SEQUENTIAL FRAMEWORKS FOR STATISTICS-BASED VALUE FUNCTION REPRESENTATION IN APPROXIMATE DYNAMIC PROGRAMMING

by

HUIYUAN FAN

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2008

To my wife Ping Wan and our son Kaili.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my supervising professor, Dr. Victoria Chen, for all her guidance, advice and financial support throughout the journey of my study at UT Arlington. I appreciate her leading me into an exciting and challenging research area. She is an excellent role model as a researcher and advisor, with her deep knowledge of my research field, clear thought process, superb technical writing and communication skills, and giving personality, and I will continually strive to reach her standard. Whenever my research progress stalled, she was always available with the new ideas to guide me in a promising direction.

I also want to acknowledge the wonderful support of my committee members. Dr. Frank Lu from the Department of Mechanical and Aerospace Engineering, and Dr. Jay Rosenbenger and Dr. Seoung Bum Kim from the Department of Industrial and Manufacturing System Engineering, provided valuable comments and advice that significantly improved my dissertation research.

I want to thank all of the former and current peers who worked with me for various lengths of time. I will always remember my friend Aihong Wen for first introducing me into the Center on Stochastic Modeling, Optimization, & Statistics (COSMOS). My friends, Venkata Pilla, Sheela Siddappa, Dachuan Shih, Hee-Su Hwang, Tai-Kuan Sung, Duraikannan Sundaramoorthi, Prashant Kumar Tarun, Prattana Punnakitikashem, Panaya Rattakorn, Sirimat Visoldilokpun, Chivalai Temiyasathit, Thuntee Sukchotrat, Chingfeng Lin, Passakorn Phananiramai, Chatabush Roongrat, et al., all made life in COSMOS enjoyable.

Invaluable thanks go to my wife, Ping Wan. Her constant support and understanding over the years gave me the motivation to pursue my second Ph.D. degree. I am forever grateful for her strength that carried our family through hard times. I also thank my son, Kaili, who always worked hard in his own studies, so that I did not have to worry about him while I was busy with my own studies.

I am also grateful to my father, my elder brother and my elder sister, who live in my hometown in remote China. Their teachings continue to make me a better person. I deeply miss my late mother, who passed on many years ago, but whose maternal love eternally remains in my heart.

July 16, 2008

ABSTRACT

SEQUENTIAL FRAMEWORKS FOR STATISTICS-BASED VALUE FUNCTION REPRESENTATION IN APPROXIMATE DYNAMIC PROGRAMMING

HUIYUAN FAN, Ph.D.

The University of Texas at Arlington, 2008

Supervising Professor: Victoria C. P. Chen

Dynamic programming (DP) was derived by Bellman in 1957 as a mathematical programming method to solve multistage decision problems. The DP concept has great potential, but it only can solve small problems exactly under very limiting restrictions such as linear dynamics, quadratic cost, Gaussian random variable, etc. With advances in computational power, a new family of dynamic programming known as approximate dynamic programming (ADP) has emerged. A real-world DP problem is often high-dimensional and stochastic with continuous state variables. To handle this type of problem, ADP methods discretize/sample the continuous state space and approximate the future value (or cost-to-go) function by some statistical modeling techniques. The earliest strategies used full finite grid discretizations with multilinear or spline interpolation. Under a statistical perspective, more efficient design of experiments methods, such as orthogonal arrays (OAs) and number theoretic methods (NTMs), combined with flexible statistical modeling methods, such as multivariate adaptive regression splines (MARS) and neural networks (NNs), enabled approximate solutions to higher-dimensional problems.

The above statistical perspective still maintains a traditional DP solution approach. By contrast, machine learning approaches evolved in the artificial intelligence community to approximately "learn" the DP solution via an iterative search. These learning based methods fall under various names, including reinforcement learning (RL), adaptive critic, and neuro-dynamic programming. These learning based ADP methods were initiated from the theories of psychology and animal learning, but now have evolved as an important branch-stream of machine learning methods. Compared with the previous ADP methods developed in statistical and operations research communities, this kind of methods can adaptively and gradually learn the DP solution with certain learning algorithms. However, the practical success of RL approaches is still limited due to extremely high computational cost.

The RL approach to ADP is sequential in nature, and this dissertation seeks to improve upon the statistical perspective by developing sequential approaches in the spirit of RL. The existing ADP methods assume fixed model structures for approximating the future value (or cost-to-go) function. In practice, this model structure is difficult to identify, in many cases requiring a time-consuming trial-and-error process. In addition, the statistical perspective requires determination of the discretization sample size in advance. The iterative approach of RL, automatically determines sample size and uses system dynamics to explore the state space. In this dissertation, two types of sequential algorithms are developed. The first type uses a sequential concept based on consistency theory to both identify the approximating model structure and determine sample size. The second type uses system dynamics to sequentially identify the state space region.

The first type of sequential algorithm builds an adaptive value function approximation while the size of the state space sample grows. In the statistical perspective to ADP, there are two components to value function approximation: (1) design of experiments and (2) statistical modeling. For design of experiments, NTM low-discrepancy sequence sampling techniques are employed because of the sequential nature in which they are generated. For statistical modeling, feed-forward NN models are used because of their consistency ability. The adaptive value function approximation (AVFA) is implemented in each stage of the backward-solving DP framework. Three different AVFA algorithms are derived based on the consistency concept and then tested on a nine-dimensional inventory forecasting problem. The first algorithm increments the size of the state space training data in each sequential step, and for each sample size a successive model search process is performed to find an optimal NN model. The second algorithm improves on the first by reducing the computation of the successive model search process. The third algorithm uses a more natural perspective of the consistency concept, where in each step, either the sample size is incremented or the complexity of the NN model is grown; an optimal NN model is not directly sought in this algorithm, but rather the consistency concept implies that convergence will yield the optimal model.

The second type of sequential algorithm conducts state space exploration forwards through the stages. The objective is to identify the appropriate region in the state space to model the future value function. The specification of this region is needed in the design of experiments component of the statistical perspective; however, in practice, this region is typically unknown. Hence, this sequential state space exploration (SSSE) approach fills an important need. Since decisions are needed to move forward through the stages, both random and optimal decisions are explored.

The SSSE algorithm is combined with the AVFA algorithm to yield a novel self-organized forward-backward ADP solution framework. This framework consists of two phases. The first phase has a single forward SSSE step using random decisions to identify an initial state space region for each stage and a single backward AVFA step to build initial future value function approximations over these regions. The second phase iterates between a forward SSSE step with optimal decisions and a backward AVFA step to update the state space regions and the future value function approximations until the state space regions stop changing. This new sequential SSSE-AVFA algorithm is also tested on a nine-dimensional stochastic inventory forecasting problem.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Dynamic Programming: The Basic Elements and the Various Types

Dynamic programming (DP) was derived by Bellman (1957) [8] as a mathematical programming method of solving multistage decision-process problems. It has been widely used in a very broad variety of applications, ranging from many branches of engineering to statistics, economics, finance, and some of the social science (Bertsekas 1995 [11]).

Problems to which DP can be applied are usually stated in the following terms (Cooper and Cooper 1981 [20]). A physical, operational, or conceptual system is considered to evolve through a series of consecutive *stages*. At each stage the system can be characterized by a relatively small set of parameters called the *state variables* or *state vector* (*state* for short). At each stage, and no matter what state the system is in, one or more *decisions* must be made. These decisions (also called *decision variables* or the *decision vector*) may depend on either stage or state or both. It is also true that the past history of the system, i.e., how it got to the current stage and state, are of no importance. When a decision is made a *return* (either profit or cost) is obtained and the system undergoes a *transition* to the next stage. The return is determined by a known single-valued function of the input state. The overall purpose of the staged process is to maximize the total profit or minimize the total cost as some function of the state and the decision variables.

The above terms actually form the basic elements for DP, which can be more exactly described as follows.

1.1.1 Stages

DP is an approach that transforms a complex optimization problem into a sequence of simpler subproblems. Stages are then used to index the subproblems. The resulting multistage problem is then solved stage by stage. Each one-stage problem is dealt with as an ordinary optimization problem with its solution helping to define the characteristics of the next one-stage problem in the sequence. Most often, the stages represent different time periods in the problem's planning horizon. For example, the problem of determining the level of inventory of a single commodity can be stated as a DP problem. If we can order only at the beginning of each month and we want an optimal ordering number for the coming year, we could decompose the problem into 12 stages, each representing the ordering decision at the beginning of the corresponding month. Sometimes the stages do not have time implications. For example, the problem of determining the shortest path for a traveler's trip from a city to another city with a certain number of intermediate stops in some optional cities can be formulated as a DP problem, where the stages can be defined to be the number of the stops made by the traveler.

As a variable, stage is often discrete. In this case, a DP problem is considered to be a discrete DP. Thus, a DP problem is discrete or continuous depending on the structure of the stage process. If the problem has a finite number of stages, we also say that the DP is finite horizon; otherwise, the DP is infinite horizon. There are situations in which the stage is considered to be a continuous variable. They are continuous DP cases that often occur when the stage is time and a decision can be made at any arbitrary time. Obviously, such a case has infinite horizon. A continuous stage structure is frequently encountered in problems which are often considered by the classical methods of calculus of variations (Cooper and Cooper 1981 [20]).

1.1.2 States

Associated with each stage of the optimization problem are the states of the process. The states reflect the information required to fully assess the consequences that the current decision has upon future actions. In the inventory problem given in the previous subsection, each stage has only one variable describing the state: the inventory level on hand of the single commodity. The shortest-path problem has the state variable representing the optional cities for each stop.

The specification of the states of the system is perhaps the most critical of a DP problem. There are no set rules for doing this. In fact, for most part, this is an art often requiring creativity and subtle insight about the problem to be studied. The essential properties that should motivate the selection of states include: i) The state should convey enough information to make future decisions without regard to how the process reached the current state (i.e., Markovian property); and ii) The number of states should be small to reduce the computational effort, namely, to alleviate the curse of dimensionality. As a variable, the state may be discrete or continuous or mixed. It may also have high dimension.

1.1.3 Decisions and Policies

Decisions are choices made for a given state in a given stage and affect the state of the next stage. In the inventory problem the decision is the amount of the commodity to order at the beginning of each month. In the shortest-path problem, the decision is the city for the next stop. The state of the system must contain all the information that is required to identify the set of allowable decisions at some given stage. Therefore, associated with each state, there is one or a set of decisions for this state. As a variable, decisions can be discrete or continuous or mixed.

The concept of a *policy* is closely related to decisions. A policy is defined as an ordered set of decisions by stage and state. For example, for the inventory problem, a policy can be the 12 ordering amounts of the commodity, dependent on the inventory levels for the 12 months, while for the shortest-path problem a policy specifies the order in which the traveler visits the cities. If the ordered set of decisions is from a mid-point stage to the final stage, it is called a *subpolicy*.

1.1.4 Transition Function

The process under study in a DP problem passes from stage to stage. As it does so it moves through one of the states to another state, as a result of the decision which is made in a given state at a given stage. What this means is that if the process is in the given state, choosing a decision for that state determines a set of states to which the process can move from the current state. The function to determine that set of states for the next stage is called a transition function or transition equation and it determines the way in which the process evolves from state to state. The transition function may be two basically different kinds, which we might call "deterministic" and "nondeterministic." The simplest case is deterministic, which can be illustrated by the shortest-path problem: If the traveler is in a given city, he will move to another city with complete certainty because the distances between each two of the associated cities are known. A nondeterministic case is the one where we do not know the outcome of a decision with certainty but the state and decision at each stage is determined as a result of some known probability distribution governing that transitions that occurs. Such a system is called a *stochastic system*. There are many examples of this kind of the systems. The inventory problem is an example if the commodity demand is a random variable. Markov processes fall in this category as well (Cooper and Cooper 1981 [20]). Transition functions are generally assumed known, but if unknown, they should be observable.

1.1.5 Stage Return, Total Return and Future Value Function

Since a DP problem is an optimization problem, there is an objective function which can be evaluated for given policies. In DP the term "return" represents the objective value. A return is a function with respect to decisions, typically, a profit, a cost, a distance, a yield, or consumption of a product. A *stage return* is something which a system generates over one stage of a process. The *total return* of a process or system depends upon the decisions that are made at each stage, i.e., the policies. The total return is some combination of the stage returns (e.g., a sum or product) which are accumulated as the process moves from state to state (or equivalently, stage to stage). In the inventory problem, the stage return can be the ordering and inventory-carrying cost for a month, and the total return can be the total of that cost for all the 12 months. For the shortest-path problem, the stage return can be the distance between the current city and the next, and the total return can be the total distance of a path chosen by the traveler. The purpose of solving a DP problem is to find the optimal total return. It should also be pointed out that it is possible for the return functions to vary from stage to stage. All that is necessary is a well-defined stage return for each stage. In addition, the return function is generally assumed to be known, but if unknown, it should be observable.

The *future value function* or *cost-to-go function* is the key to a DP solution and provides the optimal return to operate the system from a given stage to the end of the time horizon with respect to the states in that given stage. It corresponds to the optimal subpolicies and is based on "the principle of optimality." A more detailed description of this concept will be made in the next chapter.

1.1.6 The Various Types of DP

Given the different characteristics of the basic elements represented in potential problems, there can be various types of DP: discrete or continuous stages, deterministic or nondeterministic (stochastic) transitions, finite or infinite horizon, discrete or continuous states, discrete or continuous decisions. A one-factor dependent binary classification is the simplest DP decision process. A practical DP problem can usually be represented by a complicated combination of these simple classifications. One can easily see that the number of combinations could be quite large.

This dissertation focuses on finite horizon, stochastic dynamic problems with continuous state and decision spaces. Stochastic optimal control problems, such as a water reservoir network, are typically formulated this way. In water reservoir network management problems, the water reservoirs are arranged with a certain order and links, through which one reservoir can release water to the others. The amount of water flowing into the reservoirs from external sources are stochastic. The state vector consists of the water level and the stochastic external inflow for each reservoir. The decision vector is the amount of water released from a reservoir. The objective is to maintain a reservoir's water level to a target and to maximize a benefit.

1.2 "The Principle of Optimality"—the Spirit of Dynamic Programming

The spirit of DP is known as the "principle of optimality." Bellman (Bellman 1957 [8]) stated this principle as follows: "An optimal policy has the property that whatever the initial state and the initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from first decision." It is this principle of optimality that ensures the multistage DP problem to be solvable with "recurrence relations," the fundamental elements of a DP solution. There are two conditions that must be met in order for the principle of optimality to be invoked and lead to recurrence relations and, hence, the valid application of DP (Cooper and Cooper 1981 [20]). The first is separability of the objective function. The objective function must be separable in the sense that, for all stages from the current stage to the final stage, the effects of all future stages on the objective function of a multistage process depend only on the current state and upon the decisions from the current stage through the final stage. The second condition is the state separation property. This means that after a decision is made in the next stage, the state in this stage that results from that decision depends only on the state from which it transitioned and the decision made in the next stage, and does not depend on the previous state in any previous stages.

1.3 Main Advantages and Limitations

As mentioned earlier, one of the main advantages of DP is that it transforms a single high-dimensional optimization problem into a sequence of small optimization problems which can be solved one at a time. A second extremely important advantage of DP is that DP determines the absolute (global) optima rather than relative (local) optima.

There are, however, certain limitations to the use of DP. The two principle ones are mentioned herein. The first is that it assumes a fully observed system. In other words, it assumes that the system is able to make its decisions based on the full knowledge of state space. Unfortunately, in the real world the accurate and comprehensive process knowledge of complex nonlinear control system is rarely known *a priori*. The second one is the so-called "curse of dimensionality": the required DP calculations become cost-prohibitive as the number of states and decisions become large, with an exponential growth in the computation with respect to the dimension. When a real-world problem is a high-dimensional, continuous-state (and continuous-decision) stochastic DP problem, both of the above obstacles become intractable for the classical solution methods.

1.4 Challenges for Practical Applications

With the main limitations listed in the preceding section, DP is only able to solve very small problems with very few states and decisions. When a problem to be solved is high-dimensional or the state space (and decision space) is continuous, traditional DP methods as described by Bellman and the other authors (see, for example, Nemhauser 1966 [58]; Jacobs 1967 [40]; Cooper and Cooper 1981 [20]; Denardo 1982 [24]) can no longer be applicable, because the aforementioned limitations become intractable. One has to seek some "approximate" ways to solve such a problem. Given recent advances in computational power, numerical methods have been developed by many researchers to solve DP problems, which has yielded a new family of dynamic programming–approximate dynamic programming (ADP).

High-dimensional DP problems with continuous state spaces usually require proper discretization of the continuous state space, and approximation of the future value (or cost-to-go) function by some statistical modeling techniques. The earliest and also most natural strategies were to form a finite grid of disretization points in the state space, then to use multilinear or spline interpolation (see Johnson et al. 1993 [42]). When the state variables are few and the grid size can be small, it is possible to get a very precise numerical solution for such a discrete DP problem. However, when the number of state variables increases, the number of discretization points in the full grid grows exponentially (again, a form of the "curse of dimensionality"). In order to reduce the number of discretization points and the corresponding computational effort, statistical design of experiments (DoE) methods, such as orthogonal arrays (OAs), Latin hypercubes, and number theoretic methods (NTMs), were employed to more efficiently sample the continuous state space of a DP problem (Chen 1999 [15]; Cervellera et al. 2007 [14]; Wen 2005 [82]). It has been proved (Chen et al. 1999 [16]) that with OA discretizations the growth in the number of state space discretization points is reduced from exponential to polynomial in the state space dimension. Corresponding to the use of the efficient discretization methods for the state space, more effective statistical modeling methods, including multivariate adaptive regression splines (MARS) and neural networks (NN) have been applied (Chen 1999 [15]; Cervellera et al. 2007 [14]; Wen 2005 [82]).

Researchers in the fields of machine learning in the artificial intelligence community have shown a strong interest in solving a DP problem. In 1980s, they began to explore the possibility of applying the theories of psychology and animal learning to approximately solving DP (Lendaris and Neidhoefer 2004 [49]; Werbos 2004 [87]; Lee and Lee 2004 [48]). Their associated research has formed an active branch of modern ADP study and yielded ADP methods with a machine-learning perspective. In contrast to the "statistical perspective" in the previous paragraph, these kinds of methods are represented by "reinforcement learning (RL)," which was initially inspired by studies of animal learning in experimental psychology. The RL family also includes "active critic" and "neuro-dynamic programming." Although many early scientists expected the RL based ADP methods to have high potential to handle dynamic optimization problems as large and complex as what the smallest mammal brain can learn to handle (Werbos 2004 [87]), today's reality is very far from the original hopes by scientists (Werbos 2007 [89]).

Both the traditional backward-solving DP approach (including the newer statistical perspective) and the RL approach have their advantages and disadvantages in solving real-world dynamic problems. Traditional methods usually keep the same evolving frame as in an exact DP solution procedure and are appropriate for finite-horizon problems, where the backward solution procedure is applied in "batched" manner from stage to stage. In contrast, RL methods use a forward solution procedure to seek a time invariant steady-state solution and, hence, are appropriate for infinite-horizon problems. Traditional methods cannot be implemented on-line, and depend on traditional optimization algorithms, typically assuming convexity, which is not appropriate for very nonlinear problems. By contrast, RL methods are deemed good at dealing with problems with extremely high complexity, and can be implemented online by trial-and-error. However, a fatal drawback of RL methods is their costly online learning (or training). Real world systems (e.g., a very dangerous chemical process system) cannot afford a large-scale, online trial-and-error training. In short, new practical ADP methods are needed to handle the complexity of real-world dynamic optimization problems.

1.5 Objective and Overview

The ultimate objective of this dissertation research is to develop new methodologis based on a statistical perspective that integrates some advantages of the RL perspective.

Recall that the key to an ADP solution is the future value (or cost-to-go) function approximation. The shortcoming in the dominating ADP approach for approximating the value (or other) function is the assumption that the approximating model is fixed in structure. In reality, this approximation must be tuned carefully, usually by trial-and-error, in order to get a good solution. It is well known in the statistics community that model structure cannot be ignored in practice, but identification of an appropriate structure is largely ignored in the ADP literature. In this research, a sequential concept is first introduced to determine the approximating model structure. The concept has a theoretical foundation based on consistency theory that is induced from asymptotic convergence properties proven in statistical nonparametric inference (Geman et al. [35]). To realize this sequential idea a nonparametric inference technique is required. In this study feed-forward neural networks (NNs) are taken as nonparametric models. The sequential concept in the meanwhile requires a “sequential design” process for the state space sampling (discretization). The statistical perspective currently assumes a batch sampling (DoE) strategy, where the sample size of the batch is fixed in advance. The sequential concept enables both adaptive model structure identification and adaptive sample size determination. The sequential concept borrows an RL concept, first by its sequential nature, and second by its adaptive sampling scheme. For state space discretization, low-discrepancy sequence (NTM) sampling techniques are employed because of the sequential nature in which they are generated. Three algorithms are developed from the sequential concept that adaptively build the value function approximation while the size of the state space sample grows. The new adaptive value function approximation (AVFA) algorithms hence simultaneously identify model structure and determine sample size. They can be easily employed in the traditional backward solution framework to yield a new sequential framework that keeps the backward style but makes the future-value-function modeling process more optimal and automatic.

The specification of the state space region is needed by the DoE component of the ADP statistical perspective. In most practical ADP problems the state space region is typically unknown and proper specification is difficult. To handle this important practical issue, this dissertation studies a sequential method to conduct state space exploration. This sequential state space exploration (SSSE) approach moves forward, like an RL approach, stage by stage, through either random or optimal sampling of the decision space. By combining the SSSE and AVFA approaches, a fully sequential and self-organized forward-backward ADP solution framework (SSSE-AVFA) is finally produced in this research, that can iteratively update the models for the future value function and the state space region, and can solve the finite horizon ADP problems by taking advantage of the strengths from both forward and backward processes.

The proposed sequential algorithms and frameworks make it easier to apply ADP to real world problems for the following reasons. First, model structure and sample size are both difficult to specify in practice, so an AVFA algorithm eliminates the need to specify these in advance. Second, the AVFA algorithms potentially avoid unnecessary computation, due to overly-large sample sizes and the blind trial-and-error process to determine the model structure. Thirdly, in most real world applications, the state space region is unknown and is not straightforward to define, the SSSE approach and the SSSE-AVFA framework automatically determine the state space region and avoid the use of a poor model constructed over an incorrectly specified state space region. Finally, all the new algorithms and frameworks are easy to be implemented on-line, which is an advantage for applications in which the sample data only can be obtained gradually.

CHAPTER 2

DP PROBLEMS AND THE METHODS

2.1 DP Formulations

2.1.1 Finite-Horizon Model

DP can be defined as either a minimization (of cost) process or a maximization (of profit) process, with some appropriate mathematical conversion between the two optimization forms. For simplicity and without loss of generality in this study we only consider minimization problems. We start with a finite horizon deterministic DP formulation in recursive form,

$$V_t(x_t) = \min_{u_t} \{c_t(x_t, u_t) + V_{t+1}(x_{t+1})\} \tag{2.1}$$

where T is the time horizon and $t = 1, \ldots, T$, x_t is the state vector, u_t is the decision vector, $c_t(\cdot)$ is a known cost function for stage t, and $V_t(\cdot)$ is the future value function. It is assumed that the "world ends" after the final stage T. The state vector, x_t, describes the state of the system at the beginning of stage t. The decision vector, u_t, is the only variable we can control and is chosen to minimize present plus future cost. The transition from x_t to x_{t+1} is defined by

$$x_{t+1} = g_t(x_t, u_t), \tag{2.2}$$

where $g_t(\cdot)$ is a known function. The future value function (cost-to-go function), $V_t(x_t)$, provides the minimal cost of operation for stages t through T given the system is in state x_t entering stage t. The goal is to find the future value functions $V_t(\cdot)$ and the optimal decisions u_t^*. The DP is solved backwards recursively, first finding $V_T(\cdot)$, and then using it and subsequent $V_t(\cdot)$ to solve $V_{T-1}(\cdot), \ldots, V_1(\cdot)$. Once we have $V_t(x_t)$ for $t = T, \ldots, 1$

that amounts to having the surfaces $(x_t, V_t(x_t))$ and (x_t, u_t^*) for $t = 1, \ldots, T$. Given x_1, we know u_1 and $V_1(x_1)$, and $x_2 = g_1(x_1, u_1)$ gives us u_2, etc. Thus given the initial state of the system, we know the optimal operation for the system through stage T.

The simplest case is a discrete-state, discrete-decision, finite horizon deterministic DP formulation, where the state vector, x_t, and the decision vector, u_t, each take on a finite number of values. The sets of values can be different for each stage t,

$$x_t \in \left\{ x_t^1, x_t^2, \ldots, x_{(t)}^{N_x} \right\},$$
$$u_t \in \left\{ u_t^1, u_t^2, \ldots, u_{(t)}^{N_u} \right\}.$$

If the sets are different for each stage t, then the transition function, $g_t(x_t, u_t)$ will also be different for each stage t. Regardless of whether the decision is finite or discrete-infinite or continuous, the transition from x_t to x_{t+1} can only lead to points contained in the set of values for x_{t+1}. Thus for all stages, $V_t(x_t)$ only needs to be solved at a finite number of points. A discrete-state DP can also have a countably infinite number of states, which would not be directly solvable using the backward approach.

A finite horizon stochastic DP formulation employs a random variable, as follows

$$V_t(x_t) = \min_{u_t} E \left\{ c_t(x_t, u_t, \epsilon_t) + V_{t+1}(x_{t+1}) \right\}, \tag{2.3}$$

where ϵ_t is a random variable over which the expectation is taken. The transition from x_t to x_{t+1} is defined by

$$x_{t+1} = g_t(x_t, u_t, \epsilon_t). \tag{2.4}$$

For a discrete-state stochastic DP, the state vector, x_t, takes on a finite or countably-infinite number of values, so the random variable ϵ_t gives rise to *transition probabilities* (or a *transition probability matrix*) between the discrete states from one stage to the next. The expectation is calculated as a weighted summation (see, for example, Bertsekas 1995 [11].

For continuous-state DP, x_t is continuous, so, as in the countably-infinite case, it is impossible to directly solve $V_t(x_t)$ for all values of x_t using the backward approach. However, a solution may be approximated. First, we must limit ourselves to a finite number of points, N, in the range of x_t for each stage t. Then we can solve for $V_t(x_t)$ at those points and find the associated optimal decisions u_t^*. Finally, we can use some kind of approximation of the surfaces $(x_t, V_t(x_t))$ and (x_t, u_t^*). Let $\hat{V}_{t+1}(\cdot)$ and $\hat{u}_{t+1}^*(\cdot)$ be the approximations for stage $t+1$ in a deterministic DP model. For each stage $t < T$, we now solve

$$\tilde{V}_t(x_t) = \min_{u_t} \left\{ c_t(x_t, u_t) + \hat{V}_{t+1}(x_{t+1}) \right\} \tag{2.5}$$

$$\tilde{u}_{t+1}^*(x_t) = \arg\min_{u_t} \left\{ c_t(x_t, u_t) + \hat{V}_{t+1}(x_{t+1}) \right\} \tag{2.6}$$

at the finite set of N values chosen in the range of x_t. Since the minimization in Equation (2.5) depends on $\hat{V}_{t+1}(x_{t+1})$, the solution to the minimizations are already approximations. Thus we denote this version of the future value function by $\tilde{V}_t(x_t)$, and $\hat{V}_t(\cdot)$ will actually be an approximation of the function $\tilde{V}_t(\cdot)$.

For a continuous-state, finite horizon stochastic DP model, the solution procedure is similar to continuous-state, finite horizon deterministic DP, except that an expectation must be calculated. Specifically, the random variable ϵ_t in Equation (2.3) cannot be represented by a transition probability matrix. Instead, the expectation can only be computed for given state x_t and decision u_t. If ϵ_t is a discrete random variable, then the expectation is a summation. However, if ϵ_t is a continuous random variable, then an approximate solution procedure would additionally estimate the required expectation through sampling the probability distribution of ϵ_t. This is essentially a numerical integration issue.

2.1.2 Infinite-Horizon Model

For infinite horizon DP problems, everything is the same as in the previous section, except now $t = 0, 1, \ldots$ to infinity. Without loss of generality, here we directly discuss the stochastic version. Such a problem is usually formulated as a discounted model that takes the long-run return (profit or cost) of each stage into account, but the return that is received in the future is usually geometrically discounted according to a discount factor. In addition, these problems are often considered stationary, that is, the system transition equation, the return per stage, and the random disturbance statistics do not change from one stage to the next. We thus try to minimize, for instance, the total cost over an infinite number of stages, given by

$$V_1(x_1) = \min_{\{u_1, u_2, \ldots,\}} E\left\{\sum_{t=1}^{\infty} \alpha^t c(x_t, u_t, \epsilon_t)\right\}. \tag{2.7}$$

where $V_1(x_1)$ is the total cost with a initial state x_1 and $u_1, u_2, \ldots$ are a sequence of decisions corresponding to a policy, and α is the discount factor with $0 < \alpha < 1$. One also can define the cost-to-go function for the discounted infinite horizon model as

$$V_t(x_t) = \min_{\{u_t, u_{t+1}, \ldots,\}} E\left\{\sum_{k=t}^{\infty} \alpha^{k-t} c(x_k, u_k, \epsilon_k)\right\}. \tag{2.8}$$

2.2 Finite-Horizon Solution Methods

A general algorithm for solving continuous-state (or near continuous) DP systems using a statistical perspective can be stated as follows (Chen 1999 [15]). In the following steps 2(a) and 3(a), the expectation is taken over the random vector ϵ_j.

1. Using design of experiments, choose N discretization points in the state space $\{x_{jt}\}_{j=1}^{N}$ for the t-th stage, $t = 1, \ldots, T$, and $x_{jt} \in R^n$.
2. In the last stage T,
   (a) for each discretization points x_{jT}, $j = 1, \ldots, N$, solve
   $V_T(x_{jT}) = \min_{u_{jT}} E\{c_T(x_{jT}, u_{jT}, \epsilon_j)\}$, then

(b) approximate $V_T(x_T)$ with $\hat{V}_T(x_T)$, for all $x_T \in R^n$, by fitting, say, a statistical model to the data for V_T from step 2(a).

3. In each stage $t = T-1, \ldots, 1$,

   (a) for each disretization point x_{jT}, $j = 1, \ldots, N$, solve
   $\tilde{V}_t(x_{jt}) = \min_{u_{jt}} E\left\{c_t(x_{jt}, u_{jt}, \epsilon_j) + \hat{V}_{t+1}(g_t(x_{jt}, u_{jt}, \epsilon_j))\right)$, then

   (b) approximate $\tilde{V}_t(x_{jt})$ with $\hat{V}_{t+1}(x_t)$ for all $x_t \in R^n$, as in step 2(b).

In step 2, $V_T(x_T)$ can be exact. In step 3, the minimization depends on $V_{t+1}(x_{t+1})$, so it is not exact. Thus, we denote the minimum value at x_{jt} by $\tilde{V}_t(x_{jt})$ and the approximation of the future value function by $\hat{V}_t(\cdot)$.

From the previous discussion and the above algorithm it is clear that the statistical perspective numerically solves a continuous-state DP problem by discretizing the state space, so that the minimization can be carried out only for a finite number of states. However, the state space discretization can be subject to the "curse of dimensionality" as the dimension of the state space increases. Therefore, research in continuous-state DP has focused on methods that reduce computational efforts since the first uniform grid discretization methods were introduced by Bellman (1957) [8]. Specifically, Foufoula-Georgiou and Kitanidis (1988) [32] proposed an algorithm, gradient DP, which employed cubic Hermite polynomials to approximate the future value function. Johnson et al. (1993) [42] compared numerical solution methods using multilinear, Hermite gradient DP, and tensor-product cubic spline interpolation on a four-reservoir problem. They showed that cubic splines required fewer grid levels in each dimension, hence, reducing computational time. Unfortunately, the above earlier methods were based on a full grid points (i.e., a full factorial experimental design), which grows exponentially with the number of dimensions.

A breakthrough in computational intractability was presented by Chen et al. (1999) [16] and Chen (1999) [15] who enabled the first truly practical numerical solution approach for high dimensions. Their method utilizes orthogonal array (OA) experimental designs and multivariate adaptive regression splines (MARS), where OAs are special subsets of full factorial experimental designs that grow only polynomially with the number of dimensions. This greatly reduced the computational effort for high-dimensional problems. Chen et al. (1999) [16] and Chen (1999) [15] achieved good accuracy compared to using a full factorial design with tensor-product cubic spline interpolation by Johnson et al. (1993) [42]. Cervellera et al. (2007) [14] introduced the use of an alternate experimental design, Latin hypercubes, and an alternate statistical modeling method, neural networks approximation methods into Chen's approach, and obtained comparable results to using OAs and MARS on a nine-dimensional inventory forecasting problem and an eight-dimensional water reservoir problem. Finally, Cervellera et al. (2007) [14] and Wen (2005) [82] additionally studied experimental designs number-theoretic methods (NTMs) and successfully solved a thirty-dimensional water reservoir problem. The methods in this dissertation build on the success of the above methods.

2.3 Infinite Horizon Solution Methods

Infinite horizon DP problems also suffer from "the curse of dimensionality" (See, for examples, Bertsekas 1995 [11]; Denardo 1982 [24]; Nemhauser 1966 [58]). The more recent reinforcement learning (RL) ADP methods based on a machine-learning perspective are applicable to infinite horizon DP problems because the learning process can be viewed as an infinite horizon process. With recent advances in computer processor speed, RL-based ADP methods have quickly attracted wide attention.

RL was initially inspired by studies of animal learning in experimental psychology, where "reinforcement" refers to the occurrence of an event, in proper relation to a response that tends to increase the probability that the response will occur again in the same situation. In a general RL model, an agent interacts with its environment through sensors (perception) and actors (actions) (Bart and Sutton 1998 [6]; Kaelbling et al. 1996 [44]). Each interaction iteration typically includes the following: the agent receives inputs that indicate the state of the environment; the agent then selects and takes an action, which yields an output; this output changes the state of the environment, transitioning it to a "better" or a "worse" state; the latter are communicated to the agent by either a "reward" or a "penalty" from the environment, and the amount of such reward/penalty has the effect of a "reinforcement" signal to the agent. The behavior of a healthy agent tends to increase the reward part of the signal, over time, through a trial-and-error learning process.

Application of RL ideas to ADP induced the concept of an "adaptive critic." In this concept, a "critic" or "critic network" represents any parameterized network used to approximate the future value function (or cost-to-go function) of the Bellman equation or to approximate any of the other various functions related to the Bellman equation. A general adaptive critic approach usually utilizes two parametric structures called the actor and the critic. The actor consists of a parameterized control law. The critic approximates a value-related function and captures the effect that the control law will have on the future cost. At any given time the critic provides guidance on how to improve the control law. In return, the actor can be used to update the critic. With an algorithm that is designed to perform successive iterations between these two operations the system may converge to an optimal solution over time (Ferrari and Stengel 2004 [30]). A related concept is "neuro-DP," coined by Bertsekas and coworkers (Bertsekas 2001 [10]) only because of the popular use of artificial neural networks as the function approximators. The concept is identical to RL if fixed structure neural network approximators are viewed as particular examples of generic parametric approximators.

The earliest researcher who made an attempt to connect the RL concept to DP was Werbos (1977 [83], 1998 [86], 2005 [88]). While pursuing his PhD studies at Harvard University in the 1960s, Werbos started thinking about how to realize a kind of RL through building systems explicitly designed to learn an approximation to DP. Werbos's initial work sparked him to continue his research in this aspect and to develop a group of adaptive critic structures in the early 1990s (1992 [85], 1998 [86]), which have been widely used and extended by others (Lendaris and Neidhoefer [49]). In 1977 Werbos proposed a critic structure which he called "heuristic DP" (Werbos 1977 [83]). It is the most basic form of the adaptive critic approach which then triggered a kind of Bellman-equation-based approximation critic approaches (also known as "temporal difference methods"). Heuristic DP uses the parametric structure called actor (or action network) to approximate the control law, and another parametric structure called critic (or critic network) to approximate the value function. Critic training does not need a system model for its calculations, but actor training, on the other hand, requires finding the derivatives of a system model with respect to the control variables. Thus, in practice, heuristic DP converges slowly.

To improve the convergence rate of heuristic DP, Werbos (1992 [85], 2007 [89]) proposed an alternative referred to as dual heuristic programming (DHP). DHP uses the critic to approximate the derivatives of value function with respect to the state variable. The actor is used to approximate the control law, as in all other adaptive critic approaches. It requires fully-specified model based algorithms to train the critic and actor. Some applications showed that the DHP method could be more efficient than heuristic DP (Ferrari and Stengel 2004 [30]). However, due to the use of derivative information, the relationships for updating the control and value-derivative functionals are more involved and, hence, introduce additional computation. Furthermore, since the DHP critic approximates a vector functional, the problem of function approximation is more challenging. Many other methodologies have been proposed to alleviate some of the difficulties mentioned above. Global dual heuristic programming (GDHP) attempts to combine the advantages of heuristic DP and DHP (Ferrari and Stengel 2004 [30]). In GDHP the critic approximates both the value function and its derivatives. Its critic training utilizes both the heuristic DP and DHP recursions, while actor training is as same as in DHP. Therefore GDHP uses models for both critic and controller training.

In addition to the adaptive critic approaches above, Werbos and other researchers presented the "action dependent (AD)" concept leading to action-dependent heuristic DP (ADHDP) (Werbos 2004 [87]), Q learning, etc. Crudely speaking, ADHDP/Q-learning methods lead to ADP approaches which do not directly require a model of the system. Although Q learning and ADHDP were independently presented by Watkins [81] and Werbos [84] in the same year (1989), both can be derived from the same recurrence equation, i.e., the Q recurrence equation as in Werbos (2004) [87] derived from the original Bellman equation. Now researchers view them as in the same category (Lendaris and Neidhoefer 2004 [49]). In ADHDP/Q-learning methods, one can train a critic to match targets of the value function based on the Q recurrence equation, using exactly the same procedure as in heuristic DP. The actor training is simplified, in that since the control variables are inputs to the critic, the derivatives of value function with respect to the control variable are obtained directly from back-propagation through the critic. Thus ADHDP/Q-learning methods use no system models in the training process. The action dependent concept can also be combined into the other adaptive critic approaches (Ferrari and Stengel 2004 [30], Lendaris and Neidhoefer 2004 [49]). In brief, the motivation behind all these ADP approaches is to achieve faster convergence to the optimal solution and to simplify the problem of function approximation.

Other ADP research based on the machine-learning perspective includes the following. Anderson et al. combined robust control theory with RL methods and developed an RL procedure that is guaranteed to remain stable even during training (Anderson at al. 2004 [1], Anderson at al. 2004 [2]). Balakrishnan and Han applied adaptive critic based neurocontrol for distributed parameter systems with practical success in truly challenging aerospace applications (Balakrishnan and Han 2004 [4]). Rosenstein and Barto introduced a supervised learning mechanism into actor-critic structures for improving the effectiveness of learning (Rosenstein and Barto 2004 [65]). Lendaris and Neidhoefer explored application issues related to ADP, in particular, to the DHP version, and designed a nonlinear controller for a hypersonic-shaped aircraft (Lendaris and Neidhoefer 2004 [49]). Powell and Van Roy studied applying adaptive DP to handle high-dimensional problems in transportation and logistics (Powell and van Roy 2004 [64]). Saeks et al. worked with a variety of adaptive critic and adaptive DP implementations where the adaptive critic was developed to be suitable for real-time applications (Saeks et al. 2002 [68]). Si et al. studied direct neuro-DP, developed a model-independent to ADHDP approach, and successfully applied it to a wireless network call admission control in an industrial scale helicopter control (Si et al. 2004 [70], Enns and Si 2004 [26]). Tsitsiklis and Van Roy explored how to more efficiently approximate the value function in ADP for large-scale systems with some feature based methods, including feature extraction and linear approximations (Tsitsiklis and van Roy 2001 [76]). Werbos, as the most important pioneer in learning-based ADP, continues to expand his work. In a recent study, he proposed a time-lagged recurrent network and a simultaneous recurrent network to try to handle some fatal problems arising in the simple feed-forward neural network approximation in ADP (Werbos 2005 [88]).

2.4 Sequential Design and Analysis of Computer Experiments

The proposed ADP methods in this dissertation implement a sequential approach motivated by sequential methods in design and analysis of computer experiments (DACE)

(Sacks et al. 1989 [67]). DACE was developed to replace time-consuming computer models or expensive physical experiments, so as to facilitate faster exploration of a complex system. Specifically, DACE uses design of experiments (DoE) and statistical modeling to efficiently represent performance measures from a computer model, typically a simulation. The statistical model is a model of the computer model, i.e., a "model of a model" called a metamodel. The computer model is run at sample points determined by DoE, and the output responses are used to fit the metamodel. The important research issues associated with metamodeling are the type and size of the experimental design and the adequacy of the metamodel (Jin et al. 2005 [41]). In conventional metamodeling approaches, sample points are generated in batch, and a statistical model is constructed based the whole batch. However, when the computer model is extremely computational, a sequential approach can reduce the number of computer model runs. Sample points are selected sequentially, allowing the sample size to be determined as data accumulate, or in other words, it allows the sampling process to be stopped as soon as there is sufficient information (Sacks et al. 1989 [67]). Based on a sequential sampling strategy, a metamodel can be updated sequentially with new sample points and the associated response evaluations. As Sacks et al. (1989) [67] point out: "Fully sequential design is, therefore, the most natural for computer experiments." In ADP research, the iterative RL approach conducts adaptive sample size determination (but not adaptive statistical modeling), while the statistical perspective assumes batch sampling and is subject to similar issues in DACE. Hence, the desire is to combine these ideas.

Sequential DACE can be used for global metamodeling or for metamodel-based design optimization (Jin et al. 2005) [41]. Sequential design for global metamodeling focuses on sequentially improving the accuracy of metamodels over the entire design space of interest. Sequential design for metamodel-based design optimization emphasizes finding the global optimum by balancing a search for the optimal points of a metamodel and a search for unexplored regions to avoid missing promising areas due to the inaccuracy of the metamodel. For the ADP statistical perspective, the sequential design for global metamodeling is more relevant because the entire design space (state space) is of interest.

In recent years, the aforementioned sequential ideas have been attempted by a few researchers and some sequential DACE methods have been developed. Sacks and colleagues (Sacks et al. 1989 [67]) presented a simple two-stage optimal design example that is one of the earliest sequential design examples. A metamodel for predicting a physical feature of a transistor circuit was constructed based on kriging model. With this example they demonstrated that sequential design was computationally cheaper and allowed adaptiveness to the data. Currin et al. then complemented the work of Sacks et al. (Currin et al. 1988 [22], Currin et al. 1991 [23]) by using product linear and product cubic correlation functions for the kriging model and proposing a posterior entropy criterion for DoE. They showed that their two-stage design for the application example by Sacks et al. was much faster than a corresponding factorial design. Osio and Amon proposed Bayesian surrogates and a sequential-adaptive optimal sampling method based on the A-optimality criterion and kriging model (Osio and Amon 1996 [61]), and the proposed method was tested with a known analytical function approximation and a thermal design of embedded electronic packages. Lin et al. proposed a sequential exploratory experimental design method (Lin 2004 [51], Lin et al. 2004a [52], Lin et al. 2004b [53]), in which data points were identified sequentially in regions with large expected prediction errors. Jin et al. proposed two criterion for sequential sampling, namely, the maximum-scaled-distance-approach in which the importance of different design variables were considered in sampling, and the cross-validation-approach in which future points were allocated in regions where there were few observed points and prediction errors (calculated based on leave-one-out-cross-validation) were large (Jin et al. 2005 [41]). Kleijnen and van Beers developed an application-driven (or customized) sequential design method that utilized distribution-free jackknifing and cross-validation to estimate the variance for a set of candidate input combinations (Kleijnen and van Beers 2004 [45]).

2.5 Statistical Modeling Methods

Various types of statistical models have been used for approximating the future value function or cost-to-go function in their ADP studies, but almost all have been used as fixed structure parametric approximators, particularly in the aforementioned RL-based ADP literature. While a sequential DoE approach can be applied with a fixed model structure, a sequential DACE approach desires a data-adaptive identification of model structure, more akin to a nonparametric statistical modeling method. Although multilayer feed forward neural networks have been employed as fixed structure parametric approximators in ADP, they can play an adaptive role in a sequential DACE approach applied to ADP. This section describes neural networks (NNs) and other candidate statistical modeling methods, including (but not limited to): multivariate adaptive regression splines (MARS), support vector regression (SVR), radial basis functions (RBFs), and classification and regression trees (CART).

2.5.1 Neural Networks

Neural networks (NNs) form a family of powerful machine learning methods that have been applied in various tasks, like regression, classification, optimal control, etc. There are many types of NNs, depending on different topological structures, neural activation functions, training (or learning) algorithms, and so on (for general background and applications see Lippmann 1987 [54], Haykin 1999 [38], among many others). In this dissertation NNs are used to approximate the future value (or cost-to-go) function in solving a DP problem.

Multilayer feed-forward NNs are known to be universal approximators (Barron 1993 [5]). Their basic topology is an interconnection of nodes (also called neurons) organized in layers. At the input layer, the nodes are the input variables and at the output layer, the nodes are the response variable(s). In between, there is usually at least one "hidden" layer to enable modeling flexibility. An activation function defines transformations between layers (e.g., input to hidden), the simplest one being a linear function and the most common being a sigmoid function. The current research uses one-hidden layer feed-forward NNs with a single output. Such a model can be written in the general form:

$$\hat{f}(x; D_N, w) = b_2 \left(\sum_{h=1}^{H} w_h \cdot b_1 \left(\sum_{i=1}^{n} v_{ih} x_i + \theta_h \right) + \gamma \right) \quad , \tag{2.9}$$

where $x \in R^n$ denotes the input variables, $b_1(\cdot)$ and $b_2(\cdot)$ are activation functions, H is the number of hidden nodes, D_N is training data set with size N, and $w = \{v_{ih}, w_h, \theta_h, \gamma_h\}$ are coefficients that we together call the network's connection weights, without loss of generality. The above NN model is a nonlinear statistical model and its weights can be estimated with an appropriate training algorithm. The most common training algorithm is the back-propagation algorithm (see Rumelhart et al. 1986 [66]).

An NN model can learn highly complex relationship when sufficiently large amount of data are given and model architectures are properly designed. Two key problems in building an NN model are how to determine the appropriate NN architecture and how to determine the training sample size so that a real problem can be well approximated within an acceptable computational cost. The majority of the existing methods address the two problems empirically in a "one-step" manner, that is, the NN architecture and the sample size are fixed before a trial. Obviously, finding an appropriate NN architecture and sample size by trial and error is a considerable task. A few scientists have made efforts in recent years to develop algorithms to adaptively identify NN model architecture, using a sequential process. Most of them, however, are limited to a fixed training data.

For a multilayer feed-forward NN, the architecture is defined with the number of hidden layers and the number of the nodes in each hidden layer. Since this dissertation only uses one-hidden-layer feed-forward NNs, the architecture search is reduced to determining the number of hidden nodes in the one hidden layer.

The best number of hidden nodes is of course problem specific. In other words, it first depends on the complexity of the underlying function to be approximated. Too few hidden nodes will yield too simple of a model with a high statistical bias or under-fitting of the data. In machine learning terms, the overly simple NN model has high training (or learning) error that can contribute to the model's generalization error, making the model perform poorly in practice. By contrast, too many hidden nodes will yield an overly complicated model with low training (learning) error, but high generalization error due to high variance from over-fitting the data. Geman et al. (1992) [35] call the above phenomena of model performance regarding to the number of hidden nodes: "bias/variance dilemma." This concept will be discussed in more detail in the next chapter of this dissertation because it is a key to the design of the new sequential methods. The other factors that can affect the appropriate number of hidden nodes include the numbers of input and output nodes, the number of training cases (sample size), the amount of noise in the data, the type of hidden node activation function, the training algorithm, etc.

For determining the number of hidden nodes for one-hidden-layer feed-forward NNs, many authors have sought some theoretical "rules of thumb." Hecht-Nelson used Kolmogorov's theorem – which states that any function of i variables may be represented by the superposition of a set of $2i+1$ univariate functions – to derive the upper bound for the required number of hidden neurons, where i is the number of inputs, and therefore, determined that $2i+1$ should be used as the upper bound on the number of hidden nodes required for a one-hidden-layer back-propagation network (see Feng at al. 2006 [29]). Lawrence and Fredrickson (1998) [47] suggested that a best estimation for the number of hidden nodes is half of the sum of inputs and outputs. They further proposed an alternate criterion that relates the training data size to the number of hidden nodes as $N/10-i-o \leq h \leq N/2-r-o$, where N is the sample size for training, and o is the number of output nodes. Marchandani and Cao (1989) [56] derived the relationship $h = i\log_2 P$, where P is the number of training patterns (the term for "training sample size" in the pattern recognition literature). Baum and Haussler (1989) [7] presented some expressions related to training patterns, nodes, and weights under the consideration of various statistical confidence levels. Lippmann (1987) [54] indicated that the maximum number of hidden nodes for one-hidden-layer network is $O(i+1)$. Finally, BrainMaker Neural Network Software Company lists in its website (http://www.calsci.com/BrainIndex.htm) a quite complete list of "rules of thumb" for choosing an NN architecture. As the company points out in the website, these rules of thumb are impractical (even nonsensical as commented by the website) in many cases because they ignore the number of training cases, the amount of noise in the targets, and the complexity of the function.

Determining the appropriate sample size for training a NN architecture is much less studied. In theory, there is never an upper bound on the sample size for a function approximation or regression task since more samples always helps to build a better model (asymptotic convergence property). The only reason to seek a proper sample size is because of the limitations on obtaining data and on computational power. In such a situation, there is of course no reasonable method available to select a sample size. Nevertheless, Lawrence and Fredrickson (1998) [47] suggest empirical bounds, $2(i+h+o) \leq N \leq 10(i+h+o)$, to estimate the best number of training samples.

In recent years some methods on how to sequentially build NN models have been developed by a few scientists. Zhang (1994) [92] proposed a incremental learning algorithm, called "selective learning with flexible," to adaptively build a NN structure. The algorithm starts with a small structure and a small subset of the full data set, and then incrementally adds hidden nodes and training data to identify the NN structure, where the generalization error is based on the full data set. The theoretical background of the generalization error calculation for this method is unclear. Zhang and Morris (1998) [93] developed a sequential learning approach, in which hidden neurons are added one at a time. The procedure starts with a single hidden neuron and sequentially increases the number of hidden neurons until the model error is sufficiently small. The algorithm combined the classical Gram-Schmidt orthogonalization method to train the weights that link the newly added nodes. Its sequential nature, however, only addresses the NN architecture and not sample size. Liang and Huang (2006) [50] presented a online sequential learning algorithm for one-hidden-layer feed-forward NNs with additive or radial basis function hidden nodes. In their method, the hidden nodes can be sequentially added, while the hidden weights are randomly produced and the output weights are analytically determined. Ma and Khorasabi (2005) [55] proposed a constructive learning algorithm for one-hidden-layer feed-forward NNs. The hidden nodes are added with activation functions that are functions of orthonormal Hermite polynomials. The literature in evolutionary learning methods to build NN models has grown in recent years (see for example Yao (1999) [91]). The basic idea is that a population of different NN structures is evolved by an evolutionary algorithm consisting of the operations of selection, mutation, and crossover, etc. These methods usually are time-consuming due to the dependence on evolutionary algorithms. It should be noted that all these adaptive methods stress how to algorithmically update the weights while searching for the appropriate NN structure and usually require a fixed data set.

2.5.2 Other Candidate Methods

2.5.2.1 Multivariate Adaptive Regression Splines

The multivariate adaptive regression splines (MARS) algorithm was developed by Friedman (1991) [33] as a statistical modeling method for nonparametric regression. MARS uses a forward stepwise algorithm to "split" on knots, building an approximation of the form

$$\hat{f}(x) = \beta_0 + \sum_{v=1}^{M} \beta_v \left\{ \prod_{l=1}^{L_v} \left[s_{l,v} \left(x_{i(l,v)} - \kappa_{l,v} \right) \right]_+ \right\}, \tag{2.10}$$

where $[z]_+ = max\{0, z\}$ and M is the number of basis functions; for the v-th basis function, β_v is the coefficient and L_v is the number of splits in the product; and for the l-th split of the v-th basis function, the associated sign $s_{l,v}$ is either $+1$ or -1, $x_{i(l,v)}$ is the associated input variable, and $\kappa_{l,v}$ is the associated knot. A backward stepwise algorithm may also be applied to "prune" the model. Smoothing is an option to provide continuous derivatives at the knots. MARS has the added advantage of conducting variable selection while building the model. Only variables that are important for predicting the response are selected for inclusion in the model.

The key model input parameter for limiting computational effort is the maximum number of basis functions; however, this is often difficult to specify. Instead, Tsai and Chen (2005) [74] developed more flexible stopping rules based on the coefficient of determination, defined as the fraction of variability explained by the model. In addition, Tsai et al. (2004) [75] have developed a parallel computing algorithm for MARS.

2.5.2.2 Support Vector Regression

Support vector machines were originally developed to solve classification problems in supervised learning (see Vapnik 1995 [77]) and have recently been successfully extended to nonlinear regression estimation (Vapnik et al. 1997 [78]; Cristianini and Shawe-Taylor

2000 00). Via a nonlinear mapping, support vector regression seeks to map the input variables into a high-dimensional feature space, in which linear computations are conducted. The model form $\hat{f} = \langle w, \Phi(x) \rangle + b$, where w is a weight vector, b is a bias (constant) term, and function $\langle \cdot \rangle$ is the inner product in an M-dimensional feature space $\Phi(x) = \{\phi_m(x)\}_{m=1}^{M}$. A support vector regression approximation minimizes the regularized risk function $R_{reg}(w) = C\left[\frac{1}{N}\sum L_j\right] + \frac{1}{2} \parallel w \parallel^2$, where C is a pre-specified constant to control the trade-off between complexity and accuracy, and L_j is Vapnik's ε-insensitive loss function. The solution has the form

$$\hat{f}(x) = \sum_{j=1}^{N} \left(\alpha_j^* - \alpha_j\right) K\left(x_j, x\right) + b, \tag{2.11}$$

where α_j^* and α_j are Lagrange multipliers that satisfy $\alpha_j^*, \alpha_j \geq 0$ and $\alpha_j^* \alpha_j = 0$, and $K\left(x_j, x\right) = \langle \Phi\left(x_j\right), \Phi\left(x\right) \rangle$ is called a "kernel" function. Support vector machines are related to several forms of modeling, including spline models, kriging, and NNs (Wahba 1990 [79]; Gao et al. 2002 [34]; Andras 2002 [3]). Support vector regression has been recently attempted for value function approximation in ADP (Dietterich and Wang 2002 [25]; Jung and Uthmann 2004 [43]).

2.5.2.3 Radial Basis Functions

Radial basis functions were first introduced by Powell (1992) [63] to solve real multivariate interpolation problems. A radial basis function is a real-valued function whose value depends only on the distance from a center c, so that $\phi(x, c) = \phi\left(\parallel x - c \parallel\right)$, where the norm is usually Euclidean distance. Radial basis functions are typically used to build up function approximations of the form

$$\hat{f}(x) = \sum_{i=1}^{N} w_i \phi\left(\parallel x - c_i \parallel\right), \tag{2.12}$$

where the approximation function $\hat{f}(x)$ is represented as a sum of N radial basis functions, each associated with a different center c_i, and weighted by an appropriate coefficient w_i. Radial basis function models may also be interpreted as a simple single-layer type of neural network, called a radial basis function network, though it was originally developed unrelated to NNs.

2.5.2.4 Classification and Regression Trees

The classification and regression trees (CART) method was developed by Breiman et al. (1984) [13] and now has become an extremely popular data mining tool. It utilizes recursive partitioning (binary splits), which evolved from the work of Morgan and Sonquist (1963) [57] and Fielding (1977) [31] on analyzing survey data. CART has a forward stepwise procedure that adds model terms and backward procedure for pruning. The model terms partition the x-space into disjoint hyper-rectangular regions via indicator functions: $b^{+}(x;k) = 1\{x > t\}$, $b^{-}(x;t) = 1\{x \leq t\}$, where the "split-point" k defines the borders between regions. The resulting model terms are $B_m(x) = \prod_{l=1}^{L_m} [b^{s_{l,m}} \cdot (x_{v(l,m)} - k_{l,m})]_{+}$, where L_m is the number of truncated linear functions multiplied in the m-th basis function, $x_{v(l,m)}$ is the input variable corresponding to the l-th truncated linear function in the m-th basis function, $k_{l,m}$ is the knot value corresponding to $x_{v(l,m)}$, and $s_{l,m}$ is $+1$ or -1 to indicate the direction of the partition. The CART model form is then

$$\hat{f}(x) = \beta_0 + \sum_{m=1}^{M} \beta_m B_m(x), \tag{2.13}$$

The partitioning of the x-space does not keep the "parent" model terms because they are redundant.

CHAPTER 3

ADAPTIVE VALUE FUNCTION APPROXIMATION (AVFA)

3.1 Motivation and Basis

3.1.1 Nonparametric Statistical Inference and Consistent Estimation

The AVFA framework develops a sequential approach to adaptively approximate the future value function. Existing theoretical results in nonparametric statistical inference can play a significant role in the implementation of this idea. Nonparametric statistical inference is a branch of statistics associated with model-free estimation. In order to introduce its theoretical results it is necessary to define a term known within the statistical community, the "bias/variance dilemma" (Geman 1992 [35]). This dilemma states that an estimation error can be decomposed into two portions, known as bias and variance. Given a "training data set" $D_N = (x_1, y_1), ..., (x_N, y_N)$ with size N, a regression task constructs a function $f(x; D_N)$ based on D_N to approximate the real relationship between y and x. Presented a particular x, a natural measure of the effectiveness of f as a predictor of y is the mean-squared error (MSE): $E[(y - f(x; D_N))^2 | x, D_N]$. For this MSE we have:

$$E[(y - f(x; D_N))^2 | x, D_N] = E[(y - E[y|x])^2 | x, D_N] + (f(x; D_N) - E[y|x])^2. \tag{3.1}$$

In the above equation the expectation $E\left[(y - E\left[y \mid x\right])^2 \mid x, D_N\right]$ is simply the variance of y given x which we can do nothing to minimize because it is independent of our model; $(f(x; D_N) - E\left[y \mid x\right])^2$ is the squared distance to the regression function, which measures, in a natural way, the effectiveness of f as a predictor of y. The MSE of f as an estimator of the true regression $E\left[y \mid x\right]$ is $E_{D_N}\left[(f(x; D_N) - E\left[y \mid x\right])^2\right]$, where E_{D_N} represents an expectation with respect to the training set, D_N, that is, the average over the ensemble of possible D_N (for fixed sample size N). This MSE can further be decomposed into a bias and a variance term as (Geman 1992 [35]): for any x,

$$\begin{aligned}
&E_{D_N}\left[(f(x;D_N) - E[y \mid x])^2\right] \\
&\quad = (E_{D_N}[f(x;D_N)] - E[y \mid x])^2 + E_{D_N}\left[(f(x;D_N) - E_{D_N}[f(x;D_N)])^2\right] \\
&\quad = (bias)^2 + variance
\end{aligned} \tag{3.2}$$

If, on the average, $f(x;D_N)$ is different from $E[y \mid x]$, then $f(x;D_N)$ is said to be biased as an estimator of $E[y \mid x]$. On the other hand, an unbiased estimator may still have a larger MSE if the variance is large.

According to this bias/variance dilemma, either bias or variance can contribute to poor performance of a regression (or approximation) model: when sample data are given, models that are too simple in structure to represent the data lead to high bias; while models that are too complex in structure (usually guaranteed with sufficient model complexity to approach the real system) suffer from high variance, and, in turn, require more sample points.

Fortunately, nonparametric regression estimators are characterized by their being consistent for all regression problems, which motivates us to be able to properly utilize the bias/variance dilemma in some way. Consistency for a regression problem describes in what sense the estimator $f(x;D_N)$ converges to the true function $E[y \mid x]$. There are many specifications of consistency mathematically, and here only the most common two versions are listed (Geman 1992 [35]). One is "point-wise MSE," which states as: for each x

$$\lim_{N\to\infty} E_{D_N}\left[(f(x;D_N) - E[y \mid x])^2\right] = 0. \tag{3.3}$$

The other more global one is in terms of integrated MSE:

$$\lim_{N\to\infty} E_{D_N}\left[\int (f(x;D_N) - E[y \mid x])^2\,dx\right] = 0. \tag{3.4}$$

Obviously, from the above definitions, any consistent estimator will require that both bias and variance go to zero as the size of training sample increases.

3.1.2 Statistical Consistency Traces

Though the concept of consistency presented above looks simple, it provides little guidance on “how” to realize a consistent nonparametric approximation. Realizing such a consistent process is a difficult computational problem. It needs not only a gradually increasing sample size (to reduce variance error) but, more importantly, a gradually growing capacity (or complexity) of possible functions (to diminish bias). Nevertheless, the concept implies that some theoretical convergence traces exist for a consistent nonparametric approximation. Such a trace is determined by how a nonparametric estimator grows its complexity as the sample size tends to infinity. In this section a theoretical analysis is presented with a graphical illustration.

In statistical modeling, it is reasonable to evaluate the performance of a model by monitoring its generalization ability. To this end, a common statistical method, “cross validation,” is often applied (see Kohavi 1995 [46] and Seliono 2001 [69], among others). Though it is also applicable to an ADP application, the method however has an expensive computational cost, which will increase the already heavy computational load inherently encountered in an ADP solution process. The generalization ability of a model can also be evaluated by a test set that is independent from the data used to build the model, provided the test set is representative enough (Zhang and Morris 1997 [93]; Chi and Ersoy 2002 [19]). In this study, the predicted mean-squared error (called test error) from a test set is always used to evaluate the performance for the model-building process. It is defined as

$$e_{ts} = \frac{1}{M} \sum_{j=1}^{M} \left(y_j - f(x_j; D_N)\right)^2 , \tag{3.5}$$

where M is the size of the test set; and x_j and y_j, $j = 1, ..., M$, represent the sampled predictors and their target responses, respectively, in the test set. The objective is to minimize this error in order to achieve a good approximation model.

Equation (3.5) defines a random error function whose average performance can be further analyzed with the bias/variance dilemma and the consistency theory mentioned previously. For simplicity, assume that the variance of y given x is constant for any x for the system to be approximated, and is denoted with σ^2. Taking expectations for both sides of Equation (3.5), first with respect to y_j given x_j, $j = 1, ..., M$, and then with respect to the different possibilities of training data D_N (for fixed sample size N), we can get

$$\begin{aligned} E_{D_N}\left[E\left[e_{ts}\right]\right] &= \frac{1}{M}\sum_{j=1}^{M} E_{D_N}\left[E\left[(y_j - f(x_j; D_N))^2\right]\right] \\ &= \frac{1}{M}\sum_{j=1}^{M} E_{D_N}\left[\left(E\left[(y_j - E[y_j \mid x_j])^2\right] + (f(x_j; D_N) - E[y_j \mid x_j])^2\right]\right. \\ &= \sigma^2 + \frac{1}{M}\sum_{j=1}^{M} E_{D_N}\left[(f(x_j; D_N) - E[y_j \mid x_j])^2\right]. \end{aligned} \tag{3.6}$$

From the above equation, the average performance of the error function (3.5) is composed of the system's variance σ^2 and the average of the mean-squared errors of the model as an estimator of the true regression over all the test points. By using the consistency result in Equation (3.3) we can easily observe that as long as the sample size of the training data set increases to infinity, the second term of Equation (3.6) tends to zero. Mathematically, this observation can be written as

$$\begin{aligned} \lim_{N\to\infty} E_{D_N}\left[E\left[e_{ts}\right]\right] &= \lim_{N\to\infty}\left(\sigma^2 + \frac{1}{M}\sum_{j=1}^{M} E_{D_N}\left[(f(x_j; D_N) - E[y_j \mid x_j])^2\right]\right) \\ &= \sigma^2. \end{aligned} \tag{3.7}$$

This result indicates that when the model building process is consistent, the test error should converge in expectation to the system's inherent noise. This supplies a theoretical optimal value of the test error defined by Equation (3.5).

The second term of Equation (3.6) can be further decomposed in a similar way in Equation (3.2) into bias and variance parts as

$$\begin{aligned}
&\frac{1}{M}\sum_{j=1}^{M} E_{D_N}\left[(f(x_j;D_N) - E[y_j \mid x_j])^2\right] \\
&\quad = \frac{1}{M}\sum_{j=1}^{M} \left(E_{D_N}[f(x_j;D_N)] - E[y_j \mid x_j]\right)^2 \\
&\quad + \frac{1}{M}\sum_{j=1}^{M} E_{D_N}\left[(f(x_j;D_N) - E_{D_N}[f(x_j;D_N)])^2\right]. \qquad (3.8)
\end{aligned}$$

The bias and variance parts for the test error are made up by averaging the bias and variance contributions over all the test points. These two terms will inherit the bias/variance trade-off feature from the individual test points and hence allow the test error function to achieve a bias/variance trade-off feature averaged over the test points, which together with consistency will be discussed further below.

Given that a type of nonparametric modeling is selected and that the size of the training data can grow from small to large, the test error function defined with Equation (3.5) can be viewed as a function of the model complexity and the size of training data, that is, we can realize an error surface $e_{ts} = e_{ts}(\textit{model complexity}, \textit{size of training data})$ with respect to *model complexity* and *size of training data*. With the results obtained previously in Equations (3.6), (3.7) and (3.8), it is possible to have a graphical understanding of the expectation performance and the ideal shape of that error surface. We first consider a case where the data is a fixed size. From the bias/variance dilemma described earlier in this chapter and in Equation (3.8), with such a training data set, the curve of the expected generalization error (test error) is thought to be "bowl-shaped," due to the combined effects of model bias and variance with respect to the variation of model complexity: as model complexity increases, the bias term for the model decreases while the variance term increases and vice-versa (see, for example, Hastie et al. 2001 [37] for a discussion on the shape). The location where the lowest error value occurs is the optimal bias/variance trade-off point associated with an optimal model structure, i.e., optimal model complexity, for a fixed data set. For simplicity, let us further assume that the portion of the test error contributed by bias only depends on the model complexity and not on the training data size. From the fact that increasing the training data always reduces the variance, we can plot the trend of the bowl curves for different sizes of training data in the two dimensional "test error vs. model complexity" space as in Figure 3.1. From the figure one can imagine that the minimum value points on all the possible bowl curves with different sizes of training data form a trace that should asymptotically go down as long as both the size of the data and the model complexity increase to infinity. In particular, if we assume that the system has a constant variance for y given any x, from the result shown in Equation (3.7), the test error should asymptotically converge to that system's variance σ^2. This "ideal" trace can be viewed as a theoretical realization of a consistency trace.

In this dissertation, the consistency property is used to devise the sequential modeling algorithms that can realize some adaptive and "data-driven" (or "letting-data-speak") (Geman 1992 [35]) asymptotical processes. The algorithms are tuned to follow a consistency trace, so as to identify the appropriate model structure for approximating the future value functions of an ADP problem.

3.1.3 Nonparametric Neural Networks Features

Nonparametric models are a critical component of the AVFA framework because they are critical in implementing a consistent modeling process. In this dissertation, one-layer feed-forward NNs are chosen as the nonparametric models.

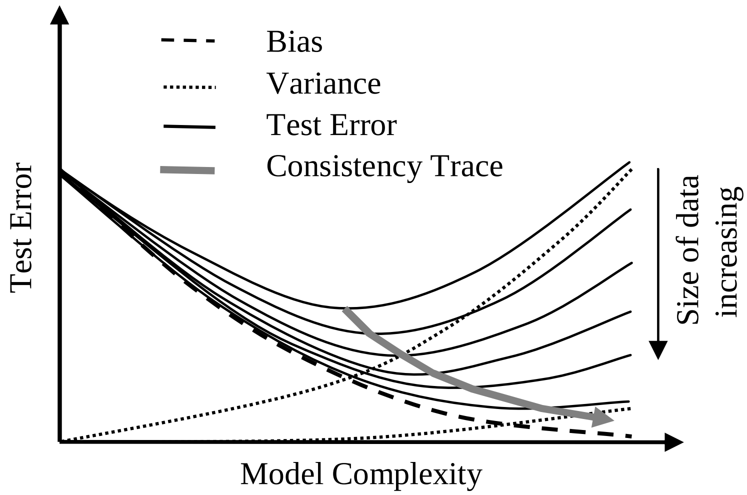


Figure 3.1. The test error curves for different sizes of training data sets, and the trace formed by the minimum error points of the error curves.

Neural networks in most cases are treated as parametric models in which the parameters are the connection weights. In this situation, a NN model has a fixed structure. As already mentioned in Chapter 2, the two most critical problems for NN modeling are how to identify an appropriate NN architecture, and how to determine that sufficient data are available. For both problems described here, the best practice is using trial and error, i.e., by training several networks, with different training sample sizes, using a test set or a cross-validation process to evaluate the generalization abilities of those candidate models, and then choosing the best among them. It can be remarked that in both kinds of existing ADP methods in which NN models have been applied, NNs were used in their parametric version (Cervellera et al. 2007 [14]; Bertsekas 2001 [10]; Si et al. 2004 [71]). Because a NN used in such a way is fixed in structure, it is unavoidable for the model builders to suffer a time-consuming trial-and-error process.

In fact, NNs can be treated as nonparametric models. Nonparametric methods are generally indexed by one or more parameters that control bias and variance. These parameters could, theoretically, be properly tuned, as functions of sample size, to ensure consistency, and to achieve a very best bias/variance tradeoff for any fixed size of the training set. Any NN with a fixed architecture is useless for such purposes. However, if the number of hidden nodes of one-layer feed-forward NNs, for instance, is taken as such a parameter, the NNs become nonparametric models. The number of hidden nodes can control NN complexity and can further be associated with the size of data set. White (1990) [90] proved that a feed-forward NN can be made consistent by suitably letting the network size grow with the size of the training set. Geman et al. (1992) [35] had further summarized about the consistency of one-hidden-layer feed-forward NNs.

3.1.4 Sequential State Space Discretization

Sequential design (or sampling) techniques are another critical component for the AVFA framework, which, together with adaptive NN nonparametric modeling, mimic a consistent model-building process. From a statistical perspective, state space discretization or sampling is equivalent to design of experiments (DoE). The traditional full grid of points, i.e., the full factorial design, is known by statisticians to be too large in practice for high dimensions. Thus, a variety of more efficient experimental designs have been developed. For DP problems one in general has no prior knowledge of the form of the future value functions, so "space-filling" methods are suggested by statisticians since they are model-independent and can identify a set of design points that are "uniformly scattered" over a design space. Space-filling methods include the aforementioned orthogonal arrays (OAs), Latin hypercubes (LHs), number-theoretic methods (NTMs), etc., and are getting more and more attention in the literature in recent years (see Chen et al. 2003 [17], 2006 [18] for a review of OAs and LHs and Wen 2005 [82] for NTMs).

In the AVFA framework, the increasing data sample size is sequentially generated by adding newly sampled data batches in each sequential step. In consideration of the

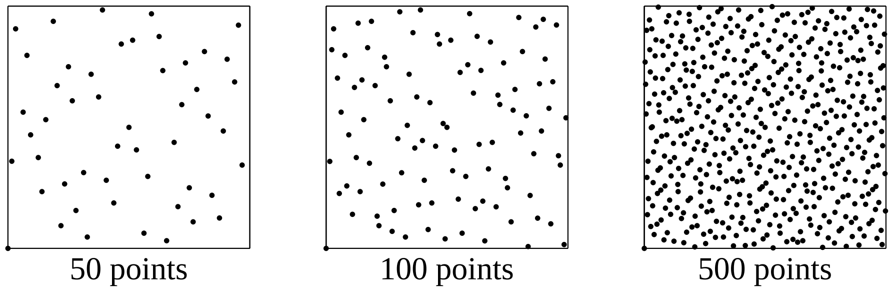


Figure 3.2. The sequential space-filling performance of a Sobol′ sequence.

fact that no prior knowledge of the state space is available, it is desired to maintain good space-filling quality while growing the sample size.

Among the various space-filling methods, NTMs are most easily adapted to a sequential sampling approach. An NTM, a.k.a. quasi-random sequence or low-discrepancy sequence, aims to create uniformly-spaced designs in $I^s \equiv [0,1]^s$ (here the domain of interest is normalized to the closed and bounded unit cube for convenience) by minimizing a "discrepancy" measure, the most common of which is star-discrepancy, defined in Niederreiter (1992) [59] as

$$D(N, P_N) = \sup_{\mathbf{w}\in[0,1]^s} \mid \frac{A(\mathbf{x}_i \leq \mathbf{w}, P_N)}{N} - v([\mathbf{0}, \mathbf{w}]) \mid, \tag{3.9}$$

where P_N is a point set with N points in $[0,1]^s$, $P_N = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N)$, $A(\mathbf{x}_i \leq \mathbf{w}, P_N)$ is a counter function that counts the number of points $\mathbf{x}_i \in P_N$ that satisfy the inequality, and $v([\mathbf{0}, \mathbf{w}])$ is the volume of the rectangle $[0, w] \in [0,1]^s$. In other words, the "discrepancy" of a points set is the largest difference between the fraction of the number of points in a subset and the volume of the subset. Point sets with lower discrepancy are deemed to be more uniformly scattered. A multitude of classes of NTMs have been proposed based on the fruitful developments within and between several mathematical disciplines such as algebraic number theory, combinatorial methods, algebraic geometry and ergodic theory. Particular low-discrepancy sequences include good lattice points (Hua and Wang 1981 [39]; Fang and Wang 1994 [27]), Hammersley sequence (Hammersley 1960 [36]), Sobol′ sequence (Sobol′ 1967 [72]), Faure sequence (Faure 1982 [28]) and Niederreiter sequences (Niederreiter 1992 [59], Niederreiter and Xing 1995 [60]), etc.

The sequential space-filling quality of NTM methods is illustrated with Figure 3.2 with a Sobol′ sequence for instance. The results show three differently sized, two-dimensional sampling data sets generated in a unit square. In all sizes (50, 100 and 500), the data points are quite well scattered at each case, indicating the desired good space-filling performance.

3.1.5 AVFA Framework

The schematic flowchart of the framework is shown in Figure 3.3. The essence of this approach is to appropriately increase both the model complexity and the training data size, so as to realize an efficiently consistent modeling building process. It is recalled that the one-hidden-layer feed-forward NNs are chosen as the nonparametric models, with their numbers of hidden nodes as adjustable parameters. (In general, any type of the nonparametric modeling can be chosen.) The state space is sequentially sampled according a low-discrepancy sequence, and the obtained data sets are used to train the models. A test set of fixed size is also prepared beforehand according to another low-discrepancy sequence that is independent of that which is used for state space sampling. A sequential modeling process from this approach starts with an initial model structure and an initial training data set. It then progresses step by step within an iterating loop that can be generically described as follows:

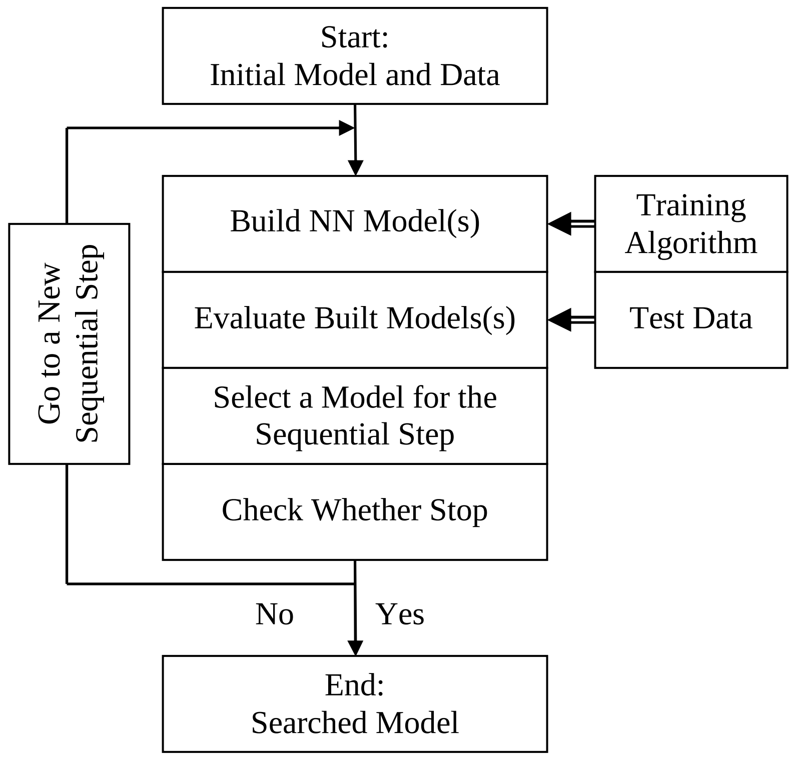


Figure 3.3. Schematic flowchart of the AVFA framework.

1. Given an updated model structure and an updated data set from the preceding sequential step, find a new model for the current sequential step among certain candidate models through certain strategies – a core part of adaptive modeling, where each candidate model must be trained with a training algorithm and evaluated with test data.
2. Check with a given criterion whether to stop the process; if yes, the process is stopped; otherwise, it goes ahead to a new sequential step.

It is noticed that an asymptotical process, as the AVFA sequential modeling process, theoretically extends to infinity. In practice, however, a stopping rule is used to "cut-off" the process as soon as the desired model performance is satisfied.

3.2 AVFA Algorithms

Let l index the sequential steps for the algorithms. The notation used in the algorithms is as follows:

NN_l — the NN model

D_{N_l} — the training data

N_l — the size of the training data

D_{ts} — the test data set

H_l — the number of hidden nodes

ΔN — an increment of the size of training data

$D_{\Delta N,l}$ — the subset of data corresponding to ΔN

ΔH — an increment of the number of hidden nodes

$e_{ts,l}$ — the test error

n^{max} — the maximal number of trial models for a successive model search process

$NN^{(i)}$ — the trial models in a successive model search process

$e_{ts}^{(i)}$ — the test errors of the trial models in a successive model search process

d — a rule index for a proper stop of a successive model search process

Given that one-hidden-layer feed-forward NNs are chosen as the models, the AVFA algorithms will involve two basic successive search processes — successive model search process (denoted as "model-search process" for simplicity) and successive data search process ("data-search process"), that are further defined as:

- **Model-Search Process**: For a given training data set, a model-search process begins with a "base" model and successively searches to find an optimal NN model among certain candidate models whose numbers of hidden nodes successively increase with an increment, denoted by ΔH, from the "base" model.

- **Data-Search Process**: A data-search process begins with a "base" data set and successively searches to find an optimal training data sample size through training a NN model of the same structure (i.e., a fixed number of hidden nodes) with different data sets that are obtained by successively adding ΔN new data points to the "base" data set.

Depending on different strategies for increasing both the model complexity and the training data size, the schematic framework of the new approach described in Section 3.1 is implemented in this dissertation via three algorithms. Each can start with the same initial conditions, i.e., a training data set D_{N_0} with size N_0, and a model NN_0 with the number of hidden nodes H_0, and then works with their individual strategies.

3.2.1 Algorithm A

Algorithm A is in a very intuitive way to implement the AVFA framework, as shown in the pseudocode. In this algorithm, the training data are set to increase with a given increment for each sequential step, and in each step a model-search process is embedded to find a optimal model under that training data set. Once the algorithm is initialized, an iterative loop conducts the following main actions:

- The training data set D_{N_l} is updated by adding ΔN new points to the preceding training data $D_{N_{l-1}}$.
- With the new data, the algorithm does a model-search process by trying n^{max} candidate models whose numbers of hidden nodes successively increase ΔH from H_{l-1}, the number of hidden nodes for the model from the previous sequential step.

In Algorithm A, the trial models always include a "base" model that has the same number of hidden nodes as in the model NN_{l-1} from the previous sequential step $l-1$, and the others are obtained by successively adding ΔH hidden nodes from that base model. After all the candidate models are trained with the data D_{N_l}, and tested with data D_{ts}, the model with the smallest value among $\left\{e_{ts}^{(1)}, e_{ts}^{(1)}, \ldots, e_{ts}^{(n^{max})}\right\}$ is chosen as the model NN_l for the current sequential step l. The test error for the chosen model for each sequential step is recorded as the performance history of the algorithm. Figure 3.4 schematically illustrates how a model-search process embedded in a sequential step in Algorithm A works to find an optimal model for the step. The figure shows two smooth bowl-shaped error curves for a current step and its preceding step, with the previous sequential model NN_{l-1} exactly at the bottom of the error curve. The model-search process will find the optimal model within the shaded range that starts from H_{l-1} and has the width n^{max} along the axis of model complexity H. Ideally and obviously, when n^{max} is large enough such that the range contains the number of the hidden nodes, say H_l, of the potential optimal model NN_l, the model-search process could successfully find the optimal model for the current sequential step. Referring to the error surface trends depicted in Figure 3.1, it is imaginable that Algorithms A empirically follows the ideal consistency trace. Given a training data set, the sequential procedure seeks to find the lowest point of the bowl curve corresponding to that data set. It then increases the data set and repeats the model-search process. As long as the associated parameters are tuned properly, this algorithm should approximately follow a consistency trace and realize an adaptive model-building process for ADP problems.

A model-search process in Algorithm A is determined by two parameters, ΔH and n^{max}. It then tries each of the n^{max} models, and chooses the best as the model for the current sequential step. As a result, Algorithm A has three important parameters, the above two for the model-search processes and the increment of training data, ΔN, to jointly control its performance.

Algorithm A can be implemented very easily due to its intuitiveness and simplicity. It however has a drawback from the included model-search processes: the entire n^{max}

Algorithm A

Input: D_{N_0}, N_0, NN_0, H_0, D_{ts}, ΔN, n^{max}, ΔH

Output: a optimal NN structure for ADP

$H_1 = H_0$, $N_1 = N_0$, $l := 0$

Repeat

- l:=l+1
- $N_l = N_{l-1} + \Delta N$
- Sample ΔN new points to get $D_{\Delta N,l}$ and
  let $D_{N_l} = D_{N_{l-1}} + D_{\Delta N,l}$
- Select NN$_l$ via a local search process:
  FOR $i = 1$ to n^{max}
    Train the model $NN^{(i)}$ with $(H_{l-1} + (i-1) \cdot \Delta H)$ hidden nodes
    Calculate their $e_{ts}^{(i)}$ with D_{ts}
  END FOR
  Identify the best one with the minimal $e_{ts}^{(i)}$
  Let the best model be NN_l
  Let the best model's test error be $e_{ts,l}$

Until a stopping criterion is satisfied

---

models can require excessive computation because the best model may appear earlier than the n^{max}-th model.

3.2.2 Algorithm B

Algorithm B was designed to reduce the extra computation in the model-search process that may occur in Algorithm A; thus, it can be considered as an improved version of Algorithm A. From the pseudocode, Algorithm B has the iterative loop with main actions very similar to Algorithm A:

- The training data set D_{N_l} is updated by adding ΔN new points to the preceding training data $D_{N_{l-1}}$.
- With the new data, the algorithm does a model-search process by trying at most n^{max} candidate models, whose numbers of hidden nodes successively increase ΔH from H_{l-1}.

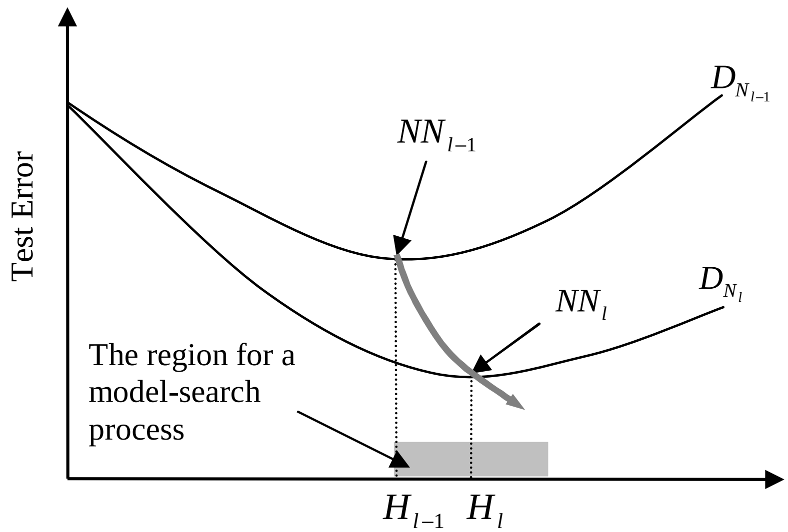


Figure 3.4. Illustration of a model-search process for finding the optimal model of a current step.

Algorithm B

Input: D_{N_0}, N_0, NN_0, H_0, D_{ts}, ΔN, n^{max}, ΔH, d
Output: a optimal NN structure for ADP

$H_1 = H_0$, $N_1 = N_0$, $l := 0$
Repeat
- l:=l+1
- $N_l = N_{l-1} + \Delta N$
- Sample ΔN new points to get $D_{\Delta N,l}$ and
  let $D_{N_l} = D_{N_{l-1}} + D_{\Delta N,l}$
- Select NN$_l$ via a local search process:

  FOR $i = 1$ to n^{max}
    Find $e_{bench}^{(i)}$ from $\left\{e_{ts}^{(i)}\right\}, i = 1, ..., i-1, \forall i \geq d$
    Train the model $NN^{(i)}$ with $(H_{l-1} + (i-1) \cdot \Delta H)$ hidden nodes
    Calculate their $e_{ts}^{(i)}$ with D_{ts}
    IF $e_{ts}^{(i)} > e_{bench}^{(i)}, \forall i \geq d$
      BREAK FOR
    END IF
  END FOR
  Identify the best one with the minimal $e_{ts}^{(i)}$
  Let the best model be NN_l
  Let the best model's test error be $e_{ts,l}$

Until a stopping criterion is satisfied

A distinction between Algorithms A and B exists in the second action, in particular, in the model-search processes. Unlike in a model-search process in Algorithm A, where each candidate is tried as long as n^{max} is specified, a model-search process in Algorithm B is given with n^{max} only to restrict the maximum number of candidate models (or trial times). The search process works almost in a same way as in Algorithm A, except a "degradation-judgment" step (i.e., "IF ... END IF" part in the pseudocode) is added between the successive candidate-model trials to promptly and timely stop the search process as soon as the trial models degrade. For each degradation-judgment, the algorithm first needs to pick a benchmarking model with certain rules from the models that have already been tried within the associated model-search process, and then compares the current trial model with that benchmarking model using test error values. If the fit of the current trial model is worse than that of the benchmarking model, then it is decided that the untried trial models will not yield an improved fit, and hence are not worth exploring; otherwise, the search process with the judgment step continues, at most until the maximal number of the trial models is reached. Once the model-search process has stopped, the model with the smallest test error value among those already tried is chosen as the model NN_l for the current sequential step l. The corresponding test error value is recorded in the performance history of the algorithm.

The interesting and crucial part of this algorithm concerns the rules for setting the benchmarking model. The degradation-judgment step added in the model-search process is not a free lunch. On the one hand, it provides a way to save computational effort. On the other hand, it may miss some better models, as some stochasticity is inherited into a model-building process by at least two main sources, the uncertainty in the data and the random models. Firstly, as was already mentioned in Section 3.1, the "smooth" bowl-shaped test error curves and the consistency trace derived from them are obtained based on some expectations. Although Algorithms A and B (even extensively, Algorithm C to be discussed later on in this section) are designed to do a model-search process to follow the theoretical test error curves, what the algorithms can do in practice is to sample the curves. In other words, the obtained error curves may not exactly be smoothly bowl-shaped (or convex) with one unique basin part where the 'theoretical" optimal model is located. Consequently, there may be "zigzag" shapes for which there are multiple locally "optimal" models. Therefore, it is not guaranteed that the degradation-judgment in Algorithm B will always conclude a correct degradation trend of the candidate models in a search process because of the data-noise caused by local optima. Secondly, the NN models selected in this study have additional randomness from the fact that the models are usually trained over some multi-mode error functions.

Fortunately, it is found that if the rules are determined in terms of "different order of best models", the issues discussed above can be addressed. Let parameter d index the different possible rules. Thus, when "the first order best model" is taken as the rule, it is denoted that $d = 1$. In this situation, the algorithm does a degradation-judgment by always taking the best model among the already tried candidate models in the associated model-search process as the benchmarking one, and comparing the current model with it. Similarly, when $d = 2$, the degradation-judgment takes the second best model (i.e., the model after the best) among the already tried candidate models as the benchmarking one and comparing the current model with it, and so on and so forth. The fact is that the lower the order of the benchmarking model is chosen as the benchmarking, the lower the computational effort, and also the higher the probability that the model-search process may miss some better models. If the order of the benchmarking model is too high order, then the model-search process tends to try the entire n^{max} models, resulting in no computational savings over Algorithm A. These pilot experiments demonstrated that the first three orders are enough to control the balance of the degradation-judgment step in this algorithm. Therefore only three rules will be considered in this study. It is noted that the benchmarking model selected by the rule described above will obviously be adaptive and dynamic (or equivalently, the benchmarking model could be from the different candidate models) during a model-search process. Also, the different rules let the degradation-judgment step start at the point where certain models have been tried and are enough to use the given rule to pick a benchmarking model. For example, if $d = 1$, the degradation-judgment step can start working at the second trial model, while if $d = 3$ it only can start working from the fourth trial model. Finally, it can be seen that ΔN, n^{max}, ΔH, and the rule index d form the four principle parameters that jointly control the performance of this algorithm.

3.2.3 Algorithm C

The AVFA framework can also be implemented with Algorithm C in that the model-search processes and data-search processes are alternately carried out. Essentially, Algorithm C follows the same consistency concept as Algorithms A and B. It however is distinguished from them by a different manner of following a consistency trace. From the pseudocode, after being initialized, Algorithm C has the iterative loop consisting of the following main actions:

- A candidate model NN_l is trained with the data set D_{N_l}, in which the former can be from an update based on a model-search process, while the latter from an update based on a data-search process.
- The trained model is tested with D_{ts} to obtain the test error $e_{ts,l}$.
- If the test error of the current model, $e_{ts,l}$, is less than the test error value of the previous-step model, i.e., if $e_{ts,l} < e_{ts,l-1}$, then the model is recorded as the l-th official sequential model and the algorithm proceeds with a new sequential step; otherwise, the algorithm realizes an update of the current model NN_l by adding

ΔH nodes in its hidden layer or an update of the current training data set D_{N_l} by adding ΔN new points, and goes back to the training action.

Algorithm C

Input: D_{N_0}, N_0, NN_0,H_0, D_{ts}, ΔN, ΔH, e^0_{ts}

Output: a optimal NN structure for ADP

$H_1 = H_0$, $N_1 = N_0$, $l := 0$

Repeat

- Build NN_l with D_{N_l}
- Calculate $e_{ts,l}$ with D_{ts}
- IF $e_{ts,l} < e_{ts,l-1}$

        $l := l + 1$
        $e^0_{ts} = e_{ts,l}$
    ELSE
        IF N_l increased
          IF $e_{ts,l} < e^0_{ts}$
            $N_l = N_l + \Delta N$
            $H_l = H_l$
          ELSE
            $N_l = N_l$
            $H_l = H_l + \Delta H$
          END IF
        ELSE
          IF $e_{ts,l} < e^0_{ts}$
            $N_l = N_l$
            $H_l = H_l + \Delta H$
          ELSE
            $N_l = N_l + \Delta N$
            $H_l = H_l$
          END IF
        END IF
        $e^0_{ts} = e_{ts,l}$
    END IF

Until a stopping criterion is satisfied

The iterative loop described above can be translated as a "models/data updating and retraining" loop that is designed to always follow a track determined by alternate (or reciprocating) model-search and data-search processes, by which a model better than the one in the preceding sequential step is filtered out as the model for the current sequential step. The following strategies are specified for the two search processes to work and mutually switch in this algorithm:

- When the tried model is from a model-search process, the algorithm continues working in this search process by trying the successive models one by one. As long as a tried model is worse than its preceding one, the model-search process stops and switches to a data-search process. In the meantime, the last model and the training data set in the model-search process are transferred to the data-search process as its base model and base training data.
- Similarly, when the tried model is from a data-search process, the algorithm continues working in this search process by trying the models, that have the same number of hidden nodes, with successive size-increasing training data sets. If it finds that a training data set generates a degraded model, then the data-search process stops and switches to a model-search process. Simultaneously, the last training data set and the model from the data-search process are passed to the model-search process as its base model and base training data.
- Evidently, when the tried model is from a switch, the algorithm starts a new search process opposite the preceding one.

As a result, with the aforementioned reciprocating searching strategy, Algorithm C allows the size of the sample data, N_l, and the number of the hidden nodes, H_l, to ingeniously increase, hopefully, to approximately follow a consistency trace as shown in Figure 3.1.

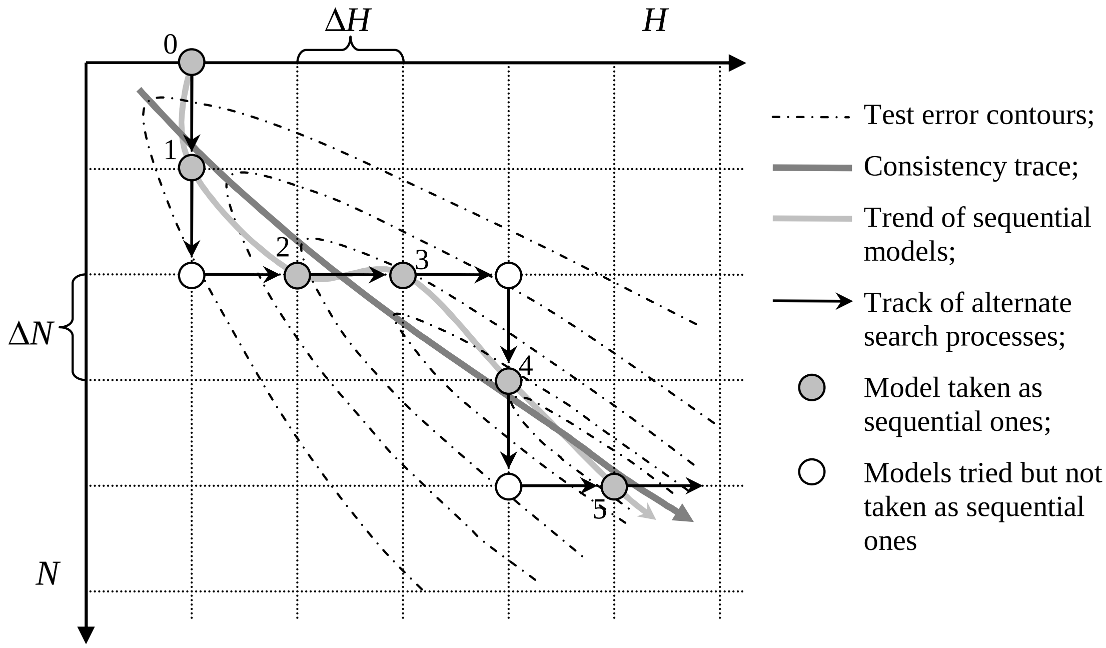


Figure 3.5. An example sequential model-building procedure performed with Algorithm C.

An example sequential model-building procedure performed with Algorithm C can be simply illustrated on a two dimensional contour plot of test error with respect to the number of hidden nodes, H, and the size of the training data, N, as shown in Figure 3.5. On the figure, the small circles indicate the models tried by the algorithm, in which the shaded are the sequential models selected while the blank are the ones only tried, and the numbers index the sequential steps. Without a loss of generality we assume that the algorithm starts from a zero-th step (an initialization step) in a data-search process. The algorithm goes ahead along this process and gets a better model under an increased data set, thus it comes to a new sequential step (the first step). When the algorithm continues with a newly increased training data set, it obtains a degraded model, and hence it switches to a model-search process. In the model-search process, the algorithm first tries a model with number of hidden nodes being increased, and finds that are better than the latest sequential one and goes to a new sequential step (the second step). It then tries another one and gets to another sequential step (the third step). When it goes on to try the next model further, the algorithm meets a worse model and switches to another data-search process. The algorithm can keep going so on and so forth, until a satisfied model is achieved. It is noted that if the algorithm starts in a model-search process, a similar track can be obtained as in the case shown above.

Some further remarks can be made about Algorithm C, referring to the example case presented in Figure 3.5.

- The sequential models constructed in Algorithm C could follow a consistency trace: A trend of these models shown in the figure presents an appearance that as the algorithm progresses, the newly found sequential models are located closer to the plotted consistency trace.
- Unlike in the model-search process in Algorithms A and B, where the algorithms need to try the base model because a new data set is always presented, a base model in Algorithm C is automatically determined by the preceding process.
- That a switch in Algorithm C occurs when a degraded model is encountered pushes the algorithm to continue growing the sample size and the number of hidden nodes. This follows the assumption that a consistency trace always increases both the size of training data and the model complexity.

Finally, Algorithm C has the two increment parameters, ΔN and ΔH, to control its performance.

3.3 Application to an Inventory Forecasting Problem

The three new AVFA sequential algorithms proposed in Section 3.2 can be easily employed by the traditional backward solution framework for ADP problems, yielding a new framework that keeps the backward style but makes the future-value-function approximating process sequential and hence more adaptive and automatic. In this section, the new algorithms were examined using a nine-dimensional stochastic inventory forecasting problem.

3.3.1 Inventory Forecasting Problem

Inventory forecasting problems are typical stochastic DP problems with continuous state space that need approximate solutions. To demonstrate the AVFA approach, a nine dimensional inventory forecasting problem is chosen. An advantage to choose this problem is that it has been frequently studied previously (Chen et al. 1999 [16], Chen 1999 [15], Cervellera et al. 2007 [14], Wen 2005 [82]) and hence provides us some good benchmarking results to be compared.

The state variables for inventory forecasting consist of the inventory levels and demand forecasts for each item. For the chosen nine-dimensional problem, three items and two forecasts for each are considered. At the beginning of stage t, let $I_t^{(i)}$ be the inventory level for item i at the beginning of the current stage. Let $D_{t,t}^{(i)}$ be the forecast made at the beginning of the current stage for the demand of item i occurring over the stage. Similarly, let $D_{t,t+1}^{(i)}$ be the forecast made at the beginning of the current stage for the demand of item i occurring over the next stage ($t+1$). Then the state vector at the beginning of stage t is:

$$x_t = \left(I_t^{(1)}, I_t^{(2)}, I_t^{(3)}, D_{t,t}^{(1)}, D_{t,t}^{(2)}, D_{t,t}^{(3)}, D_{t,t+1}^{(1)}, D_{t,t+1}^{(2)}, D_{t,t+1}^{(3)}\right)^T$$

The decision variables control the order quantities. Let $u_t^{(i)}$ be the amount of item i ordered at the beginning of stage t, then the decision vector is $u_t = \left(u_t^{(1)}, u_t^{(2)}, u_t^{(3)}\right)^T$. Each component of u_t is a function of the state variables, i.e., $u_t^{(i)} = \lambda_t^{(i)}(x_t)$, $i = 1, 2, 3$, where $\lambda_t^{(i)}(\cdot)$ is a unknown function. Put in a compact way, one can write $u_t = \lambda_t(x_t)$. For simplicity, it is assumed that orders arrive instantly.

The transition function from period t to period $t+1$ is $x_{t+1} = x_t + u_t - D$, where D is the multivariate random variable. The associated stochasticity in transitions is modeled using the multiplicative Martingale model of forecast evolution. (See Chen (1999) [15] for a detail on the stochastic modeling.)

The constraints for inventory forecasting are placed on the amount ordered, in the form of capacity constraints, and on the state variables, in the form of state space bounds. For this test problem, the capacity constraints are chosen to be restrictive, forcing interactions between the state variables, but are otherwise arbitrary. The bounds placed on the state variables in each stage are necessary to ensure accurate modeling over the state space.

The objective function is a cost function involving inventory holding costs and backorder costs. The typical V-shaped cost function common in inventory modeling is $c_v(x_t, u_t) = \sum_{i=1}^{n_I} (h_i[I_{t+1}^{(i)}]_+ + \pi_i[-I_{t+1}^{(i)}]_+)$, where $[q]_+ = \max\{0, q\}$, h_i is the holding cost parameter for item i, and π_i is the backorder cost parameter for item i. The smoothed version of the objective function described in Chen (1999) [15] is employed here.

3.3.2 Computational Setup

The experiments in which the CPU times were compared were conducted on a Dual 2.6 GHz Linux workstation. The MATLAB software system (www.mathworks.com) was used to code the AVFA algorithms to approximate the future value functions for the selected inventory forecasting problem, and then to simulate ADP solutions based on the obtained future value functions. MATLAB has a neural network (NN) toolbox that was used to train models. The training algorithm of NNs is based on back propagation (see Lippmann 1987 [54]). For all of the NN models the activation functions for the hidden nodes and for the output nodes were sigmoid and linear, respectively. A training process was limited to 100 times of network weight updates, which is the default value by the toolbox and is thought to be a sufficiently large value for the training algorithm to converge for most practical NN modeling tasks.

The time horizon of the nine-dimensional inventory problem is three, so only two future value functions, i.e., for the second stage and the third (last) stage, respectively, needed to be approximated. In accordance with the backward solution framework, the future value function model for the third stage was built first. Based on that model, the model for the second stage was then built. For both stages the training data used to build the approximate future value functions were obtained based on the design points from a nine-dimensional Sobol′ sequence. The test data consisted of 2000 design points from a nine-dimensional Hammersley sequence, with the consideration that these test data should overlap as little as possible with the training data.

For a comparison purposes, the case of a fixed structure and fixed training data set using NN modeling by Cervellera et al. (2007) [14] for future value function approximations of the same inventory forecasting problem under the traditional backward framework was rerun and tested under the same computational conditions as for the new AVFA algorithms. In this case, the NNs had 40 hidden nodes and the training data consisted of 1331 points from a Sobol′ sequence. Cervellera et al. (2007) [14] set the number of the training updates to 1000 for this fixed structure case, so this setting was used for this case only. The test data set used by the new method was also used to evaluate the fixed structure NNs.

3.3.3 Approximations of Future Value Functions

The selection of the initial parameters such as the size of training data N_0 and the number of hidden nodes H_0 would of course impact the corresponding algorithm performance. However, their influence on the algorithms was expected to be weaker than the other algorithmic control parameters. Thus, $N_0 = 100$ and $H_0 = 2$ were arbitrarily set for all studied cases of all algorithms. In addition, the increment of the training data points, ΔN, should be important for each new algorithm. Herein, it is assumed that its controllability for each algorithm could somehow be compensated by a proper adjustment of the increment of hidden nodes and by other algorithmic parameters, and hence $\Delta N = 50$ throughout this study, only for simplicity.

For Algorithm A, the influential algorithmic parameters are the two parameters, ΔH and n^{max}, for the model-search processes. The experimentation of this algorithm was then designed to investigate different values of these parameters. In particular, five cases from combinations of the different values of these two parameters, i.e., (ΔH, n^{max})= (1,3), (1,4), (1,5), (2,3), (2,4), were studied. For Algorithm B, it is assumed that when the maximum number of models n^{max} was sufficiently large, it was no longer a critical control parameter for the algorithm. In this study n^{max} was taken as 5 by trial and error. The influential algorithmic parameters for this algorithm were then the increment of the hidden nodes for the successive search process, ΔH, and the rule index d for the benchmarking models. Six cases from combinations of the different values of these parameters, i.e., (ΔH, d)=(1,1), (1,2), (1,3), (2,1), (2,2), and (2,3), were investigated for Algorithm B. For Algorithm C only the increment of hidden nodes, ΔH, was a critical control parameter. Three cases of this parameter, i.e., ΔH=1, 2, 3, were tested.

For convenience of discussion in the remainder of this chapter, the fixed structure case in Cervellera et al. (2007) [14] is represented with "FS." Each of the specified cases for new algorithms are represented with a string composed of two segments in which the first indicates the algorithm and the second specifies the selected parameter(s). For example, the string "A-($\Delta H = 1$, $n^{max} = 3$)" denotes the case of $\Delta H = 1$ and $n^{max} = 3$ for Algorithm A. All the studied cases are listed as:

A-($\Delta H = 1$, $n^{max} = 3$), A-($\Delta H = 1$, $n^{max} = 4$),

A-($\Delta H = 1$, $n^{max} = 5$), A-($\Delta H = 2$, $n^{max} = 3$), A-($\Delta H = 2$, $n^{max} = 4$)

for Algorithm A;

B-($\Delta H = 1$, $d = 1$), B-($\Delta H = 1$, $d = 2$),

B-($\Delta H = 1$, $d = 3$), B-($\Delta H = 2$, $d = 1$),

B-($\Delta H = 2$, $d = 2$), B-($\Delta H = 2$, $d = 3$)

for Algorithm B; and

C-($\Delta H = 1$), C-($\Delta H = 2$), C-($\Delta H = 3$)

for Algorithm C.

Algorithms A, B and C with the above parameter-specific cases were used to fulfill the tasks of the approximations of the future value functions for the chosen nine-dimensional inventory forecasting problem. For each experimental case, 50 trial runs were performed under the consideration that different random initializations of the network weights may result in different models, as the error functions on which the training algorithm tries to find a minimum are usually multimodal. Each run stopped when the size of training data reached 1350, chosen to assure a relevant comparison to the fixed structure case, which used a training sample of 1331 data points.

The experimental results of the fourteen selected cases for the three proposed algorithms were demonstrated and compared with the fixed structure NN method in Figures 3.6-3.9. Figure 3.6 shows the boxplots of the training errors (i.e., the MSE of the models over the training data) and the boxplots of the test errors for the NN models achieved by these algorithms. For clarity, Figure 3.7 only shows the eleven cases for Algorithms A and B because the results of Algorithm C were much worse. In addition, an outlier in the test error boxplot of case FS at stage 3 was excluded. Figure 3.8 shows the boxplots of the numbers of the hidden nodes in the NN models. The computational effort for the experimental cases is presented in Figure 3.9, in which the runtimes were recorded for each entire model-building procedure over both stages.

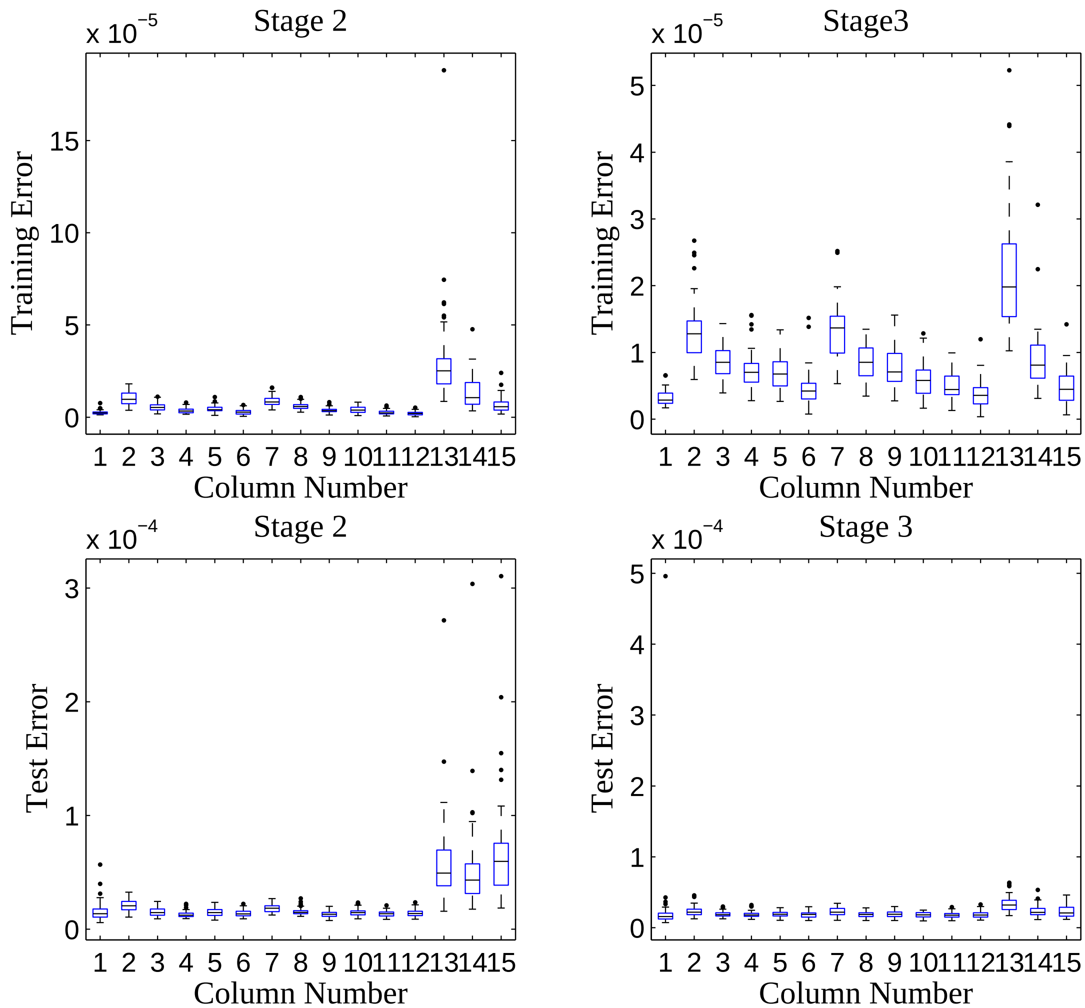


Figure 3.6. Boxplots of the training errors and the test errors for the NN models from the fourteen experimental cases by Algorithms A, B and C and the fixed structure case for future value function approximations.

Note: In Figures 3.6-3.9, the column numbers and the corresponding experimental cases are as follows:

1 — FS	9 — B-($\Delta H = 1$, $d = 3$)
2 — A-($\Delta H = 1$, $n^{max} = 3$)	10 — B-($\Delta H = 2$, $d = 1$)
3 — A-($\Delta H = 1$, $n^{max} = 4$)	11 — B-($\Delta H = 2$, $d = 2$)
4 — A-($\Delta H = 1$, $n^{max} = 5$)	12 — B-($\Delta H = 2$, $d = 3$)
5 — A-($\Delta H = 2$, $n^{max} = 3$)	13 — C-($\Delta H = 1$)
6 — A-($\Delta H = 2$, $n^{max} = 4$)	14 — C-($\Delta H = 2$)
7 — B-($\Delta H = 1$, $d = 1$)	15 — C-($\Delta H = 3$)
8 — B-($\Delta H = 1$, $d = 2$)	

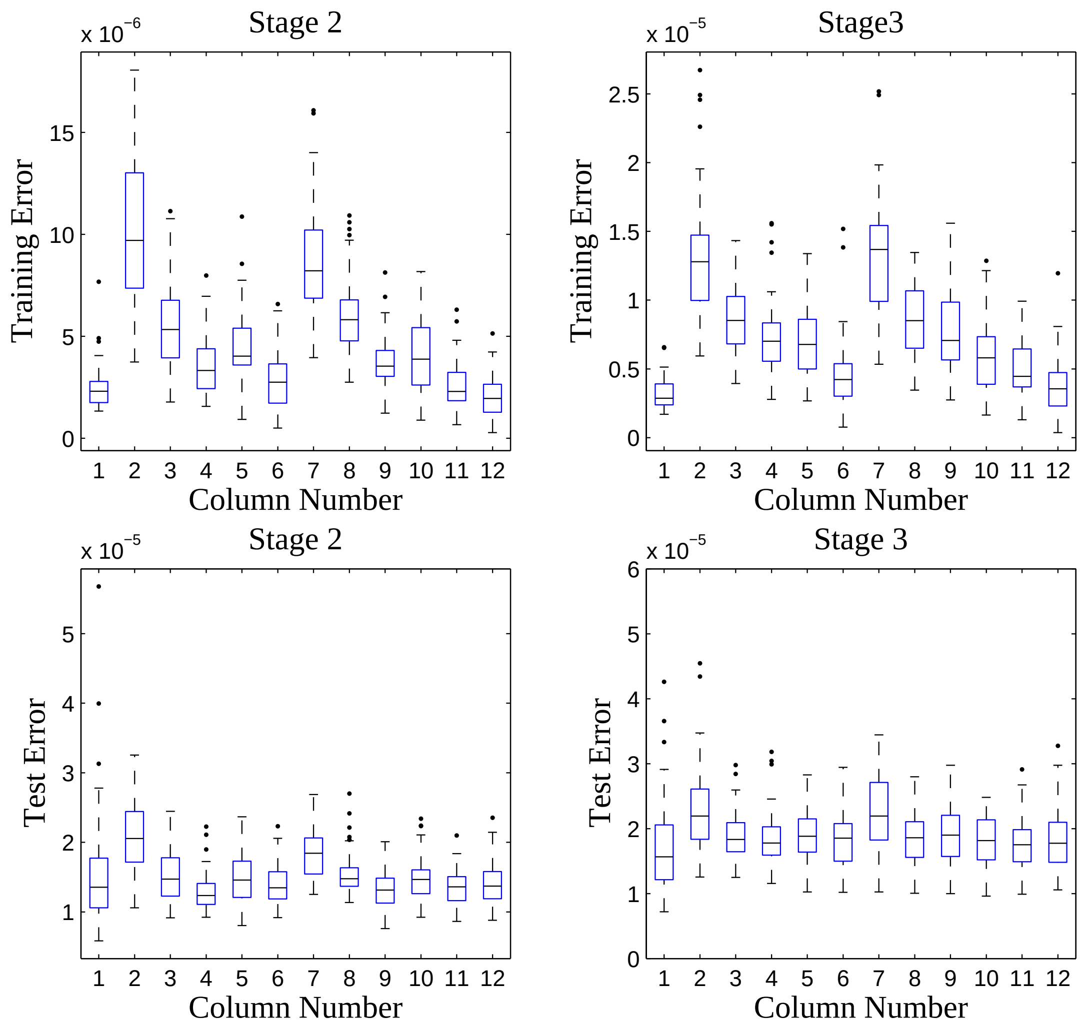


Figure 3.7. Boxplots of the training errors and the test errors for the NN models from the eleven experimental cases by Algorithms A and B and the fixed structure case for future value function approximations.

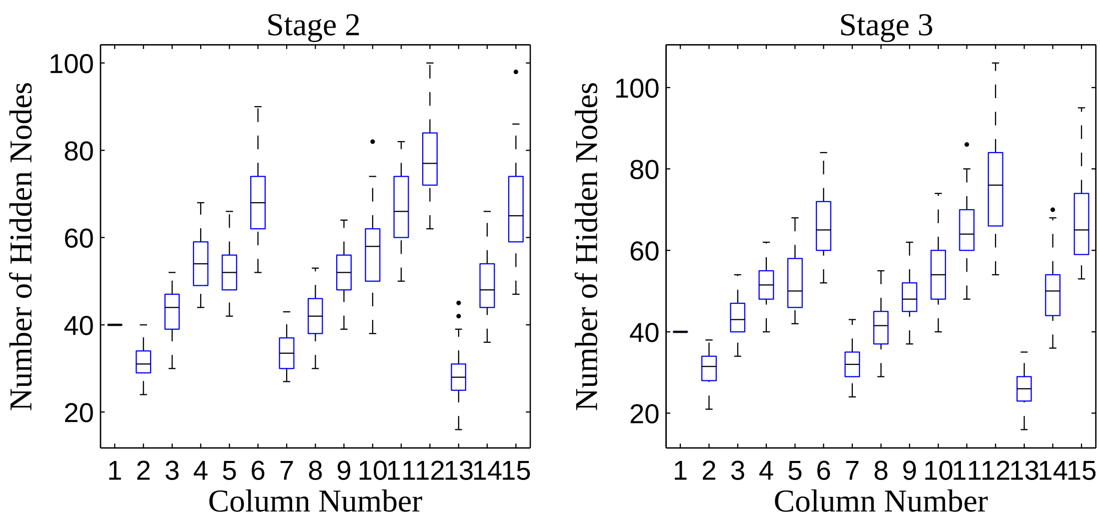


Figure 3.8. Boxplots of the numbers of hidden nodes for the NN models from the fourteen experimental cases by Algorithms A, B and C and the fixed structure case for future value function approximations.

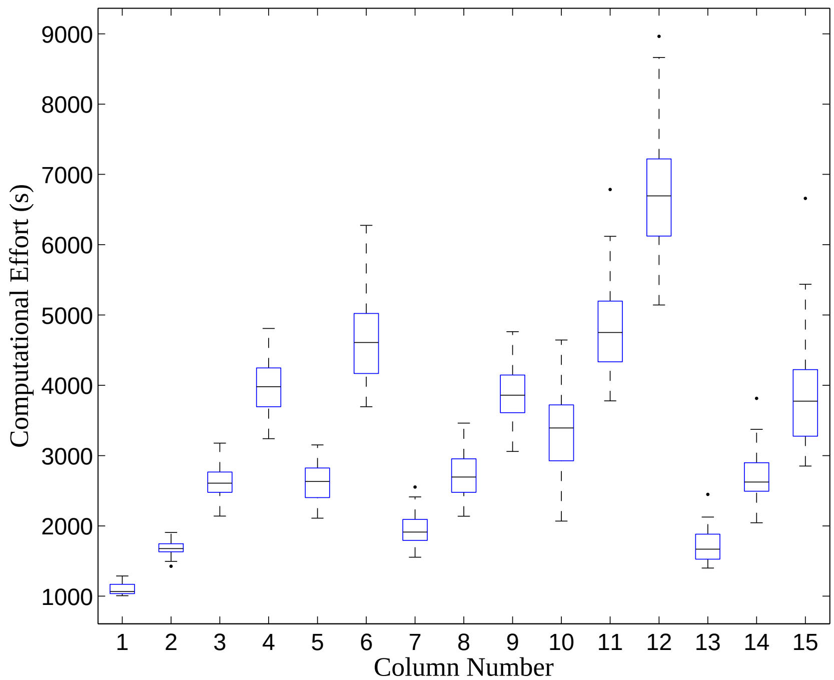


Figure 3.9. Boxplots of the computational effort (seconds) for the NN models from the fourteen experimental cases by Algorithms A, B and C and the fixed structure case for future value function approximations.

3.3.3.1 Observations for Each Algorithm

For the five experimental cases for Algorithm A, some observations with respect to the two studied parameters ΔH and n^{max} can be made as follows:

- When ΔH was given, for the different n^{max} the results showed that the training errors and the test errors decreased, on average, as n^{max} became larger, as shown in Figure 3.7. The numbers of hidden nodes and the computational effort also tended to have larger values as n^{max} became larger, as shown in Figures 3.8 and 3.9, indicating that a better model (in terms of lower test errors) needed more hidden nodes and a larger computational effort.
- When n^{max} was given for the two different ΔH, the results by cases A-($\Delta H = 1$, $n^{max} = 4$) and A-($\Delta H = 2$, $n^{max} = 4$) showed that the training errors and the test errors decreased as ΔH became larger, that is, when n^{max} was given, the models tended to be better (again in terms of lower test errors) with a larger increment of hidden nodes for the model-search processes. Also, the numbers of hidden nodes and the computational effort tended to have larger values as ΔH became larger.

The six cases for Algorithm B with the two studied parameters ΔH and d have the following observations:

- When ΔH was given for the three different d, the training errors exhibited decreasing trends as d became larger, as shown in Figure 3.7. However, the test errors exhibited rather complicated trends in contrast to the training errors. Except for the cases of $\Delta H = 1$ for stage 2 where the test errors decreased as d became larger, when $\Delta H = 2$ for stage 2 and both $\Delta H = 1$ and $\Delta H = 2$ for stage 3, the test errors decreased as d increased from 1 to 2 and increased (at least do not obviously decrease) as d increased from 2 to 3. This may indicate that a d that is too large could result in over-fitting since the corresponding training errors still decreased.

The numbers of hidden nodes and the computational effort tended to have larger values as d became larger, as shown in Figures 3.8 and 3.9.

- When d was given for the two different ΔH, the results showed that the training errors exhibited decreasing trends as ΔH became larger; whereas the test errors decreased as ΔH from 1 changed to 2 and not obviously decreased as ΔH from 2 changed to 3. Again, too large of a ΔH could cause over-fitting. The numbers of hidden nodes and the computational effort tended to have larger values as ΔH became larger.

The three cases for Algorithm C with the studied parameter ΔH present some trends as follows: As ΔH became larger, the training errors decreased, as shown in Figure 3.6. Although not shown, the test errors for both stages decreased with ΔH from 1 to 2, and increased with ΔH from 2 to 3. The numbers of hidden nodes and the computational effort tended to have larger values as ΔH became larger, as shown in Figures 3.8 and 3.9.

3.3.3.2 Overall Observations on Model Accuracy

For the adaptively constructed NN models, their performance could first be assessed by test errors, under the assumption that a lower test error means a model with higher approximation accuracy. It is reasonable for us to assume that the fixed structure case (FS) was carefully constructed (i.e., with many trial-and-error experiments), so as to achieve high accuracy. Comparing the experimental cases to case FS with the results shown in Figures 3.6 and 3.7, the following observations can be made:

- Four cases (A-($\Delta H = 1$, $n^{max} = 4$), A-($\Delta H = 1$, $n^{max} = 5$), A-($\Delta H = 2$, $n^{max} = 3$), A-($\Delta H = 2$, $n^{max} = 4$)) for Algorithm A, and five cases (B-($\Delta H = 1$, $d = 2$), B-($\Delta H = 1$, $d = 3$), B-($\Delta H = 2$, $d = 1$), B-($\Delta H = 2$, $d = 2$), B-($\Delta H = 2$,

$d = 3$)) for Algorithm B achieved the NN models that are statistically equivalent to or better than the fixed structured case FS;

- One case (A-($\Delta H = 1$, $n^{max} = 3$)) for Algorithm A and one case (B-($\Delta H = 1$, $d = 1$)) for Algorithm B had higher test errors, on average, and hence were considered worse than case FS;
- Unfortunately the three cases (C-($\Delta H = 1$), C-($\Delta H = 2$) and C-($\Delta H = 3$)) for Algorithm C were rather worse than case FS and the other experimental cases, with higher values in both training error and test error, especially at stage 2.

The "good" cases, that is the cases equivalent to or better than case FS in model accuracy, can provide some rules of thumb for selecting the corresponding algorithmic parameters:

- For Algorithm A, when $\Delta H = 1$, it seems that the other parameter n^{max} should be larger than 3. Too small of an n^{max} could cause the algorithm to produce inferior models (e.g., case A-($\Delta H = 1$, $n^{max} = 3$)). The results demonstrated that, when $\Delta H = 2$, n^{max} should be equal to or larger than 3. The results with those selected experimental cases did not indicate any upper limits on n^{max} with respect to the two ΔH values.
- For Algorithm B, when $\Delta H = 1$, the parameter d should be larger than 1. d taken as 1 could cause the algorithm to produce inferior models (e.g., case B-($\Delta H = 1$, $d = 1$)). As was mentioned in Section 3.3.3.1 and shown in Figure 3.7, when d was taken from 2 to 3 the test errors at stage 3 presented a upward trend. This fact indicated that in this situation, d taken as 3 produced inferior models, implying that d should not be larger than 3. The cases of $\Delta H = 2$ also lead to the same rule that d should not be larger than 3.

3.3.3.3 Overall Observations on Computational Effort

Computational effort is also a crucial criterion to assess the AVFA framework. Focusing on the nine "good" cases obtained above, from the results shown in Figure 3.9 some points are noted as follows:

- Three cases (A-(ΔH = 1, n^{max} = 4), A-(ΔH = 2, n^{max} = 3)) for Algorithm A and one case (B-($\Delta H = 1$, $d = 2$)) for Algorithm B were the most computationally efficient.
- One case (B-($\Delta H = 2$, $d = 3$)) for Algorithm B was the most time-consuming.
- The remaining five cases (A-(ΔH = 1, n^{max} = 5), A-(ΔH = 2, n^{max} = 4) for Algorithm A and B-($\Delta H = 1$, $d = 3$), B-($\Delta H = 2$, $d = 1$), B-($\Delta H = 2$, $d = 2$) for Algorithm B) were between the above extreme cases, and thus were moderate in computational effort.

Here it must be stated that it is unfair to directly compare the computational effort of the new algorithms to the fixed structure NN case with the results shown in Figure 3.9. The calculations for the computational effort for case FS did not consider the time taken by a trial-and-error process usually required in building a fixed structural NN model. This trial-and-error process is known to be time-consuming. Nevertheless, some rough and qualitative comparisons can still be made. To this end, it was assumed that, if case FS had to construct several approximations via trial-and-error to reach the current structure, each of the trials would have taken nearly same computational effort as in a FS case shown. Focusing on the nine "good" cases, it is easy to make the following arguments:

- If case FS had required three trial-and-error attempts to reach the current structure, then its real computational effort might be larger, on average, than the three most computationally efficient cases for Algorithms A and B, i.e., cases A-($\Delta H = 1$, $n^{max} = 4$), A-($\Delta H = 2$, $n^{max} = 3$), B-($\Delta H = 1$, $d = 2$).

- Similarly, if case FS had required, for example, seven trial-and-error attempts, its real computational effort might be larger than the five cases, i.e., A-($\Delta H = 1$, $n^{max} = 5$), A-($\Delta H = 2$, $n^{max} = 4$), B-($\Delta H = 1$, $d = 3$), B-($\Delta H = 2$, $d = 1$), B-($\Delta H = 2$, $d = 2$), that were thought to be moderate in computational effort.
- Finally, more than nine trial-and-error attempts would yield a real computational effort larger than the most time-consuming case, i.e., B-($\Delta H = 2$, $d = 3$), and it is not a stretch to imagine that more than nine trial-and-error attempts would be needed in practice.

Therefore, in reality the new AVFA sequential algorithms are more likely to require less total computational effort than the fixed structure method.

3.3.3.4 A Tradeoff on Model Accuracy and Computational Effort

In many cases, for a large-scale ADP problem, a tradeoff is needed between acceptable computational effort and model approximation accuracy for a real-world DP problem. In order to investigate the performance of the nine "good" cases for Algorithms A and B in this aspect, the mean values of the test errors and the computational times are averaged over the two stages of the models for the nine cases and plotted in Figure 3.10. The mean test error values obtained for the nine experimental cases actually are quite small (less than 1.7×10^{-5}) and quite close, testifying that all nine cases had a good model accuracy. The results further show the following:

- Case A-($\Delta H = 1$, $n^{max} = 5$) formed a front (or lucky case) that is very near the origin, and thus is the best case in a tradeoff on model approximation accuracy and computational effort.

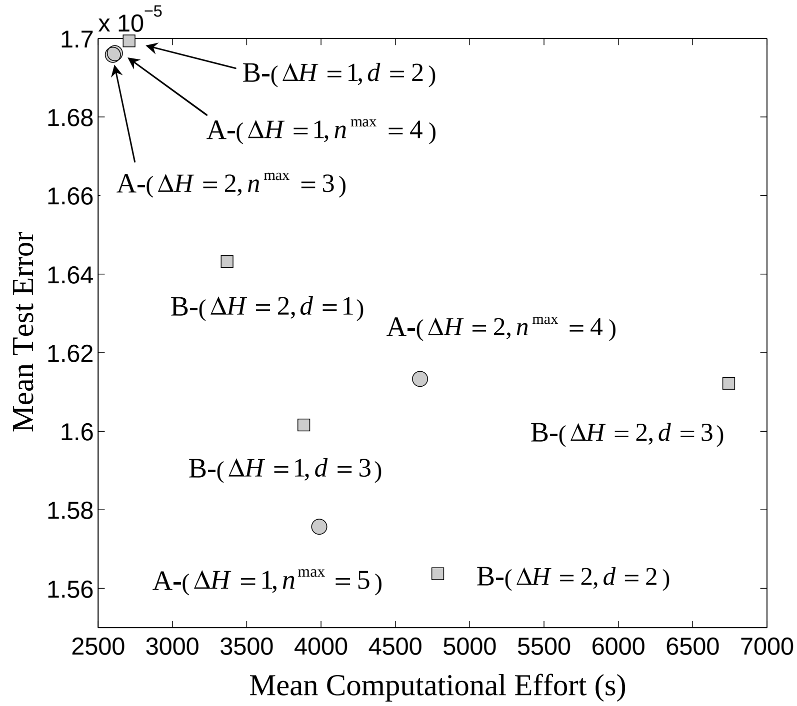


Figure 3.10. Mean test errors versus mean computational effort averaged over two stages for the NN model sets from the nine experimental cases by Algorithms A and B for future value function approximations.

- Considering the fact that the mean test error values for all the cases are very close, it can be concluded that all the cases except case B-($\Delta H = 2$, $d = 3$) formed a family of eight cases with good tradeoff performance since they are in general close each other and close to the origin.
- Obviously, case B-($\Delta H = 2$, $d = 3$) was quite poor with regard to computational effort compared with the others.
- Cases A-($\Delta H = 1$, $n^{max} = 4$), A-($\Delta H = 2$, $n^{max} = 3$), and B-($\Delta H = 1$, $d = 2$) clustered at the extreme area of model accuracy side, revealing that they were most computationally efficient cases among the "good" cases, as mentioned in Section 3.3.3.3.

3.3.3.5 Additional Remarks

Some additional remarks can be made from the above obtained results as follows:

- **On Hidden Nodes**: As was shown in Figure 3.8, the numbers of hidden nodes for the nine "good" cases were very dependent on the associated algorithmic parameters. Usually for a given training data size, more hidden nodes requires more computational effort in training. This effect can easily be seen by comparing Figures 3.8 and 3.9. An important finding on the hidden nodes is that the numbers of hidden nodes were quite insensitive to the model approximation accuracy, at least for those "good" cases: Within values ranging somewhere from 40 to 70, Algorithms A and B achieved acceptable models (here case B-($\Delta H = 2$, $d = 3$) was excluded for its bad tradeoff ability).
- **On Algorithms A and B**: Algorithm B was proposed originally to avoid extra computations that could occur in the model-search process of Algorithm A. The computational effort results for the two algorithms, shown in Figure 3.9, explicitly demonstrated this point. Comparing case A-($\Delta H = 1$, $n^{max} = 5$) for Algorithm A, with cases B-($\Delta H = 1$, $d = 2$) and B-($\Delta H = 1$, $d = 3$) in that $n^{max} = 5$ for Algorithm B, respectively, it can be seen that case A-($\Delta H = 1$, $n^{max} = 5$) exhibited clearly larger computational time than case B-($\Delta H = 1$, $d = 2$) and slightly larger computational time than case B-($\Delta H = 1$, $d = 3$). More importantly, Algorithm B was designed to tune its performance using the two parameters ΔH and d, and the introduction of the parameter d made the algorithm easy to tune because this parameter takes on very few values, i.e., 1, 2 and 3. By contrast, Algorithm A, which was designed to use the two parameters ΔH and n^{max} to tune its performance, tested the parameter n^{max} at values from 1 to 5. Hence, Algorithm B is both easier to use and more computationally efficient.

3.3.4 ADP Solution Simulations

The nine "good" cases for Algorithms A and B from Section 3.3.3 were further examined by using the associated NN models to simulate test scenarios for the inventory forecasting problem.

The test scenarios consisted of 100 initial state vectors, each simulating 1000 demand forecast ϵ sequences over which a mean cost is calculated to assess the quality of the resulting ADP solution. This setup is the same as that used in Cervellera et al. (2007) [14].

From Section 3.3.3 50 different sets of NN models were obtained for each experimental case (including the fixed structure NN case). For each of the nine "good" cases and the fixed structure case, the best and worst approximations, according to the average of the final test errors of the two stages, were selected for the ADP solution simulations. Thus, each experimental case yielded two simulated ADP solutions over the test scenarios. The solutions are denoted by adding "b" (best) and "w" (worst) in the corresponding notation for their experimental cases. All the simulated ADP solutions can then be listed as:

FS-b, FS-w for fixed structure NN case;

A-($\Delta H = 1$, $n^{max} = 4$)-b, A-($\Delta H = 1$, $n^{max} = 5$)-b,

A-($\Delta H = 2$, $n^{max} = 3$)-b, A-($\Delta H = 2$, $n^{max} = 4$)-b,

A-($\Delta H = 1$, $n^{max} = 4$)-w, A-($\Delta H = 1$, $n^{max} = 5$)-w,

A-($\Delta H = 2$, $n^{max} = 3$)-w, A-($\Delta H = 2$, $n^{max} = 4$)-w

for Algorithm A; and

B-($\Delta H = 1$, $d = 2$)-b, B-($\Delta H = 1$, $d = 3$)-b,

B-($\Delta H = 2$, $d = 1$)-b, B-($\Delta H = 2$, $d = 2$)-b,

B-($\Delta H = 2$, $d = 3$)-b,

B-($\Delta H = 1$, $d = 2$)-w, B-($\Delta H = 1$, $d = 3$)-w, B-($\Delta H = 2$, $d = 1$)-w, B-($\Delta H = 2$, $d = 2$)-w,

B-($\Delta H = 2$, $d = 3$)-w

for Algorithm B.

The simulations were conducted using the 100 given initial state vectors. A simulated solution provided 100 mean costs (corresponding to the 100 initial state vectors), each over 1000 sequences. To assess the ADP solution quality, 36 AVFA ADP solutions taken from throughout this chapter were each simulated. For each initial state vector, the smallest mean cost achieved by a set of 36 simulated ADP solutions was used as the "true" mean cost for computing the "absolute error" as measure of ADP solution quality. Among the 36 ADP solutions, 20 are examined in this section: 4 best cases and 4 worst cases for Algorithm A, 5 best cases and 5 worst cases for Algorithm B, 1 best case and 1 worst case for the fixed structure method. The boxplots of the absolute errors for the 20 simulated solutions are shown in Figure 3.11.

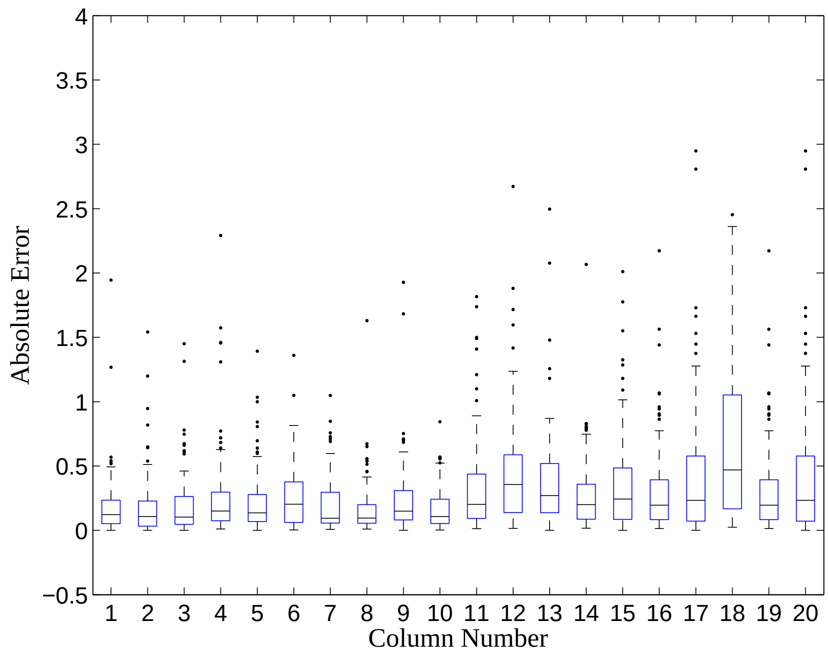


Figure 3.11. Boxplots that illustrate the distributions of absolute errors for all 18 simulated ADP solutions by Algorithms A and B and 2 simulated ADP solutions by the fixed structure method.

Note: The column numbers and the corresponding simulation cases are as follows:

1 — FS-b	11 — FS-w
2 — A-($\Delta H = 1$, $n^{max} = 4$)-b	12 — A-($\Delta H = 1$, $n^{max} = 4$)-w
3 — A-($\Delta H = 1$, $n^{max} = 5$)-b	13 — A-($\Delta H = 1$, $n^{max} = 5$)-w
4 — A-($\Delta H = 2$, $n^{max} = 3$)-b	14 — A-($\Delta H = 2$, $n^{max} = 3$)-w
5 — A-($\Delta H = 2$, $n^{max} = 4$)-b	15 — A-($\Delta H = 2$, $n^{max} = 4$)-w
6 — B-($\Delta H = 1$, $d = 2$)-b	16 — B-($\Delta H = 1$, $d = 2$)-w
7 — B-($\Delta H = 1$, $d = 3$)-b	17 — B-($\Delta H = 1$, $d = 3$)-w
8 — B-($\Delta H = 2$, $d = 1$)-b	18 — B-($\Delta H = 2$, $d = 1$)-w
9 — B-($\Delta H = 2$, $d = 2$)-b	19 — B-($\Delta H = 2$, $d = 2$)-w
10 — B-($\Delta H = 2$, $d = 3$)-b	20 — B-($\Delta H = 2$, $d = 3$)-w

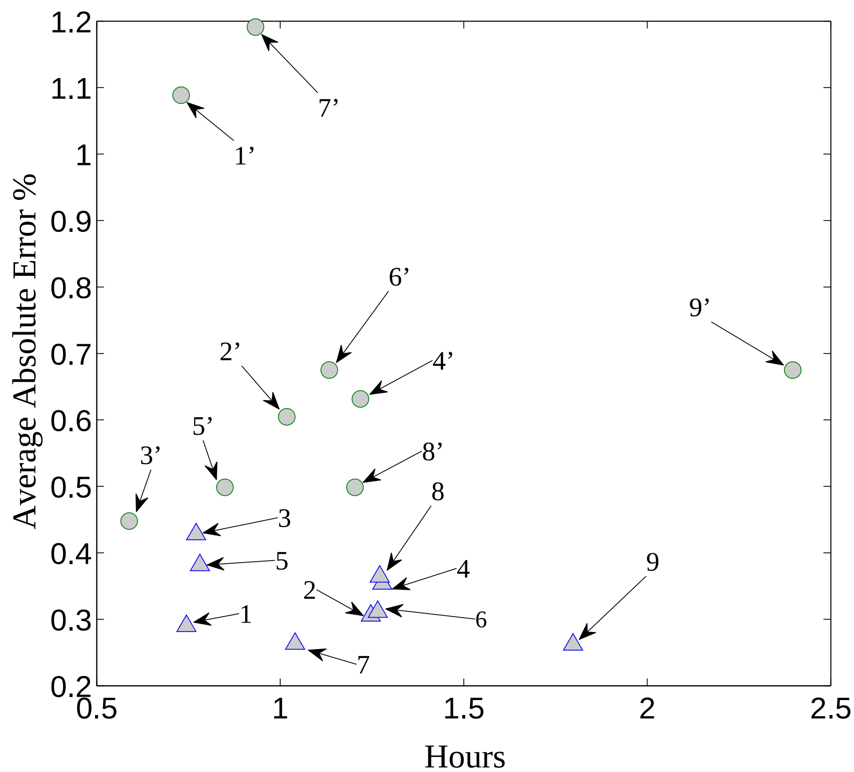


Figure 3.12. Average absolute error as a percentage of 61.81 against the computational time (hours) required to build the associated "best" and "worst" NN model sets for the nine "good" experimental cases by Algorithms A and B and for the fixed structure case. Note: The index numbers and the corresponding simulation cases are as follows:

1 — A-($\Delta H = 1$, $n^{max} = 4$)-b	1' — A-($\Delta H = 1$, $n^{max} = 4$)-w
2 — A-($\Delta H = 1$, $n^{max} = 5$)-b	2' — A-($\Delta H = 1$, $n^{max} = 5$)-w
3 — A-($\Delta H = 2$, $n^{max} = 3$)-b	3' — A-($\Delta H = 2$, $n^{max} = 3$)-w
4 — A-($\Delta H = 2$, $n^{max} = 4$)-b	4' — A-($\Delta H = 2$, $n^{max} = 4$)-w
5 — B-($\Delta H = 1$, $d = 2$)-b	5' — B-($\Delta H = 1$, $d = 2$)-w
6 — B-($\Delta H = 1$, $d = 3$)-b	6' — B-($\Delta H = 1$, $d = 3$)-w
7 — B-($\Delta H = 2$, $d = 1$)-b	7' — B-($\Delta H = 2$, $d = 1$)-w
8 — B-($\Delta H = 2$, $d = 2$)-b	8' — B-($\Delta H = 2$, $d = 2$)-w
9 — B-($\Delta H = 2$, $d = 3$)-b	9' — B-($\Delta H = 2$, $d = 3$)-w

The following basis is used to compare the model accuracy for these simulation results:

- Two experimental cases are said to be equivalent when they have both equivalent “best” and ”worst” simulated solutions (of course, a model can be deemed better/worse than another when the former has both better/worse “best” and ”worst” simulated solutions than the latter, but the current results did not include such a pure situation);
- A model is said to be comparable to another model if the former has one simulated solution equivalent or better and one simulated solution worse than the latter.

Comparing the nine experimental cases for Algorithms A and B with the fixed structure experimental case, from the results shown in Figure 3.11, the following is observed:

- Three cases, namely A-($\Delta H = 2$, $n^{max} = 3$) for Algorithm A, B-($\Delta H = 1$, $d = 2$) and B-($\Delta H = 2$, $d = 2$) for Algorithm B, were equivalent to case FS.
- The remaining cases were comparable to case FS.

It is noted that for the experimental case B-($\Delta H = 2$, $d = 1$), its “worst” NN model set gave in an inferior ADP solution compared to the case FS and the other experimental cases, while the corresponding result for its “best” NN model set, in contrast, was among the better ADP solutions. Therefore, this experimental case is comparable to the others. Nevertheless, it can be argued that this case may not have good robustness compared to the others. In summary, all nine experimental cases were equivalent or at least comparable to the fixed structure case for these test scenarios.

The smallest mean-costs for 100 initial state vectors obtained above averaged 61.81. The average absolute error as a percentage of 61.81 for each simulated solution for Algorithms A and B was also studied against the run time required to build the associated “best” or “worst” NN models sets, in order to further observe the tradeoff performance of the compared experimental cases in ADP solution quality and in computational effort, as depicted in Figure 3.12. (The fixed structure case was excluded in this comparison for the same reason as mentioned in Section 3.3.3.3.) From the results, the following points were derived:

- The two pairs of simulated solutions, (A-(ΔH = 2, n^{max} = 3)-b, A-(ΔH = 2, n^{max} = 3)-w) and (B-(ΔH = 1, d = 2)-b, B-(ΔH = 1, d = 2)-w), with their members close one to another, were most close to the origin, indicating that in such a tradeoff viewpoint, the two experimental cases A-($\Delta H = 2$, $n^{max} = 3$) and B-($\Delta H = 1$, $d = 2$) were the "lucky" bests;
- The pairs of simulated solutions (A-($\Delta H = 1$, $n^{max} = 5$)-b, A-($\Delta H = 1$, $n^{max} = 5$)-w), (A-($\Delta H = 2$, $n^{max} = 4$)-b, A-($\Delta H = 2$, $n^{max} = 4$)-w), (B-($\Delta H = 1$, $d = 3$)-b, B-($\Delta H = 1$, $d = 3$)-w), and (B-($\Delta H = 2$, $d = 2$)-b, B-($\Delta H = 2$, $d = 2$)-w) together with the above lucky two pairs were clustered together in an area quite close to the origin, implying that the associated 6 experimental cases, A-($\Delta H = 2$, $n^{max} = 3$), B-($\Delta H = 1$, $d = 2$), A-($\Delta H = 1$, $n^{max} = 5$), A-($\Delta H = 2$, $n^{max} = 4$), B-($\Delta H = 1$, $d = 3$), and B-($\Delta H = 2$, $d = 2$), in general had good tradeoff performance.
- The two pairs of the simulated solutions (A-($\Delta H = 1$, $n^{max} = 4$)-b, A-($\Delta H = 1$, $n^{max} = 4$)-w) and (B-($\Delta H = 2$, $d = 1$)-b, B-($\Delta H = 2$, $d = 1$)-w) were quite disparate in the average absolute error direction. Thus the associated model cases, A-($\Delta H = 1$, $n^{max} = 4$) and B-($\Delta H = 2$, $d = 1$) may lack robustness in ADP solution quality.
- The pair of simulated solutions (B-($\Delta H = 2$, $d = 3$)-b, B-($\Delta H = 2$, $d = 3$)-w) was far from its fellows in the hours direction, implying that the experimental case B-($\Delta H = 2$, $d = 3$) was the most time-consuming case.

The above observations are, in principle, consistent with what was obtained for the nine cases from the previous results. For example, the good tradeoff groups defined in Figure 3.10 and Figure 3.12 shared at least 6 of the same members; the bad tradeoff case B-($\Delta H = 2$, $d = 3$) in computational effort was detected in both above tradeoff figures; and the cases A-($\Delta H = 1$, $n^{max} = 4$) and B-($\Delta H = 2$, $d = 1$) that had disparate performance between their "best" and "worst" simulated solutions in average absolute error in Figure 3.12 also exhibit disparate results in Figure 3.11.

3.3.5 An Extensive Study on Stopping Criteria

The AVFA algorithms were mainly studied by stopping at a given number of training data points (1350), so that a comparison of the new algorithms with the fixed structure method could be made under the similar conditions. This type of stopping rule is equivalent to specifying a maximum number of sequential steps, provided the increment of the training data set, ΔN, is fixed. (For Algorithm C the two aforementioned stopping criteria are not equivalent.) Though somewhat naive, these stopping criteria are easy ways to stop a sequential algorithm like Algorithms A and B.

In practice, one would desire a more generic, adaptive and automatic stopping criterion that is applicable to any of the AVFA algorithms. Such a criterion may use a model-performance measure, e.g., test error or R^2 (R^2 is the coefficient of determination in regression analysis, and in this study it means the test R^2, i.e., the R^2 based on the test data), to form a stopping parameter, denoted as S_l hereafter, that can be queried in each sequential step l and can be given a threshold value at which the algorithm would stop.

As was mentioned previously, due to the inherent stochasticity in a model-building process, a model performance measure is usually random as well. Direct use of it as a stopping criterion is inappropriate, since a stopping parameter should ideally vary robustly and monotonically with model performance. To enhance robustness, a "moving-band" concept was proposed, which is like a "moving average" concept from statistics. The moving-band is defined as the difference between the maximum and the minimum values of a performance measure for given number of successive sequential steps within a window moving along with the sequential steps of the algorithm. Let MB represent a moving-band and p denote a performance measure, for sequential step l. Then a moving band can mathematically be written as

$$MB_l(p : r) = max\{p_{l-r+1}, ..., p_{l-1}, p_l\} - min\{p_{l-r+1}, ..., p_{l-1}, p_l\}, \tag{3.10}$$

where r is the length of the moving window.

Based on this concept, a stopping criterion was presented as:

$$S_l = MB_l(R^2 : r) \leq c \tag{3.11}$$

where the stopping parameter S_l relied on the model performance measure R^2, and the threshold (or error tolerance) c given beforehand. Since a model's R^2 value is always in [0,1], this criteria has the advantage that it can easily be given with a threshold for stopping a sequential algorithm.

Two experimental cases, namely A-($\Delta H = 2$, $n^{max} = 3$) for Algorithm A and B-($\Delta H = 1$, $d = 2$) for Algorithm B, were arbitrarily chosen to test the newly defined stopping criterion. The two new experimental cases with the stopping criterion were denoted as A-($\Delta H = 2$, $n^{max} = 3$)(R^2-stop) and B-($\Delta H = 1$, $d = 2$)(R^2-stop), respectively. While the length of the moving-band r and the threshold c were given through trial-and-error as $r = 6$ and $c = 0.0002$, respectively, the initial conditions and the other algorithmic parameters were kept identical to their original experimental cases. Fifty trial runs were performed for each of the new experimental cases. The experimental results were compared with the corresponding original experimental cases and the fixed structure case in Figures 3.13-3.15, in which Figure 3.13 showed the boxplots of the test errors, Figure 3.14 showed the boxplots of the numbers of the hidden nodes and the size of training data, and Figure 3.15 showed the computational effort, for the obtained NN model sets. The two new experimental cases were also simulated over the test scenarios as described in Section 3.3.4, with their "best" and "worst" model sets among the 50 runs. This yielded four simulated solutions denoted as A-($\Delta H = 2$, $n^{max} = 3$)(R^2-stop)-b, B-($\Delta H = 1$, $d = 2$)(R^2-stop)-b, and A-($\Delta H = 2$, $n^{max} = 3$)(R^2-stop)-w, B-($\Delta H = 1$, $d = 2$)(R^2-stop)-w. The obtained results were compared in Figure 3.16 with the simulated solutions for the original experimental cases where the training data size was given for stopping the algorithms, and with the simulated solutions for the fixed structure case.

From the results, under the given stopping threshold value and window length of the moving-band for the defined stopping criterion, the new experimental cases by Algorithms A and B successfully found the NN model sets that were equivalent or better than the corresponding original cases and the fixed structure case. The detailed observations from those results include:

- The new experimental cases were equivalent to or slightly better than their original cases in model accuracy, especially at stage 3, as shown in Figure 3.13.
- The number of hidden nodes and the size of the training data for the new experimental cases were equivalent to or larger than in their original cases, as shown in Figure 3.14.
- The computational effort of the new experimental cases was larger than their original cases, as shown in Figure 3.15.

The above results are reasonable, since according to the consistency theory mentioned in Section 3.1 the new experimental cases with higher model accuracy should require more training data and more hidden nodes, and consequently more computational effort. The simulation results on the ADP solution quality over the test scenarios show that the experimental cases with the new stopping criterion performed a little bit better than or comparably to their original experimental cases and the fixed structure case.

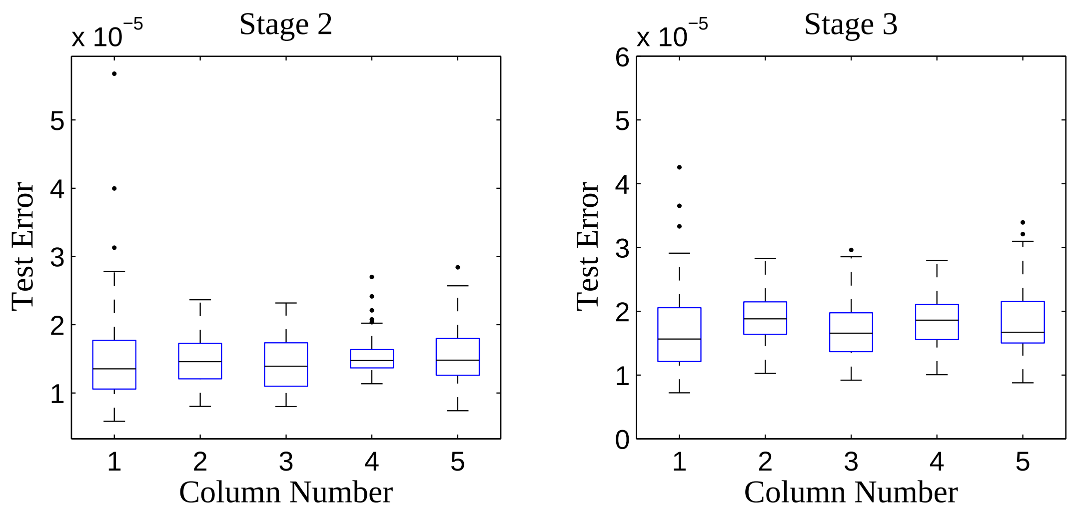


Figure 3.13. Boxplots of the training errors and the test errors for the NN model sets for two experimental cases by Algorithms A and B using the new stopping criterion in comparison with the experimental cases by Algorithms A and B using the given training data size as the stopping criterion and the fixed structure case for future value function approximations.
Note: In Figures 3.13-3.15, the column numbers and the corresponding experimental cases are as follows:
1 — FS
2 — A-($\Delta H = 2$, $n^{max} = 3$)
3 — A-($\Delta H = 2$, $n^{max} = 3$)(R^2-stop)
4 — B-($\Delta H = 1$, $d = 2$)
5 — B-($\Delta H = 1$, $d = 2$)(R^2-stop)

The results for the threshold with three different values, namely, $c = 0.0002$, 0.0003 and 0.0005, were also shown in boxplots of the test errors over the 50 runs in Figure 3.17. When c was 0.0002, the test errors, on average, were close to the fixed structure case, so this value of c is used in the current study.

Finally, it is worth mentioning that another performance measure, namely test error, can be used to form a similar stopping criterion. In this case, the R^2 in Equation (3.11) can be replaced by e_{ts}, with a scaling technique to make sure that the resultant moving-band is within a reasonable range, for instance, in [0 1].

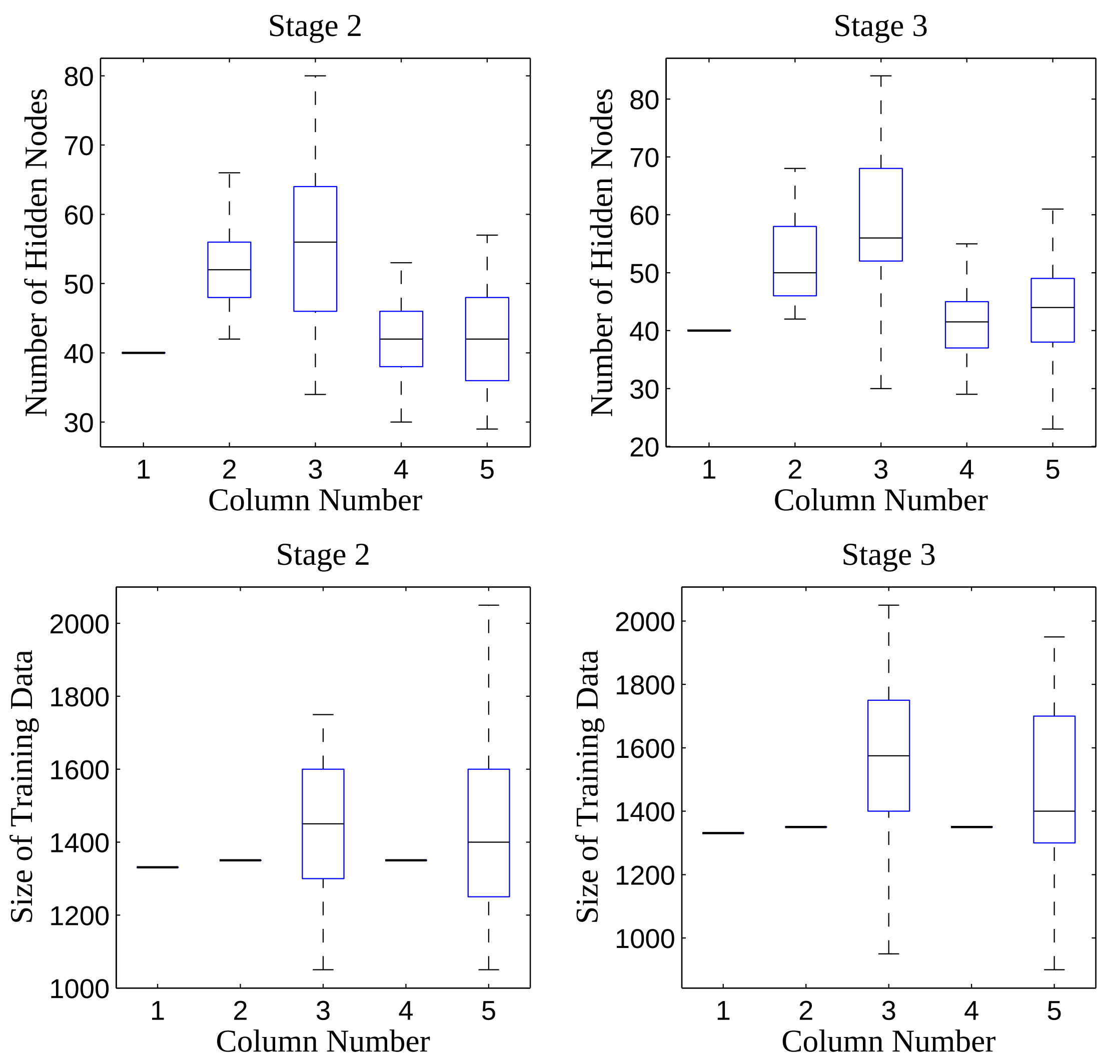


Figure 3.14. Boxplots of the numbers of hidden nodes and the size of training data for the NN sets for two experimental cases by Algorithms A and B using the new stopping criterion in comparison with the experimental cases by Algorithms A and B using the given training data size as the stopping criterion and the fixed structure case for future value function approximations.

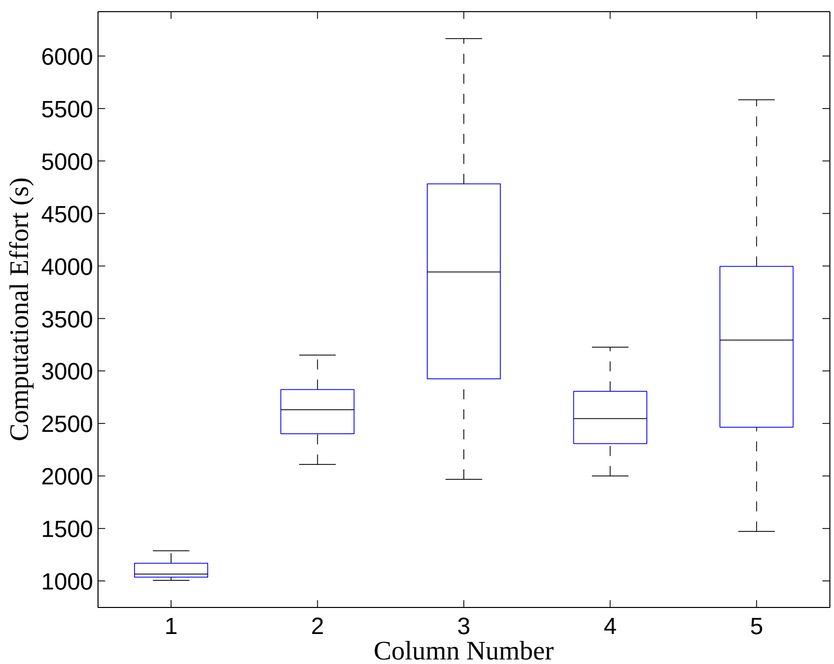


Figure 3.15. Boxplots of the computational effort (seconds) for the NN sets for two experimental cases by Algorithms A and B using the new stopping criterion in comparison with the experimental cases by Algorithms A and B using the given training data size as the stopping criterion and the fixed structure case for future value function approximations.

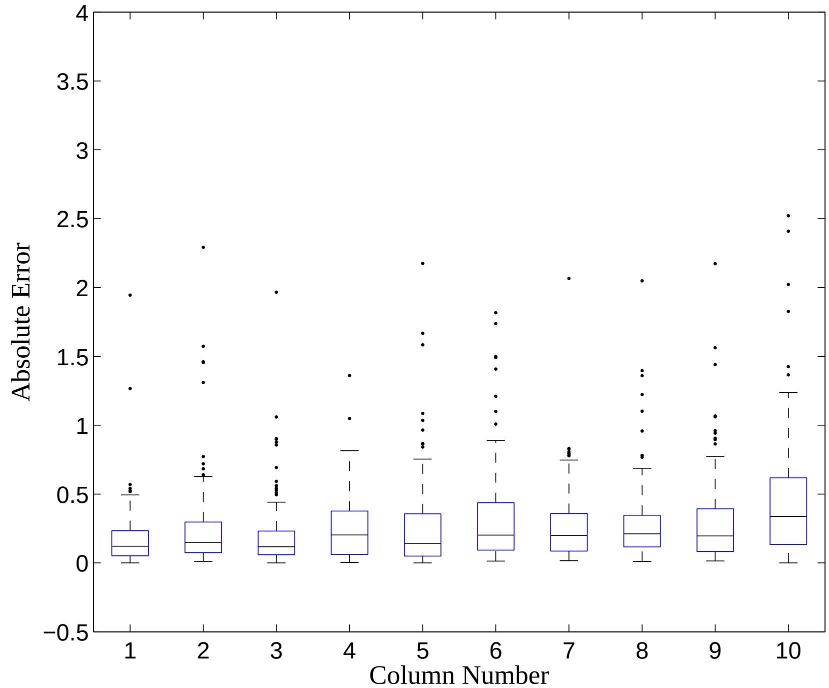


Figure 3.16. Boxplots that illustrate the distributions of absolute errors for four simulated ADP solutions by Algorithms A and B using the new stopping criterion four simulated ADP solutions by Algorithms A and B using the given training data size as the stopping criterion and two simulated ADP solutions by the fixed structure method.

Note: The column numbers and the corresponding simulation cases are as follows:

1 — FS-b	6 — FS-w
2 — A-($\Delta H = 2$, $n^{max} = 3$)-b	7 — A-($\Delta H = 2$, $n^{max} = 3$)-w
3 — A-($\Delta H = 2$, $n^{max} = 3$)(R^2-stop)-b	8 — A-($\Delta H = 2$, $n^{max} = 3$)(R^2-stop)-w
4 — B-($\Delta H = 1$, $d = 2$)-b	9 — B-($\Delta H = 1$, $d = 2$)-w
5 — B-($\Delta H = 1$, $d = 2$)(R^2-stop)-b	10 — B-($\Delta H = 1$, $d = 2$)(R^2-stop)-w

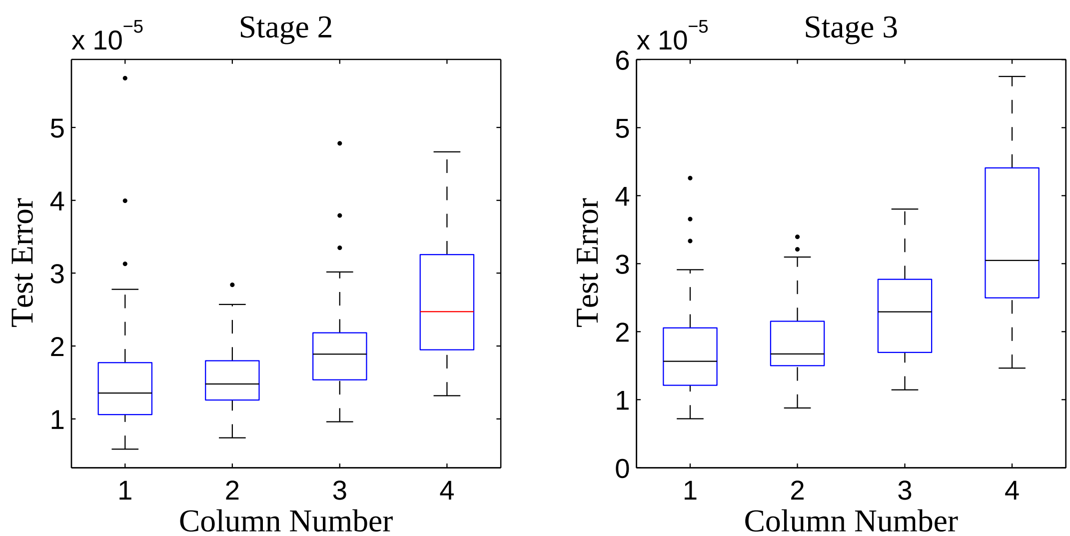


Figure 3.17. Boxplots of the test errors for the NN model sets for the experimental case by Algorithm B using the new stopping criterion with three different stopping threshold values in comparison with the fixed structure case.

Note: The column numbers and the corresponding experimental cases are as follows:

1 — FS

2 — B-($\Delta H = 1$, $d = 2$)(R^2-stop), $c = 0.0002$

3 — B-($\Delta H = 1$, $d = 2$)(R^2-stop), $c = 0.0003$

4 — B-($\Delta H = 1$, $d = 2$)(R^2-stop), $c = 0.0005$

3.3.6 Illustrations of Consistency Features

As all the proposed AVFA algorithms theoretically should follow a consistency trace, in this section, some consistency features for Algorithms A and B are illustrated. These illustrations utilize the aforementioned experiments plus some additional experiments with longer runs. Each of the longer runs was stopped at the 91-st sequential step, resulting in 4850 training data points. Longer run results were obtained for all four "good" cases for Algorithm A and only one case (i.e., B-($\Delta H = 1$, $d = 2$)) for Algorithm B. Two cases A-($\Delta H = 1$, $n^{max} = 4$) and A-($\Delta H = 2$, $n^{max} = 4$) for Algorithm A were chosen to be demonstrated, in which the former was moderate in performance while the latter averaged the highest numbers of the hidden nodes among the experimental cases for Algorithm A, and therefore was thought to have higher risk of deviating from a consistency trace due to overfitting of the training data. The only case for Algorithm B was chosen arbitrarily. For each of the chosen experimental cases, its longer run test error history together with the "shorter" 50-run test error histories recorded from the corresponding experiments in Section 3.3.3 were presented in Figures 3.18-20. For the chosen case for Algorithm B the corresponding experimental results for the algorithm using "R^2-stop" were available and hence were also used in these illustrations. Figure 3.21 showed the longer run test error history for the case for Algorithm B used in Figure 3.20 and the 50-run test error histories recorded from the experiments for the related "R^2-stop" case in Section 3.3.5. The results in these figures demonstrated that for all the selected cases, the test error histories presented trends that strongly support the assumption that the new algorithms follow consistency traces. In particular, the following observations can be made:

- The test error histories from all the longer runs for the selected three experimental cases smoothly stretched toward the direction of increasing both hidden nodes and the training data, which is the same direction as a statistical consistency trace derived in Section 3.1.2.

- The test error histories from the 50 runs of the experimental cases in Section 3.3.3 and of the experimental case in Section 3.3.5 closely followed the associated longer run histories, implying that they have the same trend as the longer runs.

In addition, case A-($\Delta H = 2$, $n^{max} = 4$) for Algorithm A did not present an inferior trend compared to the other chosen cases, implying that overfitting was not an issue.

The NN model sets obtained by the longer runs of the three experimental cases were simulated under the test scenarios used in Section 3.3.4, and the results were shown in Figure 3.22, in comparison with the "best" simulated solutions for the chosen cases and the fixed structure case. The absolute errors of the simulation results from the longer-run model sets were lower than those from the "best" simulated solution among the chosen experimental cases for Algorithms A and B and the fixed structure case. This again indicated that Algorithms A and B were consistent: Running Algorithms A or B longer will yield a more precise model, provided the computational requirement is affordable.

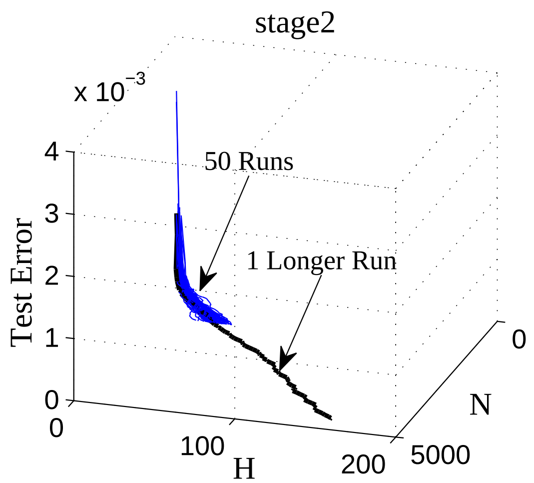


Figure 3.18. Test errors histories of a longer run and the 50 runs for the NN model sets for experimental case A-($\Delta H = 1$, $n^{max} = 4$) by Algorithm A.

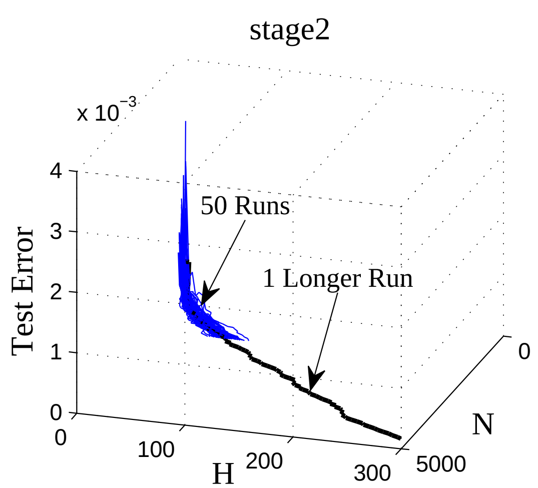


Figure 3.19. Test errors histories of a longer run and the 50 runs for the NN model sets for experimental case A-($\Delta H = 2$, $n^{max} = 4$) by Algorithm A.

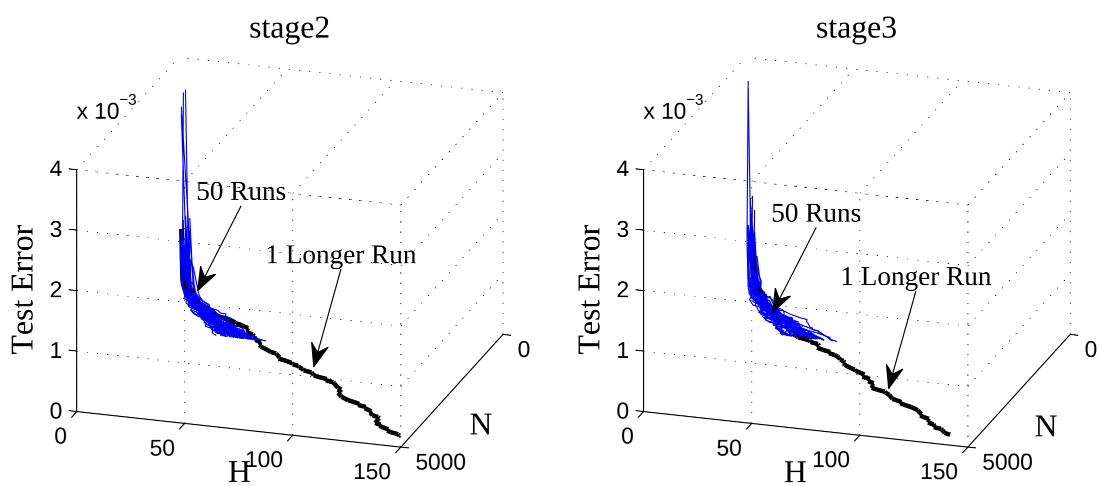


Figure 3.20. Test errors histories of a longer run and the 50 runs for the NN model sets for experimental case B-($\Delta H = 1$, $d = 2$) by Algorithm B.

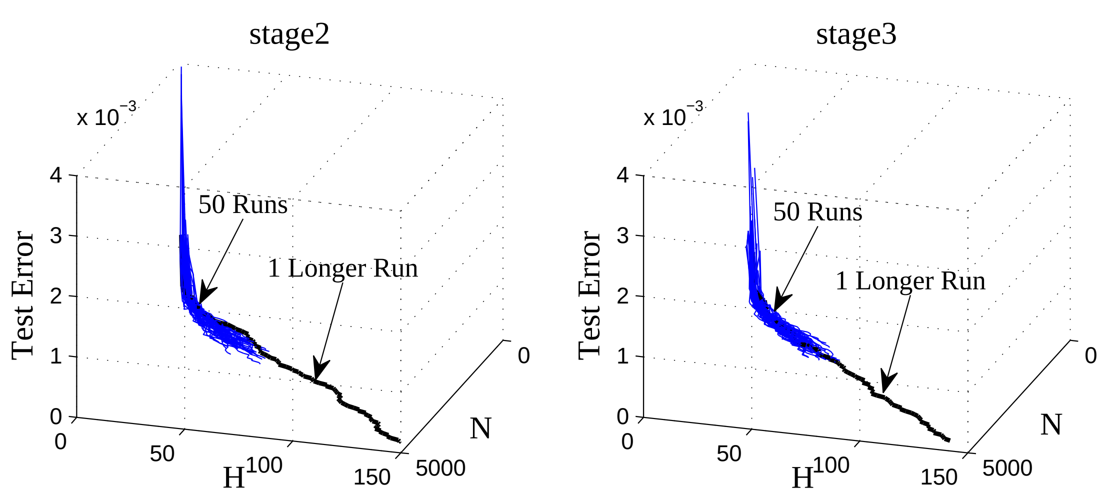


Figure 3.21. Test errors histories of a longer run and the 50 runs for the NN model sets for experimental case B-($\Delta H = 1$, $d = 2$)(R^2-stop) for Algorithm B.

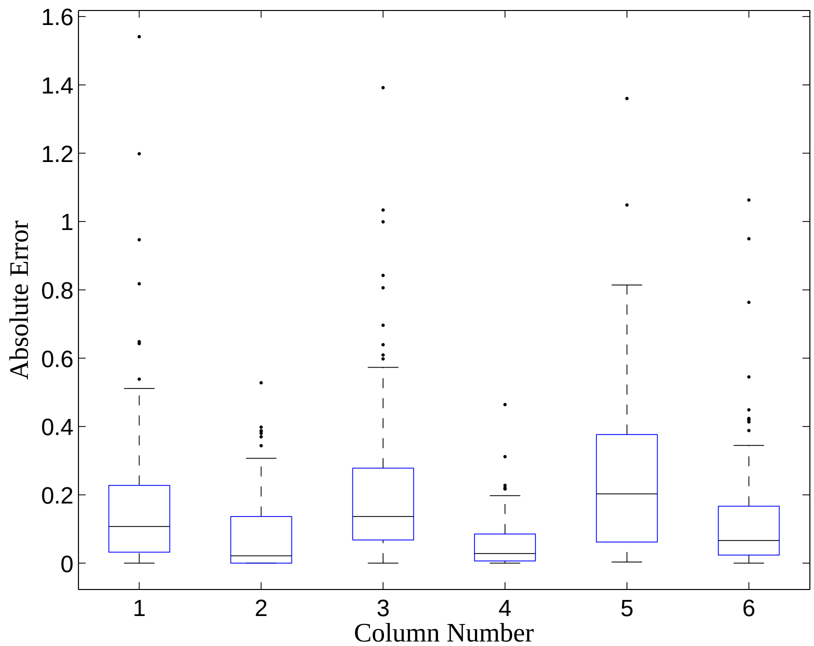


Figure 3.22. Boxplots that illustrate the distributions of absolute errors for the simulated ADP solutions of the "best" NN model sets for experimental cases A-($\Delta H = 1$, $n^{max} = 4$), A-($\Delta H = 2$, $n^{max} = 4$) and B-($\Delta H = 1$, $d = 2$) in comparison with simulated ADP solutions of their longer run NN model sets.
Note: The column numbers and the corresponding cases are as follows:
1 — A-($\Delta H = 1$, $n^{max} = 4$)-b
2 — A-($\Delta H = 1$, $n^{max} = 4$), one longer run
3 — A-($\Delta H = 2$, $n^{max} = 4$)-b
4 — A-($\Delta H = 2$, $n^{max} = 4$), one longer run
5 — B-($\Delta H = 1$, $d = 2$)-b
6 — B-($\Delta H = 1$, $d = 2$), one longer run

3.3.7 Discussions on Algorithm C

Algorithm C has the same theoretical background as Algorithms A and B, and was thought to be more adaptive than Algorithms A and B since it gradually varies both the model complexity and the training data size. The fact that the experimental cases by Algorithms C were clearly worse than the cases by Algorithms A and B is unclear and needs further study. Nevertheless, Algorithm C is still considered as a potentially good sequential algorithm. This opinion is based on a few of the clues from some additional experiments.

To explore Algorithm C beyond the given training data size limit in Section 3.3.3, for each of the three experimental cases of this algorithm (i.e., C-($\Delta H = 1$), C-($\Delta H = 2$) and C-($\Delta H = 3$)), an additional single run of the algorithm with "R^2-stop" studied in Section 3.3.5 was conducted with the stopping threshold c taken as 0.0002. The test error values of the NN model sets for the inventory forecasting problem obtained from these three runs were compared in Figure 3.23 with the corresponding 50-run results for the three experimental cases and the fixed structure case discussed in Section 3.3.3. The models resulting from each of the additional runs had the test error values lower than the smallest test error values achieved by the model sets established from the corresponding 50 runs for the three experimental cases. It was observed that the single runs usually took much longer than in their associated 50-run cases. It is hence believed that the algorithm could do well at least in model accuracy, though it may perform very poor in computational effort. It also implies that any attempt to improve this algorithm might focus on a decrease of its computational times.

The consistency trend of Algorithm C is also illustrated in Figure 3.24, in which the test error history of the additional single run and the 50-run test error histories of the experimental case C-($\Delta H = 2$) with respect to the number of hidden nodes, H, and the size of the training data, N, recorded in Section 3.3.3 were plotted. A good trend can been seen from this figure, which verifies the assumption of consistency for Algorithm C.

The NN model sets obtained by the additional single runs of the three experimental cases were simulated with the same test scenarios used in Section 3.3.4. For each of the three experimental cases for Algorithm C, the best and worst NN models from the 50 runs were also simulated with the same test scenarios, yielding six simulated solutions denoted as:

C-($\Delta H = 1$)-b, C-($\Delta H = 2$)-b, C-($\Delta H = 3$)-b,

C-($\Delta H = 1$)-w, C-($\Delta H = 2$)-w, C-($\Delta H = 3$)-w.

The absolute errors of all the above nine simulated solutions were shown in Figure 3.25 together with the simulated solutions for the fixed structure case. Among the three single run simulated solutions, two, i.e., the simulated solutions of the single runs for experimental cases C-($\Delta H = 1$) and C-($\Delta H = 2$), were better than the "best" simulated solutions for the experimental cases for Algorithm C. In particular, the single run for case C-($\Delta H = 2$) was comparable with the simulated solutions for the fixed structure case. This means that Algorithm C is able to do as well as Algorithms A and B in ADP solution quality.

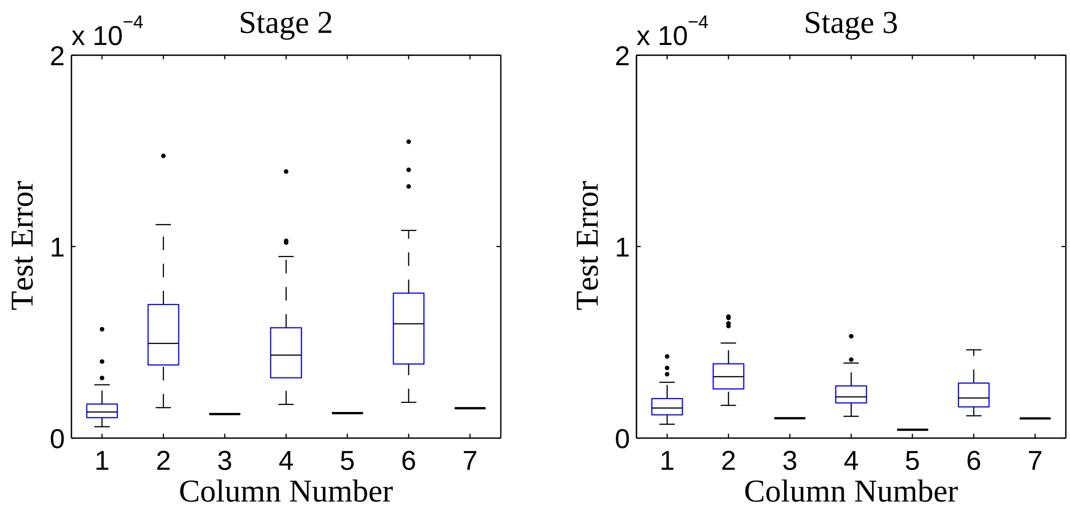


Figure 3.23. Test error values and boxplots for the NN model sets for the single runs and the three experimental cases by Algorithm C and the fixed structure case for future value function approximations.
Note: The column numbers and the corresponding cases are as follows:
1 — FS
2 — C-($\Delta H = 1$)
3 — C-($\Delta H = 1$), R^2-stop, one run
4 — C-($\Delta H = 2$)
5 — C-($\Delta H = 2$), R^2-stop, one run
6 — C-($\Delta H = 3$)
7 — C-($\Delta H = 3$), R^2-stop, one run

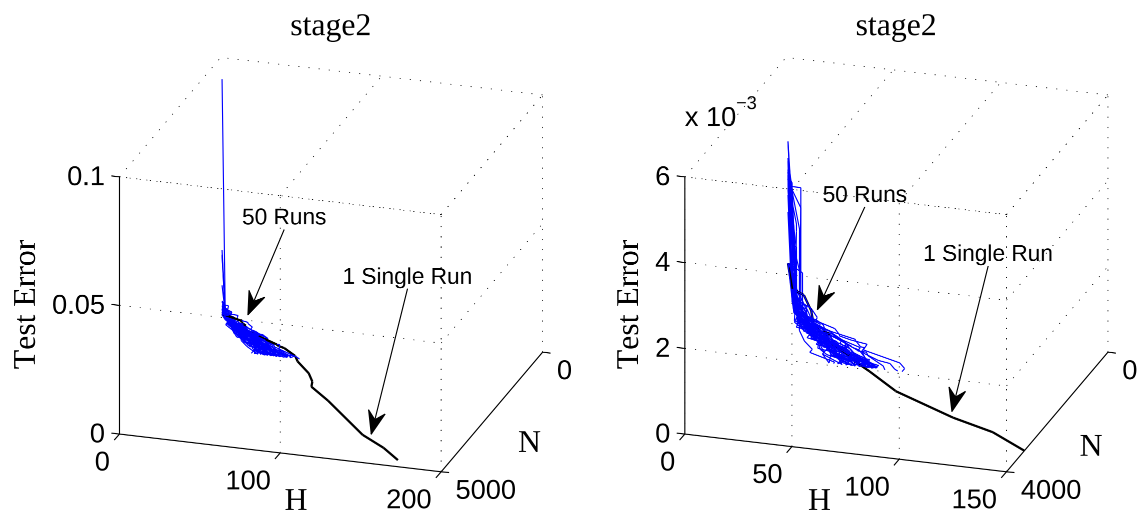


Figure 3.24. The test errors histories for the NN model sets of the single run and the 50-run experimental case C-($\Delta H = 2$) for Algorithm C.

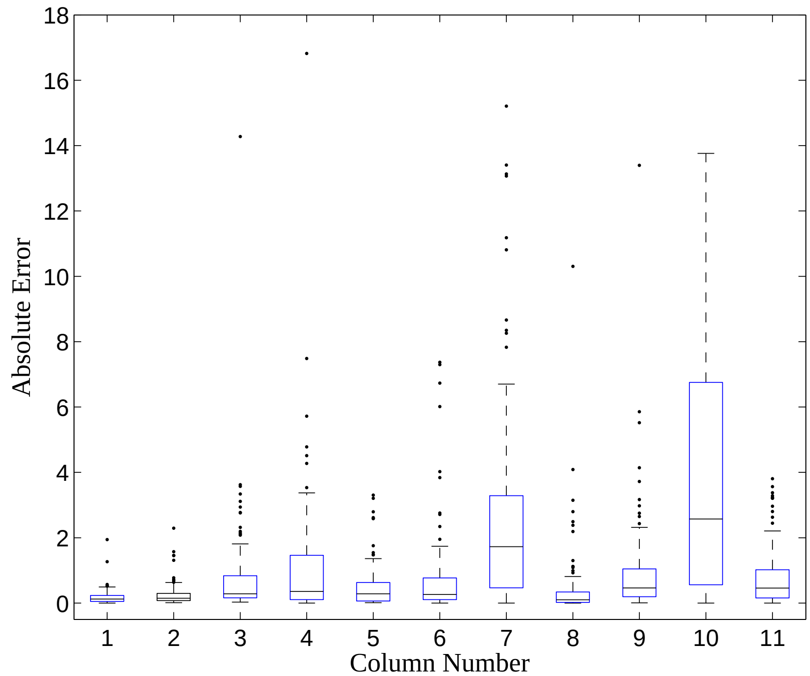


Figure 3.25. Boxplots that illustrate the distributions of absolute errors for the simulated ADP solutions by the three single runs and the associated experimental cases for Algorithm C and by the fixed structure case.

Note: The column numbers and the corresponding cases are as follows:

1 — FS-b
2 — FS-w
3 — C-($\Delta H = 1$)-b
4 — C-($\Delta H = 1$)-w
5 — C-($\Delta H = 1$), R^2-stop, one run
6 — C-($\Delta H = 2$)-b
7 — C-($\Delta H = 2$)-w
8 — C-($\Delta H = 2$), R^2-stop, one run
9 — C-($\Delta H = 3$)-b
10 — C-($\Delta H = 3$)-w
11 — C-($\Delta H = 3$), R^2-stop, one run

CHAPTER 4

SEQUENTIAL STATE SPACE EXPLORATION (SSSE)

The adaptive value function approximation (AVFA) algorithms discussed in the previous chapter can both adaptively identify the model structure and state space sample size for the future value function approximation of a finite horizon ADP problem, assuming its state space region is given for each stage. Unfortunately, in most practical ADP problems the state space region is typically unknown. Though it is crucial to the solution of an ADP problem, an appropriate specification of the state space region is not straightforward. In this chapter, a second type of sequential framework is developed to conduct state space exploration forward through the stages, and hence can identify the appropriate state space regions for modeling the future value functions. This forward sequential state space exploration (SSSE) approach induces an iterative forward-backward framework (called "SSSE framework") for solving finite-horizon ADP problems. By combining the SSSE and AVFA approaches in the statistical perspective, a self-organized "learning" of an ADP solution can be realized.

4.1 Forward Exploration

As the SSSE approach is also a sequential procedure, without loss of generality, as in Chapter 3, take l to index the sequential steps for a forward exploration process as well. The following notation is used to describe such a process:

$DF_{t,l}$, $DF_{t+1,l}$ — the state space sample data sets for stage t and $t+1$, respectively

$NF_{t,l}$, $NF_{t+1,l}$ — the sizes of the sample data sets for stage t and $t+1$, respectively

ΔNF — an increment of the size of the sample data

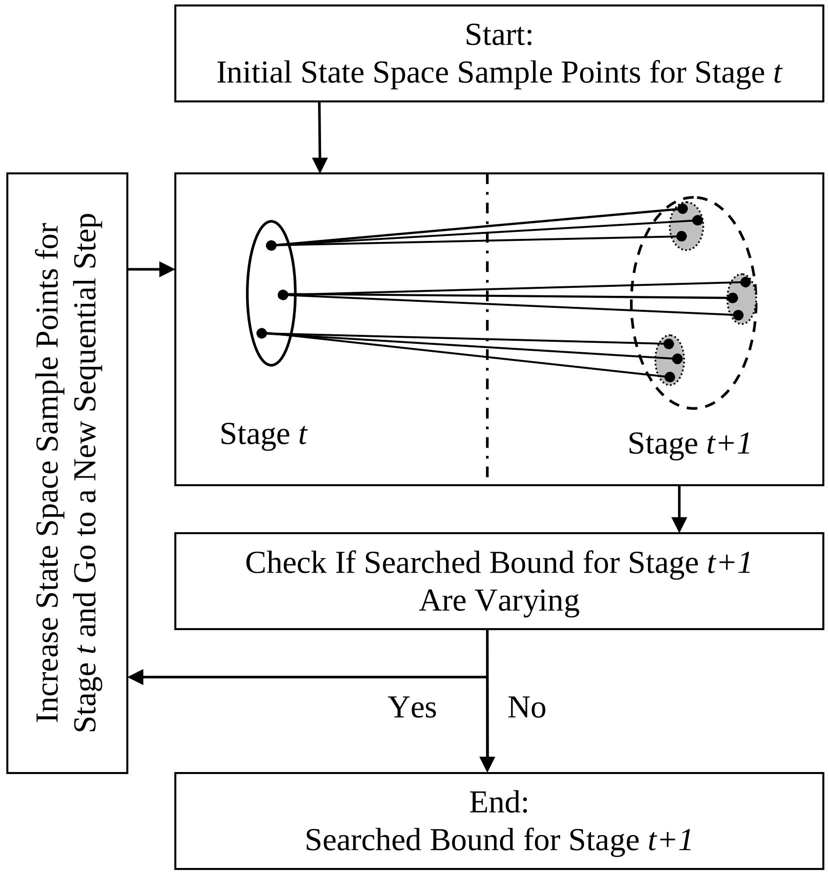


Figure 4.1. Flowchart of a forward sequential state space exploration process.

$DF_{\Delta NF,l}$ — the subset of data corresponding to ΔNF

$bu_{t+1,l}$ — the upper bound vector of state space by a sequential step for stage $t+1$

$bl_{t+1,l}$ — the lower bound vector of state space by a sequential step for stage $t+1$

$BU_{t+1,l}$ — the upper bound vector of state space updated for stage $t+1$

$BL_{t+1,l}$ — the lower bound vector of state space updated for stage $t+1$

n_d — a sample size of decision space points per a state point

n_ϵ — a sample size of ϵ points per a state-decision combination

It is assumed that the state space region for the first stage is known. Given the state space region for the current stage t, the SSSE approach propagates forward to explore the next stage $t+1$. In this SSSE framework, the state space region is defined by the upper and lower bound vectors in which the upper and lower bounds of each state variable are stored, respectively. Figure 4.1 illustrates the schematic flowchart for a forward SSSE process. In general, such a process sequentially samples the unknown state space region for the next stage based on the known state space region of the current stage. The state space sample points of the current stage t are gradually increased during the process. The transition to the next stage $t+1$ depends on the decision and a realization of the random variable ϵ. For each sampled point in the state space of the current stage t, the SSSE approach follows several trajectories induced by multiple decision-ϵ combinations. All the obtained sets of the state points for stage $t+1$ are collected and used to calculate the state space bound vectors for stage $t+1$. The sequential process continues until the bound vectors no longer change. The sequential state space sampling for the current stage employs the space-filling, low discrepancy methods (NTMs) used in Chapter 3, for the same reasons. Two basic sequential processes, denoted as "SSSE Process I" and "SSSE Process II," are considered in this dissertation, where they differ in how they obtain the decisions to propagate forward.

4.1.1 SSSE Process I

This is an SSSE process that purely samples the decision space. The process is described as follows:

- Start with a given initial sample data $DF_{t,0}$;
- For a current sequential step l with the state space sample data $DF_{t,l}$ of the size $NF_{t,l}$:
    1. For each of the $NF_{t,l}$ state space sample points in $DF_{t,l}$, sample n_d decision space points;

2. To each of the $NF_{t,l} \times n_d$ state-decision combinations, randomly assign n_ϵ ϵ points;
3. Transition each of the $NF_{t,l} \times n_d \times n_\epsilon$ state-decision-ϵ combinations to get the state space sample points for the next stage, $DF_{t+1,l}$, with the size $NF_{t+1,l} = NF_{t,l} \times n_d \times n_\epsilon$;
4. Calculate the sampled state space upper/lower bounds, $bu_{t+1,l}$ and $bl_{t+1,l}$, from the state space points, $DF_{t+1,l}$;
5. Update the sequentially identified upper/lower bounds, $BU_{t+1,l}$ and $BL_{t+1,l}$, in such a way:
   (a) If in the first sequential step, i.e. $l = 0$, then

   $BU_{t+1,l} = bu_{t+1,l}$ and

   $BL_{t+1,l} = bl_{t+1,l}$;

   (b) Otherwise:

   $BU_{t+1,l} = \max\{BU_{t+1,l-1}, bu_{t+1,l}\}$ and

   $BL_{t+1,l} = \max\{BL_{t+1,l-1}, bl_{t+1,l}\}$;
6. If $l > k_1$, where k_1 is a positive integer given for stopping the process, check whether the sequential identified upper/lower bounds are varying within certain sequential steps, i.e., whether $BU_{t+1,l} = BU_{t+1,l-k_1}$ AND $BL_{t+1,l} = BL_{t+1,l-k_1}$ is true:
   (a) If yes, then stop the process, output the identified upper/lower bounds;

   (b) Otherwise, go to a new step by updating l and $DF_{t+1,l}$ as:

   $l = l + 1$

   $DF_{t+1,l} = DF_{t+1,l-1} + DF_{\Delta NF,l-1}$

There are two ways to sample the decision space for a state space point in substep 1 of SSSE Process I. The first is with a purely random sampling strategy in which the points are obtained from a random seed, usually with a uniform distribution to provide coverage of the decision space. The second is with a design of experiments (DoE) technique, where in theory any DoE technique for state space discretization mentioned in Chapters 2 can be used. Again, NTMs are employed here.

4.1.2 SSSE Process II

Given that a future value function approximation for each stage is available, SSSE Process II can be described as follows:

- Start with a given initial sample data $DF_{t,0}$;
- For a current sequential step l with the state space sample data $DF_{t,l}$ of the size $NF_{t,l}$:
  1. For each of the $NF_{t,l}$ state space sample points in $DF_{t,l}$, use the proper value function and the available future value function approximation for stage t to get a "optimal" decision space point;
  2. To each of the $NF_{t,l}$ state-decision combinations, randomly assign n_ϵ ϵ sequence points;
  3. Transition each of the $NF_{t,l} \times n_\epsilon$ state-decision-ϵ combinations to get the state space sample points for the next stage, $DF_{t+1,l}$, with the size $NF_{t+1,l} = NF_{t,l} \times n_\epsilon$;
  4. Calculate the sampled state space upper/lower bounds, $bu_{t+1,l}$ and $bl_{t+1,l}$, from the state space points, $DF_{t+1,l}$;
  5. Update the sequentially identified upper/lower bounds, $BU_{t+1,l}$ and $BL_{t+1,l}$, in such a way:
     (a) If in the first sequential step, i.e. $l = 0$, then

     $BU_{t+1,l} = bu_{t+1,l}$ and

     $BL_{t+1,l} = bl_{t+1,l}$;

(b) Otherwise:

$BU_{t+1,l} = \max\{BU_{t+1,l-1}, bu_{t+1,l}\}$ and

$BL_{t+1,l} = \max\{BL_{t+1,l-1}, bl_{t+1,l}\}$;

6. If $l > k_2$, where k_2 is a positive integer given for stopping the process, check whether the sequential identified upper/lower bounds are varying within certain sequential steps, i.e., whether $BU_{t+1,l} = BU_{t+1,l-k_2}$ AND $BL_{t+1,l} = BL_{t+1,l-k_2}$ is true:

(a) If yes, then stop the process, output the identified upper/lower bounds;

(b) Otherwise, go to a new step by updating l and $DF_{t+1,l}$ as:

$l = l + 1$

$DF_{t+1,l} = DF_{t+1,l-1} + DF_{\Delta NF,l-1}$

Unlike SSSE Process I, where multiple decisions are sampled for each point in the state space of the current stage, SSSE Process II only propagates forward with one decision, the optimal decision. Since the ADP goal is to estimate the future value functions, the appropriate state space region should be limited to the optimal decisions that lead to the future value functions.

4.2 SSSE Framework

A forward-backward iterative SSSE framework is presented that employs both types of SSSE processes together with any backward ADP solution approach. This framework simultaneously identifies the appropriate state space regions and approximates the future value functions using the backward ADP solution approach.

The flowchart of the proposed SSSE framework was shown in Figure 4.2. It consists of two phases. The first phase has a single forward step and a single backward step. This forward step moves forward stage by stage via SSSE Process I. The purpose of this forward step is to identify an initial realistic state space region for each stage. The state space regions resulting from SSSE Process I should be larger than what is needed for approximating the future value function because the decisions are randomly sampled instead of optimized. However, a larger initial region ensures sufficient exploration of the state space. These initial regions (bounds) are used in the subsequent backward step to build the initial future value function approximations, which are then used in the second phase to enable the first iteration of SSSE Process II with optimized decisions. The second phase iterates between a forward step and a backward step. The forward step moves forward stage by stage through implementing SSSE Process II for each stage, which uses the available future value function approximations from the first phase or from the previous iteration of the second phase to optimize the decisions and determine the appropriate bounds. In each iteration, the state space bounds identified by SSSE Process II are used to update the saved overall state space bounds and the backward step builds new future value function approximations over these updated regions. The two steps repeat until the bounds stabilize.

In both phases of the SSSE framework, the backward step can employ any ADP backward solution technique, including the sequential AVFA algorithms discussed in Chapters 3.

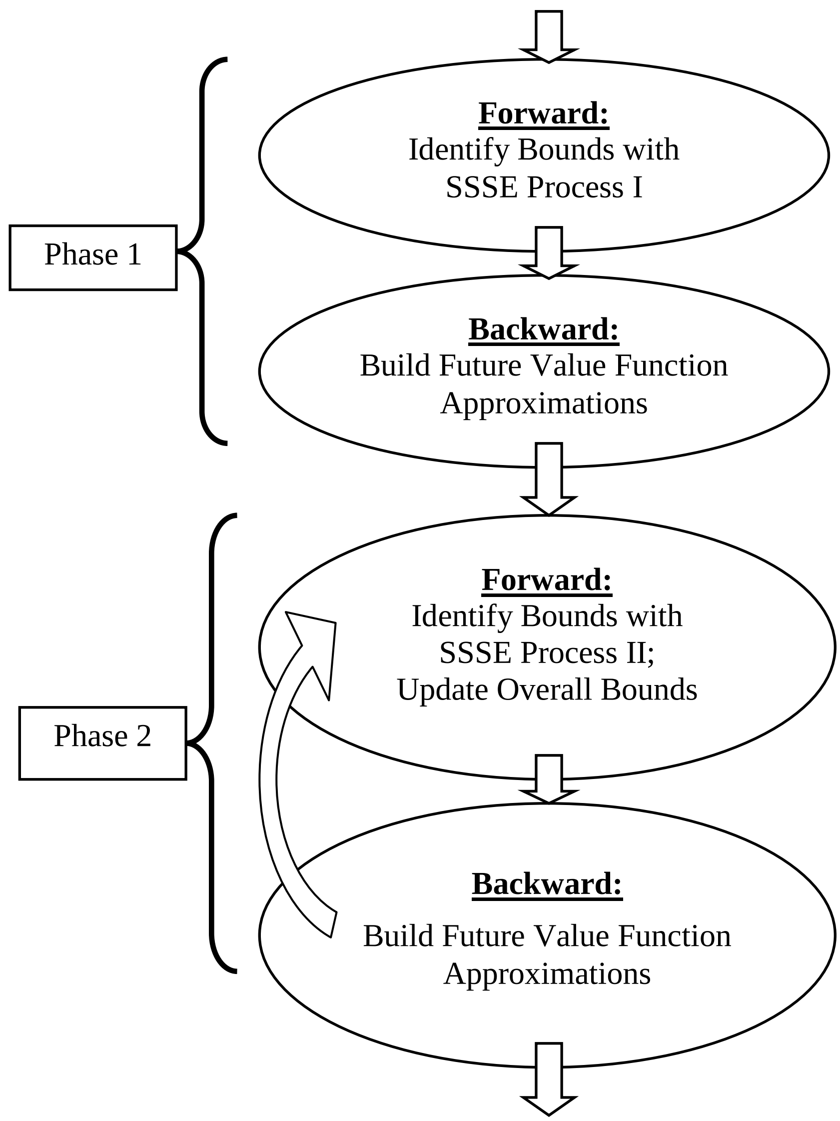


Figure 4.2. Flowchart of a forward-backward iterative SSSE framework.

Let L index the iterations in the second phase of the SSSE framework, $BUF_{t,L}$ and $BLF_{t,L}$ represent the upper and lower bound vectors of state space identified by SSSE Process II in a forward step for stages $t = 2, \ldots, T$ (as is mentioned, it is assumed that the state space region is known for the first stage), $BUI_{t,L}$ and $BLI_{t,L}$ represent the saved overall upper and lower bound vectors of the state space for stages $t = 2, \ldots, T$. The forward-backward iterative loop in the second phase is explained as follows:

- Start with a initial future value function approximations from the first phase;
- For a current loop iteration L:
  1. Conduct a forward step to get the state space bounds, $BUF_{t,L}$ and $BLF_{t,L}$, identified by SSSE Process II for stages $t = 2, \ldots, T$;
  2. Update the saved overall state space bounds $BUI_{t,L}$ and $BLI_{t,L}$ in such a way:
     (a) If in the first loop ietartion, i.e. $L = 1$, then
     $BUI_{t,L} = BUF_{t,L}$ and
     $BLI_{t,L} = BLF_{t,L}$ for stages $t = 2, \ldots, T$;
     (b) Otherwise:
     $BUI_{t,L} = \max\{BUI_{t,L-1}, BUF_{t,L}\}$ and
     $BLI_{t,L} = \max\{BLI_{t,L-1}, BLF_{t,L}\}$ for stages $t = 2, \ldots, T$;
  3. Conducted a backward step to build the future value function approximations under the saved overall bounds for stages $t = 1, \ldots, T$ with an ADP backward solution technique;
  4. If $L > k_3$, where k_3 is a positive integer given for breaking the loop, check whether the saved overall bounds are varying within certain iterations, i.e., whether $BUI_{t,L} = BUI_{t,L-k_3}$ AND $BLI_{t,L} = BLI_{t,L-k_3}$ is true, $\forall t = 2, \ldots, T$:
     (a) If yes, then break the loop, output the saved overall bounds and the associated future value function approximation models;
     (b) Otherwise, go to a new loop iteration with $L = L + 1$

This framework combines the statistical and machine learning perspectives in the following aspects:

- The pure sampling of the decision space by SSSE Process I in the forward step in the first phase is similar to the sampling of the decision space in machine learning based ADP approaches.
- The backward step in both phases conducts a statistical perspective based ADP solution.
- The second phase iterates to refine the knowledge of the unknown state and decision spaces, like a machine learning perspective, but additionally takes advantage of the optimization and modeling used by a statistical perspective in order to converge faster.

More importantly, this framework can learn a finite-horizon ADP solution more efficiently than a pure machine learning based approach and with more self-organization ability than a pure statistical perspective based approach. It should be noted here that the existing machine learning based methods are actually intended for infinite-horizon ADP and cannot easily handle a finite-horizon problem (Bart and Sutton 1998 [6]).

4.3 SSSE-AVFA Applied to an Inventory Forecasting Problem

The SSSE framework can employ the AVFA algorithms in the backward step. The combination of SSSE and AVFA framework yields a fully sequential and self-organized framework for ADP. In this section, the combined SSSE-AVFA algorithm is applied to solve the same nine-dimensional inventory forecasting problem as described in Chapter 3, where the state space region for the first stage was kept the same as in Chapter 3, while the state space regions for the second and third stages were determined by the SSSE framework.

4.3.1 Implementation Strategies

The backward steps in both phases of the SSSE framework were realized by Algorithm B with its algorithmic parameters taken the same as in experimental case "B-($\Delta H = 1$, $d = 2$)" and its initial conditions taken the same as in Chapter 3, i.e., an increment of design points of 50 (i.e., $\Delta N = 50$), 100 initial training data points ($N_0 = 100$) and 2 initial hidden nodes ($H_0 = 2$) for each NN model. The stopping criterion for Algorithm B was "R^2-stop" with the threshold c taken as 0.0002. All these settings were shown to be reasonable in Chapter 3.

For the forward steps of both phases, both SSSE Processes I and II began with 100 initial sample points ($NF_{t,0} = 100$), incremented by 50 state space sample points ($\Delta NF = 50$), and sampled 50 ϵ points for each state-decision combination ($n_\epsilon = 50$). In addition, for SSSE Process I in the forward step of the first phase, 50 decision space points were sampled for each state space point ($n_d = 50$). The above settings for SSSE Processes I or II in the forward step were taken to be complementary to the settings of the AVFA algorithm in the backward step.

The sequential processes in the forward and backward steps shared the same nine-dimensional Sobol$'$ sequence that was used in Chapter 3 for sequentially obtaining the training data from the state space. It should be noted that although the exact same experimental design sequence is used, the actual points in the state space will vary with the identified state space bounds. The test data required in the selected AVFA algorithm were taken from the 2000 Hammersley sequence points, again, as in Chapter 3, but now scaled using the bounds identified by the SSSE approach.

All the ϵ points required by both SSSE Processes I and II in the framework were sequentially sampled based on the same ϵ realizations used to simulate the ADP solution in Chapter 3.

SSSE Process I in the forward step of the first phase utilized a three-dimensional Sobol′ sequence with a large enough size to sequentially sample the decision space. As there are two constraints for the decision space for the chosen ADP problem, each sampled decision space point must be checked for feasibility under the given constraints. For the inventory forecasting problem, the two constraints have the form, $u_t^i + u_t^j \leq C_k$, where $i = 1, 2, j = 2, 3, i \neq j$ and $C_k, k = 1, 2$ are two positive constants, A "mirror" or "reflection" method was used to treat a point that violated a constraint. In addition, there are nonnegativity constraints $u_t^1 \geq 0$ and $u_t^2 \geq 0$. Let us take one constraint, $u_t^1 + u_t^2 \leq C_1$ for instance to explain this method. This constraint requires that the decision produced must fall in the triangle area formed with the two axes and the line by $u_t^1 + u_t^2 = C_1$ in the u_t^1-vs.-u_t^2 coordinate space. On the other hand, the decision points yielded from the Sobol′ sequence would spread to the entire square area formed by the two axes and the lines $u_t^1 = C_1$ and $u_t^2 = C_1$. Consequently, all the decision points on the upper right triangle of the square would violate the constraint. The "mirror" or "reflection" method maps an infeasible point into the feasible region by "reflecting" the point across the violated constraint. For this inventory forecasting problem, this maintains the "space-filling" feature of the Sobol′ sequence in the feasible region. Other options for handling infeasible designed decisions include simply ignoring them or projecting them onto the boundary of the constrained area. An example of a more complex method for handling infeasible designed decisions is given by Pilla et al. (2008)[62]. The appropriate choice will always be application-specific.

For both SSSE Processes I and II, if bounds did not change in 5 sequential steps (i.e. $k_1 = k_2 = 5$), then the process stopped. For the forward-backward loop in the second phase, if the overall bounds did not change in 3 loop iterations ($k_3 = 3$), then the loop was broken, and the entire algorithm ends.

4.3.2 Experimental Results and Discussions

The experiments were conducted with the same computational setup used in Chapter 3. The combined SSSE-AVFA algorithm successfully completed after 15 loop iterations.

4.3.2.1 Observations on Bound Search

Among the nine state variables (i.e., $I_t^{(1)}$, $I_t^{(2)}$, $I_t^{(3)}$, $D_{t,t}^{(1)}$, $D_{t,t}^{(2)}$, $D_{t,t}^{(3)}$, $D_{t,t+1}^{(1)}$, $D_{t,t+1}^{(2)}$, $D_{t,t+1}^{(3)}$), the lower bounds for the six demand forecast variables were known to have lower bounds of zero. Hence the SSSE approach was used to identify upper bounds for all nine and lower bounds for only the three inventory variables ($I_t^{(1)}$, $I_t^{(2)}$, $I_t^{(3)}$). The evolving histories of the state space bounds identified by SSSE Process I in the forward step of the first phase are shown in Figure 4.3. An example realization of SSSE Process II in the forward step of the second phase is shown in Figure 4.4. It can be seen from Figure 4.3 that SSSE Process I stabilized in 17 sequential steps for stage 2 and 18 sequential steps for stage 3. In particular, the upper bounds for the five state variables $I_t^{(1)}$, $I_t^{(3)}$, $D_{t,t+1}^{(1)}$, $D_{t,t+1}^{(2)}$, $D_{t,t+1}^{(3)}$ in both stage 2 and stage 3 stabilize in one sequential step. The example case of SSSE Process II shown in Figure 4.4 was from the 10th loop iteration, which stabilized in 27 sequential steps for both stage 2 and stage 3. The SSSE Process II bounds identified by all fifteen forward steps are shown in Figure 4.5. Some additional observations on the bounds can be made from this figure:

- For the first three state variables, $I_t^{(1)}$, $I_t^{(2)}$, $I_t^{(3)}$, all had quite unstable lower bounds for both stage 2 and stage 3, indicating more variability on the lower end of the distribution, while $I_t^{(1)}$ had little variability for stage 2 and instability for stage 3, $I_t^{(2)}$ had little variability, and $I_t^{(3)}$ had unstable upper bounds.
- The three state variables, $D_{t,t}^{(1)}$, $D_{t,t}^{(2)}$, $D_{t,t}^{(3)}$, had unstable upper bounds.

- The three state variables, $D_{t,t+1}^{(1)}$, $D_{t,t+1}^{(2)}$, and $D_{t,t+1}^{(3)}$, had little variability in their upper bounds.

Evolving histories of the saved overall state space bounds from the second phase of the SSSE framework are shown in Figure 4.6, on which the bounds identified from the first phase of the framework and the bounds used in Chapter 3 are also shown. The results demonstrated that the framework could find the unknown bounds as expected. The following can be observed from Figure 4.6:

- As expected, the bounds obtained through SSSE Process I in the forward step of the first phase in general produced larger regions than what were needed for approximating the future value functions. This was true in particular for the first three state variables $I_t^{(1)}$, $I_t^{(2)}$, $I_t^{(3)}$. (For the other six state variables the SSSE framework found the bounds that were almost the same as those from the first phase.)
- The state space regions identified by the SSSE framework turned out to be larger than those used in Chapter 3. This implies that the appropriate regions of the state space for modeling the future value functions should be larger than what was assumed before.

4.3.2.2 Discussions on NN Models

The experimental run of the SSSE framework resulted in a sequence of sixteen sets of NN models, each of which includes two models used to approximate the future value functions for stage 2 and stage 3. With different state space bounds from the different backward steps, these NN model sets obtained from the associated backward steps were also tested with different test sets, though the test sets were generated using the same set of Hammersley sequence points. It is therefore hard to compare their performance in terms of test error. However, if the test errors show an overall decrease over the course of the SSSE-AVFA algorithm, then this demonstrates the improvement in the future value approximations achieved by proper identification of the state space regions. Three typical model sets were selected for this purpose. The first one was the model set from the first phase where the bounds were very preliminary and the model set should be rough. The second was the model set from the first loop iteration of the second phase where the framework started to use optimal decisions to search the bounds, and thus the model sets should become better compared to the one from the first phase. The third was the one from the final loop iteration of the second phase, i.e.,the final output model set upon completion of the SSSE-AVFA algorithm. The three model sets are denoted as "Model-0", "Model-1" and "Model-15", respectively. The test error values attained by the three model sets over the associated test sets are presented in Figure 4.7. It can be seen in the figure that Model-1 had lower test error than Model-0, and Model-15 had lower test error than Model-1. Hence, the SSSE framework appears to be successful.

As was already mentioned, the regions of the state space used in Chapter 3 were quite different with the ones identified by the SSSE framework. Assuming that the state space regions obtained by the SSSE framework are more appropriate for the problem, an additional experiment was conducted to study how the NN models based on the state space regions from Chapter 3 would perform over the more appropriate SSSE-identified regions. The 50 NN model sets from the fixed structure experimental case in Chapter 3 were utilized to predict the test set used by the SSSE framework in building the final output model set. This added experimental case is denote as "FS-add" in correspondence with the original fixed structure case "FS" discussed in Chapter 3. The boxplot of the 50 test errors for the added experimental case were plotted in Figure 4.8 together with the boxplot of the 50 test errors for the original fixed structure case obtained in Chapter 3 and the test error values of the three selected NN model sets from Figure 4.7. Although the original fixed structure case performed well over its test set with the “older” bounds, its 50 NN model sets had extremely poor accuracy in predicting the test set with the bounds identified by the SSSE framework. This is not a surprise since the original fixed structure is only valid over the “older” bounds, and prediction at any point outside these bounds is extrapolation — a phenomenon that is avoided in statistical modeling. However, assuming that the SSSE-identified bounds are more appropriate, the evolution of the inventory forecasting system could access points anywhere within these bounds. Consequently, if a model set was built under inexact bounds, it might perform very poorly and lead to a poor ADP solution. By contrast, the three model sets identified during the execution of the SSSE-AVFA algorithm had comparable test error values being compatible to the original fixed structure case. Hence, the SSSE framework, with its adaptive and automatic identification of the state space regions of the ADP problem, effectively avoided extrapolation caused by models constructed over inexact bounds.

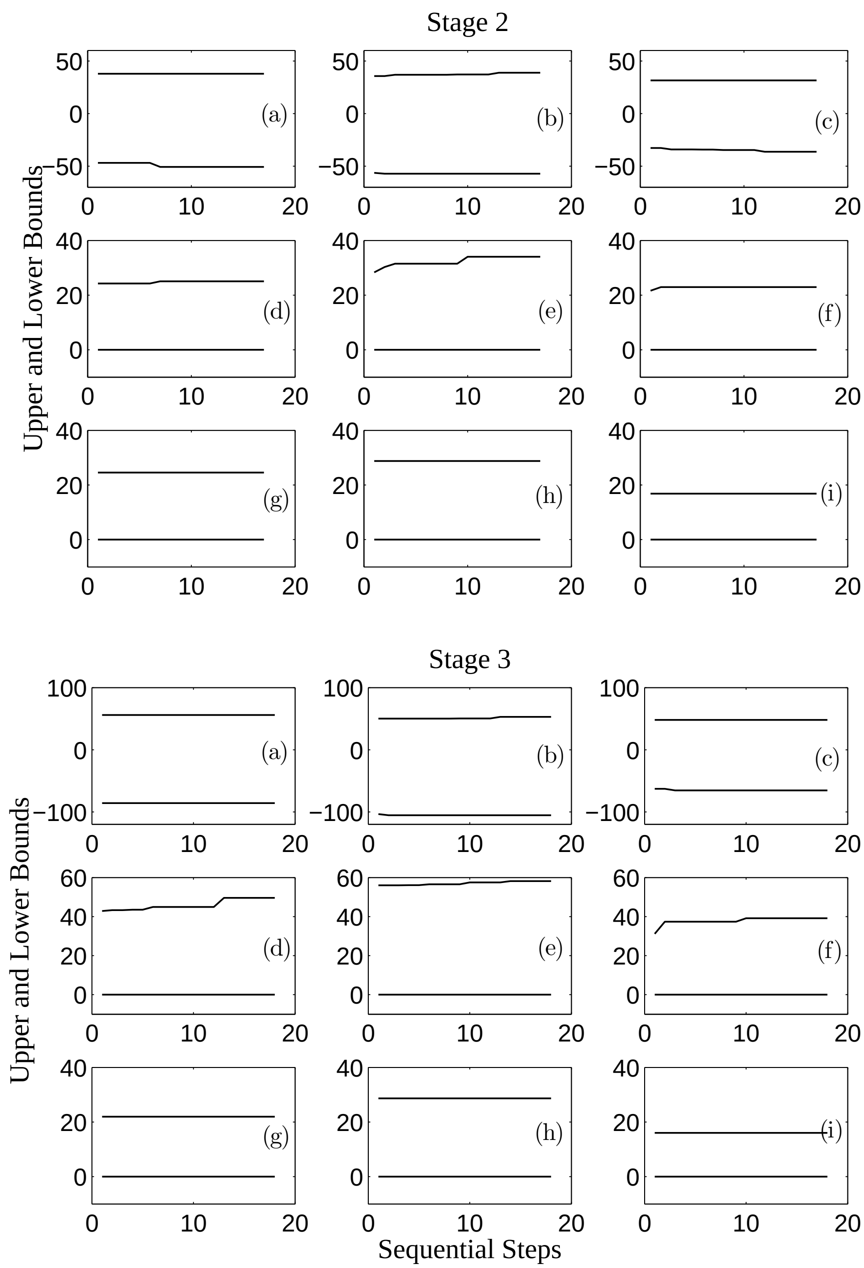


Figure 4.3. Evolving histories of the bounds of state space from SSSE Process I in the forward step of the first phase of the SSSE framework for the nine-dimensional inventory forecasting problem.

Notes: In Figures 6.2, 6.3 and 6.5, the subplots and the corresponding state variables are as follows:

(a) — $I_t^{(1)}$; (b) — $I_t^{(2)}$; (c) — $I_t^{(3)}$;

(d) — $D_{t,t}^{(1)}$; (e) — $D_{t,t}^{(2)}$; (f) — $D_{t,t}^{(3)}$;

(g) — $D_{t,t+1}^{(1)}$; (h) — $D_{t,t+1}^{(2)}$; (i) — $D_{t,t+1}^{(3)}$;

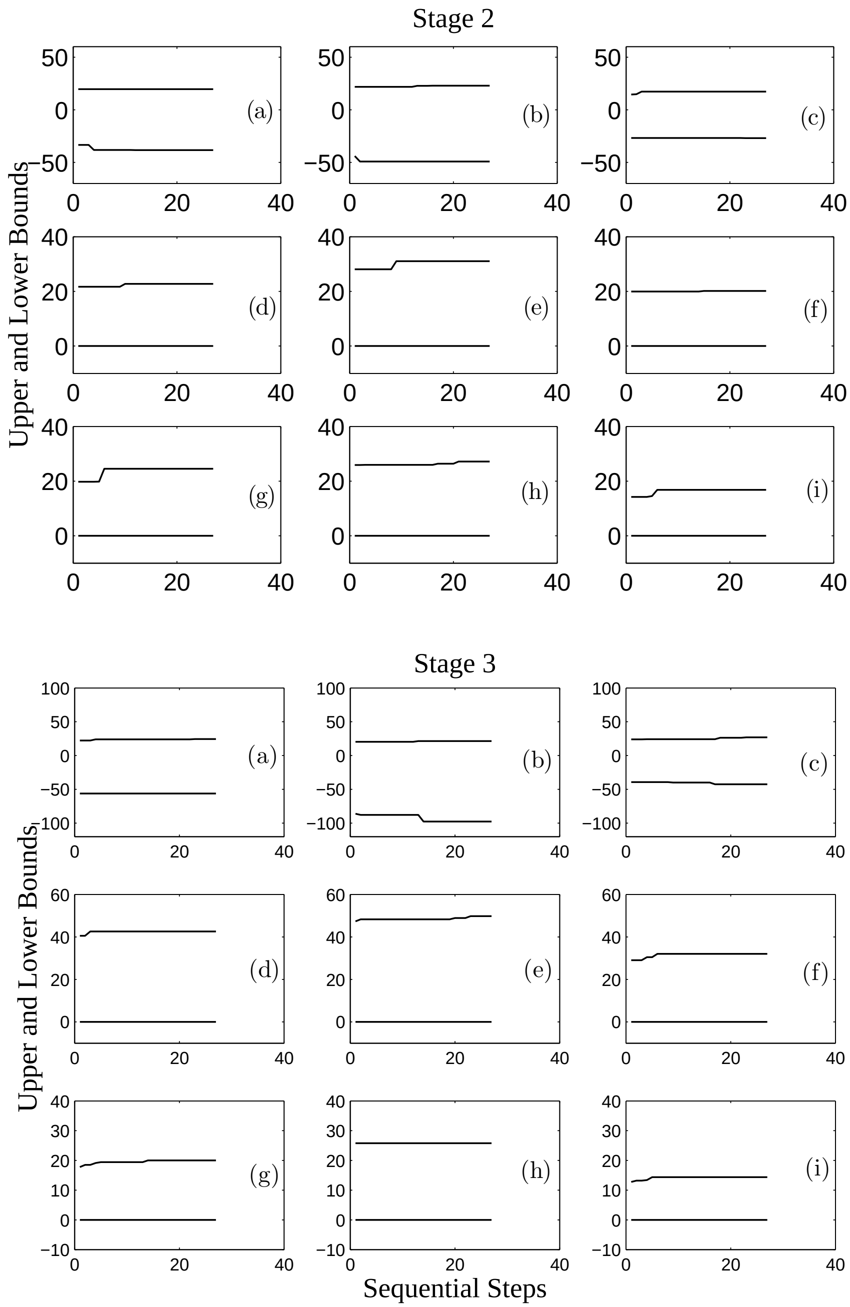


Figure 4.4. An example case for evolving histories of the bounds of state space from SSSE Process II in the forward step of the second phase of the SSSE framework for the nine-dimensional inventory forecasting problem.

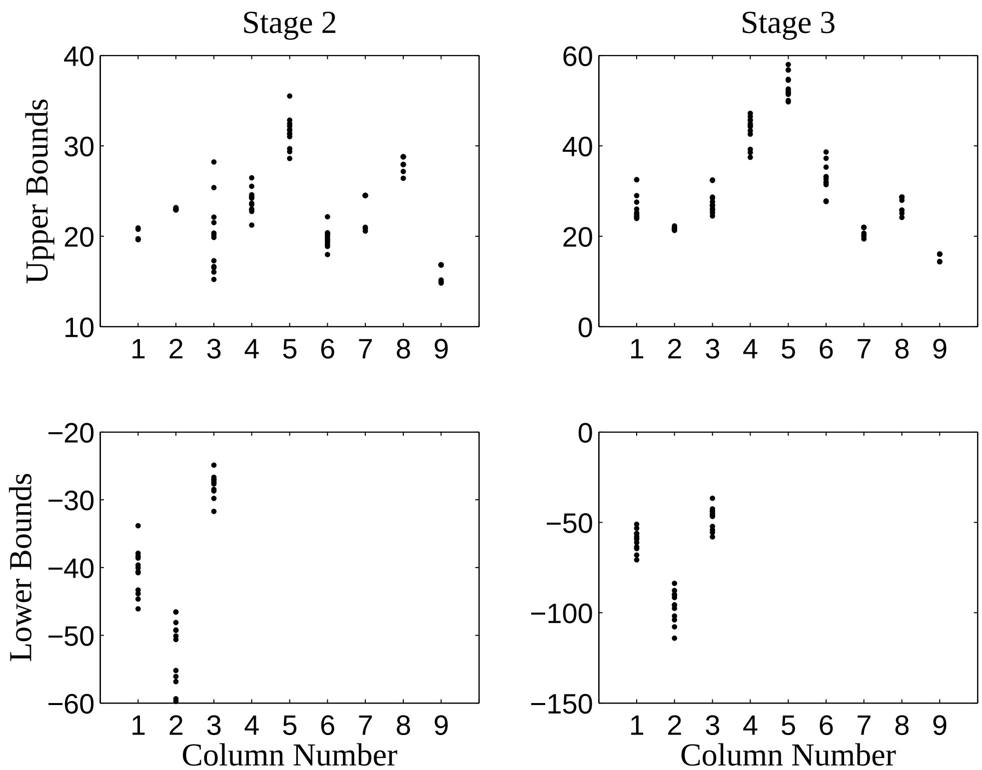
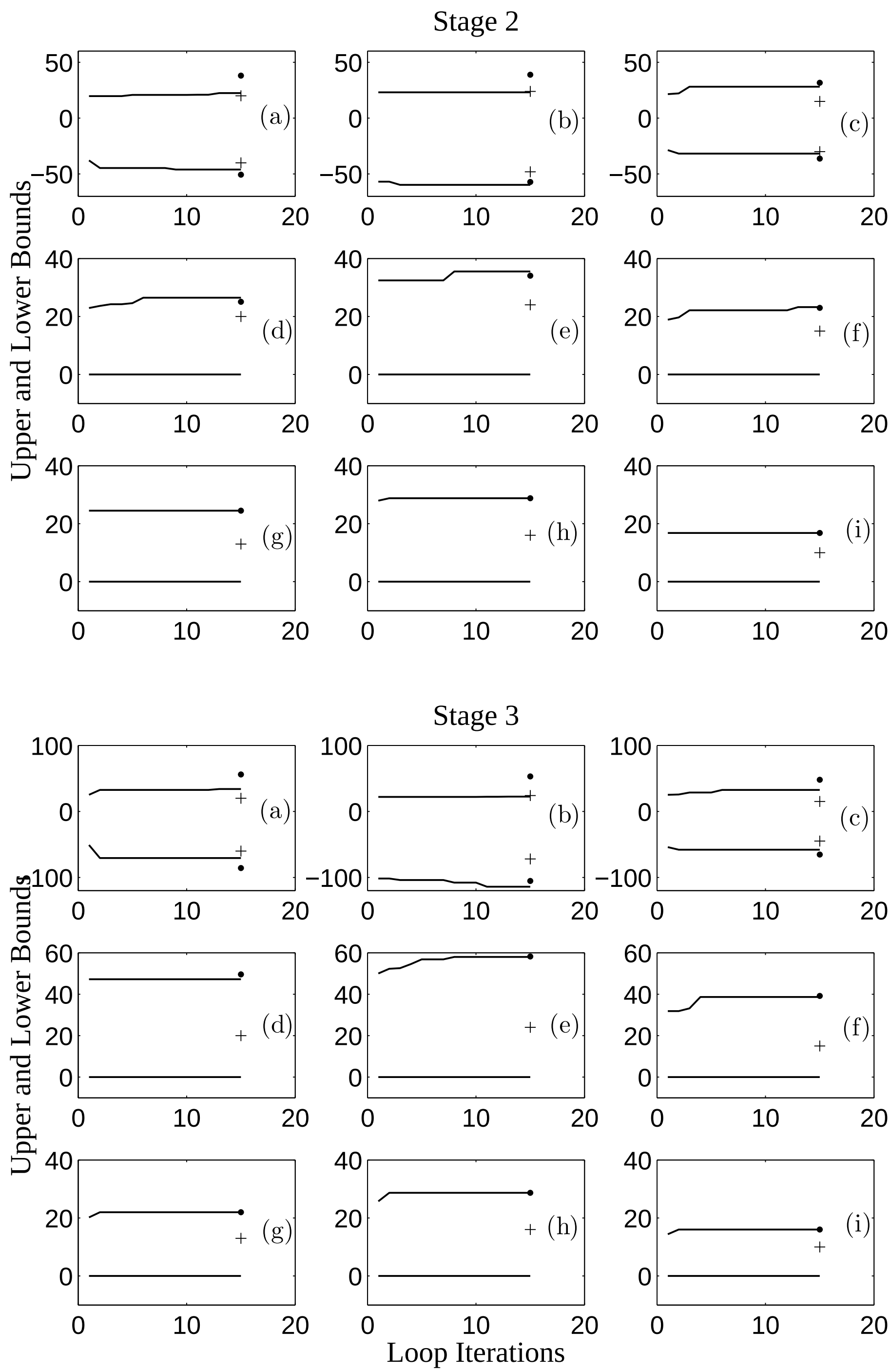


Figure 4.5. The bounds identified by all fifteen forward steps from SSSE Process II in the second phase of the SSSE framework for the nine-dimensional inventory forecasting problem.

Notes: The column numbers and the corresponding state variables are as follows:

1 — $I_t^{(1)}$	4 — $D_{t,t}^{(1)}$	7 — $D_{t,t+1}^{(1)}$
2 — $I_t^{(2)}$	5 — $D_{t,t}^{(2)}$	8 — $D_{t,t+1}^{(2)}$
3 — $I_t^{(3)}$	6 — $D_{t,t}^{(3)}$	9 — $D_{t,t+1}^{(3)}$



(● — the bounds identified by the first phase; + — the bounds used in Chapter 5)

Figure 4.6. Evolving histories of the overall state space bounds by the second phase of the SSSE framework for the nine-dimensional inventory forecasting problem.

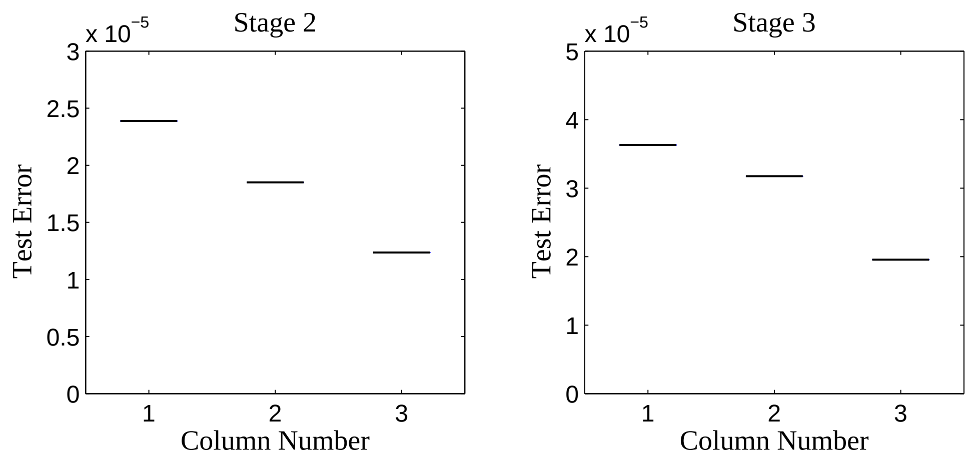


Figure 4.7. Test error values for three representative NN model sets over their associated test sets identified by the SSSE framework during its experimental run.
Note: The column numbers and the corresponding simulation cases are as follows:
1 — Model-0
2 — Model-1
3 — Model-15

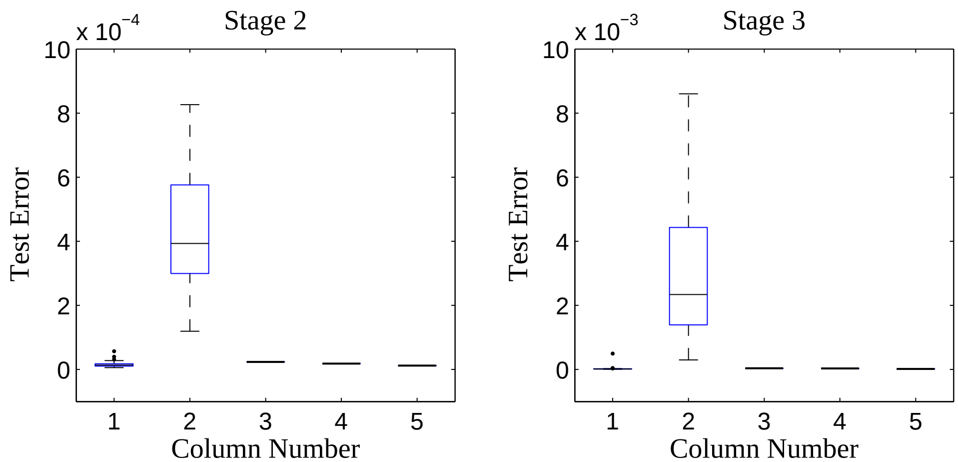


Figure 4.8. Boxplots of the test errors from the added case in which the 50 NN model sets from the fixed structure case in Chapter 3 were used to predict the test set that the SSSE framework used to build its final output model set, compared to the boxplots of the the test errors from the original fixed structure case in Chapter 3, and the test errors from the three selected model sets by the SSSE-AVFA algorithm.
Note: The column numbers and the corresponding simulation cases are as follows:
1 — FS
2 — FS-add
3 — Model-0
4 — Model-1
5 — Model-15

CHAPTER 5

SUMMARY AND FUTURE WORK

This dissertation developed novel methodology for solving approximate dynamic programming (ADP) problems based on a statistical perspective. A sequential concept was first presented based on some theoretical results from nonparametric statistical inference and consistent estimation. This concept was used to develop an adaptive value function approximation (AVFA) framework, from which three AVFA algorithms were proposed that can sequentially and adaptively approximate the future value functions of an ADP problem. A second kind of sequential concept was then presented for a sequential state space exploration (SSSE) framework that seeks to identify the appropriate state space regions over which to model the future value functions. Two basic SSSE processes were proposed that, combined with a backward ADP solution technique, produced an iterative forward-backward framework. The SSSE framework can solve a finite-horizon ADP problem with an unknown state space region, and can be fully sequential and self-organized when combined with an AVFA algorithm.

Section 1 of Chapter 3 first described the theoretical background and motivation for the new AVFA approach. Statistical consistency traces in terms of test errors were derived from the characteristics possessed by nonparametric inference and the definition of consistent estimation. Neural networks were introduced as a kind of nonparametric model, one of the two critical components needed to support the AVFA framework. The other critical support component of AVFA, is sequential experimental design techniques, for which number-theoretic methods (NTMs) were selected. The basic framework of the AVFA approach was outlined in the end of the section.

Given that one-hidden layer feed-forward neural networks were employed for non-parametric statistical modeling and that NTM low-discrepancy sequences were used to obtain sequential data from a continuous state space, Section 2 of Chapter 3 proposed three AVFA algorithms:

- **Algorithm A** was in a very intuitive way to follow a consistency trace, in which the training data were set to increase with an given increment for each sequential step, and in each step a successive model-search process was performed to find an optimal model under that given data.
- **Algorithm B** was an improvement over Algorithm A, which introduced a stopping mechanism for the successive model-search process to reduce computation.
- **Algorithm C** conducted alternate successive model-search and data-search processes to more naturally follow a consistency trace.

In Section 3 of Chapter 3, the proposed three AVFA algorithms were examined using a nine-dimensional stochastic inventory forecasting problem. Among the three new algorithms, Algorithms A and B were shown to outperform the existing fixed-structure based methods:

- Proper model structure and sample size were adaptively identified.
- The automated adaptiveness of the AVFA approach avoided unnecessary and uncertain computation usually needed by a blind trial-and-error process for determining model structure or sample size.

The performance of Algorithm C was discussed at the end of the section, where some additional experimental results indicated that the algorithm still has promise, but requires further study.

Chapter 4 first proposed two basic SSSE processes, SSSE Processes I and II, among which the former propagates the states via a sample of decisions while the latter propagates the states via optimal decisions. A SSSE framework was then proposed by combining the two basic processes with a backward solution ADP technique. The SSSE framework consisted of two phases, where the first phase is used to obtain initial state space regions and initial future value function approximations, and the second phase iteratively updates the state space regions and the associated future value function approximations. In both phases, a SSSE process was used to search the state spaces forward through the stages, and a backward ADP solution technique was used to build the future value function approximation for each stage under the latest saved state space region. Finally, the SSSE framework was combined with the AVFA algorithm to solve the nine-dimensional inventory forecasting problem.

There are several directions for further work based on the AVFA and SSSE sequential approaches:

- **To Improve the Three AVFA Algorithms**: For the three new AVFA algorithms, Algorithms A and B can be studied more to fine tune their parameters, and obviously much work must be done for Algorithm C. One initially foreseeable strategy to improve Algorithm C is to add more parameters to adaptively adjust one or both of the data-search and models-search processes.
- **To Improve the SSSE Framework**: The presented results on the SSSE framework are preliminary. Hence, the following work is planned:
  - **A comprehensive study on sequential parameters**: This study will focus on finding the optimal control parameters, in particular for the forward steps.
  - **A study on better update rules for the state variable bounds**: The current framework used the update rule that can be stated in brief as "always keep the higher/lower value for the upper/lower bound of a state variable." This rule is sensitive to outliers. To reduce this risk, a percentile concept is considered to replace the current update rule. According to this concept, a percentile value will be taken to update the bounds of a state variable.

- **To Study Other Candidates for Nonparametric Modeling**: All the modeling methods mentioned in Chapter 2, namely, multivariate adaptive regression splines (MARS), support vector machines, radial basis functions, and classification and regression trees (CART), can be properly treated as nonparametric models, and hence can in theory replace the NN models in our current AVFA algorithms. A comparison study on these techniques with the existing NN models should be conducted.
- **To Extend the New Approach to More Complicated ADP Problems**: A potential real application of the AVFA and SSSE approaches is the large-scale ADP problem from a research project (GOALI: Statistically Parsimonious Adaptive Dynamic Programming for Minimizing the Environmental Impact of Airport Deicing Activities) funded by the National Science Foundation and led by my supervising professor, Dr. Victoria Chen.

REFERENCES

[1] Anderson, C. W., M. Kretchmar, P. Young, and D. Hittle (2004). "Robust reinforcement learning using integral-quadratic." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 337–358.

[2] Anderson, C. W., D. Hittle, M. Kretchmar, and P. Young (2004). "Robust reinforcement learning for heating, ventilation, and air conditioning control of buildings." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 517–534.

[3] Andras, P. (2002). "The equivalence of support vector machine and regularization neural networks." *Neural Processing Letters*, **15(2)**, pp. 97–104.

[4] Balakrishnan, S. N., and D. Han (2004). "Adaptive critic based neural network for control-constrained agile missile." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 463–478.

[5] Barron, A. R. (1993). "Universal approximation bounds for superpositions of a sigmoidal function." *IEEE Transactions on Information Theory*, **39**, pp. 930–945.

[6] Bart G., and R. S. Sutton (1998). *Reinforcement Learning: a Introduction.* MIT Press, Cambridge, MA.

[7] Baum, E. B., and D. Hausslee (1898). "What net size gives valid generalization?." *Neural Computation*, **1(1)**, pp. 151–160.

[8] Bellman, R. E. (1957). *Dynamic Programming.* Princeton: Princeton University Press.

[9] Berry, M. J., and G. Linoff (1997). *Data Mining Techniques.* John Wiley Sons, NY.

[10] Bertsekas, D. P. (2001). *Dynamic Programming and Optimal Control.* Athena Scientific, MA.

[11] Bertsekas, D. P. (1995). "dynamic programming: an overview." In *Proceedings of Sixth International Conference on Chemical Process Control*, (J. B. Rawlings, B. A. Ogunnaike, and J. W. Eaton, eds.)

[12] Blum, A. (1992). *Neural Networks in C++.* Wiley, NY.

[13] Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone (1984). *Classification and Regression Trees*, Belmont, Wadsworth, CA.

[14] Cervellera, C., A. Wen, and V. C. P. Chen (2007). "Neural network and regression spline value function approximations for stochastic dynamic programming." *Computers and Operations Research*, **34**, pp. 70–90.

[15] Chen, V. C. P. (1999). "Application of MARS and Orthogonal Arrays to Inventory Forecasting Stochastic Dynamic Programs." *Computational Statistics and Data Analysis*, **30**, pp. 317–341.

[16] Chen, V. C. P., D. Ruppert, and C. A. Shoemaker (1999). "Applying Experimental Design and Regression Splines to High-Dimensional Continuous-State Stochastic Dynamic Programming." *Operations Research*, **47**, pp. 38–53.

[17] Chen, V. C. P., K. L. Tsui, R. R. Barton, and J. K. Allen (2003). "A review of design and modeling in computer experiments." In *Handbook of Statistics: Statistics in Industry*, (R. Khattree and C. R. Rao, eds.), **22**, Elsevier Science, Amsterdam, pp. 231–261.

[18] Chen, V. C. P., K. L. Tsui, R. R. Barton, and M. Mechesheimer (2006). "A review of design, modeling and applications of computer experiments." *IIE Transactions*, **38**, pp. 273–291.

[19] Chi, H.., and O. K. Ersoy (2002). "Determining the optimum number of hidden nodes in constructive functional-link networks using validation and VC dimension." ECE Technical Reports, TR-ECE 02-06, Purdue University (Available in: http://docs.lib.purdue.edu/ecetr/166).

[20] Cooper, L., and M. W. Cooper (1981). *Introduction to Dynamic Programming.* Pergamon Press, NY.

[21] Cristianini, N., and J. Shawe-Taylor (2000). *An Introduction to Support Vector Machines.* Cambridge University Press, UK.

[22] Currin, C., T. Mitchell, M. Morris, and D. Ylvisaker (1988). "A Bayesian approach to the design and analysis of computer experiments." Technical Report ORNL-6498, Oak Ridge National Laboratory.

[23] Currin, C., T. Mitchell, M. Morris, and D. Ylvisaker (1991). "Bayesian prediction of deterministic function, with applications to the design and analysis of computer experiments." *Journal of the American Statistical Association*, **86(416)**, pp. 409–435.

[24] Denardo, E. V. (1982). *Dynamic Programming, Models and Applications.* Dover Publications, NY.

[25] Dietterich, T., and X. Wang (2002). "Batch value function approximation of support vectors." *14th Conference of Neural Network Information Process System*, pp. 1491–1498.

[26] Enns, R., and J. Si (2004). "Direct neural dynamic programming." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on*

Computation Intelligence), (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.). Wiley-IEEE Press, NY, pp. 535–560.

[27] Fang, K. T., and Y. Wang (1994). *Numeric-Theoretic Methods in Statistica*, Chapman Hall, NY.

[28] Faure, H. (1982). "Discrepance de suites associees a un systeme de numeration (en dimensions)." *ACTA Arithmetica*, XLI, pp. 337–351.

[29] Feng, C. H. J., C. Cox, Z. G. Yu, and A. Rusiak (2006). "Selection and validation of predictive regression and neural network models based on designed experiments." *IIE Transactions*, **38**, pp. 13–23.

[30] Ferrari, S., and R. F. Stengel (2004). "Model-based adaptive critic designs." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 65–95.

[31] Fielding, A. (1977). "Binary segmentation: the automatic interaction detector and related techiques for exploring data structure." *The Analysis of Survey Data, Vol. I: Exploring Data Structure*, Wiley, NY, pp. 221–258.

[32] Foufoula-Georgiou, E. and P. K. Kitanidis (1988). "Gradient Dynamic Programming for Stochastic Optimal Control of Multidimensional Water Resources Systems." *Water Resources Research*, **24**, pp. 1345–1359.

[33] Friedman, J. H. (1991). "Multivariate adaptive regression splines (with discussion)." *Annals of Statistics*, **19**, pp. 1–141.

[34] Gao, J. B., S. R. Gunn, and C. J. Harris (2002). "A probability framework for SVM regression and error bar estimation." *Machine Learning*, **46**, pp. 71–89.

[35] Geman, S., E. Bienenstoch, and R. Doursat (1992). "Neural Networks and the Bias/Variance Dilemma." *Neural Computation*, **4**, pp. 1–58.

[36] Hammersley, J. M. (1960). "Monte Carlo methods for solving multivariable problems." *Annals of New York Academy Science*, **86**, pp. 844–874.

[37] Hastie, T., R. Tibshirani, and J. Friedman (2001). *The Element of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, NY.

[38] Haykin, S. S. (1999). *Neural Networks: A Comprehensive Foundation*, second edition. Prentice Hall, Upper Saddle River, NJ.

[39] Hua, L. K., and Y. Wang (1981). *Application of Number Theory to Numerical Analysis*. Springer-Verlag, Berlin, Germany.

[40] Jacobs, O. L. R. (1967). *An Introduction to Dynamic Programming*. Chapman and Hall, London, UK.

[41] Jin, R., W. Chen, and A. Sudjianto (2005). "An efficient algorithm for construction optimal design of computer experiments." *Journal of Statistical Planning and Inferences*, **134(1)**, pp. 268–287.

[42] Johnson, S. A., J. R. Stedinger, C. A. Shoemaker, Y. Li, and J. A. Tejada-Guibert (1993). "Numerical Solution of Continuous-State Dynamic Programs Using Linear and Spline Interpolation." *Operations Research*, **41**, pp. 484–500.

[43] Jung, T., and T. Uthmann (2004). "Experiments in value function approximation with sparse support vector regression." *Proceedings of European Conference on Machine Learning*, pp. 180–191.

[44] Kaelbling, L. P., M. L. Littman, and A. W. More (1996). "Reinforcement learning: A survey." *Journal of Artificial Intelligence Research*, **4**, pp. 237–285.

[45] Kleijnen, J. P. C., and W. C. M. van Beers (2004). "Application-driven sequential designs for simulation experiments: Kriging metamodeling." *Journal of the Operational Research Society*, **55(9)**, pp. 876–883.

[46] Kohavi, R. (1995). "A study of cross-validation and bootstrap for accuracy estimation and model selection." In *Proceedings of the Fourth International Joint Conference on Artificial Intelligence*, **2(12)**, pp. 1137–1143.

[47] Lawrence, J., and J. Fredrickson (1998). *BrainMaker Users Guide and Reference Manual*, 7th edn., California Scientific Software, Nevada City, CA.

[48] Lee, J. M., and J. H. Lee (2004). "Approximate dynamic programming strategies and their applicability for process control: A review and future direction." *International Journal of Control, Automation, and System*, **2(3)**, pp. 267–278.

[49] Lendaris, G. G., and J. C. Neidhoefer (2004). "Guidance in the Use of adaptive critics for control." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 97–124.

[50] Liang, N. Y., G. B. Huang, P. Saratchandran, and N. Sundararajan (2006). "A fast and accurate online sequential learning algorithms for feedforwad networks." *IEEE Transactions on Neural networks*, **17(6)**, pp. 1141–1423.

[51] Lin, Y. (2004). *An Efficient Robust Concept Exploration Method and Sequential Exploratory Experimental Design*, Ph.D. Dissertation, Mechanical Engineering, Georgia Institute of Technology, Atlanta.

[52] Lin, Y., F. Mistree, J. K. Allen, K. L. Tsui, and V. C. P. Chen (2004). "Sequential metamodeling in engineering design." *10th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Aug. 30-Sept. 1, 2004, Albany, NY, AIAA-Paper 2004-4303.

[53] Lin, Y., V. C. P. Chen, K. L. Tsui, F. Mistree, and J. K. Allen (2004). "A sequential exploratory experimental design method: development of appropriate empirical models in design." *ASME Design Engineering Technical Conference*, Salt Lake City, Utah, Paper No. DETC2004-57527.

[54] Lippmann, R. P. (1987). "An Introduction to Computing with Neural Nets." *IEEE ASSP Magazine*, pp. 4–22.

[55] Ma, L., and K. Khorasabi (2005). "A fast and accurate online sequential learning algorithms for feedforwad networks." *IEEE Transactions on Neural Networks*, **16(4)**, pp. 821–833.

[56] Marchandani, G., and W. Cao (1989). "On hidden nodes for neural nets." *IEEE Transactions on Circuits and Systems*, **36(5)**, pp. 661–664.

[57] Morgan, J. N., and J. A. Sonquist (1963). "Problem in the analysis of survey data, and proposal." *Journal of the American Statistical Association.*, **58**, pp. 415–434.

[58] Nemhauser, G. L. (1966). *Introduction to Dynamic Programming.* John Wiley and Sons, NY.

[59] Niederreiter, H. (1992). *Random Number Generation and Quasi-Monte Carlo Methods.* SIAM, PA.

[60] Niederreiter, H., and C. P. Xing (1995). "Low-disprepancy sequences ontained from algebric function field over finite fields." *Acta Arithmetica*, **72**, pp. 281–298.

[61] Osio, I. G., and C. H. Amon (1996). "An engineering design methodology with multistage Bayesian surrogates and optimal sampling." *Research in Engineering Design*, **8**, pp. 189–206.

[62] Pilla, V. L., J. M. Rosenberger, V. C. P. Chen, and B. C. Smith (2008). "A Statistical Computer Experiments Approach to Airline Fleet Assignment." *IIE Transactions*, **40(5)**, pp. 524–537.

[63] Powell, M. J. D. (1992). "The theory of radial basis functionapproximation in 1990." *Advances in Numerical Analysis, Volume 2: Wavelets, Subdivision Algorithms and Radial Basis Functions*, (W. A. Light, ed.), Oxford University Press, UK.

[64] Powell, W. B., and B. van Roy (2004). "Approximate dynamic programming for high-dimensional resource allocation problems." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 235–260.

[65] Rosenstein, M. T., and A. G. Barto (2004). "Supervised actor-critic reinforcement learning." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 359–380.

[66] Rumelhart, D. E., G. E. Hinton, and R. J. Williams (1986). "Learning internal representations by error propagation." In *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*, (D. E. Rumelhart and J. L. McClelland, eds.), Cambridge, MIT Press, MA, pp. 319–362.

[67] Sacks, J., W. J. Welch, T. J. Mitchell and H. P. Wynn (1989). "Design and Analysis of Computer Experiments," *Statistical Science*, **4**, pp. 409–423.

[68] Saeks, R., C. Cox, J. Neidhoefer, P. Mays, and J. Murray (2002). "Adaptive critic control of a hybrid electric vehicle." *IEEE Transaction on Intelligent Transportation Systems*, **3(4)**.

[69] Seliono, R. (2001). "Feedforward neural network construction using cross validation." *Neural Computation*, **13**, pp. 2865–2877.

[70] Si, J., D. Liu, and L. Yang (2004). "Direct neural dynamic programming." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.). Wiley-IEEE Press, NY, pp. 125–151.

[71] Si, J., A. G. Barto, W. B. Powell, and D. Wunsch (eds.) (2004). *Handbook of Learning and Approximate Dynamic Programming (IEEE Press Series on Computation Intelligence)*. Wiley-IEEE Press, NY.

[72] Sobol′, I. M. (1967). "The distribution of points in a cube and the approximate evaluation of integrals." *USSR Computational Mathematics and Mathematical Physics*, **7**, pp. 784–802.

[73] Swingler, K. (1996). *Applying Neural Networks: A Practice Guide*. Academic Press, London.

[74] Tsai, J. C. C., and V. C. P. Chen (2005). "Flexible and robust implementations of multivariate adaptive regression splines within a wastewater treatment stochastic dynamic program." *Quality and Reliability Engineering International*, **21**, pp. 689–699.

[75] Tsai, J. C. C., V. C. P. Chen, J. Chen, and M. B. Beck (2004). "Stochastic dynamic programming formulation for a wastewater treatment decision-making framework." *Annals of Operations Research, Special Issue on Applied Optimization under Uncertainty*, **132**, pp. 204–221.

[76] Tsitsiklis, J. N., and B. van Roy (2001). "Regression methods for pricing complex American-style options." *IEEE Transactions on Neural Networks*, **12(4)**, pp. 694–703.

[77] Vapnik, V. (1995). *The Natural of Statistical Learning Theory*. Springer-Verlag, NY.

[78] Vapnik, V., S. Golowich, and A. Smola (1997). "Support method for prediction approximation regression estimation, and signal processing." *Advances in Neural Information Processing System*, (M. Mozer, M. Jordan, and T. Petsche, eds.), **9**, MIT Press, MA.

[79] Wahba, G. (1990). *Spline Models for Observational Data*. CBMS-NSF Reginal Conference Series in Applied Mathematics, **59**, SIAM, PA.

[80] Wang, Y., and K. T. Fang (1994). "Number theoretic methods in applied statistics." *Chinese Annals of Mathematics Series*, **B 11**, pp. 41–55.

[81] Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. Ph.D. Dissertation, Combridge University, Cambridge, UK.

[82] Wen, A. (2005). *Statistics-Based Approach to Stochastic Optimal Control Problems*, Ph.D. Dissertation, University of Texas at Arlington.

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

[84] Werbos, P. J. (1989). "Neural networks for control and system identification." In *IEEE Proceedings of CDC89*, IEEE.

[85] Werbos, P. J. (1992). "Approximate dynamic programming for real-time control and neural modeling." *Handbook of Intelligent Control*, (D. A. White and D. A. Sofge, eds.), Van Nostrand Reinhold, NY, pp. 493–525.

[86] Werbos, P. J. (1998). "Stable adaptive control using new critic designs." Technical Report: arXiv: adap-org/9810001 v1 25 Sep.

[87] Werbos, P. J. (2004). "ADP: Goals, opportunities and principles." *Handbook of Learning and Approximate Dynamic Programming (IEEE Press series on Computation Intelligence)*, (J. Si, A. G. Barto, W. B. Powell and D. Wunsch, eds.), Wiley-IEEE Press, NY, pp. 3–44.

[88] Werbos, P. J. (2005). "Backwards differentiation in AD and neural Net: Past link and new opportunities." *Automatic Differentiation: Application, Theory, and Tools*, (H. M. Bucker, G. Corliss, P. Hovland, U. Naumann, and B. Norris, eds.), Springer (LNCS), NY.

[89] Werbos, P. J. (2007). "Using ADP to understand and replicate brain intelligence: the next level design." In *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL 2007)*, pp. 209–216.

[90] White, H. (1990). "Connectionist nonparametric regression: multilayer feedforward networks can learn arbitrary mappings." *Neural Networks*, **3**, pp. 535–549.

[91] Yao, X. (1999). "Evolving artificial neural networks." *Proceedings of the IEEE*, **9(87)**, pp. 1423–1447.

[92] Zhang, B. T. (1994). "An incremental learning algorithm that optimizes network size and sample size in one trial." *Procedings of Conference on Neural Networks (ICNN-94)*, pp. 215–220.

[93] Zhang, J., and A. J. Morris (1998). "A sequential learning approach for single hidden layer neural networks." *Neural Networks*, **11(1)**, pp. 65–80.

BIOGRAPHICAL STATEMENT

Huiyuan Fan was born in Shaanxi, P. R. China, in 1961. He received his B.S. degree in Power Engineering and his first Ph.D. degree in Power Engineering and Engineering Thermal Physics, both from Xi'an Jiaotong University, Xi'an, P. R. China, in 1983 and 2000, respectively. He began his studies for his second Ph.D. degree in Industrial Engineering at the University of Texas at Arlington in 2005. After he completed his undergraduate study, he worked in a petrochemical corporation in P. R. China as a mechanical engineer for about ten years. Then he moved back to Xi'an Jiaotong University and worked there for seven years. In 2000 and 2001 he was a visiting scholar at universities in Hong Kong and Finland. He joined Technion-Israel Institute of Technology from 2001 to 2003, where he worked as a Lady Davis Postdoctoral Research Fellow. During 2003 to 2005, he was with the Department of Mechanical and Aerospace Engineering at the University of Texas at Arlington as a senior research scientist. His current research interests include: data mining and statistical analysis, operations research, design and analysis of computer experiments; neural computation, evolutionary optimization; computational fluid dynamics, detonation engine simulation, turbo and jet engine simulation and design, turbomachinery; air quality prediction. He is an author or co-author of over 20 international journal papers, an awardee of the 2003 Excellent Doctoral Dissertation of China, and a member of AIAA, ASME and IIE. He is married to Ping Wan and has one son, Kaili.