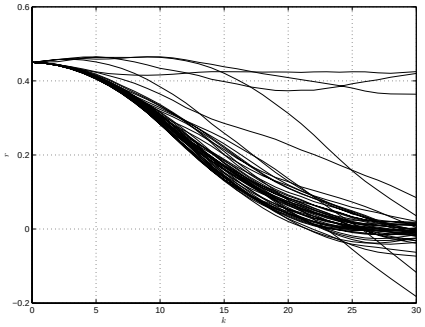


(b) $\hat{K}(r, 0, 0.4)$

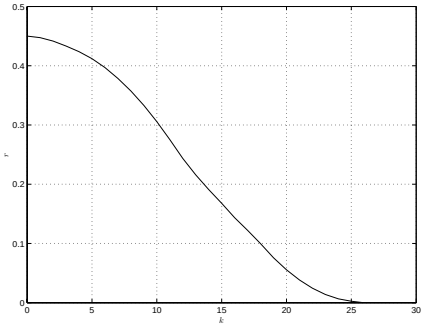
Figure 2.11: $\hat{K}(x, y, z)$ for $L = 550000$



(a) $L = 1000, 2000, \ldots, 30000$



(b) $L = 50000, 100000, \ldots, 550000$



(c) $L = 550000$

Figure 2.12: Trajectories starting from $r = 0.45$m.

cannot stay at the middle. When $L \geq 50000$, the ball will go to the middle of the beam and stay there for a while occasionally. When $L \geq 450000$, the trajectory can always reach the target. The ball can reach the middle faster when L is larger. When $L = 550000$, the trajectory is very smooth and it is a good approximation of the optimal trajectory.

■

2.7 Conclusions

In dynamic programming of discrete-time systems, one has to consider sequences of functions J_k^* and v_k^*. This is the cause of "curse of dimensionality". In this chapter, after introducing functions K_ϵ, V_ϵ^*, and μ_ϵ^*, we established our algorithm ADPDN(ϵ). Only one controller is trained in algorithm ADPDN(ϵ). Besides the critic $\hat{J}$ and the action $\hat{\mu}$, which are approximations of V_ϵ^* and μ_ϵ^*, respectively, a network $\hat{K}$ is used in the algorithm to approximate the function K_ϵ. K_ϵ will give the region where the states are controllable and will indicate how long the ϵ-optimal control sequence will be. When $K_\epsilon(x) = k$, we have $V_\epsilon^*(x) = J_k^*(x)$ and $\mu_\epsilon^*(x) = v_k^*(x)$. So V_ϵ^* and μ_ϵ^* are piecewise functions. But the most important fact is, V_ϵ^* (and μ_ϵ^*) is really a single function, not a sequence of functions. Thus, in our algorithm ADPDN(ϵ), instead of considering sequences of functions J_k^* and v_k^*, we will only deal with functions V_ϵ^* and μ_ϵ^*. Furthermore, μ_ϵ^* is always an admissible controller, even though v_∞^* may not be admissible. Hence, by using V_ϵ^* to replace functions J_k^*, we reduce the curse of dimensionality while we can control the system. By choosing a suitable value of ϵ, we can achieve adaptive dynamic programming with a bounded error, i.e., ϵ-adaptive dynamic programming with desired accuracy.

The functions $V_\epsilon^*(x)$ and $\mu_\epsilon^*(x)$ satisfy (2.4.11) and (2.4.12). When $K_\epsilon(x) > 1$, they have the same form as the HJB equation. However, when $K_\epsilon(x) = 1$, $V_\epsilon^*(x)$ and $\mu_\epsilon^*(x)$ satisfy (2.4.15) and (2.4.16), which will be solved from the problem $\min\{U(x,u)\colon F(x,u) = 0\}$.

At the beginning of our ADPDN(ϵ) algorithm, it may not find the optimal control strategy. It may not even drive the state along the right path to the target. But after a few rounds of running, it will know how to reach the target. When it has enough experience, it will find a good approximation to the optimal controller.

In the numerical experiments, algorithm ADPDN(ϵ) is applied to nonlinear unstable discrete-time systems with non-quadratic utility functions. The algorithm ADPDN(ϵ) works well on these systems. The ϵ-optimal controllers are found by algorithm ADPDN(ϵ).

3 THE ϵ-ADAPTIVE DYNAMIC PROGRAMMING FOR DISCRETE-TIME SYSTEM WITH DISCOUNT FACTOR IN ITS PERFORMANCE COST

In this Chapter, we study the ϵ-optimal control and ϵ-adaptive dynamic programming for the systems with discount factor in their performance cost functions. It is a generalization of the results of Chapter 2. All the results in this Chapter have been published in the papers "Discrete-time ϵ-adaptive dynamic programming algorithm using neural networks" ([13]) and. "Adaptive dynamic programming algorithm for discrete-time systems with ϵ-error bound and discount factor in the performance cost" ([14]).

3.1 Introduction

In this chapter, we study dynamic programming of discrete-time, time - invariant systems given by

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

where the system functions F is continuous and the origin is an equilibrium point of the system. There will be a discount factor γ, $0 < \gamma \leq 1$, in the performance cost of the system. The control target is the origin and we will try to drive the system to reach the target in finite but unspecified number of steps. We will generalize the results and algorithm of Chapter 2 to the case with discount factor $0 < \gamma \leq 1$.

In Section 3.2, we give some statements of the optimal control problem for discrete-time systems. We recall some details about dynamic programming which will be useful in this chapter. Then we show that the optimal

performance indexes J_k^* for different control steps k form a nonincreasing sequence. The limit of J_k^* as $k \to \infty$ equals the infimum J_∞^* of all performance indexes and satisfies Bellman's principle of optimality. In Section 3.3, we introduce the functions $K_\epsilon(x)$ and V_ϵ^*. V_ϵ^* is an approximation of J_∞^* with error less than ϵ such that $V_\epsilon^* \geq J_\infty^* \geq V_\epsilon^* - \epsilon$. The value of $K_\epsilon(x)$ gives the length of optimal control sequence starting from x with cost V_ϵ^*. The associated controller $\mu_\epsilon^*(x)$ is admissible and will be used as the only controller in all steps of an ϵ-optimal control sequence. Then we establish our algorithm of adaptive dynamic programming with an error bound according to ϵ or ϵ-*adaptive dynamic programming* for discrete-time systems using neural networks (ADPDN(ϵ)). Numerical experiments are given in Section 3.4 to evaluate the performance of the algorithm ADPDN(ϵ). We apply our algorithm ADPDN(ϵ) to a nonlinear unstable system $x_{k+1} = x_k + \sin(x_k + u_k)$ with utility function $U(x, u) = |x| + u^2$ and the discount factor $\gamma = 0.5$. Our algorithm performs well in the experiments. We conclude the present chapter in Section 3.5.

3.2 Problem Statement

In this chapter, we will study the discrete-time systems

$$x_{k+1} = F(x_k, u_k), \ k = 0, 1, 2, \ldots, \tag{3.2.1}$$

where $x_k \in \mathbb{R}^n$ is the state and $u_k \in \mathbb{R}^m$ is the control. The system function $F(x, u)$ is continuous and $F(0, 0) = 0$. Hence $x = 0$ is an equilibrium state of the system (3.2.1) under control $u = 0$.

Let x be an initial state and $\underline{u} = (u_0, u_1, ..., u_{N-1})$ be a sequence of controls. Then system (3.2.1) gives a trajectory starting from x: $x_0 = x$,

$x_1 = F(x_0, u_0)$, $x_2 = F(x_1, u_1)$, $\ldots$, $x_N = F(x_{N-1}, u_{N-1})$. We call the number of elements in control sequence $\underline{u}$ the length of $\underline{u}$ and denote it as $|\underline{u}|$. Then $|\underline{u}| = N$. For convenience, we also say that the length of the associated trajectory x_0, x_1, $\ldots$, x_N is N. We denote the final state of the trajectory as $x^{(e)}(x, \underline{u})$, i.e., $x^{(e)}(x, \underline{u}) = x_N$.

The control target is the origin $x = 0$. For an initial state $x_0 = x$, a control sequence $\underline{u}$ is called an admissible control sequence of x if $x_N(x, \underline{u}) = 0$. Let $\mathfrak{A}_x = \{\underline{u}\colon x_N(x, \underline{u}) = 0\}$ be the set of all admissible control sequences of x. Let $\mathfrak{A}_x^{(k)} = \{\underline{u}\colon x_N(x, \underline{u}) = 0, |\underline{u}| = k\}$ be the set of all admissible control sequences of x with length k. Then $\mathfrak{A}_x = \cup_{1 \le k < \infty} \mathfrak{A}_x^{(k)}$.

Suppose that system (3.2.1) has a utility function $U(x, u)$, $U(x, u) \ge 0$ for any (x, u). Assume that $U(x, u)$ is continuous and $U(0, 0) = 0$. Suppose γ is a discount factor, $0 < \gamma \le 1$. Then the performance cost with discount factor γ from a starting state x under the control sequence $\underline{u} = (u_0, u_1, ..., u_{N-1})$ is defined as

$$J(x, \underline{u}) = \sum_{k=0}^{N-1} \gamma^k U(x_k, u_k), \tag{3.2.2}$$

where $x_0 = x$, and $x_k = F(x_{k-1}, u_{k-1})$ for $k = 1, 2, ..., N$.

For any given initial state x, the aim of optimal control is to find an admissible control sequence $\underline{u} \in \mathfrak{A}_x$ to minimize the performance cost $J(x, \underline{u})$. For $k = 1, 2, ...$, let

$$J_k^*(x) = \min_{\underline{u} \in \mathfrak{A}_x^{(k)}} \{J(x, \underline{u})\}.$$

Then $J_k^*(x)$ is the optimal performance cost among all of the trajectories starting from x with length k and ending at the origin. By Bellman's principle

of optimality,

$$J_k^*(x) = \min_u\{U(x,u) + \gamma J_{k-1}^*(F(x,u))\}. \tag{3.2.3}$$

Now, define a control sequence

$$\underline{v}_k^*(x) = \arg\min_{\underline{u}}\{J(x,\underline{u})\}, \tag{3.2.4}$$

and define a single control vector

$$v_k^*(x) = \arg\min_u\{U(x,u) + \gamma J_{k-1}^*(F(x,u))\}. \tag{3.2.5}$$

Then we have

$$\begin{cases} J_k^*(x) = J(x,\underline{v}_k^*(x)), \\ J_k^*(x) = U(x,v_k^*(x)) + \gamma J_{k-1}^*(F(x,v_k^*(x))). \end{cases} \tag{3.2.6}$$

It is easy to see that Bellman's principle of optimality (3.2.4) is equivalent to

$$\underline{v}_k^*(x) = (v_k^*(x), \underline{v}_{k-1}^*(F(x,v_k^*(x))). \tag{3.2.7}$$

Consequently, we have

$$\underline{v}_k^*(x) = (v_k^*(x_0), v_{k-1}^*(x_1), ..., v_1^*(x_{k-1})), \tag{3.2.8}$$

where $x_0 = x$ and $x_{j+1} = F(x_j, v_{k-j}^*(x_j))$ for $j = 0, 1, ..., k-1$.

Dynamic programming is solved by determining $\underline{v}_k^*(x_0)$. The first step is to determine the function $v_1^*(\cdot)$. For any given state vector x, $v_1^*(x)$ is the optimal solution of the following problem

$$\min_u U(x,u) \ \text{ subject to } F(x,u) = 0. \tag{3.2.9}$$

The associated cost is

$$J_1^*(x) = U(x, v_1^*(x)).$$

After the functions $v_1^*(\cdot)$ and $J_1^*(\cdot)$ have been determined, we can determine $v_j^*(\cdot)$ and $J_j^*(\cdot)$ for $j = 2, 3, ..., k$ by applying formulas (3.2.4) and (3.2.6). Then we can determine $\underline{v}_k^*(x_0) = (v_k^*(x_0), v_{k-1}^*(x_1), ..., v_1^*(x_{k-1}))$. Applying $v_k^*(\cdot)$ to x_0 gets $v_0^* = v_k^*(x_0)$. Applying v_0^* to the system (3.2.1) gets $x_1 = F(x_0, u_0^*)$. Then applying $v_{k-1}^*(\cdot)$ to x_1 gets $v_1^* = v_{k-1}^*(x_1)$, and applying v_1^* to the system gets $x_2 = F(x_1, u_1^*)$. Repeat this process, we get the optimal control sequence $\underline{v}_k^*(x_0) = \{u_0^*, u_1^*, ..., u_{N-1}^*\}$.

But then we have to face to the problem of "curse of dimensionality". We have to record all the functions $v_j^*(\cdot)$, $j = 1, 2, ..., k$, even only one control trajectory is wanted. In most real world applications, big k is necessary. But for big k, the storage required is huge. There are also lots of calculation needed to find all the $v_j^*(\cdot)$'s.

One way to avoid the problem of curse of dimensionality is try to find

$$J_\infty^*(x) = \min_{\underline{u}}\{J(x, \underline{u})\}$$

and then use

$$v_\infty^*(x) = \arg\min_{\underline{u}}\{J(x, \underline{u})\}$$

as the unique controller. But in most cases, the trajectories under the control of v_∞^* can not reach the target in finite steps. They are only asymptotic convergence or even not convergence. However, the following fact is easy to observe.

Fact 3.1 *Let $\epsilon > 0$ be an arbitrary positive number. For each $k = 1, 2, \cdots$, there is a region $\mathcal{T}_k^{(\epsilon)}$ such that*

$$J_\infty^*(x) \le J_k^*(x) \le J_\infty^*(x) + \epsilon \text{ if } x \in \mathcal{T}_k^{(\epsilon)}. \tag{3.2.10}$$

The Fact 3.1 means that if a state x is inside the region $\mathcal{T}_k^{(\epsilon)}$, then the optimal trajectory which reach the target in k steps will have a performance cost close to the optimal cost with an error bound ϵ. In the following section, we will develop an ϵ-*adaptive dynamic programming algorithm* based on the Fact 3.1.

3.3 The ϵ-Optimal Cost V_ϵ^* and the ϵ-Adaptive Dynamic Programming Algorithm

Define

$$V_\epsilon^*(x) = J_k^*(x), \text{ if } x \in \mathcal{T}_k^{(\epsilon)} \setminus \mathcal{T}_{k-1}^{(\epsilon)}, \tag{3.3.1}$$

and

$$\mu_\epsilon^*(x) = v_k^*(x), \text{ if } x \in \mathcal{T}_k^{(\epsilon)} \setminus \mathcal{T}_{k-1}^{(\epsilon)}, \tag{3.3.2}$$

where $k = 1, 2, \ldots$. Then V_ϵ^* and μ_ϵ^* are piecewise functions. For a state $x \in \mathcal{T}_k^{(\epsilon)} \setminus \mathcal{T}_{k-1}^{(\epsilon)}$, if we apply controller $\mu_\epsilon^*(\cdot)$ on it, then in fact it is $v_k^*(x)$ applied. We denote the next state by x_1, i.e. $x_1 = F(x, v_k^*(x))$. According to the discussion on the previous section (see (3.2.8)), $\underline{v}_k^*(x) = (v_k^*(x_0), v_{k-1}^*(x_1), ..., v_1^*(x_{k-1}))$ is an optimal control with length k for the initial state $x_0 = x$. Thus, $(v_{k-1}^*(x_1), ..., v_1^*(x_{k-1}))$ must be an optimal control with length $k-1$ for the initial state x_1. Hence $x_1 \in \mathcal{T}_{k-1}^{(\epsilon)}$. Repeat this process, using controller $\mu_\epsilon^*(\cdot)$ to control the state x_1 to get x_2, and then x_3, ..., and so on. Finally, we know that the controller $\mu_\epsilon^*(\cdot)$ can always drive a state towards the target. Furthermore, we can rewrite the formulas (3.2.3) and (3.2.5) as

$$V_\epsilon^*(x) = \min_u \{U(x,u) + \gamma V_\epsilon^*(F(x,u))\}, \tag{3.3.3}$$

$$\mu_\epsilon^*(x) = \arg\min_u \{U(x,u) + \gamma V_\epsilon^*(F(x,u))\}. \tag{3.3.4}$$

(3.3.4) gives the way to obtain μ_ϵ^* from V_ϵ^*. In particular, when $x \in \mathcal{T}_1^{(\epsilon)}$, $\mu_\epsilon^*(x) = v_1^*(x)$ is the optimal solution of the problem (3.2.9) and $V_\epsilon^*(x) = J_1^*(x) = U(x, v_1^*(x))$.

Now we will introduce a numerical algorithm of ϵ-approximative dynamic programming for discrete-time systems. Two neural networks $\hat{J}$ and $\hat{\mu}$ will be used to approximate the ϵ-optimal cost V_ϵ^* and ϵ-optimal controller μ_ϵ^*, respectively. In other words, $\hat{J}$ will approximate $J_k^*(x)$ and $\hat{\mu}$ will approximate $v_k^*(x)$ if $x \in \mathcal{T}_k^{(\epsilon)}$. To indicate whether $x \in \mathcal{T}_k^{(\epsilon)}$, we introduce the third neural network $\hat{K}$. The value of $\hat{K}(x)$ is k if and only if $x \in \mathcal{T}_k^{(\epsilon)}$.

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

The following is the algorithm ADPDN(ϵ), *ϵ-adaptive dynamic programming for discrete-time systems using neural networks.* Roughly speaking, the idea of this algorithm is that for an initial state close to the target, a short control sequence will be applied, and for an initial state far away from the target, a long control sequence will be used.

Algorithm ADPDN(ϵ) (ϵ-adaptive dynamic programming for discrete time systems using neural networks)

B00 Initialization. Solve problem (3.2.9) to determine the functions $J_1^*(x)$

and $v_1^*(x)$. Then let $\hat{J}(x) = J_1^*(x)$, $\hat{\mu}(x) = v_1^*(x)$, and

$$\hat{K}(x) = \begin{cases} 0, & \text{if } x = 0, \\ 1, & \text{otherwise.} \end{cases}$$

B01 Choose randomly an array of initial states $x_0 = (x_0^{(1)}, x_0^{(2)}, \ldots, x_0^{(r)})$ from the entire state space.

B02 For $k = 1, 2, \ldots, \text{maxK}$, do B03–B06.

B03 Run the system from the array of initial states x_0 under the control of $\hat{\mu}$. Record the resultant array of state $x_1 = (x_1^{(1)},\ x_1^{(2)},\ \ldots,\ x_1^{(r)})$. Calculate the associated costs $C_1^{(i)} = U(x_0^{(i)}, \hat{\mu}(x_0^{(i)})) + \gamma \hat{J}(x_1^{(i)})$. Record each $k_0^{(i)} = \hat{K}(x_0^{(i)})$ and $k_1^{(i)} = \hat{K}(x_1^{(i)})$.

B04 Update $\hat{J}$ and $\hat{K}$. For each $i = 1, \ldots, r$,

if $J_1^*(x_0^{(i)}) \leq C_1^{(i)} + \epsilon$ then

$$\begin{cases} \hat{J}(x_0^{(i)}) = J_1^*(x_0^{(i)}), \\ \hat{K}(x_0^{(i)}) = \begin{cases} 0, & \text{if } x_0^{(i)} = 0, \\ 1, & \text{if } x_0^{(i)} \neq 0, \end{cases} \end{cases}$$

else if $k_1^{(i)} \leq \max(k_0^{(i)} - 1, 1)$ then

$$\begin{cases} \hat{J}(x_0^{(i)}) = C_1^{(i)}, \\ \hat{K}(x_0^{(i)}) = k_1^{(i)} + 1, \end{cases}$$

endif

B05 Update $\hat{\mu}$. For each $i = 1, \ldots, r$,

if $J_1^*(x_0^{(i)}) \leq \hat{J}(x_0^{(i)}) + \epsilon$ then

$$\hat{\mu}(x_0^{(i)}) = v_1^*(x_0^{(i)}),$$

else

$$\hat{\mu}(x_0^{(i)}) = \arg\min_u \{U(x_0^{(i)}, u) + \gamma \hat{J}(F(x_0^{(i)}, u)); \\ \hat{K}(F(x_0^{(i)}, u)) \leq \max(\hat{K}(x_0^{(i)}) - 1, 1)\},$$

endif

B06 Let $x_0 = x_1$.

B07 Go to B01 until the process converges.

From lines B01 to B07, there are two levels of loops. The outer loop, starting on line B01 and ending on line B07, will give initial states x_0 and then go into the inner loop. The inner loop run the system maxK steps. As we have mentioned before, for an initial state x, if one cannot control the system to reach the target within maxK steps starting from x, then we will consider x to be uncontrollable.

The body of the inner loop, from line B03 to B05, will run the system from x_0 under the control of $\hat{\mu}$ and update the networks according to the results. There are two different ways to adjust the networks. When $J_1^*(x_0^{(i)}) \leq C_1^{(i)} + \epsilon$, one will consider $x_0^{(i)} \in \mathcal{T}_1^{(\epsilon)}$ and $V_\epsilon^*(x_0(i)) = J_1^*(x_0(i))$. Otherwise, the condition $k_1^{(i)} \leq \max(k_0^{(i)} - 1, 1)$ implies that $x_1^{(i)}$ is one step closer to the target $x = 0$. Then $\hat{J}$ and $\hat{\mu}$ will satisfy the formulas (3.3.3) and (3.3.4), and the system will go to the target one more steps from x_0 than from x_1.

The initial state x_0 is chosen randomly on line B01. Suppose that the associated random probability density function is non-vanished everywhere. Then we can assume that all the states will be explored. So we know that the resulting networks tend to satisfy the formulas (3.3.3) and (3.3.4) for all state vector x. The limit of $\hat{J}$ and $\hat{\mu}$ will approximate the optimal ones V_ϵ^* and μ_ϵ^*, respectively.

The network $\hat{K}$ is very important in algorithm ADPDN(ϵ). It is used to check whether a state $x_0^{(i)}$ belongs to $\mathcal{T}_1^{(\epsilon)}$ or $\mathcal{T}_k^{(\epsilon)}$ for $k > 1$. It will also determine when a state can be controlled to reach the final target by finite control step.

The final outputs of the algorithm ADPDN(ϵ) are three neural networks. $\hat{J}$ and $\hat{\mu}$ give the approximation to ϵ-optimal cost V_ϵ^* and the associated ϵ-optimal controller μ_ϵ^*, respectively. Meanwhile, $\hat{K}$ indicates the region where optimal control exists and how long it will go for controlling x to reach the target. For a state x, if $\hat{K}(x) = \text{maxK}$ then we can consider x to be uncontrollable. If $\hat{K}(x) < \text{maxK}$, then $\hat{K}(x)$ is an estimate of the number of control steps to drive x to the target. Before applying the controller $\hat{\mu}$, use $\hat{K}$ to check whether the initial state is controllable. Then use $\hat{\mu}$ as controller to control it.

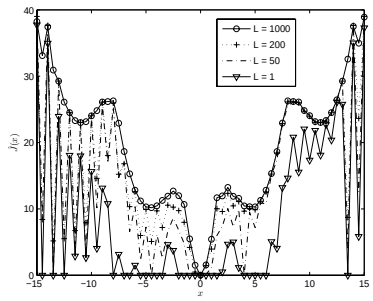
3.4 Numerical Experiments

To evaluate the performance of the algorithm ADPDN(ϵ), we select a nonlinear unstable system with non-quadric utility for numerical experiment. We consider the system

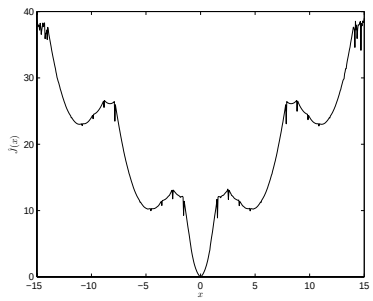
$$x_{k+1} = F(x_k, u_k) \triangleq x_k + \sin(x_k + u_k), \tag{3.4.1}$$

where $x_k, u_k \in \mathbb{R}$, and $k = 0, 1, 2, \ldots$. The utility function is $U(x, u) = |x| + u^2$. The discount factor is $\gamma = 0.5$. Since $F(0, 0) = 0$, 0 is an equilibrium state of system (3.4.1). But (3.4.1) is unstable at $x = 0$, since $(\partial F/\partial x)(0, 0) = 2 > 1$.

We choose the value of error bound to be $\epsilon = 0.1$. The region of states considered here is $|x| \leq 15$ and the number of control steps is restricted to



(a) $L = 1, 50, 200, 1000$

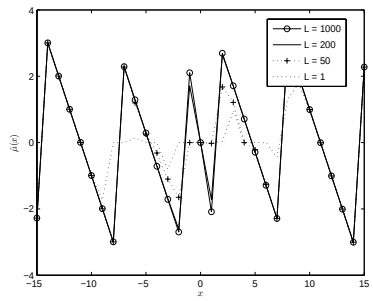


(b) $L = 1500$, ..., 2000

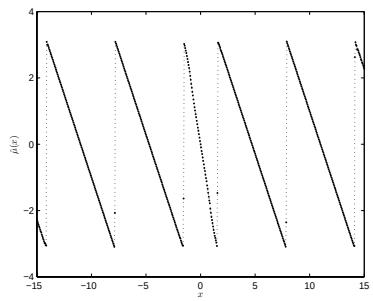
Figure 3.1: $\hat{J}(x)$, $x \in [-15, 15]$

20. The size of the set of initial states x_0 (line B04 of the algorithm) will be 5 each time, i.e., at the beginning of the each inner loop iteration, we choose 50 initial states randomly.

The neural networks $\hat{K}$, $\hat{J}$ and $\hat{\mu}$ will be trained from the observation while the algorithm is running. We expect that they will have the ability to keep the old feature when they are trained by some new data. For this purpose, radial basis function neural networks are adopted. Each network will be a linear combination $\sum_i w_i \phi(\|x - c_i\|/\rho_i)$, where w_i are weights of neurons, $\phi(r) = \exp(-r^2)$ is the basis function, c_i is the center and ρ_i is the



(a) $L = 1, 50, 200, 1000$



(b) $L = 1500$, ..., 2000

Figure 3.2: $\hat{\mu}(x)$, $x \in [-15, 15]$

width of each neuron.

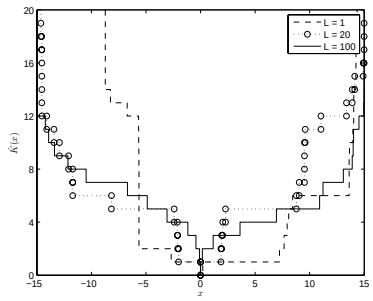
Since $F(x, u) = x + \sin(x+u)$, the implicit functions according to $F(x, u) = 0$ are

$$u(x) = f^{(i)}(x) = 2i\pi + \sin^{-1}(-x) - x,$$

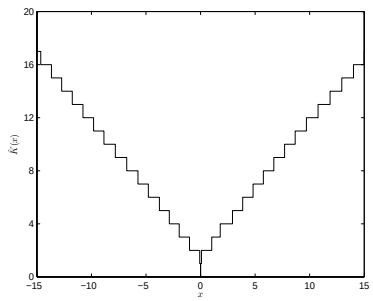
and

$$u(x) = g^{(i)}(x) = (2i+1)\pi - \sin^{-1}(-x) - x,$$

where $-1 \leq x \leq 1$, and $i = 0, \pm 1, \pm 2, \ldots$. It is easy to see that $\mathcal{T}_1 = [-1, 1]$



(a) $L = 1, 20, 200$



(b) $L = 2000$

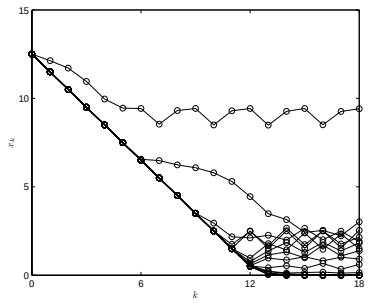
Figure 3.3: $\hat{K}$, $x \in [-15, 15]$

and

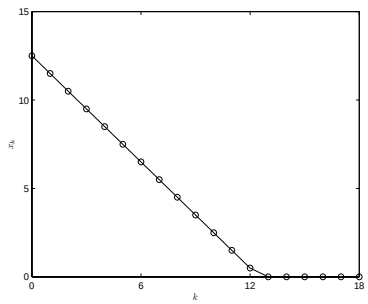
$$\begin{cases} J_1^*(x) = \min\{U(x,u) \colon F(x,u) = 0\} \\ \qquad\quad = |x| + (-x + \sin^{-1}(-x))^2, \\ v_1^*(x) = -x + \sin^{-1}(-x), \end{cases}$$

for $x \in \mathcal{T}_1$. Since the value of $\sin(x+u)$ is between -1 an 1, one can easily to find that $\mathcal{T}_k = [-k, k]$ for any $k = 1, 2, \ldots$.

On line B05 of algorithm ADPDN(ϵ), we need to find $\arg\min_u[U(x,u) + \gamma\hat{J}(w)]$, where $w = \hat{F}(x,u)$. Now $U(x,u) = |x| + u^2$, $F(x,u) = x + \sin(x+u)$.



(a) $L = 10, 20, ..., 200$



(b) $L = 1500, ..., 2000$

Figure 3.4: Trajectories starting from $x_0 = 12.5$

So we will find the minimum

$$\min_u \{|x| + u^2 + \gamma \hat{J}(w)\}, \tag{3.4.2}$$

where $w = x + \sin(x + u)$. Since sin is a period function with periodic 2π, we know that the minimum of (3.4.2) will be reached only when $u \in [-\pi.\pi]$.

We perform our ADPDN(ϵ) algorithm and save $\hat{J}$, $\hat{K}$ and $\hat{\mu}$ at the end of each outer loop iteration, i.e., save them in line B07 of the algorithm. Denote the number of iterations of the outer loop by L. Then we obtain sequences of $\hat{J}$, $\hat{K}$ and $\hat{\mu}$ for $L = 1, 2, \cdots$. They will approximate $V_\epsilon^*(x)$, $K_{(\epsilon)}(x)$ and

$\mu_\epsilon^*(x)$, respectively.

Figs. 3.1 and 3.2 are graphs of $\hat{J}(x)$ and $\hat{\mu}(x)$, respectively. In Fig. 3.1(a), $L = 1, 100, 200, 500, 1000$. In Fig. 3.2(a), $L = 1, 50, 200, 1000$. Figs. 3.1(b) and 3.2(b) are the graphs of $\hat{J}(x)$ and $\hat{\mu}(x)$ when $1500 \leq L \leq 2000$. They are almost same for different L. They are approximations of $V_\epsilon^*(x)$ and $\mu_\epsilon^*(x)$, respectively. In fact, if we set the convergent threshold to be $\|\hat{J}_{old} - \hat{J}_{new}\|_2 < 0.001$, the algorithm will stop when $L = 1437$. Obviously, both $V_\epsilon^*(x)$ and $\mu_\epsilon^*(x)$ are piecewise functions.

Fig. 3.3 is the graph $\hat{K}$. In Fig. 3.3(a), $L = 1, 20, 200$. Fig. 3.3(b) is the final result of $\hat{K}$ when $L = 2000$, which can be considered as $K_{(\epsilon)}$. When $|x|$ is small, $\hat{K}(x)$ is small, i.e., less steps of controls are needed to drive x to the target, and when $|x|$ is big, more steps are needed to control the system to the target. Fig. 3.4 is the trajectory starting from initial state $x_0 = 12.5$ under the control law $\hat{\mu}$. In Fig. 3.4(a), $L = 10, 20, ..., 200$. When $L \leq 70$, the trajectory cannot reach the target in 10 steps. When $L \geq 80$, the trajectory can reach the target in 10 steps. Fig. 3.4(b) are the trajectory starting from $x_0 = 12.5$ for $1500 \leq L \leq 2000$. They are all so close that they seem to be just one trajectory. Hence, when $L \geq 1500$, the trajectory will arrive the target in 6 steps and will be quite close to the optimal trajectory.

3.5 Conclusions

In dynamic programming of discrete-time systems, one has to consider sequences of functions J_k^* and v_k^*. This is the reason of "curse of dimensionality". In this chapter, after introducing functions V_ϵ^* and μ_ϵ^*, we establish our algorithm ADPDN(ϵ). Only one controller is trained in algorithm

ADPDN(ϵ). Besides the critic $\hat{J}$ and action $\hat{\mu}$, a network $\hat{K}$ is used in the algorithm. $\hat{K}$ will give the region where the states are controllable and will indicate how long the ϵ-optimal control sequence will be. When $\hat{K}(x) = k$, we have $V_\epsilon^*(x) = J_k^*(x)$ and $\mu_\epsilon^*(x) = v_k^*(x)$. So V_ϵ^* and μ_ϵ^* are piecewise functions. But the most important fact is, V_ϵ^* (or μ_ϵ^*) is really one function, not a sequence of functions. Thus, in our algorithm ADPDN(ϵ), instead of considering the sequences of functions J_k^* and v_k^*, we will only deal with the functions V_ϵ^* and μ_ϵ^*. Furthermore, μ_ϵ^* is an admissible controller, even though v_∞^* may be not admissible. Hence, by using V_ϵ^* to replace functions J_k^*, we reduce the curse of dimensionality while we can control the system. And by choosing suitable value of ϵ, we can achieve ϵ-approximative dynamic programming with the desired accuracy.

The functions $\mu_\epsilon^*(x)$ and $V_\epsilon^*(x)$ satisfy the equations (3.3.3) and (3.3.4). When $\hat{K}(x) > 1$, they have the same form with HJB equation. However, when $\hat{K}(x) = 1$, $\mu_\epsilon^*(x)$ and $V_\epsilon^*(x)$ satisfy (3.2.9), will be solved from the problem $\min\{U(x,u) \colon F(x,u) = 0\}$.

In the numerical experiments, algorithm ADPDN(ϵ) is applied to a nonlinear unstable discrete-time system with non-quadratic utility, and the associated discount factor in the performance cost is $\gamma = \frac{1}{2}$. The algorithm ADPDN(ϵ) works well on this system. The ϵ-optimal controller is found very successfully by algorithm ADPDN(ϵ).

4 DYNAMIC PROGRAMMING FOR DESCRETE-TIME SYSTEM WITH QUADRATIC UTILITY

In Chapter 2 and Chapter 3, we introduced our ϵ-adaptive dynamic programming for discrete-time systems and designed the algorithm ADPDN(ϵ). In ADPDN(ϵ), Since only one performance cost function is used in ϵ-adaptive dynamic programming, ϵ-adaptive dynamic programming is helpful to overcome the curse of dimensionality. However, the time expense of ADPDN(ϵ) is still big. In this Chapter, we will introduce an iterative algorithm for dynamic programming which will run faster than ADPDN(ϵ), providing that the utility function of the system is a positive definite quadratic form. All the results in this Chapter have been written in the papers "Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using globalized dual heuristic programming" ([23]) and "Optimal control of unknown nonaffine nonlinear discrete-time systems based on adaptive dynamic programming" ([39]).

In this chapter we generalize our ϵ-adaptive dynamic programming to discrete-time systems with discount factor $0 < \gamma \leq 1$ in the performance cost. Then, an iterative adaptive dynamic programming (ADP) algorithm is introduced to solve the ϵ-adaptive optimal control problem with convergence analysis. As a condition to realize the iterative algorithm, the utility function is restricted to be a quadratic function in this chapter. The implementation of the iterative algorithm via globalized dual heuristic programming (GDHP) technique is presented by using three neural networks, which will approximate at each iteration the cost function, the control law, and the unknown nonlinear system, respectively

4.1 Problem Statement

Firstly, we give a brief description of our problem. The statements in this section are similar with those in section 2.2 but we have to consider the discount factor γ in the associated equations and formulas here.

Consider the system

$$x_{k+1} = F(x_k, u_k), \ k = 0, 1, 2, \ldots, \tag{4.1.1}$$

where $x_k \in \mathbb{R}^n$ is the state and $u_k \in \mathbb{R}^m$ is the control. The system function $F(x, u)$ is continuous and $F(0, 0) = 0$. Hence $x = 0$ is an equilibrium state of the system (4.1.1) under control $u = 0$.

Definition 4.1 *A nonlinear dynamical system is said to be stabilizable on an open set $\Omega \subset \mathbb{R}^n$ which contained 0 as an interior point, if for any initial condition $x_0 \in \Omega$, there exists a control sequence $u_0, u_1, \cdots$, $u_i \in \mathbb{R}^m$, $i = 0, 1, \cdots$, such that the state $x_k \to 0$ as $k \to \infty$.*

It is desired to find the control law $u(x)$ which minimizes the infinite horizon cost function given by

$$J(x_k) = \sum_{p=k}^{\infty} \gamma^{p-k} U(x_p, u_p), \tag{4.1.2}$$

whereγ is the discount factor with $0 < \gamma \leq 1$, U is the utility function given by

$$U(x, u) = x^T Q x + u^T R u, \tag{4.1.3}$$

and Q and R are positive definite matrices.

For optimal control problems, the designed feedback control must not only stabilize the system on Ω, but also guarantee that (4.1.2) has finite value, i.e., the control must be admissible.

Definition 4.2 *A control law $u(x)$ of system (4.1.1) is said to be admissible with respect to performance cost (4.1.2) on Ω if $u(x)$ is continuous on Ω, $u(0) = 0$, u stabilizes (4.1.1) on Ω, and for any $x_0 \in \Omega$, $J(x_0)$ is finite.*

Note that equation (4.1.2) can be written as

$$\begin{aligned} J(x_k) &= x_k^T Q x_k + u_k^T R u_k + \gamma \sum_{p=k+1}^{\infty} \gamma^{p-k-1} U(x_p, u_p) \\ &= x_k^T Q x_k + u_k^T R u_k + \gamma J(x_{k+1}). \end{aligned} \tag{4.1.4}$$

According to Bellman's optimality principle, the optimal cost function $J^*(x_k)$ satisfies the HJB equation

$$J^*(x_k) = \min_{u_k} \left\{ U(x_k, u_k) + \gamma J^*(x_{k+1}) \right\}. \tag{4.1.5}$$

Besides, the optimal control law u^* can be derived from the gradient of the right-hand side of (4.1.5) with respect to u_k as

$$\frac{\partial (x_k^T Q x_k + u_k^T R u_k)}{\partial u_k} + \gamma \left(\frac{\partial x_{k+1}}{\partial u_k} \right)^T \frac{\partial J^*(x_{k+1})}{\partial x_{k+1}} = 0. \tag{4.1.6}$$

Then, we have

$$u^*(x_k) = -\frac{\gamma}{2} R^{-1} \left(\frac{\partial x_{k+1}}{\partial u_k} \right)^T \frac{\partial J^*(x_{k+1})}{\partial x_{k+1}}. \tag{4.1.7}$$

By substituting (4.1.7) into (4.1.5), the HJB equation becomes

$$\begin{aligned} J^*(x_k) = x_k^T Q x_k &+ \frac{\gamma^2}{4} \left(\frac{\partial J^*(x_{k+1})}{\partial x_{k+1}} \right)^T \frac{\partial x_{k+1}}{\partial u_k} R^{-1} \\ &\times \left(\frac{\partial x_{k+1}}{\partial u_k} \right)^T \frac{\partial J^*(x_{k+1})}{\partial x_{k+1}} + \gamma J^*(x_{k+1}), \end{aligned} \tag{4.1.8}$$

where $J^*(x_k)$ is the optimal cost function corresponding to the optimal control law $u^*(x_k)$.

Noticed that when we dealing with the linear quadratic regulator problems, the HJB equation reduces to the Riccati equation which can be efficiently solved. For the general nonlinear case, however, it is considerably difficult to cope with the HJB equation directly. However, by the assumption that the utility function $U(x, u)$ is a quadratic form, we can develop an iterative ADP algorithm to solve it in next section, basing on Bellman's optimality principle.

4.2 Neuro-Optimal Control Based on Iterative Adaptive Dynamic Programming Algorithm

Now we begin the derivation of our iterative ADP algorithm. Firstly, we start with the initial cost function $V_0(\cdot) = 0$ and obtain the law of single control vector $v_0(x_k)$ as follows:

$$v_0(x_k) = \arg\min_{u_k}\{x_k^T Q x_k + u_k^T R u_k + \gamma V_0(x_{k+1})\}. \tag{4.2.1}$$

Then we update the cost function as

$$\begin{aligned} V_1(x_k) &= \min_{u_k}\{x_k^T Q x_k + u_k^T R u_k + \gamma V_0(F(x_k, u_k)))\} \\ &= x_k^T Q x_k + v_0^T(x_k) R v_0(x_k). \end{aligned} \tag{4.2.2}$$

Next, for $i = 1, 2, \cdots$, the algorithm iterates according to

$$\begin{aligned} v_i(x_k) &= \arg\min_{u_k}\{x_k^T Q x_k + u_k^T R u_k + \gamma V_i(x_{k+1})\} \\ &= -\frac{\gamma}{2} R^{-1} \left(\frac{\partial x_{k+1}}{\partial u_k}\right)^T \frac{\partial V_i(x_{k+1})}{\partial x_{k+1}}. \end{aligned} \tag{4.2.3}$$

and

$$\begin{aligned} V_{i+1}(x_k) &= \min_{u_k}\{x_k^T Q x_k + u_k^T R u_k + \gamma V_i(x_{k+1})\} \\ &= x_k^T Q x_k + v_i^T(x_k) R v_i(x_k) + \gamma V_i(F(x_k, v_i(x_k))). \end{aligned} \tag{4.2.4}$$

In the above recurrent iteration, i is the iteration index, while k is the time index. The cost function and control law are updated until they converge to the optimal ones. In the following part, we will present the convergence proof of the iteration between (4.2.3) and (4.2.4) with the cost function $V_i \to J^*$ and the control law $v_i \to u^*$ as $i \to \infty$.

Lemma 4.1 *Let $\{\mu_i\}$ be an arbitrary sequence of control laws and $\{v_i\}$ be the control laws as in (4.2.3). Define V_i as in (4.2.4) and Λ_i be*

$$\Lambda_{i+1}(x_k) = x_k^T Q x_k + \mu_i^T(x_k) R ??_i(x_k) + \gamma \Lambda_i(F(x_k, \mu_i(x_k))). \tag{4.2.5}$$

If $V_0(\cdot) = \Lambda_0(\cdot) = 0$, then $V_{i+1}(x) \leq \Lambda_{i+1}(x)$, $\forall i$.

Proof. It can be derived by noticing that V_{i+1} is the result of minimizing the right-hand side of (4.2.4) with respect to the control input u_k, while Λ_{i+1} is a result of an arbitrary control input. ■

Lemma 4.2 *Let the sequence $\{V_i\}$ be defined as in (4.2.4). If the system is controllable, there is an upper bound Y such that $0 \leq V_i(x_k) \leq Y$, $\forall i$.*

Proof. Let $\eta(x_k)$ be any admissible control input, and let $V_0(\cdot) = Z_0(\cdot) = 0$, where V_i is updated as in (4.2.4) and Z_i is updated by

$$Z_{i+1}(x_k) = x_k^T Q x_k + \eta^T(x_k) R \eta(x_k) + \gamma Z_i(x_{k+1}). \tag{4.2.6}$$

Noticing the difference between Z_i and Z_{i+1} is

$$\begin{aligned} Z_{i+1}(x_k) - Z_i(x_k) &= \gamma(Z_i(x_{k+1}) - Z_{i-1}(x_{k+1})) \\ &= \gamma^2(Z_{i-1}(x_{k+2}) - Z_{i-2}(x_{k+2})) \\ &= \gamma^3(Z_{i-2}(x_{k+3}) - Z_{i-3}(x_{k+3})) \\ &\quad \vdots \\ &= \gamma^i(Z_1(x_{k+i}) - Z_0(x_{k+i})) \\ &= \gamma^i Z_1(x_{k+i}), \end{aligned} \tag{4.2.7}$$

we can obtain

$$\begin{aligned} Z_{i+1}(x_k) &= \gamma^i Z_1(x_{k+i}) + Z_i(x_k) \\ &= \gamma^i Z_1(x_{k+i}) + \gamma^{i-1} Z_1(x_{k+i-1}) + Z_{i-1}(x_k) \\ &\quad \cdots \\ &= \gamma^i Z_1(x_{k+i}) + \gamma^{i-1} Z_1(x_{k+i-1}) + \gamma^{i-2} Z_1(x_{k+i-2}) \\ &\qquad + \cdots + \gamma Z_1(x_{k+1}) + Z_1(x_k), \end{aligned} \tag{4.2.8}$$

and therefore,

$$\begin{aligned} Z_{i+1}(x_k) &= \sum_{j=0}^{i} \gamma^j Z_1(x_{k+j}) \\ &= \sum_{j=0}^{i} \gamma^j (x_{k+j}^T Q x_{k+j} + \eta^T(x_{k+j}) R \eta(x_{k+j})) \\ &\leq \sum_{j=0}^{\infty} \gamma^j (x_{k+j}^T Q x_{k+j} + \eta^T(x_{k+j}) R \eta(x_{k+j})). \end{aligned} \tag{4.2.9}$$

Since that $\eta(x_k)$ is an admissible control input, i.e., $x_k \to 0$ as $k \to \infty$, we have

$$Z_{i+1}(x_k) \leq \sum_{j=0}^{\infty} \gamma^j Z_1(x_{k+j}) \leq Y, \ \forall i. \tag{4.2.10}$$

By using Lemma 4.1, we get

$$V_{i+1}(x_k) \leq Z_{i+1}(x_k) \leq Y, \ \forall i, \tag{4.2.11}$$

and so complete the proof. ■

Basing on Lemma 4.1 and Lemma 4.2, we now present the convergence proof of the cost function sequence.

Theorem 4.1 *Define the sequence $\{V_i\}$ as in (4.2.4) with $V_0(\cdot) = 0$, and the control law sequence $\{v_i\}$ as in (4.2.3). Then, we can conclude that $\{V_i\}$ is a nondecreasing sequence satisfying $V_{i+1} \geq V_i$, $\forall i$.*

Proof. Define a new sequence as

$$\Phi_{i+1}(x_k) = x_k^T Q x_k + v_{i+1}^T(x_k) R v_{i+1}(x_k) + \gamma \Phi_i(x_{k+1}) \tag{4.2.12}$$

with $\Phi_0(\cdot) = V_0(\cdot) = 0$. Now, we will show that $\Phi_i(x_k) \leq V_{i+1}(x_k)$.

First, we prove that it holds for $i = 0$. Since

$$V_1(x_k) - \Phi_0(x_k) = x_k^T Q x_k + v_0^T(x_k) R v_0(x_k) \geq 0, \tag{4.2.13}$$

we have

$$V_1(x_k) \geq \Phi_0(x_k). \tag{4.2.14}$$

Second, we assume that it holds for $i-1$, i.e., $V_i(x_k) \geq \Phi_{i-1}(x_k)$, $\forall x_k$. Then, for i, from (4.2.4) and (4.2.12), we get

$$V_{i+1}(x_k) - \Phi_i(x_k) = \gamma \left(V_i(x_{k+1}) - \Phi_{i-1}(x_{k+1}) \right) \geq 0, \tag{4.2.15}$$

i.e.,

$$V_{i+1}(x_k) \geq \Phi_i(x_k). \tag{4.2.16}$$

Thus, (4.2.16) is true for any i by mathematical induction.

Furthermore, according to Lemma 4.1, we know that $V_i(x_k) \leq \Phi_i(x_k)$. Combining with (4.2.16), we have

$$V_{i+1}(x_k) \geq \Phi_i(x_k) \geq V_i(x_k) \tag{4.2.17}$$

and complete the proof. ■

According to Lemma 4.2 and Theorem 4.1, we can obtain that $\{V_i\}$ is a monotonically nondecreasing sequence with an upper bound, and therefore, its limit exists. Here, we define it as $lim_{i\to 1} V_i(x_k) = V_\infty(x_k)$ and present the following theorem.

Theorem 4.2 *Let the cost function sequence $\{V_i\}$ be defined as in (4.2.4). Then, its limit satisfies*

$$V_\infty(x_k) = \min_{u_k}\{x_k^T Q x_k + u_k^T R u_k + \gamma V_\infty(x_{k+1})\}. \tag{4.2.18}$$

Proof. For any u_k and i, according to (4.2.4), we can derive

$$V_i(x_k) \leq x_k^T Q x_k + u_k^T R u_k + \gamma V_{i-1}(x_{k+1}). \tag{4.2.19}$$

Combining with

$$V_i(x_k) \leq V_\infty(x_k), \ \forall i, \tag{4.2.20}$$

which is obtained from (4.2.17), we have

$$V_i(x_k) \leq x_k^T Q x_k + u_k^T R u_k + \gamma V_\infty(x_{k+1}), \ \forall i. \tag{4.2.21}$$

Let $i \to \infty$, we can obtain

$$V_\infty(x_k) \leq x_k^T Q x_k + u_k^T R u_k + \gamma V_\infty(x + k + 1). \tag{4.2.22}$$

Note that in the above equation, u_k is chosen arbitrarily, thus, it implies that

$$V_\infty(x_k) \leq \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma V_\infty(x_{k+1}) \right\}. \tag{4.2.23}$$

On the other hand, since the cost function sequence satisfies

$$V_i(x_k) = \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma V_{i-1}(x_{k+1}) \right\} \tag{4.2.24}$$

for any i, considering (4.2.20), we have

$$V_\infty(x_k) \geq \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma V_{i-1}(x_{k+1}) \right\} \quad , \forall i. \tag{4.2.25}$$

Let $i \to \infty$, we can get

$$V_\infty(x_k) \geq \min_{u_k} \left\{ x_k^T Q x_k + u_k^T R u_k + \gamma V_\infty(x_{k+1}) \right\}. \tag{4.2.26}$$

Basing on (4.2.23) and (4.2.26), we can conclude that (4.2.18) is true. ■

Remark 4.1 *We have just proved that the cost function $V_\infty(x_k)$ satisfies the HJB equation. Thus it is the optimal cost function. Accordingly, we say that the cost function sequence converges to the optimal one which is the solution of HJB equation, i.e., $\lim_{i\to\infty} V_i(x_k) = J^*(x_k)$. Then, according to (4.1.7) and (4.2.3), we can conclude that the corresponding control law sequence also converges to the optimal one, i.e., $v_i \to u^*$ as $i \to \infty$.*

Now let us introduce a neural network realization of the iterative ADP algorithm via GDHP technique based on the formulas (4.2.1)-(4.2.4).

For carrying out the iterative ADP algorithm, we need to use neural networks as the function approximation structure to approximate both $v_i(x_k)$ and $V_i(x_k)$.

Let the number of hidden layer neurons be denoted by l, the weight matrix between the input layer and hidden layer be denoted by ν, and the weight matrix between the hidden layer and output layer be denoted by ω. Then, the output of three-layer NN is formulated as

$$\hat{F}(X, \nu, \omega) = \omega^T \sigma\left(\nu^T X\right), \tag{4.2.27}$$

where $\sigma\left(\nu^T X\right) \in \mathbb{R}^l$, $[\sigma(z)]_q = (e^{z_q} - e^{-z_q})/(e^{z_q} + e^{-z_q})$, $q = 1, 2, \cdots, l$, are the activation functions.

Now, we implement the iterative ADP algorithm via GDHP technique. It consists of model network, critic network and action network, which are all chosen as three-layer feedforward NNs. The whole structure diagram is shown in Fig. 4.1, where

$$DER = \left(\frac{\partial \hat{x}_{k+1}}{\partial x_k} + \frac{\partial \hat{x}_{k+1}}{\partial \hat{\nu}_i(x_k)} \frac{\partial \hat{\nu}_i(x_k)}{\partial x_k}\right)^T. \tag{4.2.28}$$

In order to avoid the requirement of knowing $F(x_k, u_k)$, we should train the model network before carrying out the main iterative process. For given x_k and $\hat{\nu}_i(x_k)$, we can obtain the output of the model network as

$$\hat{x}_{k+1} = \omega_m^T \sigma\left(\nu_m^T \left[x_k^T \hat{\nu}_i^T(x_k)\right]^T\right). \tag{4.2.29}$$

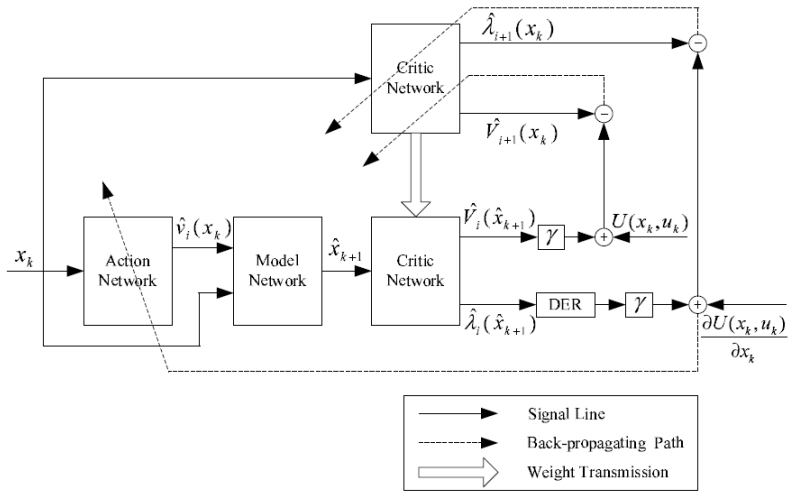


Figure 4.1: The structure diagram of the iterative GDHP algorithm

where m indicate the neural network for approximating the model. We define the error function of the model network as

$$e_{mk} = \hat{x}_{k+1} - x_{k+1}. \tag{4.2.30}$$

The weights of model network are updated to minimize the following performance measure:

$$E_{mk} = \frac{1}{2} e_{mk}^T e_{mk}. \tag{4.2.31}$$

Using the gradient-based adaptation rule, the weighs can be updated as

$$\omega_m(j+1) = \omega_m(j) - \alpha_m \left[\frac{\partial E_{mk}}{\partial \omega_m(j)} \right], \tag{4.2.32}$$

$$\nu_m(j+1) = \nu_m(j) - \alpha_m \left[\frac{\partial E_{mk}}{\partial \nu_m(j)} \right], \tag{4.2.33}$$

where $\alpha_m > 0$ is the learning rate of the model network, and j is the iterative step for updating the weight parameters.

The weights of model network are kept unchanged after the training process is finished.

The critic network is used to approximate both $V_i(x_k)$ and its derivative $\partial V_i(x_k)/\partial x_k$, which is denoted as $\lambda_i(x_k)$. The input of critic network is x_k, while the output is given by

$$\begin{bmatrix} \hat{V}_i(x_k) \\ \hat{\lambda}_i(x_k) \end{bmatrix} = \begin{bmatrix} \omega_{ci}^{1T} \\ \omega_{ci}^{2T} \end{bmatrix} \sigma(\nu_{ci}^T x_k) = \omega_{ci}^T \sigma(\nu_{ci}^T x_k), \tag{4.2.34}$$

where $\omega_{ci} = [\omega_{ci}^1 \omega_{ci}^2]$. Hence, we have

$$\hat{V}_i(x_k) = \omega_{ci}^{1T} \sigma(\nu_{ci}^T x_k) \tag{4.2.35}$$

and

$$\hat{\lambda}_i(x_k) = \omega_{ci}^{2T} \sigma(\nu_{ci}^T x_k). \tag{4.2.36}$$

The target functions can be written as

$$V_{i+1}(x_k) = x_k^T Q x_k + v_i^T(x_k) R v_i(x_k) + \gamma \hat{V}_i(\hat{x}_{k+1}) \tag{4.2.37}$$

and

$$\begin{aligned} \lambda_{i+1}(x_k) &= \frac{\partial \left(x_k^T Q x_k + v_i^T(x_k) R v_i(x_k) \right)}{\partial x_k} + \gamma \frac{\partial \hat{V}_i(\hat{x}_{k+1})}{\partial x_k} \\ &= 2Qx_k + 2 \left(\frac{\partial v_i(x_k)}{\partial x_k} \right)^T R v_i(x_k) \\ &\quad + \gamma \left(\frac{\partial \hat{x}_{k+1}}{\partial x_k} + \frac{\partial \hat{x}_{k+1}}{\partial \hat{v}_i(x_k)} \frac{\partial \hat{v}_i(x_k)}{\partial x_k} \right)^T \hat{\lambda}_i(\hat{x}_{k+1}) \end{aligned} \tag{4.2.38}$$

Then, the error functions can be defined as

$$e_{cik}^1 = \hat{V}_i(x_k) - V_{i+1}(x_k) \tag{4.2.39}$$

and

$$e_{cik}^2 = \hat{\lambda}_i(x_k) - \lambda_{i+1}(x_k). \tag{4.2.40}$$

The objective function to be minimized for critic network is

$$E_{cik} = (1-\theta)E_{cik}^1 + \theta E_{cik}^2, \tag{4.2.41}$$

where

$$E_{cik}^1 = \frac{1}{2} e_{cik}^{1T} e_{cik}^1 \tag{4.2.42}$$

and

$$E_{cik}^2 = \frac{1}{2} e_{cik}^{2T} e_{cik}^2. \tag{4.2.43}$$

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

$$\omega_{ci}(j+1) = \omega_{ci}(j) - \alpha_c \left[(1-\theta)\frac{\partial E_{cik}^1}{\partial \omega_{ci}(j)} + \theta \frac{\partial E_{cik}^2}{\partial \omega_{ci}(j)} \right], \tag{4.2.44}$$

$$\nu_{ci}(j+1) = \nu_{ci}(j) - \alpha_c \left[(1-\theta)\frac{\partial E_{cik}^1}{\partial \nu_{ci}(j)} + \theta \frac{\partial E_{cik}^2}{\partial \nu_{ci}(j)} \right], \tag{4.2.45}$$

where $\alpha_c > 0$ is the learning rate of the critic network, j is the inner-loop iterative step for updating the weight parameters, and $0 \leq \theta \leq 1$ is a parameter that adjusts how HDP and DHP are combined in GDHP. When $\theta = 0$, the training of the critic network reduces to a pure HDP, while $\theta = 1$ does the same for DHP.

In action network, x_k is used as the input and the output is

$$\hat{v}_i(x_k) = \omega_{ai}^T \left(\nu_{ai}^T x_k \right). \tag{4.2.46}$$

The target control input is given by

$$v_i(xk) = -\frac{\gamma}{2}R^{-1}\left(\frac{\partial \hat{x}_{k+1}}{\partial u_k}\right)^T \frac{\partial \hat{V}_i(\hat{x}_{k+1})}{\partial \hat{x}_{k+1}}. \tag{4.2.47}$$

The error function of the action network can be defined as

$$e_{aik} = \hat{v}_i(x_k) - v_i(x_k). \tag{4.2.48}$$

The weights of action network are updated to minimize

$$E_{aik} = \frac{1}{2}e_{aik}^T e_{aik}. \tag{4.2.49}$$

Similarly, the weight update algorithm is

$$\omega_{ai}(j+1) = \omega_{ai}(j) - \alpha_a \left[\frac{\partial E_{aik}}{\partial \omega_{ai}(j)}\right], \tag{4.2.50}$$

$$\nu_{ai}(j+1) = \nu_{ai}(j) - \alpha_a \left[\frac{\partial E_{aik}}{\partial \nu_{ai}(j)}\right], \tag{4.2.51}$$

where $\alpha_a > 0$ is the learning rate of the action network, and j is the inner-loop iterative step for updating the weight parameters.

Remark 4.2 *According to Remark 4.1, $V_i \to J^*$ as $i \to \infty$. Since $\lambda_i(x_k) = \partial V_i(x_k)/\partial x_k$, we can conclude that the sequence $\{\lambda_i\}$ is also convergent with $\lambda_i \to \lambda^*$ as $i \to \infty$.*

Remark 4.3 *Since we cannot implement the iteration until $i \to \infty$ in practical applications, we should run the algorithm with a prespecified accuracy ϵ to test the convergence of the cost function sequence. When $|V_{i+1}(x_k) - V_i(x_k)| < \epsilon$, we consider the cost function sequence has converged sufficiently and stop running the iterative GDHP algorithm.*

4.3 Numerical Experiments

In this section, two examples are provided to demonstrate the effectiveness of the iterative GDHP algorithm.

Example 4.3.1

Consider the following nonlinear system:

$$x_{k+1} = x_k + \sin(x_k + u_k), \tag{4.3.1}$$

where $x_k \in \mathbb{R}$, $u_k \in \mathbb{R}$, $k = 1, 2, \cdots$. The utility function is chosen as $U(x_k, u_k) = x_k^T x_k + u_k^T u_k$. It can be seen that $x_k = 0$ is an equilibrium state of system (4.3.1). However, the system is unstable at this equilibrium, since $(\partial x_{k+1}/\partial x_k)|_{(0,0)} = 2 > 1$.

We choose three-layer feedforward NN as model network, critic network and action network with the structures 2-8-1, 1-8-2, and 1-8-1, respectively, and implement the algorithm at time instant $k = 0$. The initial weights of the three NNs are all set to be random in $[-1, 1]$. Note that the model network should be trained firstly. We train the model network for 100 time steps using 500 data samples under the learning rate $\alpha_m = 0.1$. After the model network is trained, its weights are kept unchanged. Then, let the discount factor $\gamma = 1$ and the adjusting parameter $\theta = 0.5$, we train the critic network and action network for 120 iterations (i.e., for $i = 1, 2, \cdots, 120$) with 2000 training steps for each iteration to make sure that the given accuracy $\epsilon = 10^{-6}$ is reached. In the training process, the learning rate $\alpha_c = \alpha_a = 0.05$. The convergence processes of the cost function and its derivative of GDHP algorithm are shown in Fig. 4.2, for $k = 0$ and $x_0 = 1.5$. We can see that

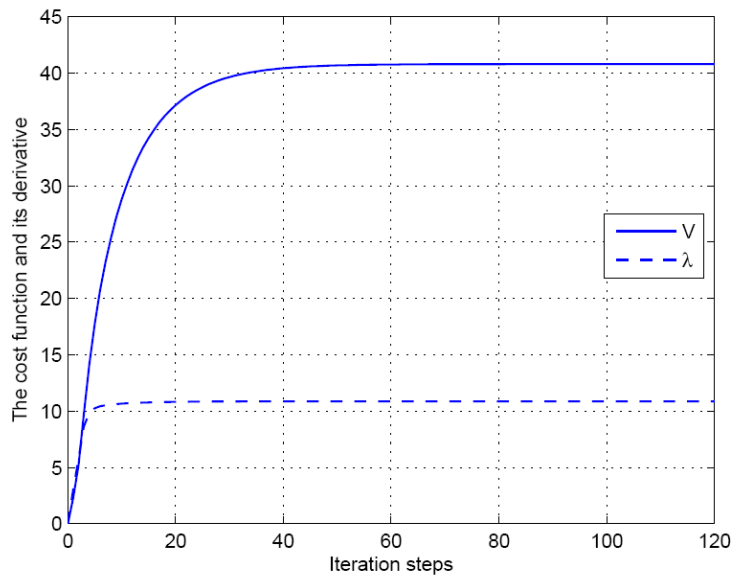


Figure 4.2: The convergence processes of the cost function and its derivative of the iterative GDHP algorithm

the iterative cost function sequence does converge to the optimal value quite rapidly, which also indicates the validity of the iterative GDHP algorithm. For the same problem, the iterative GDHP algorithm takes about 16 seconds while HDP takes about 117 seconds before satisfactory results are obtained.

Moreover, in order to make comparison with DHP algorithm, we also present the controller designed by DHP algorithm. Then, for given initial state $x_0 = 1.5$, we apply the optimal control laws designed by GDHP and DHP techniques to the system for 15 time steps, respectively, and obtain the state curves as shown in Fig. 4.3. The corresponding control curves are shown in Fig. 4.4. It can be seen from the simulation results that the controller derived by GDHP algorithm has better performance than DHP algorithm.

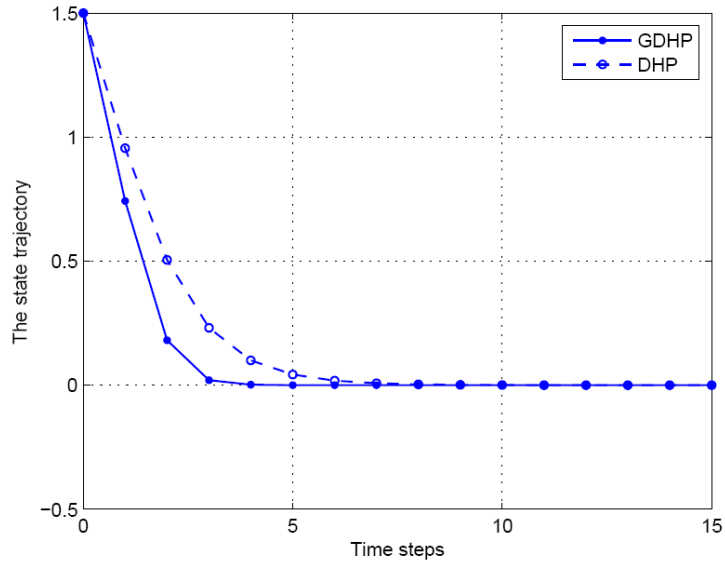


Figure 4.3: The state trajectory x

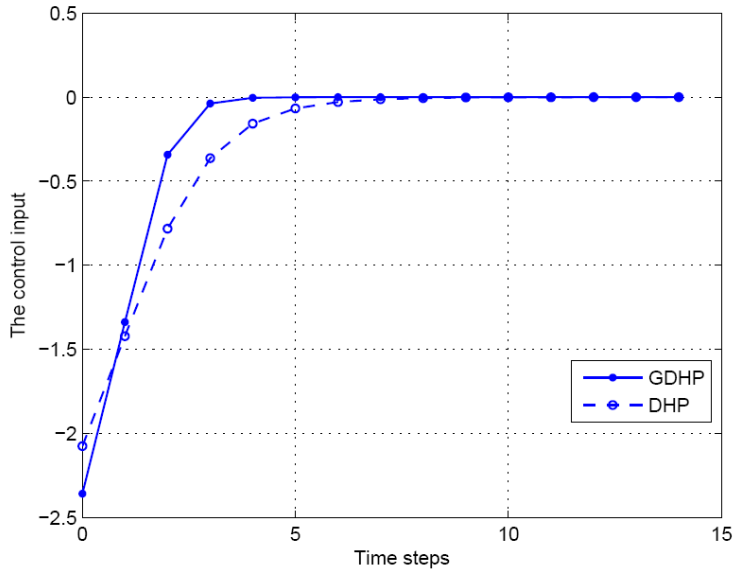


Figure 4.4: The control input u

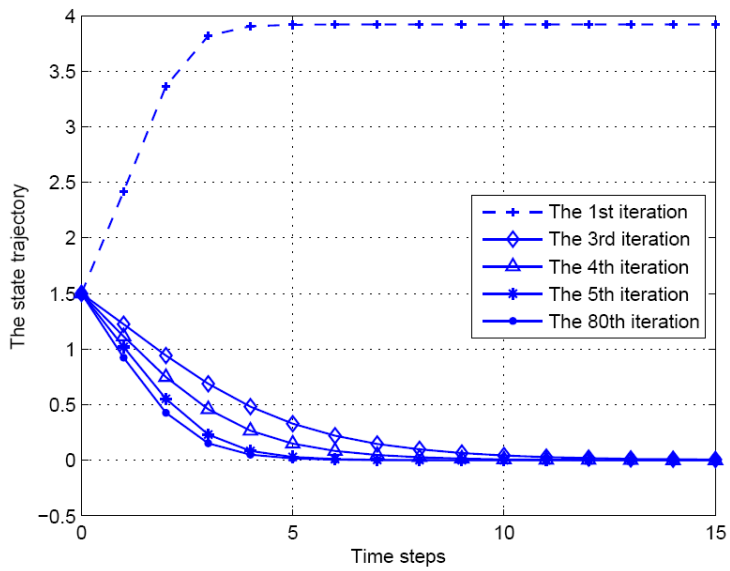


Figure 4.5: The state trajectory x

To show the discount factor has evident impact on our iterative algorithm, in this case, we choose the discount factor $\gamma = 0.9$ and set the other parameters the same as above. Then, we train the critic network and action network for 80 training cycles and find that the given accuracy $\epsilon = 10^{-6}$ has been reached, which demonstrates that smaller discount factor can insure quicker convergence of the cost function sequence. Next, we will show the discrepancy of the state and control curves under different iterations to prove the usefulness of the iterative algorithm. For the same initial state $x_0 = 1.5$, we apply different control laws to the controlled plant for 15 time steps and obtain the simulation results as follows. The state curves are shown in Fig. 4.5, and the corresponding control inputs are shown in Fig. 4.6. From the simulation results, we can see that the closed-loop system is divergent when using the control law obtained in the first iteration. However, the system responses become better and better as the iteration numbers increasing from

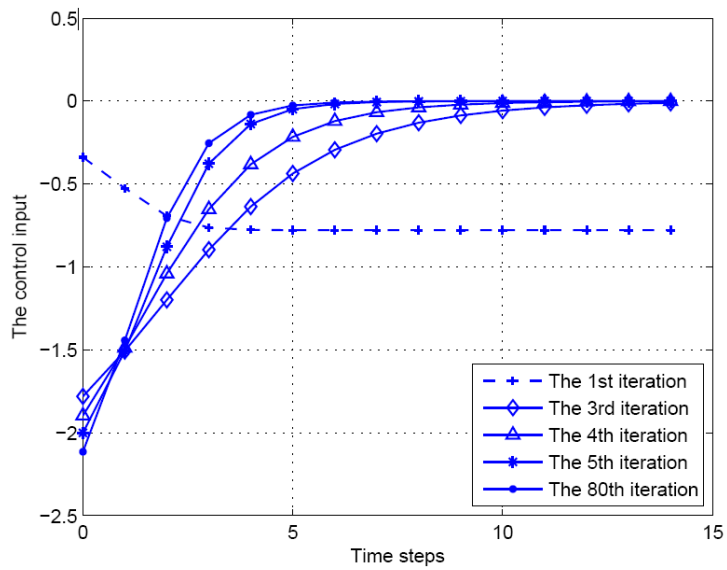


Figure 4.6: The control input u

3 to 80. Besides, the responses basically remain unchanged when the iteration number is larger than 5, which verifies the effectiveness of the proposed iterative GDHP algorithm.

Example 4.3.2

Consider the nonlinear discrete-time system given by

$$x_{k+1} = \begin{bmatrix} -x_{1k}x_{2k} \\ 1.5x_{2k} + \sin(x_{2k}^2 + u_k) \end{bmatrix} \tag{4.3.2}$$

where $x_k = [x_{1k}x_{2k}]^T \in \mathbb{R}^2$, $u_k \in \mathbb{R}$, $k = 1, 2, \cdots$. The utility function is also set as $U(x_k, u_k) = x_k^T x_k + u_k^T u_k$.

In this example, we also choose three-layer feedforward NN as model network, critic network and action network, but with the structures 3-8-2, 2-8-3, 2-8-1, respectively. Here, we train the critic network and action network

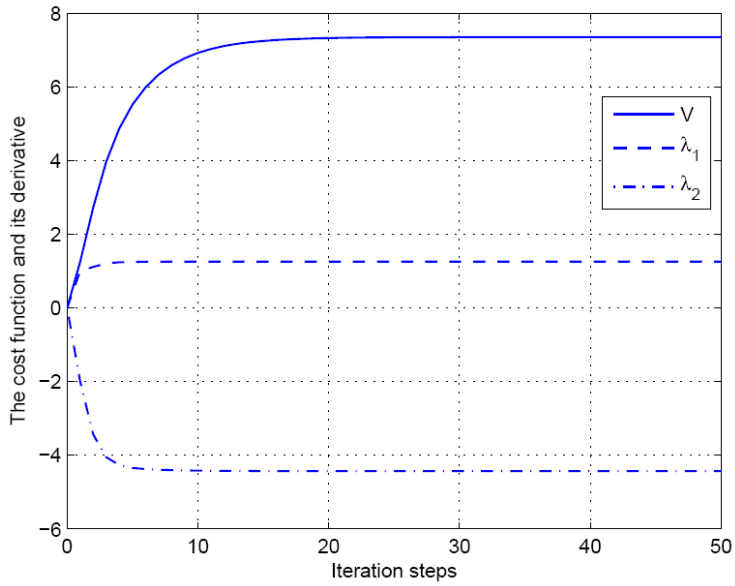


Figure 4.7: The convergence processes of the cost function and its derivative of the iterative GDHP algorithm

for 50 training cycles when keeping other parameters the same as the above example. The convergence processes of the cost function and its derivative of the iterative GDHP algorithm are shown in Fig. 4.7, which verify the statements of Theorems 4.1-4.2 and Remarks 4.1-4.2. Furthermore, for given initial state $x_{10} = 0.5$ and $x_{20} = -1$, we apply the optimal control law designed by the iterative GDHP algorithm to system (4.3.2) for 25 time steps, and obtain the state curves and the corresponding control curve are shown in Fig. 4.8 and Fig. 4.9, respectively. These simulation results verify the excellent performance of the controller derived by the iterative GDHP algorithm.

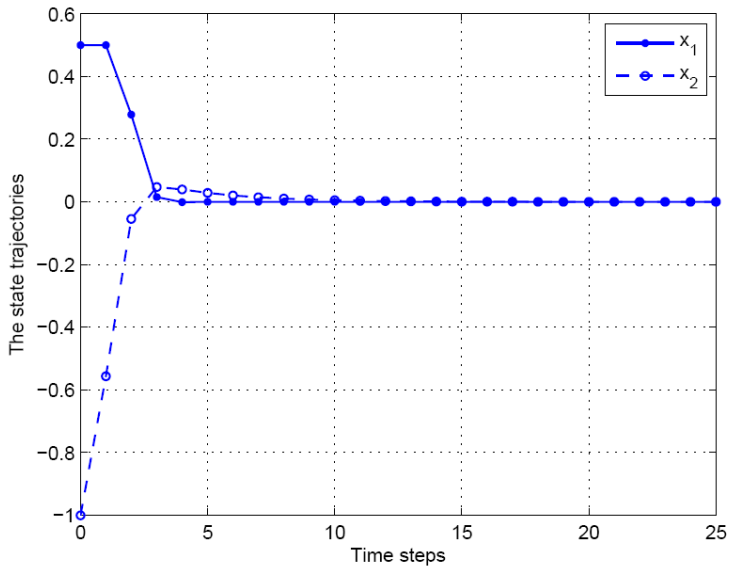


Figure 4.8: The state trajectories x_1 and x_2

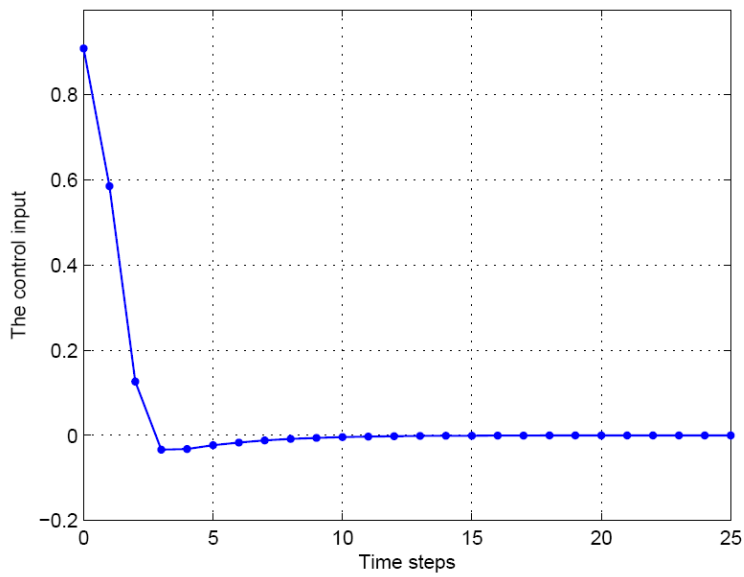


Figure 4.9: The control input u

4.4 Conclusions

In this chapter, an effective iterative ADP algorithm with convergence analysis is given to design the near optimal controller for unknown nonaffine nonlinear discrete-time systems with discount factor in the cost function. The GDHP technique is introduced to implement the algorithm. Three NNs are used as parametric structures to approximate the cost function, the control law and identify the unknown nonlinear system, respectively. The simulation studies demonstrated the validity of the proposed optimal control scheme.

5 WAVELET BASIS FUNCTION NEURAL NETWORKS FOR SEQUENTIAL LEARNING

In this Chapter, we develop the wavelet basis function neural networks (WBFNNs). It is analogous to radial basis function neural networks (RBFNNs) and to wavelet neural networks (WNNs). In WBFNN, both the scaling function and the wavelet function of a multiresolution approximation (MRA) are adopted as the basis to approximate functions. A sequential learning algorithm for WBFNNs is presented and compared to the sequential learning algorithm of RBFNNs. The results of the experiments show that WBFNNs have better generalization property and require shorter training time than RBFNNs. All the results in this Chapter have been published in the paper "Wavelet basis function neural networks for sequential learning" ([12]).

5.1 Introduction

Radial basis function neural networks (RBFNNs) are used to approximate complex functions directly from the input-output data with a simple topological structure ([7, 9, 29, 30, 31, 37]). RBFNNs have good generalization ability as compared to the multilayer feedforward networks. In a RBFNN, a function $f(x)$ is approximated as

$$\hat{f}(x) = \sum_i w_i \phi\left(\frac{\|x - a_i\|}{b_i}\right), \tag{5.1.1}$$

where $\phi(r)$ is the basis function. The most common used basis function is the Gaussian function $\exp(-r^2/2)$. In sequential learning, a neural network is trained to approximate a function while a series of training sample pairs

are randomly drawn and presented to the network. The sample pairs are learned by the network one by one. There are several different sequential learning algorithms for RBFNNs ([9, 15, 16, 24, 25, 28, 33, 34]).

Wavelet neural networks (WNNs) are also used to approximate functions by a single basis function ([6, 17, 20, 40, 48, 49]). In a WNN, a function $f(x)$ is approximated as

$$\hat{f}(x) = \sum_i w_i \phi\left(\frac{x - a_i}{b_i}\right), \tag{5.1.2}$$

where $\phi(x)$ is the basis function coming from wavelet theory ([10, 26]) – the scaling function, the wavelet function, or the basis function of continuous wavelet transform. WNNs can approximate functions more accurately and they have better generalization property than RBFNNs. But all the existing training algorithms are not especially for sequential learning and the orthogonal properties of wavelets have not been used in these algorithms.

In this chapter, we study the *wavelet basis function neural networks* (WBFNNs) for sequential learning. Both the scaling function ϕ and the wavelet function ψ are used as basis functions in WBFNN. Functions ϕ and ψ are orthogonal to each other. In a wavelet decomposition of a function, they will give approximations in different level of details, i.e., coarse and fine approximations, respectively. In a WBFNN, a function $f(x)$ is approximated as

$$\hat{f}(x) = \sum_i \alpha_i \phi\left(\frac{x - a_i}{b_i}\right) + \sum_i \beta_i \psi\left(\frac{x - c_i}{d_i}\right). \tag{5.1.3}$$

A sequential learning algorithm is produced from the orthogonal properties of multiresolution approximation and Mallat's formula of wavelet decomposition ([10, 26]).

The chapter is organized as follows. Section 5.2 gives a brief review of wavelet theory. Section 5.3 presents the definition of WBFNNs and provides a sequential learning algorithm for WBFNNs. Section 5.4 shows quantitative performance comparisons between the sequential learning algorithm of WBFNNs and RBFNNs. Section 5.5 summarizes the conclusion.

5.2 Multiresolution Approximation and Wavelets

In this section a brief review of the wavelets theory is given (cf. [10, 26] for details). A *wavelet basis* is constructed with a *multiresolution approximation* (MRA) of function space. An MRA 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 $\langle f, g\rangle = \int f(x)g(x)dx$. An 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

(S2.1) $\cup_j V_j$ is dense in L^2, i.e., $\overline{\lim_j V_j} = L^2$.

(S2.2) $\cap_j V_j = \{0\}$.

(S2.3) $f(x) \in V_j \iff f(2x) \in V_{j+1}$ for all $j \in \mathbb{Z}$.

(S2.4) $f(x) \in V_j \iff f(x - 2^{-j}k) \in V_j$ for all $j, k \in \mathbb{Z}$.

(S2.5) 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 $\{V_j\}$.

(S2.6) 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 $\{\phi_j^k \colon k \in \mathbb{Z}\}$ is an orthonormal basis of V_j.

(S2.7) Let W_0 be the orthogonal complement of V_0 in V_1, i.e., $W_0 \perp V_0$ and $V_1 = V_0 \oplus W_0$, where the symbol $\oplus$ denotes the orthogonal direct sum of function spaces. Then there exists a function $\psi \in W_0$ such that $\langle \phi, \psi \rangle = 0$, $\|\psi\| = 1$, and $\{\psi(x-k) \colon k \in \mathbb{Z}\}$ is an orthonormal basis of W_0. ψ is called the *wavelet function* of the MRA $\{V_j\}$.

(S2.8) Let $\psi_j^k(x) = 2^{j/2}\psi(2^j x - k)$ for $j, k \in \mathbb{Z}$. Let $W_j = \overline{\mathrm{Span}\{\psi_j^k : k \in \mathbb{Z}\}}$. Then the set of functions $\{\psi_j^k \colon k \in \mathbb{Z}\}$ is an orthonormal basis of W_j, $W_j \perp V_j$ and $V_{j+1} = V_j \oplus W_j$.

(S2.9) For any $J, H \in \mathbb{Z}$, $J < H$, we have

$$V_H = V_J \oplus W_J \oplus W_{J+1} \oplus \cdots \oplus W_{H-1}.$$

Furthermore,

$$L^2 = \overline{\lim_{H \to \infty} V_H} = V_J \oplus (\oplus_{k=J}^{\infty} W_k).$$

(S2.10) 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 level j. We have $\lim_{j \to \infty} P_j f = f$.

(S2.11) 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 level j. We have $P_j f + Q_j f = P_{j+1} f$. Consequently, for any $J, H \in \mathbb{Z}$,

$$P_H f = P_J f + \sum_{J \le k < H} Q_k f.$$

(S2.12) Let $f \in L^2$. Since $P_j f \in V_j$ and $\{\phi_j^k \colon k \in \mathbb{Z}\}$ is an orthonormal basis of V_j, we have

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

where

$$C_{j,k} = \langle f, \phi_j^k \rangle. \tag{5.2.1}$$

(S2.13) Let $f \in L^2$. Since $Q_j f \in W_j$ and $\{\psi_j^k \colon k \in \mathbb{Z}\}$ is an orthonormal basis of W_j, we have

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

where

$$D_{j,k} = \langle f, \psi_j^k \rangle. \tag{5.2.2}$$

(S2.14) Let $f \in L^2$. For any given $J, H \in \mathbb{Z}$, we have the approximation

$$f(x) \approx \sum_{k \in \mathbb{Z}} C_{J,k} \phi_J^k(x) + \sum_{j=J}^{H} \sum_{k \in \mathbb{Z}} D_{j,k} \psi_j^k(x). \tag{5.2.3}$$

(5.2.3) is called the wavelet approximation of f with resolutions from J to H. It is also called the wavelet decomposition of f with resolutions from J to H. The coefficients $C_{j,k}$ and $D_{j,k}$ in (5.2.3) can be calculated by the inner product given in (S2.12) and (S2.13). But in numerical computation, it will take a long time to obtain all these coefficients by inner products. Fortunately, there is a fast algorithm, i.e., the "fast wavelet decomposition" of Mallat and Daubechies.

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 $\{\phi_1^k\}$ of V_1. For $k \in \mathbb{Z}$, define

$$\begin{cases} h_k = \langle \phi(x), \phi_1^k(x) \rangle, \\ g_k = \langle \psi(x), \phi_1^k(x) \rangle. \end{cases} \tag{5.2.4}$$

Then

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

and

$$\begin{cases} \phi(x) = \sum_{k\in\mathbb{Z}} h_k \phi_1^k(x), \\ \psi(x) = \sum_{k\in\mathbb{Z}} g_k \phi_1^k(x). \end{cases} \tag{5.2.5}$$

It was proved that in the wavelet decomposition (5.2.3), the weights $C_{j,k}$ and $D_{j,k}$ of different resolution levels have the following relationship:

$$\begin{cases} C_{j-1,m} = \sum_{k\in\mathbb{Z}} h_{k-2m} C_{j,k}, \\ D_{j-1,m} = \sum_{k\in\mathbb{Z}} g_{k-2m} C_{j,k}. \end{cases} \tag{5.2.6}$$

Formula (5.2.6) is the so called *fast wavelet decomposition algorithm* of Mallat and Daubechies. It describes the way to obtain the coefficients $C_{j,k}$ and $D_{j,k}$ for lower resolution level from the coefficients with higher resolution level. Now we have two ways to obtain the coefficients $C_{j,k}$ and $D_{j,k}$ in the wavelet decomposition (5.2.3). We can calculate all the $C_{j,k}$ and $D_{j,k}$ by inner product according to (5.2.1) and (5.2.2). We can also obtain $C_{j,k}$ and $D_{j,k}$ by fast wavelet decomposition (5.2.6), in case we have already had the values of $C_{j,k}$ and $D_{j,k}$ for some higher resolution level.

5.3 Wavelet Base Function Neural Networks and Sequential Learning

WBFNN is analogous to both RBFNN and WNN. Same with the WNNs, the structure of WBFNNs is based on the theory of wavelets. Instead of one

basis function, both the scaling function and the wavelet function of a multiresolution approximation are used as basis functions in a WBFNN. Besides, we use not only the ability of wavelets to approximate functions, but also the orthogonal properties and the relationships between the approximations of different resolution levels. Similar to RBFNNs, WBFNNs can be employed for sequential learning, but WBFNNs can be trained faster, perform more accurately, and have better generalization properties.

The structure of WBFNNs are based on the approximation (5.2.3). J and H are given to indicate the minimum and maximum of the resolution levels. There are two types of neurons in a WBFNN, equipped with activation function ϕ_j^k and ψ_j^k, respectively.

For purpose of real world application, we assume that the domain X of the function $f(x)$ is finite. Suppose $X = [X_{\min}, X_{\max}]$. We will choose Daubechies' wavelets with compact support as the basis. The Daubechies' wavelets have the following properties.

(S3.1) Suppose that the scaling function ϕ and wavelet function ψ form a Daubechies' wavelet. Let h_k be the coefficients defined in (5.2.4). Then there is a positive integer K_s such that $h_k \neq 0$ only when $k = 0, \ldots, K_s$. Furthermore, the support sets of ϕ and ψ are supp $\phi = [0, K_s)$ and supp $\psi = [1 - K_s, 1)$, respectively.

(S3.2) For $j, k \in \mathbb{Z}$,

$$\begin{aligned}
&\text{supp } \phi_j^k = [2^{-j}k, 2^{-j}(k + K_s)),\\
&\text{supp } \psi_j^k = [2^{-j}(k + 1 - K_s), 2^{-j}(k + 1)),\\
&\text{supp } \phi_j^k \cap \text{supp } \psi_j^k = [2^{-j}k, 2^{-j}(k + 1)).
\end{aligned}$$

When J is small enough, we have $X \subseteq [2^{-J}k_0, 2^{-J}(k_0 + 1))$ for certain

choice of k_0. We only need consider those neurons with $X \cap \text{supp } \phi_j^k \neq \emptyset$ or $X \cap \text{supp } \psi_j^k \neq \emptyset$. So the neurons of resolution level J in the network are $\phi_J^{k_0-k}$ and $\psi_J^{k_0+k}$ for $0 \leq k \leq K_s - 1$. We call these neurons the root neurons. All the other neurons are $\psi_j^k(x)$ for $J < j \leq H$.

The neurons ϕ_j^k and ψ_j^k which will fire on an input x only if x is inside the interval supp $\phi_j^k = [2^{-j}k, 2^{-j}(k+K_s))$ or supp $\psi_j^k = [2^{-j}(k+1-K_s), 2^{-j}(k+1))$. The length of these intervals are $2^{-j}K_s$. During the training of the network, for a training sample pair (x_i, y_i), we need to adjust the root neurons and those neurons such that

$$x_i \in [2^{-j}(k+1-K_s), 2^{-j}(k+1)). \tag{5.3.1}$$

It is easy to see that (5.3.1) is equivalent to

$$2^j x_i - 1 < k \leq 2^j x_i + K_s - 1. \tag{5.3.2}$$

Hence the output of the whole network is

$$\begin{aligned}
\hat{f}(x) &= \sum_{k=0}^{K_s-1} \left(\varpi_k \phi_J^{k_0-k}(x) + w_{J,k_0+k}\psi_J^{k_0+k}(x)\right) \\
&\quad + \sum_{j=J+1}^{H} \sum_{k=\lfloor 2^j x \rfloor}^{\lfloor 2^j x + K_s - 1 \rfloor} w_{j,k} k \psi_j^k(x), \\
&= \sum_{k=0}^{K_s-1} \varpi_k \phi_J^{k_0-k}(x) + \sum_{j=J}^{H} \sum_{k=\lfloor 2^j x \rfloor}^{\lfloor 2^j x + K_s - 1 \rfloor} w_{j,k} k \psi_j^k(x),
\end{aligned} \tag{5.3.3}$$

where $\lfloor K \rfloor$ means the biggest integer not bigger than K. For each j, the number of k satisfying (5.3.2) is always K_s. Hence, since we limit that $J \leq j \leq H$, for each input x (or training sample (x_i, y_i)), at most $(H - J + 2)K_s$ neurons will fire (or be adjusted during the training).

A WBFNN will grow during the training. It starts with root neurons $\phi_J^{k_0-k}$, $0 \le k < K_s$, and sequentially increases (or decreases, when some existing neurons have very small significance) the number of neurons until the approximation error is sufficiently small. The growth of neurons will depend on the error of the approximation, the position of existing neurons, and the resolution level of existing neurons. To adjust the weights of neurons, we use the formulas (5.2.1), (5.2.2), and (5.2.6). Now we give our growing and pruning algorithm for sequential learning of WBFNNs. Given $e_{\min}$ to be the minimal error of the approximation and given $\delta_{\min}$ to be the minimal significance of the neurons. The network is initialized as $\hat{f}(x) = \sum_{k=0}^{K_s-1} \varpi_k \phi_J^{k_0-k}(x)$, with $\varpi_k = 0$. The sequence of sample pairs $\{(x_i, y_i)\}$ are chosen randomly with respect to certain probability distribution.

SLWBF Algorithm (Sequential Learning of Wavelet Basis Function neural networks)

A01 For each pair (x_i, y_i), do Steps A02–A03.

A02 Compute the overall network output

$$\hat{f}(x_i) = \sum_k \varpi_k \phi_J^{k_0-k}(x_i) + \sum_{j,k} w_{j,k} \psi_j^k(x_i).$$

Find the error $e = y_i - \hat{f}(x_i)$.

A03 If $|e| > e_{\min}$ then do steps A04–A08

A04 Find the maximal resolution level H_1 of existing neurons which will fire at x_i. If $H_1 < H$ then $H_1 = H_1 + 1$.

A05 Define the error function

$$E(x) = \max\{0, 1 - 2^{H_1}|x - x_i|\}e.$$

A06 From $j = H_1$ to $j = J$, compute

$$\triangle C_{j,k} = \begin{cases} \langle E(x), \phi^k_{H_1} \rangle, & \text{if } j = H_1, \\ \sum_{s\in\mathbb{Z}} h_{s-2k} \triangle C_{j+1,s} & \text{if } J \le j < H_1, \end{cases}$$
$$\triangle D_{j,k} = \begin{cases} \langle E(x), \psi^k_{H_1} \rangle, & \text{if } j = H_1, \\ \sum_{s\in\mathbb{Z}} g_{s-2k} \triangle C_{j+1,s}, & \text{if } J \le j < H_1, \end{cases}$$

where $2^j x_i - 1 < k \le 2^j x_i + K_s - 1$.

A07 Adjust the weights by $\varpi_k = \varpi_k + \triangle C_{J,k_0-k}$ and $w_{j,k} = w_{j,k} + \triangle D_{j,k}$.

A08 If $|w_{j,k}| \ge \delta_{\min}$ and there is no neuron with activation function ψ^k_j, add a neuron $w_{j,k}\psi^k_j$.

If $|w_{j,k}| < \delta_{\min}$ and the neuron $w_{j,k}\psi^k_j$ exists, prune this neuron.

The parameters $e_{\min}$, $\delta_{\min}$ and H determine the accuracy of the algorithm. Smaller $e_{\min}$ and $\delta_{\min}$ give better approximation while larger H gives smaller errors.

5.4 Numerical Experiments

In this section, numerical experiments are performed for the SLWBF algorithm. The experiments run in a Dell Optiplex 745 computer using MATLAB. As a comparison of our WBFNNs and SLWBF algorithm, we choose sequential learning algorithm "minimal resource allocation networks" (MRAN) [37] of RBFNNs. In [37], MRAN algorithm was compared with some other sequential learning algorithms of RBFNNs, such as the resource allocation network (RAN) [28] and the resource allocation network via extended Kalman filter (RANEKF) [15], and so on. The results in [37] show that MRAN has better performance in terms of generalization, network size, and training

speed, whenever the input data are uniformly or not-uniformly distributed. Let

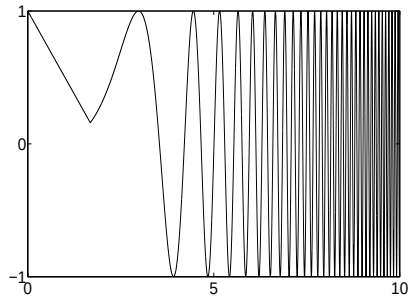
$$f(x) = \begin{cases} \max\{1 - x/2, \sin(x^4/50)\}, & \text{if } 0 \le x \le 2; \\ \sin(x^4/50), & \text{if } 2 < x \le 10. \end{cases} \tag{5.4.1}$$

Fig. 5.1(a) is the graph of $f(x)$. We will approximate $f(x)$ by RBFNNs and WBFNNs, respectively, while the training data are generated under uniform distribution or Gaussian distribution $N(4, 1.25^2)$.

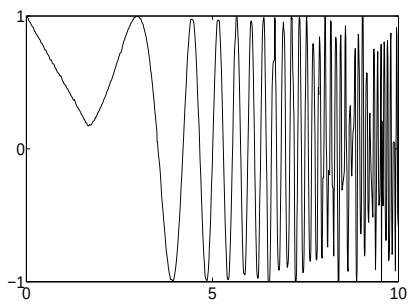
In each case, 10000 training pairs (x_i, y_i) $(i = 1, \cdots, 10000)$ are taken from the range $X = [0, 10]$ under according probability distribution. For testing the resulting neural networks, we use 1000 data which uniformly distributed in the same range $x \in [0, 10]$. In the performance of SLWBF algorithm, the wavelet adopted is Haar wavelet, which is the wavelet with the smallest support among Daubechies' wavelets. For Haar wavelet, the parameter $K_s = 2$. Hence both the structure of WBFNNs and the calculation of algorithm SLWBF are very simple in this case.

Fig. 5.1(b)–(e) show the resulting approximation $\hat{f}(x)$ after presenting 10000 training pairs. Fig. 5.1(b) is the approximation by RBFNN and the training sequence obeys uniform distribution. Fig. 5.1(c) is the approximation by WBFNN and the training sequence also obeys uniform distribution. In (d) and (e) of Fig. 5.1, the training data sequences obey Gaussian distribution $N(4, 1.25^2)$ and the network is a RBFNN in (d) and a WBFNN in (e). The results show that WBFNNs give better approximation than RBFNNs.

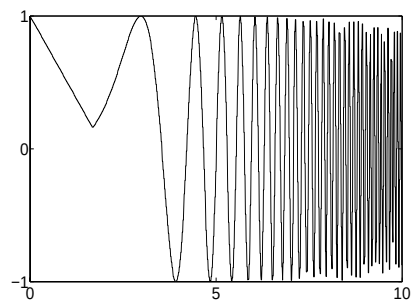
Fig. 5.2 gives the comparison of approximation mean square error $\int_0^{10}(f(x) - \hat{f}(x))^2 dx/10$. In Fig. 5.2(a), the training data obey uniform distribution and in Fig. 5.2(b), the training data obey Gaussian distribution. In both cases of uniform distributed and Gaussian distributed samples,



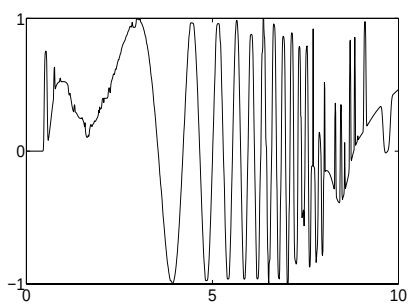
(a) $f(x)$



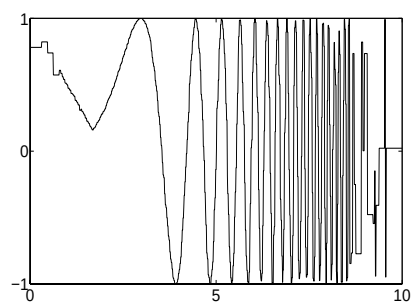
(b) RBF & uniform distribution



(c) WBF & uniform distribution

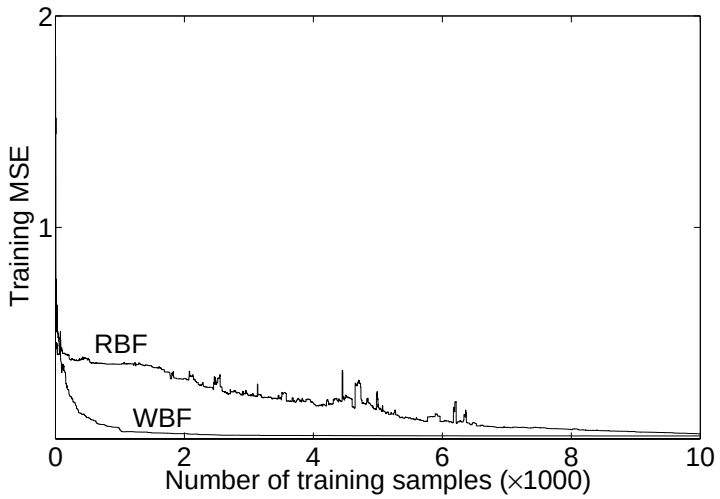


(d) RBF & Gaussian distribution

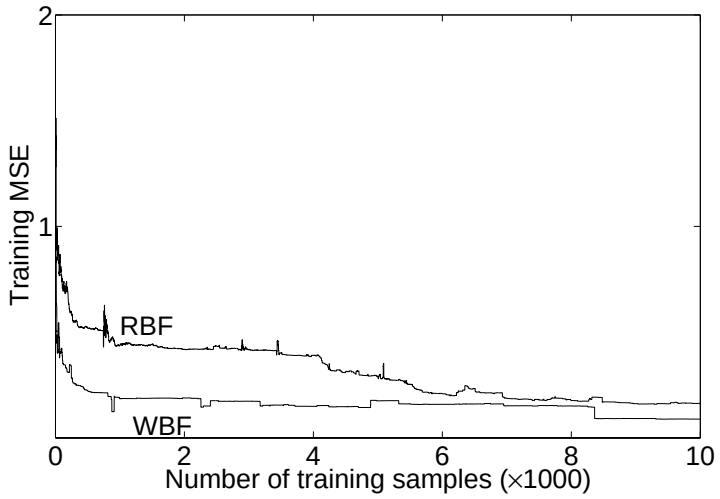


(e) WBF & Gaussian distribution

Figure 5.1: $f(x)$ and $\hat{f}(x)$



(a) Training data with uniform distribution



(b) Training data with Gaussian distribution

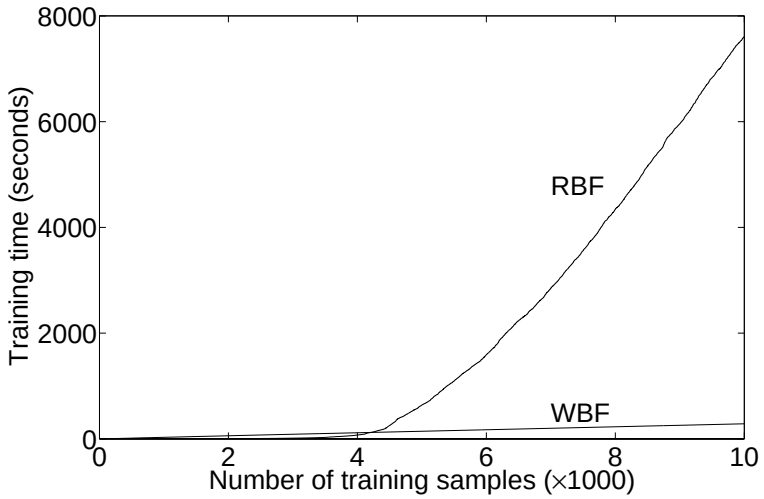
Figure 5.2: Comparison of mean square errors

WBFNNs perform better approximation and converge earlier.

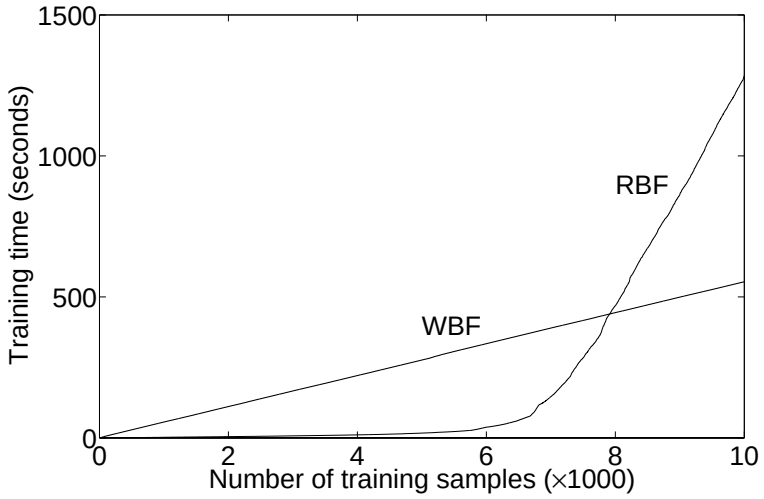
Fig. 5.3 is the comparison of training time for RBFNNs and WBFNNs. We can see that the training time of WBFNNs algorithm is linear and the time of RBFNNs algorithm looks like a square function. This is because in the MRAN algorithm, extended Kalman filter is applied to justify the parameters of neurons, whose training time is proportional to the square of the number of neurons; while in the SLWBF algorithm, the time to learn from a single training data is fixed.

Fig. 5.4 is the comparison of the number of neurons. At the beginning, RBFNNs have less neurons than WBFNNs. But finally, RBFNNs have more neurons. We can see that at the beginning of the training, SLWBF algorithm requires more neurons while it gives higher accuracy in the approximation and faster converge speed. In the end, RBFNNs require more neurons but still provide less accuracy in the approximation.

According to the performance comparison results, WBFNNs have better approximation accuracy. When the training data obey uniform distribution, although WBFNNs and RBFNNs give similar approximation errors after all 10000 training samples have been applied, WBFNNs converge faster. When the training data obey normal distribution, WBFNNs give smaller errors. Hence we know that WBFNNs have better generalization than RBFNNs have. In both cases, WBFNNs can give better approximation of functions from fewer training data pairs. The time to train a WBFNN is proportional to the number of training data pairs. But the time for training a RBFNN is nonlinear and will grow faster when the number of training data grows. For the size of the network, RBFNNs have fewer neurons at the beginning and finally, RBFNNs have more neurons than WBFNNs have.

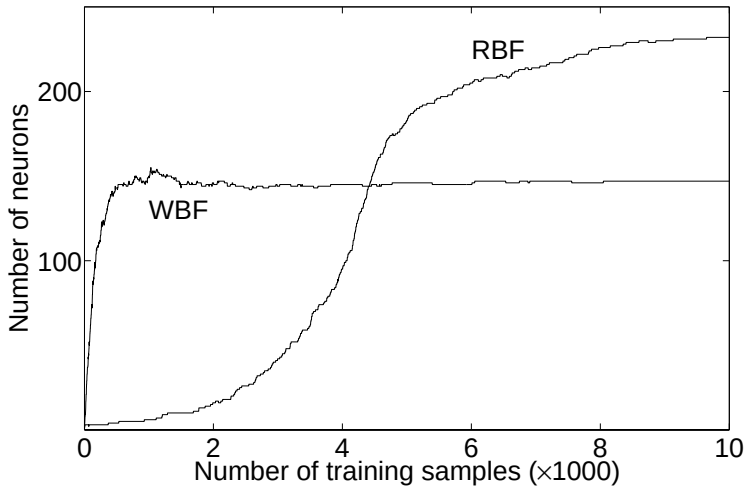


(a) Training data have uniform distribution

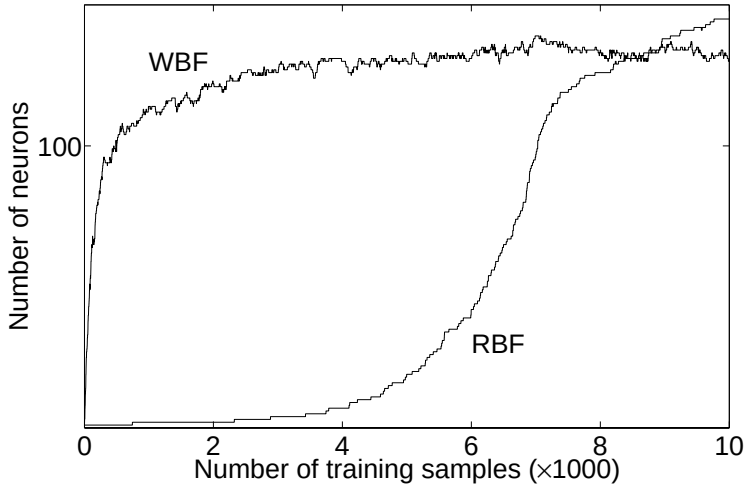


(b) Training data have Gaussian distribution

Figure 5.3: Comparison of training time



(a) Training data have uniform distribution



(b) Training data have Gaussian distribution

Figure 5.4: Comparison of numbers of neurons

5.5 Conclusions

In this chapter, a new kind of neural networks, WBFNN is defined. A sequential learning algorithm called SLWBF algorithm is presented for WBFNNs. WBFNN is a development of both RBFNN and WNN. Both the scaling function and wavelet function of an MRA are applied to construct a WBFNN. Since wavelets have the ability to approximate functions in multiresolution, WBFNN can give better approximation of functions with better generalization.

Performance of the algorithm SLWBF has been compared with the algorithm MRAN. The training sequences coming from both uniform and Gaussian distributions are used for numerical experiments. The results indicate that WBFNNs give more accurate approximation, faster converge, and less training time, while they require smaller size of networks. Furthermore, SLWBF gives better generalization performance especially when the input data are not uniformly distributed.

6 CONCLUDING REMARKS

In this dissertation, we addressed three research topics.

We start our research by studying optimal control and dynamic programming. Optimal control and dynamic programming for complex dynamical systems is difficult due to the "curse of dimensionality" and the "over optimal" problem. The aim of our work is to overcome these two difficulties. We introduce a novel ϵ-optimal performance index function $V_\epsilon^*(\cdot)$ as an approximation of the optimal performance index function. The associated ϵ-optimal controller $\mu_\epsilon^*(\cdot)$ can always control the system state to approach the equilibrium state stably, while the performance index is close to the optimal performance index within an error bound according to ϵ. We call this method to be the ϵ-adaptive dynamic programming method. Since only one performance cost function is used in ϵ-adaptive dynamic programming, ϵ-adaptive dynamic programming is helpful to overcome the curse of dimensionality. In Chapter 2, we establish the ϵ-adaptive dynamic programming theory for the nonlinear discrete-time systems with no discount factor in the performance index, i.e., the discount factor in the performance cost is $\gamma = 1$. In Chapter 3, we generalize our results of the ϵ-adaptive dynamic programming to the systems with discount factor $0 < \gamma \leq 1$ in the performance index. Numerical simulations have been performed in both Chapter 2 and Chapter 3. The results of the simulations show that our ϵ-adaptive dynamic programming method works well in both cases of $\gamma = 1$ and $0 < \gamma \leq 1$.

The results on ϵ-optimal dynamic programming provide stable controllers in the sense of ϵ-optimal. By using a single performance cost function in ϵ-adaptive dynamic programming, ϵ-adaptive dynamic programming is also

helpful to overcome the curse of dimensionality. However, the time expense of the ϵ-optimal dynamic programming algorithm is still big. Thus, we turn to our second topic that we will design an algorithm for the dynamic programming which runs faster. We set certain restriction on the system. It is assumed that the utility function of system is a positive definite quadratic function. Under this assumption, an iterative adaptive dynamic programming (ADP) algorithm using globalized dual heuristic programming (GDHP) technique is introduced to obtain the optimal controller with convergence analysis in terms of cost function and control law. In order to implement the iterative algorithm, a neural network is constructed first to identify the unknown nonlinear system. Then, based on the learned system model, two other neural networks are used as parametric structures to facilitate the implementation of the iterative algorithm, which aims at approximating at each iteration the cost function and the control law, respectively. A simulation example is provided to verify the effectiveness of the presented optimal control scheme dynamic programming (ADP) algorithm using globalized dual heuristic programming (GDHP) technique is introduced.

The last part of this thesis is about wavelet neural network. In the numerical simulations in our research on dynamic programming, we use neural network to approximate the functions, such as the performance cost function and the optimal controller. The neural networks will be trained from sequences of input-output data. So the generalization ability of the neural networks is important for our purpose. Radial basis function neural networks (RBFNNs) and wavelet neural networks (WNNs) are well known neural networks that have good generalization ability. Hence we designed a wavelet basis function neural networks (WBFNNs) for sequential learning. Both the scaling function ϕ and the wavelet function ψ are used as basis functions in

WBFNN. Functions ϕ and ψ are orthogonal to each other. In a wavelet decomposition of a function, they will give approximations in different level of details, i.e., coarse and fine approximations, respectively. A sequential learning algorithm is produced from the orthogonal properties of multiresolution approximation and Mallat's formula of wavelet decomposition.

CITED LITERATURE

[1] M. Abu-Khalaf and F. Lewis, "Nearly optimal control laws for nonlinear systems with saturating actuators using a neural network HJB approach," *Automatica*, vol. 41, no. 5, pp. 779–791, May 2005.

[2] A. Al-Tamimi and F. Lewis "Discrete-time nonlinear HJB solution using approximate dynamic programming: convergence proof," *Proc. IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, Honolulu, HI, Apr. 2007, pp. 38–43.

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

[4] R. Beard, G. Saridis, and J. Wen, "Galerkin approximations of the generalized Hamilton-Jacobi-Bellman equation," *Automatica*, vol. 33, no. 12, pp. 2159–2177, Dec. 1997.

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

[6] S. A. Billings and H. Wei, "A new class of wavelet networks for nonlinear system identification," *IEEE Trans. Neural Networks*, vol. 16, no. 4, pp. 862–874, July 2005.

[7] D. S. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks," *Complex Syst.*, vol. 2, pp. 321–355, 1988.

[8] A. E. Bryson and Y.-C. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*, New York, NY: Hemisphere-Wiley, 1975.

[9] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 302–309, Mar. 1991.

[10] I. Daubechies, *Ten Lectures on Wavelets*, CBMS-NSF Regional Conference Series in Applied Mathematics, Philadelphia: SIAM Press, vol. 61, 1992.

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

[12] N. Jin and D. Liu, "Wavelet basis function neural networks for sequential learning," *IEEE Trans. Neural Networks*, Vol. 19, No. 3, pp. 523-528, 2008.

[13] N. Jin and D. Liu, "Discrete-time ϵ-Adaptive dynamic programming algorithm using neural networks," *Proceedings of IEEE International Symposium on Intelligent Control 2008 (ISIC 2008)*, San Antonio, TX, Sept. 2008, pp.1085-1090.

[14] N. Jin, D. Liu, and Y. Ma, "Adaptive dynamic programming algorithm for discrete-time systems with ϵ-error bound and discount factor in the performance cost," *Proceedings of IEEE International Conference on Networking, Sensing and Control, 2009 (ICNSC '09)*, Okayama, Japan, March 2009, pp. 189 - 194.

[15] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Comput.*, vol. 5, no. 6, pp. 954–975, Nov. 1993.

[16] N. B. Karayiannis and G. W. Mi, "Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques," *IEEE Trans. Neural Networks*, vol. 8, no. 6, pp. 1492–1506, Nov. 1997.

[17] H. R. Karimi, B. Moshiri, B. Lohmann, and P. J. Maralani, "Haar wavelet-based approach for optimal control of second-order linear systems in time domain," *J. Dyna. & Control Sys.*, vol. 11, no. 2, pp. 237–252, Apr. 2005.

[18] 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*, Tulsa, OK, Aug. 2002, vol. 2, pp. II338–II341.

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

[20] Y. Lin and F. Wang, "Modular structure of fuzzy system modeling using wavelet networks," *Proc. 2005 IEEE Networking, Sensing and Control*, Tuscon, AZ, Mar. 2005, pp. 671–676.

[21] D. Liu, "Approximate dynamic programming for self-learning control," *ACTA Automatica Sinica*, vol. 31, no. 1, pp. 13–18, Jan. 2005.

[22] D. Liu, "Neural network-based adaptive critic designs for self-learning control," *Proceedings of the 9th International Conference on Neural Information Processing*, Singapore, Nov. 2002, pp.1252-1256. (Invited paper)

[23] D. Liu, D. Wang, D. Zhao, Q. Wei, and N. Jin, "Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems

using gobalized dual heuristic programming," submitted for publication.

[24] Y. Lu, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks," *Neural Comput.*, vol. 9, no. 2, pp. 461–478, 1997.

[25] Y. Lu, N. Sundararajan, and P. Saratchandran, "Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm," *IEEE Trans. Neural Networks*, vol. 9, no. 2, pp. 308–318, Mar. 1998.

[26] S. Mallat, "Multiresolution approximation and wavelet orthonomal bases of L^2," *Trans. Amer. Mat. Soc.*, vol. 315, pp. 69–87, 1989.

[27] J. J. Murray, C. J. Cox, G. G. Lendaris, and R. Saeks, "Adaptive dynamic programming," *IEEE Trans. Syst., Man, Cybern. - Part C: App & Reviews*, vol. 32, no. 2, pp. 140–153, May 2002.

[28] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, no. 2, pp. 213–225, 1991.

[29] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481–1497, Sept. 1990.

[30] M. J. D. Powell, "Radial basis functions for multivariable interpolation: A review," *Algorithms for Approximation*, J. C. Mason and M. G. Cox, Eds., Oxford: Clarendon Press, 1987, pp. 143–167.

[31] M. J. D. Powell, "Radial basis functions approximations to polynomials," *Proc. 12th Biennial Numerical Analysis Conf.*, Dundee, Scotland, June 1987, pp. 223–241.

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

[33] A. Roy, S. Govil, and R. Miranda, "A neural-network learning theory and a polynomial time RBF algorithm," *IEEE Trans. Neural Networks*, vol. 8, no. 6, pp. 1301–1313, Nov. 1997.

[34] M. Salmerón, J. Ortega, C. G. Puntonet, A. Prieto, and I. Rojas, "SSA, SVD, QR-cp, and RBF model reduction," *Lecture Notes in Comput. Sci.*, Berlin: Springer, vol. 2415, pp. 589–594, 2002.

[35] 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.

[36] 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.

[37] N. Sundararajan, P. Saratchandran, and Y. Lu, *Radial Basis Function Neural Networks with Sequential Learning: MRAN and Its Applications*, Singapore: World Scientific, 1999.

[38] F. Wang, N. Jin, D. Liu, and Q. Wei, "Adaptive dynamic programming for finite-horizon optimal control of discrete-time nonlinear systems with ϵ-error bound," *IEEE Trans. Neural Networks*, vol. 22, no. 1, pp. 24–36, 2011.

[39] D. Wang, D. Liu, Q. Wei, D. Zhao, and N. Jin, "Optimal control of unknown nonaffine nonlinear discrete-time systems based on adaptive dynamic programming," submitted for publication.

[40] F. Wang and H. Kim, "Implementing adaptive fuzzy logic controllers with neural networks: A design paradigm," *Journal of Intelligent and Fuzzy Systems*, vol. 3, no. 2, pp. 165–180, 1995.

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

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

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

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

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

[46] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, PhD. Thesis*, Harvard Univ., Cambridge, MA, 1974.

[47] W. M. Wonham, "Random differential equations in control theory," *Probabilistic Methods in Applied Mathematics*, A. T. Bharucha-Reid, Ed. New York: Academic Press, 1970.

[48] J. Zhang, G. G. Walter, Y. Miao, and W. N. W. Lee, "Wavelet neural networks for function learning," *IEEE Trans. Signal Processing*, vol. 43, no. 6, pp. 1485–1497, June 1995.

[49] Q. Zhang and A. Benveniste, "Wavelet networks," *IEEE Trans. Neural Networks*, vol. 3, no. 6, pp. 889–898, Nov. 1992.

VITA

NAME:	Ning Jin
EDUCATION:	Ph.D., Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois, 2011.
	Ph.D., Mathematics, East China Normal University, China, 1990.
	M.S., Mathematics, East China Normal University, China, 1987.
	B.S., Mathematics, East China Normal University, China, 1984.
TEACHING:	Department of Mathematics, Nanjing University, China, 1992-2001.
	Department of Electrical and Computer Engineering, University of Illinois at Chicago, Chicago, Illinois, Fall 2011.
EXPERIENCE:	Department of Computer Science, Hong Kong Baptist University, Hong Kong, 2001.
	Department of Mathematics, Statistics and Computer Science, University of Illinois at Chicago, Chicago, Illinois, 2002-2004.
PUBLICATIONS:	Feiyao Wang, Ning Jin, Derong Liu, and Qinglai Wei, "Adaptive dynamic programming for finite-horizon optimal control of discrete-time nonlinear systems with ϵ-error bound," *IEEE Trans. Neural Networks,* vol. 22, no. 1, pp. 24–36, 2011.
	Ning Jin, Derong Liu, and Yingying Ma, "Adaptive dynamic programming algorithm for discrete-time systems with ϵ-error bound and discount factor in the performance cost," *Proceedings of IEEE International Conference on Networking, Sensing and Control, 2009 (ICNSC '09)*, Okayama, Japan, March 2009, pp. 189 - 194.

Zhongyu Pang, Derong Liu, Ning Jin, and Zou Wang, "A Monte Carlo particle model associated with neural networks for tracking problems," *IEEE Transactions on Circuits and Systems-I*, vol 55, no. 11, pp. 3421-3429, 2008.

Ning Jin and D. Liu, "Discrete-time ϵ-adaptive dynamic programming algorithm using neural networks," *Proceedings of IEEE International Symposium on Intelligent Control 2008 (ISIC 2008)*, San Antonio, TX, Sept. 2008, pp.1085-1090.

Derong Liu, Ning Jin, "ϵ-Adaptive Dynamic Programming for discrete-time systems," IJCNN 2008, Hong Kong, China, June 1-6, 2008, pp. 1417-1424.

Ting Huang, Derong Liu, Hossein Javaherian and Ning Jin, "Neuro sliding mode control of the engine torque," Proceedings of the 17th World Congress of The International Federation of Automatic Control, Seoul, Korea, July 6-11, 2008, pp. 9453-9458.

Ning Jin and Derong Liu, "Wavelet basis function neural networks for sequential learning," *IEEE Transactions on Neural Networks*, vol. 19, No. 3, pp. 523-528, 2008.

Ning Jin, Derong Liu, Zhongyu Pang and Ting Huang, "Wavelet basis function neural networks," International Joint Conference on Neural Networks (IJCNN2007), Orlando, Florida, Aug. 2007, pp. 500–505.

Zhongyu Pang, Derong Liu, Ning Jin and Zhuo Wang, "Neural network strategy for sampling of particle filters on the tracking problem," Proceeding of International Joint Conference on Neural Networks (IJCNN2007), Orlando, Florida, Aug. 2007, pp. 254–259.

Ning Jin, Derong Liu, Ting Huang and Zhongyu Pang, "Discrete-time approximate dynamic programming using wavelet basis function neural networks," Proceeding of IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, April 1-5, 2007, Hawaii, pp. 135–143.

Derong Liu and Ning Jin, "Finite Horizon Discrete-Time Approximate Dynamic Programming," On Proceeding of 21st IEEE International Symposium on Intelligent Control, Munich, Germany, Oct. 4-6, 2006, pp. 446–451.

Ding Wang, Derong Liu, Qinglai Wei, Dongbing Zhao, and Ning Jin, "Optimal control of unknown nonaffine nonlinear discrete-time systems based on adaptive dynamic programming," submitted for publication.

Derong Liu, Ding Wang, Dongbing Zhao, Qinglai Wei, and Ning Jin, "Neural-network-based optimal control for a class of unknown discrete-time nonlinear systems using gobalized dual heuristic programming," submitted for publication.