INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
Neural networks for system modeling and control: A user’s perspective

Donne, Jeffrey David, Ph.D.

The Ohio State University, 1991
Neural Networks for System Modeling and Control: A User’s Perspective

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

by

Jeffrey David Donne, B.S., M.S.

* * * * *

The Ohio State University

1991

Dissertation Committee:
Professor Ümit Özgüner
Professor Kevin M. Passino
Professor Ashok Krishnamurthy

Approved by

Adviser
Department of Electrical Engineering
To Mom, who taught me perseverance;

To Dad, who teaches me integrity and commitment;

and most importantly,

To Vicki, Jamie, and Samantha,

who teach me laughter, loving, and living.
ACKNOWLEDGEMENTS

If it is indeed possible, I would like to express my gratitude to those people who have had a most profound effect in helping me to attain my goals.

I would like to thank my adviser, Dr. Ümit Özgüner for his support throughout this work, as well as for his key involvement in helping to make Ohio State a leader in control education. He and the other professors involved in the Control Research Laboratory have provided us with an environment that invites collaboration, and with the facilities to do creative work. In addition, I would like to thank the members of my committee, Dr. Kevin Passino and Dr. Ashok Krishnamurthy for their critical review, and comments and suggestions. Support from Dr. Raymond Montgomery at NASA, Langley Research Center, and for his interest in this research is also gratefully acknowledged.

I have been fortunate to work with many, many extremely talented researchers, both fellow grad students and visiting scholars, with whom I have shared the “joy and pain” of graduate school. Special thanks goes to Dr. Jian-Xin Xu for collaborating on much of the content of Chapters IV and V. Also, thanks to David Schoenwald for sharing his extensive knowledge of flexible robots and to Ken Cheung for the many discussions related to system identification. I have found many new friends along the way, too numerous to name, and I thank all of you.
However, the most important contributions to my well-being have come on the non-academic side, from both family and friends. My parents, although with no formal education beyond the high school level, knew well the value of education, and strived to give their children chances that they never had. I gladly share this accomplishment with you! In addition, my wife's family has been particularly helpful and understanding in this rather difficult time for us as a family. They have helped to teach my children the true value of the extended family. I would also like to thank my many friends, especially Michael and B.J., Pat and Bruce, and David in Albuquerque for their words of encouragement, for continuing to include me in the fun, and for giving me a shoulder to lean on.

Finally, and most importantly, I reserve my most heartfelt gratitude for my wife Vicki and wonderful children, Jamie and Samantha. My children have given me joy in so many ways, and have never failed to make me smile, even on the darkest of days. What a dull existence this would be without you! However, words fail when I express my gratitude to Vicki for her support, love, understanding and perseverance. Very few people could have made the many sacrifices for the benefit of everyone but herself, and assumed the many roles that she has with so much dignity. She has excelled at every job. May this work allow you to enjoy the fruits of your labor.

The research contained herein was supported in part by NASA, Langley Research Center grant NGT-50409, The Jet Propulsion Lab, The Ohio Supercomputer Center and by the Electrical Engineering Department of the Ohio State University.
VITA

May 25, 1960 .................................................. Born—Pittsburgh, Pennsylvania, U.S.A.

1980 - 1982 .................................................. Undergraduate Teaching Assistant,
             Dept. of Mathematics,
             The Ohio State University,
             Columbus, Ohio

June 1982 .................................................. B.S.E.E., The Ohio State University,
             Columbus, Ohio

1982 - 1983 .................................................. Graduate Teaching Assistant,
             Dept. of Mathematics,
             The Ohio State University,
             Columbus, Ohio

August 1983 .................................................. M.S.E.E., The Ohio State University,
             Columbus, Ohio

1983 - 1987 .................................................. Member of Technical Staff (MTS),
             Sandia National Laboratories,
             Albuquerque, NM

1985 - 1987 .................................................. Adjunct Assistant Professor,
             University of New Mexico,
             Albuquerque, NM

1987 - 1989 .................................................. Graduate Teaching Associate,
             Dept. of Electrical Engineering,
             The Ohio State University,
             Columbus, Ohio

1989 - Present .............................................. NASA Graduate Research Assistant,
             Dept. of Electrical Engineering,
             The Ohio State University,
             Columbus, Ohio
PUBLICATIONS


FIELDS OF STUDY

Major Field: Electrical Engineering

Major Area of Specialization: ..........Control and Systems Theory
Minor Areas of Specialization: ..........Communication Theory

Mathematics
# Table of Contents

**DEDICATION** ................................................................. ii

**ACKNOWLEDGEMENTS** ................................................. iii

**VITA** .................................................................................. v

**LIST OF FIGURES** .......................................................... x

**LIST OF TABLES** ............................................................. xiii

## CHAPTER

<table>
<thead>
<tr>
<th>I</th>
<th>Problem Overview</th>
<th>......................................................... 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Introduction</td>
<td>.................................................. 1</td>
</tr>
<tr>
<td>1.2</td>
<td>Contributions</td>
<td>.............................................. 5</td>
</tr>
<tr>
<td>1.3</td>
<td>Organization of Dissertation</td>
<td>.................................. 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>II</th>
<th>Neural Networks for Control Systems</th>
<th>........................................... 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Neural Net Overview</td>
<td>............................................. 11</td>
</tr>
<tr>
<td>2.1.1</td>
<td>Computational Aspects of Neural Nets</td>
<td>................................ 13</td>
</tr>
<tr>
<td>2.1.2</td>
<td>Training Algorithms for Neural Nets</td>
<td>...................................... 19</td>
</tr>
<tr>
<td>2.2</td>
<td>Designing Neural Net Experiments</td>
<td>............................................. 26</td>
</tr>
<tr>
<td>2.2.1</td>
<td>Input Data Selection for Accurate Functional Mappings</td>
<td>................................ 28</td>
</tr>
<tr>
<td>2.2.2</td>
<td>The Neural Net Convergence Problem</td>
<td>........................................ 31</td>
</tr>
<tr>
<td>2.2.3</td>
<td>Validation of the Neural Mapping</td>
<td>........................................... 33</td>
</tr>
<tr>
<td>2.3</td>
<td>Common Structures for Neural Net Control</td>
<td>........................................ 36</td>
</tr>
<tr>
<td>2.3.1</td>
<td>Replacing Existing Controllers with a Neural Net</td>
<td>................................ 37</td>
</tr>
<tr>
<td>2.3.2</td>
<td>Exact Modeling of Nonlinearities</td>
<td>........................................... 38</td>
</tr>
<tr>
<td>2.3.3</td>
<td>Inverse Dynamics</td>
<td>............................................. 39</td>
</tr>
<tr>
<td>2.3.4</td>
<td>Indirect Adaptive Control</td>
<td>............................................. 41</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

FIGURE

1  A Typical Neuron with Sigmoid Nonlinearity ................................. 14
2  The Multilayer Network ................................................................. 17
3  Neural Net Supervised Learning ....................................................... 19
4  Existing Controller Replaced by a Neural Net Controller ................. 38
5  Implementation of the Inverse Plant Using Neural Nets ..................... 40
6  Indirect Adaptive Control of Nonlinear Plants using Neural Nets ...... 41
7  Block Diagram of the Basic Input/Output Configuration ..................... 48
8  ARMAX Model Parametrization ....................................................... 50
9  Block Diagram of the Prediction Error Method .................................. 53
10 A Conventional FIR Representation ............................................... 60
11 A Neural FIR Filter ........................................................................... 60
12 Frequency Response of Three Filter Types ....................................... 67
13 Second Order System Block Diagram .............................................. 68
14 Plots of ARMAX and Neural Net Models (top) and Error Plots of both Models (bottom) ................................................................. 69
15 The Random Walk Process ............................................................... 70
16 Simulation of the Random Walk Process ......................................... 72
17 Error Plots for Several Model Structures of the Random Walk Process 75
<table>
<thead>
<tr>
<th>Page</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>Tracking a Free-Falling Body</td>
<td>82</td>
</tr>
<tr>
<td>19</td>
<td>Error Plots from Monte Carlo Simulations</td>
<td>84</td>
</tr>
<tr>
<td>20</td>
<td>Plots of ARMAX and Neural Net Models for JPL Structure Example (top) and Error Plots (bottom)</td>
<td>86</td>
</tr>
<tr>
<td>21</td>
<td>Underparametrized ARMAX and Neural Net Model Outputs (top) and Error Plots (bottom)</td>
<td>87</td>
</tr>
<tr>
<td>22</td>
<td>Plots of the Cross-Validation Process</td>
<td>88</td>
</tr>
<tr>
<td>23</td>
<td>The Flexible One-Link Manipulator</td>
<td>98</td>
</tr>
<tr>
<td>24</td>
<td>Identification Stage Using Neural Nets</td>
<td>102</td>
</tr>
<tr>
<td>25</td>
<td>Identification and Control Using Neural Nets</td>
<td>104</td>
</tr>
<tr>
<td>26</td>
<td>Identification Results for System with Nominal Payload</td>
<td>110</td>
</tr>
<tr>
<td>27</td>
<td>Flexible One-Link Manipulator: Position with LQR Control (top), Position with LQR and Neural Net Control (middle), and Control Signal (bottom)</td>
<td>111</td>
</tr>
<tr>
<td>28</td>
<td>Identification Block with Accelerometer Input</td>
<td>114</td>
</tr>
<tr>
<td>29</td>
<td>Slewing Maneuvers with Variable Payload</td>
<td>117</td>
</tr>
<tr>
<td>30</td>
<td>Step response of system with parameter uncertainty (dashed line) compared to nominal system response (solid line)</td>
<td>134</td>
</tr>
<tr>
<td>31</td>
<td>Step response of system with VSC</td>
<td>137</td>
</tr>
<tr>
<td>32</td>
<td>Chattering control signal (solid line) and the corresponding average value (dashed line), r=1.0</td>
<td>138</td>
</tr>
<tr>
<td>33</td>
<td>Block diagram of Feedback linearized system with VSC and Neural Net compensation</td>
<td>141</td>
</tr>
<tr>
<td>34</td>
<td>Step response of system with VSC and Neural Net Compensation</td>
<td>142</td>
</tr>
<tr>
<td>35</td>
<td>Output of Neural Net controller with step reference input</td>
<td>143</td>
</tr>
</tbody>
</table>
36 System input with VSC only (dashed line) and with both VSC with ANN compensation (solid line); \( r = 1.0 \) ................................................ 143

37 JPL/AFAL Flexible Structure Testbed ........................................................ 150

38 Rib Numbering Convention and Location of Sensors and Actuators . 150
## List of Tables

<table>
<thead>
<tr>
<th>TABLE</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Training Results for the Random Walk Neural Net</td>
</tr>
<tr>
<td>2</td>
<td>Cross-Validation Results for the Random Walk</td>
</tr>
<tr>
<td>3</td>
<td>Parameters Used for Tracking Example</td>
</tr>
<tr>
<td>4</td>
<td>Parameters of the One-Link Manipulator</td>
</tr>
<tr>
<td>5</td>
<td>Physical Parameters of the OSU One-Link Manipulator</td>
</tr>
<tr>
<td>6</td>
<td>Modal Frequency as a Function of Payload</td>
</tr>
<tr>
<td>7</td>
<td>Effect of Payload Variation on Training Time</td>
</tr>
<tr>
<td>8</td>
<td>Modes included in the system model</td>
</tr>
</tbody>
</table>
CHAPTER I

Problem Overview

1.1 Introduction

To meet the challenges of increasing demands on high performance and sophistication of control systems, the designers of control systems must introduce new and innovative schemes for solving such problems. The notion of a single central computing facility for all control objectives becomes unrealistic as we increase the requirements of performance, accuracy, and reliability. Each of these areas presents a formidable challenge in itself, but combinations of these ideas offer immense difficulties both computationally and algorithmically. The field of neural networks is an emerging technology that can be used to relieve the burden of the central processor while also providing additional capabilities not available using conventional techniques for control. Neural nets can be used in concert with or in lieu of conventional control systems. This dissertation addresses the use of neural nets in control applications, focusing on both their benefits and limitations in a variety of paradigms for both system modeling and control.

From the wide range and complexity of problems being considered in today's technology, it is clear that mankind is at the threshold of a new age. For example, as we
approach the year 2000, it is possible that man’s permanent presence in space will have become a reality. Systems such as the NASA Space Station, as well as applications of coordinated robots and lightweight robots in space push our knowledge of nonlinear and large scale systems to the limit. Moreover, on earth, today’s automobiles are being equipped with more sophisticated subsystems such as anti-lock brakes, adaptively controlled automatic transmissions, etc. These problems force engineers to address several emerging and inter-related issues: control of nonlinear systems, control of large scale systems, with the sometimes conflicting goal of maintaining a high performance sophisticated design. Neural nets are the focus of this work which addresses some of the issues and concerns raised above, offering solutions that can either replace the conventional methods or work together with these systems for computational efficiency while maintaining a high degree of sophistication.

One of the more difficult issues to address is that of nonlinear system modeling and control. While there is a large body of work concerning linear system theory, there is a relatively small amount of literature for general approaches to model realization and controller design for nonlinear systems. This can be attributed to the difficulty of deriving identification algorithms that are applicable to a reasonably large class of nonlinear systems. Additionally, the methods that are used often have modeling errors inherent in the process due to the difficulty in quantifying sources of nonlinearities a priori. However, neural nets used as functional mappings from input space to output space can be implemented as easily for linear mappings as for nonlinear mappings; thus using the input/output data from a plant, one can use a neural net to accomplish
this mapping which results in a nonlinear dynamic model that can then be used for simulation or nonlinear controller design. This notion can be extended to filtering theory, where neural nets are used to realize the output estimate of a system for both linear and nonlinear systems. Hence, neural nets offer the control system designer a new tool for system analysis and design, where the techniques are similar for both linear and nonlinear systems.

Artificial neural networks are comprised of dense interconnections of simple processing elements called neurons. They are biologically inspired with the hope of achieving human-like performance for computing. The appeal of neural nets for control applications is in their ability to "learn" a system's behavior based on input/output data. The learning process involves the use of a training algorithm which is an information processing function to enable the neural net to perform an accurate mapping from an input space \( \{ x_i | x_i \in X \} \) to an output space \( \{ y_i | y_i \in Y \} \) based on training examples \( (x_1, y_1), (x_2, y_2), \ldots, (x_j, y_j) \) from the mapping where \( y_j = f(x_j) \) and \( f(\cdot) \) is the functional mapping. This type of training is referred to as supervised training since the network output is compared to a desired value and iterated until the error is small between the net output and the desired value. Unsupervised training implements a mapping from an input space to an output space, but without the use of the training examples. Instead, the network learns the correct mapping based on the minimization of a cost function.

Conventional analytic system models require precise mathematical expressions to describe system behavior. Conversely, neural nets require no knowledge of the math-
mathematical basis of the system and therefore can replace conventional models in a wide variety of control tasks where the system is not well known in precise mathematical terms. Moreover, since neural nets can learn arbitrary mappings with any desired degree of accuracy [22], both systems with unknown dynamics and high dimensional systems which must be truncated in simulation can be more accurately realized.

Neural network research has its origin in the 1940's, but recently new training algorithms such as the backpropagation algorithm have been the driving force behind the resurgence in the research of neural nets. The popularity of backpropagation stems from its ability to train networks to learn multidimensional mappings. Backpropagation was originally introduced in 1969 by Bryson and Ho [10], and independently rediscovered by Werbos in 1974 [70] and Parker in the mid 1980's [51]. While the algorithm was known as early as 1969, it was not applied to neural networks until it was again rediscovered by Rumelhart and the PDP Research Group in 1985 [56, 57]; hence, much of the credit for developing the algorithm into the technique that is used today is given to the PDP Group. Their work is described in detail in [56], most likely one of the most widely referenced works in neural nets to date.

Neural nets used in control systems applications are not without limitations. In closed-loop control, one must be concerned with providing the neural net with inputs from the data set from which it was trained. Often times the neural net is trained in an open loop setting, then implemented in the closed-loop setting, making its performance difficult to anticipate. Thus, neural nets in control should not be thought of as a panacea; there are many tasks for which they are not appropriate. How-
ever, in certain paradigms they offer advantages in ease of implementation, accuracy, and modularity, as well as computational speed advantages through parallel implementations over conventional techniques. The control system engineer is ultimately responsible for choosing the most appropriate paradigms in which one can exploit the many benefits of neural networks. It is with this thought that we proceed.

1.2 Contributions

This dissertation is written with the user in mind, such that it concentrates on developing theory and applications for successful implementation of neural networks for system identification and control. In [39], Lippmann mentions that due to the theorem of Kolmogorov, it was proven that any continuous function of \( N \) variables can be computed using only linear summations and nonlinear but continuously increasing functions of only one variable. However, the method to be used for implementation of such functions is not described. In [14], the Stone-Weierstrass Theorem, from classical real analysis, is used to show that certain neural network architectures are capable of approximating arbitrary functions. Furthermore, in [22], there is a proof that any function can be realized with any desired degree of accuracy with a neural net. Armed with the above information, in Chapter II we address the methods used to realize these functional mappings using multilayer perceptrons and the backpropagation algorithm as the training algorithm. A definition is given for training to within bound \( e \) (\( T.T.W.B.e \)), which is an empirical means of determining when training should be stopped. Then, conditions are imposed on selection of the input data set such that the neural net interpolates well for data outside the training data
Further contributions can be found in Chapter III where ARMA-type models are implemented using neural networks for system identification and prediction. This chapter gives comprehensive comparisons of conventional techniques with the neural net methods and shows that in nearly all cases, the neural nets perform at least as well for linear systems, and for nonlinear systems, neural nets are clearly advantageous over the conventional realizations with respect to output error. This has implications in nonlinear filter theory in which often times nonlinear filters such as the extended Kalman filter (EKF) require extensive calculations when compared to the neural net methods.

While Chapter III uses neural nets in open-loop identification problems, Chapters IV and V approach control problems where neural nets are used in closed-loop control settings. The contributions from Chapter IV can be found in the area control for nonlinear systems with partially known dynamics. Simulations on a one link flexible robot arm demonstrate that accurate tracking and vibration damping can be achieved even when varying the manipulator payload. The methods used are composite control and identification schemes, using both conventional and neural net methods. This work is among the first to address neural net control of flexible manipulators and offers insight into the possible advantages and implementation considerations for the neural-based strategy.

In Chapter V, neural nets are used with conventional robust control techniques for control of nonlinear systems with uncertain parameters. Neural nets are used
in this configuration for synthesis of an equivalent control signal from the variable structure controller (VSC) which has the effect of significantly reducing chattering of the control signal, resulting in more accurate control.

1.3 Organization of Dissertation

The dissertation is organized in the following way. Chapter II begins with a section that gives a brief overview of neural nets, focusing on both computational aspects and training algorithms. The next section addresses the neural net experiment design process, addressing issues such as input data selection, neural net topology, and the often debated topic of neural net convergence. The next section focuses on neural nets in control applications, discussing several of the more common paradigms for neural nets in control using a general block diagram approach.

Chapter III addresses the topic of dynamic system modeling of linear and nonlinear systems using neural nets. Dynamical systems can be realized in predictor form by neural nets using a regression of prior inputs, outputs and errors as neural net inputs, allowing the network to learn an internal representation in weight space of the dynamics of the process under study. The neural net output represents a one-step-ahead predictor. The chapter begins with a section giving a brief overview of system identification. Section 3.2 follows with a discussion of some common model structures for linear systems, along with their neural net representations. The next section gives a performance analysis of the linear structures using several examples where the conventional methods are compared with neural methods, with error variance as the figure of merit. Then, Section 3.4 addresses system modeling and identification for
nonlinear systems. It is in this section that the benefits of a neural net approach becomes apparent. There are few identification methods for nonlinear systems that are applicable to a wide range of problems, so the neural net methods offer the control designer a new means for modeling nonlinear systems. This chapter ends with a section containing examples in which the neural nets are compared with conventional methods for system identification and output estimation using a large space structure example. The issue of underparametrization is investigated via example. In every case, the neural nets trained using the methods outlined in Chapter II are shown to have superior performance over the conventional methods with respect to output error.

Chapter IV addresses control of flexible systems with partially known dynamics. Control and identification is discussed as a two stage process in which the system is first identified using a neural net in a supervised training mode. Control then is accomplished using a model-based control scheme for the known dynamics and a neural net-based scheme to provide "corrective" control of the unknown system dynamics. Simulation on a flexible manipulator are performed first for a constant payload, then for a variable payload, showing the ability of the neural net controller and identifier to adapt to a changing environment.

The next chapter, Chapter V addresses systems where neural nets are used together with conventional closed-loop robust control methods. We discuss control of nonlinear systems with uncertainty using a combination of feedback linearization and variable structure control (VSC). We first show that feedback linearization is not an
effective means of control for systems with uncertainty, so a variable structure controller is added to compensate for the uncertainty. The role of neural nets is examined to provide compensation to the VSC to minimize chattering of the control signal. It is demonstrated via simulation that the controller with neural net compensation has the advantage of reducing the tracking error of the system while also smoothing the control signal.

The final chapter, Chapter VI summarizes the dissertation. Included is a brief conclusion section and a section detailing suggestions for further research.

An appendix is included to give structural details of the JPL/AFAL Flexible Ribbed Antenna Structure, which is used in the examples of Chapter III.
CHAPTER II

Neural Networks for Control Systems

The purpose of this chapter is to introduce the fundamental properties of neural networks and to serve as a basis for the chapters to follow which address applications specific to control. This chapter discusses the computational tools necessary for neural net implementations and gives a general (but by no means comprehensive) overview of many of the various types of neural nets. The backpropagation algorithm is presented as the primary means of training the neural nets contained within. Since the focus of this dissertation is on the use of neural nets in control applications, the neural nets discussed in detail are mapping neural nets. The neural net performs a nonlinear mapping from an input space to an output space. We assume for the purposes of this chapter that the mapping is known, that is, representative inputs and outputs are available and are used to "train" the neural network.

The chapter is organized by starting in Section 2.1 with a discussion of neural computations and the various training algorithms used. Section 2.2 broaches the subject of neural net experiment design to answer the question of how to accomplish a desired mapping with a reasonable amount of accuracy. Finally, in Section 2.3, some general structures for neural nets in a control systems paradigm are discussed. A brief summary concludes this chapter.
2.1 Neural Net Overview

Artificial neural net models have been the subject of a vast amount of research in hope of achieving human-like performance for processing of algorithms. Many names are used to describe these structures: neural nets, connectionist models, distributed processing models, and neuromorphic systems to name a few. Neural nets are biologically inspired, and organized to resemble the anatomy of the brain. The models are dense interconnections of simple building blocks that constitute the computational element of the process. The simplest node sums several weighted inputs and passes the result through a nonlinear element. Research into the area of neural nets has its origins in the early 1940's [42] but the recent resurgence in research is due to analog VLSI implementation techniques, and new net topologies and algorithms using parallel computation schemes for a variety of applications requiring high speed and performance. In particular, in the mid 1980's, the backpropagation algorithm for multilayer perceptrons opened new areas that were previously thought to be unsolvable.

Neural net research is divided into three basic areas: applications, algorithm development, and hardware implementations. This dissertation addresses only applications, specifically the use of neural nets in control. Many neural net control applications exist, among them are some novel ideas with varying degrees of success. For example, there are adaptive controllers and self-tuning regulators implemented in [24, 28], multitudes of robotic applications for control [33, 17, 44] and identification [19], and system identification [47], to name a small subset of the existing control
There are several types of neural nets characterized by their topology and algorithm used for training. Training a neural net (also called teaching or neural net learning) is the process of adjusting the weights to improve performance. To teach a neural net in a supervised mode, one presents the network a training data set consisting of pairs of inputs and desired outputs; weights are adjusted to minimize a cost criterion (such as minimum squared error over the data set) and to produce a consistent response. With supervised learning, one generally assumes that target values are available for the output nodes. This assumption is a fundamental problem for neural nets under supervised learning, since for a large number of applications, these values cannot be found prior to training. For example, feedforward control of a robot requires that the torques needed to achieve the desired trajectory are known if one is training a neural net to complete this task. In general, these torques are not known, instead, one is provided with both the current state of the system in terms of joint coordinates and the goal of the experiment. Thus, other methods which do not use a target output are necessary to train the neural net.

In [29], Jordan attacked the problem of unsupervised learning, where the weights are adjusted without the use of a desired output vector. He proposes the use of minimization techniques on performance or cost functions, using the neural network to calculate the gradient of the cost function. Unsupervised learning algorithms are more practical from a biological viewpoint, but the majority of successful application examples to date use supervised learning due to their simplicity and ease of
implementation. Unsupervised learning is discussed further in Chapter IV.

A main emphasis of this dissertation is toward the use of neural nets in the approximation of a bounded mapping $f : S \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$. Here, $f$ is a function which takes inputs from a compact set $S$ in an $n$–dimensional Euclidean space and maps into a bounded subset $f(S)$ in an $m$–dimensional Euclidean space. The accuracy of the approximation is discussed in terms of an error surface. These mappings are further discussed in Section 2.2.

2.1.1 Computational Aspects of Neural Nets

The processing element, called a neuron, is a nonlinear element with many inputs and one output. Neural network models are dense interconnections of these neurons. The artificial neuron was designed to mimic the first order characteristics of the biological neuron. The simplest type of element sums $N$ weighted inputs to determine the activation level, $a$ of the neuron. The result is passed through an activation function that can be a linear gain, a threshold device, or, to more accurately simulate the nonlinear transfer characteristics of a biological neuron [39], a sigmoid nonlinearity, defined in Figure 1. This process is depicted in the figure, with output $y$ calculated from

$$y = \frac{1}{1 + e^{-(\sum w_i x_i)}}.$$  

(2.1)

Neural net models are developed by fanning out the output of a processing node so that it is the input of another processing element. The weights are adapted according to specific adaptation laws or learning rules to improve performance. Once
the topology of a model is developed, it can be “trained” using a known data set until the weights converge to a desired performance specification. Current research is aimed at development of adaptation laws, new net topologies, and applications which exploit these architectures. Also of interest is selection of the data set which adequately spans the working space of the neural net.

Neural net research has its roots in the early 1940’s. McCulloch and Pitts [42] published the first systematic study of artificial neural networks, using neurons now referred to as perceptrons. A great amount of optimism toward the field of neural nets stemmed from work done in the early 1960’s. Learning rules were developed by Rosenblatt [55], and Widrow et al. [72] made several interesting demonstrations using neural nets. This initial optimism was tempered in 1969 by Minsky and Papert [45] who, after in depth analysis proved that there are severe restrictions on what sort of mappings can be achieved by a single layer perceptron. This problem was not overcome until hidden layers were added, and training algorithms for multi-layer perceptrons were developed in the mid 1980’s. These new algorithms rekindled a resurgence in neural net research after the work by Minsky brought neural net research to a virtual standstill for several years.
One of the researchers who helped to rekindle research into neural nets was Hopfield, whose net, the Hopfield Net can be used either as an associative memory or in the application of optimization problems such as the classic "traveling salesman problem," for example. The Hopfield net usually has weights assigned by a strict rule based on the exemplar set, mainly to help the net converge to a solution. Hopfield nets can give rise to spurious states when used with inputs outside the exemplar set which could prove disastrous in closed loop control applications. For this reason, the Hopfield net is not used in the applications of this dissertation.

The Single Layer Perceptron which was used in the work by Minsky is a neural net that can be used as a mapping from input space to output space with either continuous valued or binary inputs. It can be used as a classifier to determine which class an unknown input belongs. A decision region divides the output space into two classes A and B where the regions are separated by a hyperplane in the multi-dimensional space spanned by the input variables. In the two input case, the hyperplane becomes a line, and the equation for decision making can be derived from the output equation for the perceptron. The perceptron output equation is:

\[ y = f_h \left[ \sum_{i=1}^{N} w_i x_i - \theta \right] \]  

where \( N \) is the number of inputs, \( w \) is the weight to be multiplied by the input \( x \), and \( \theta \) is a threshold or offset variable. For this net, \( f_h(\cdot) \) represents a hard limiter nonlinearity (as opposed to the sigmoid nonlinearity of Figure 1) so that for two inputs \( (N = 2) \), the decision region becomes a line defined by

\[ x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}. \]
Notice that the line depends on both the weights and the threshold, which can be either fixed or adaptive.

Using structures similar to the single layer perceptron, Widrow advanced the field of digital filtering with his introduction of the Adaline (Adaptive Linear Neuron) and the now famous Widrow-Hoff algorithm, also known as the Least Mean Squares algorithm. The Adaline uses a tapped delay line as the input to a structure that is actually a single linear neuron. These neurons find a multitude of applications in adaptive signal processing and noise canceling. Like Minsky and Papert, they discovered the linear separability problem which is the task of separating the input patterns into a predetermined number of regions. One such example is the exclusive-or problem for two input, single output systems. A single layer perceptron cannot separate the inputs to perform the desired mapping. This problem was overcome by using the Madeline, a multilayer perceptron structure, again with linear neurons. Recently, Widrow has used nonlinear neurons in a variety of applications such as the "Truck Backer-Upper," [48] a nonlinear controller which learns to back a trailer truck to a loading dock.

The **Multilayer perceptrons** of Figure 2 below are the building blocks for the majority of the work contained in this dissertation. In the figure, each circle represents a neuron as characterized in Figure 1, and the lines connecting the circles each have an associated weight. The additional capability of multilayer perceptrons stems from the nonlinearities used within nodes. If nodes were linear, then single layer perceptrons could do all of the tasks of the multilayer perceptrons. The layers would just represent
a matrix multiplication of the weights to reduce to the single-layer perceptron. Mul-
tilayer perceptrons, on the other hand, can represent arbitrarily complex mappings
from input to output, which is implied by the both the Kolmogorov theorem [21]
and from the Stone-Weierstrass Theorem [47, 14]. (See the next section for details of
these theorems.) However, the method used to determine this mapping is not at all
clear [39].

![Multilayer Network Diagram]

Figure 2: The Multilayer Network

We use the following notation and terminology when discussing multilayer per-
ceptrons. The circles of the input layer are linear neurons which means that they
accept the input \( x_i \) and pass it to the next layer through a weighted connection. The
number of layers refers to the number of weight layers. Therefore, a three layer net
is a neural net with two hidden layers, a linear input layer and an output layer. We let the number of inputs be $N_0$, there are $N_1$ nodes in hidden layer 1, $N_2$ nodes in hidden layer 2, and $N_3$ output nodes. Using the notation of Narendra [47], this net has topology $N_{N_0,N_1,N_2,N_3}$. The calculation through the net from input to output is called forward propagation. Let $x_i$, $i = 1, \ldots, N_0$ be the inputs to the neural net and $y_j$, $j = 1, \ldots, N_3$ be the outputs of the neural net. Each input is multiplied by a weight, $w_{mn}^k$, which is the weight connecting the $m$th input from layer $k$ to the $n$th node in layer $k+1$. The matrix of weights connecting layer $k$ to layer $k+1$ is indicated by $W_{k,k+1}^k$. Thus, the output of the $i$th node in hidden layer 1, $z_i^1$ is given by

$$z_i^1 = f_s\left(\sum_{j=1}^{N_0} w_{ij}^0 x_j\right) \quad (2.4)$$

where $f_s$ again represents the sigmoid nonlinearity. Then we can indicate the output of all nodes in hidden layer 1 using the following vector notation:

$$Z^1 = \mathcal{F}(W_{01}X) \quad (2.5)$$

where $Z^1$ is a vector of outputs from the nodes of hidden layer 1, and $\mathcal{F}$ is a diagonal operator of sigmoid functions. Extending this notation to a three layer net, the forward propagation equation for a three layer neural net is

$$Y = \beta \mathcal{F} \{ W_{23} \mathcal{F} \{ W_{12} \mathcal{F}(W_{01}X) \} \}. \quad (2.6)$$

In (2.6), $\beta$ represents the scale factor, used to scale the inputs and outputs prior to training on the same range of the sigmoid nonlinearity.
2.1.2 Training Algorithms for Neural Nets

Much of the recent excitement of neural net research is due to claims that neural nets can automatically learn from their mistakes. This is not entirely correct, and therefore it must be clarified. For most of the multilayer neural nets used in this dissertation, supervised learning is implemented as in Figure 3, where at each output node, the neural net output is compared with a desired output value, and the error is used to adjust the weights to improve performance. In this context, one can say that a neural net learns from its mistakes, but it is a systematic algorithm called backpropagation that allows for this learning procedure where internal representations are learned based on the input data set. These learning representations make it plausible to train networks to perform such difficult tasks such as nonlinear system identification, a task that until recently has been too difficult to accomplish using traditional techniques.

![Figure 3: Neural Net Supervised Learning](image)

Many purists believe that if an artificial neural network is to mimic the function of biological neural nets, then it is not practical to use supervised training for a neural net. D. O. Hebb was among the first to propose the idea of unsupervised learning for artificial neural networks. Prior to the work of Hebb, it was generally believed that learning in a biological system involved some type of physical change to the neurons,
but no one was able to clearly state how this would take place. Hebb proposed, based on physiological and psychological research, that a synapse (weight) connecting two neurons is strengthened whenever both of those neurons fire. (This notation is taken from biological brain research; a neuron is said to "fire" when it has maximal output). Hebbian learning is sometimes referred to as correlation learning since a synapse is strengthened according to the correlation between the excitation levels of the neurons that it connects. More recently, Jordan [29] accomplishes unsupervised learning by incorporating a set of constraints into the learning rule and determines the net output by minimizing the constraint which is given in the form of a cost function.

Much more prevalent in the literature are applications of neural nets involving supervised learning. In [55], Rosenblatt devised a learning rule for adapting the weights in a single layer perceptron. This algorithm serves as a starting point for learning algorithms which depend on the "delta rule". In this algorithm, weights and thresholds are initialized to a small random value. An input is applied, outputs are calculated and compared to a desired value, \( d(k) \) and weights are adapted according to the delta rule. With this rule, the difference between the desired and actual outputs of a node is calculated. If the difference is zero, then nothing is done, otherwise, the difference is multiplied by the input to that node, \( x_i \). This quantity is further multiplied by a learning rate coefficient \( \eta \) to allow control over the size of the weight changes. This relation for the \( i \)th connection weight is calculated as:

\[
\begin{align*}
    w_i(k + 1) &= w_i(k) + \Delta_i \\
    \Delta_i &= \eta [y(k) - d(k)] x_i
\end{align*}
\]  
\( (2.7) \)
The Backpropagation algorithm is a generalized delta rule. It is a systematic method of training multilayer neural nets using a gradient descent technique which minimizes errors between a desired output value and the output of a neural net, using a training set of input-output pairs called exemplars. Derivations of the algorithm are prevalent in the literature and thus are omitted. (See [56, 73, 53] for a rigorous mathematical treatment or [69] for a simpler approach.) A brief description of the backpropagation algorithm used for training multilayer neural nets is provided in this section.

With backpropagation as with any training algorithm, the objective is to find a rule that maps an input to the desired output. With neural net training, one presents the net with an input/output pair; the net produces its own output and compares this output with the target value. If there is a non-zero difference, then the weights are adjusted to minimize the error. To minimize the error, one uses a gradient descent rule which calculates the weight change in proportion to the derivative of error measure with respect to each weight. This corresponds to a steepest descent rule on a surface in weight space where the height of a surface is equivalent to the error measure.

Define the error measure $E_T$ as

$$E_T = \frac{1}{2} \sum_{S} \| e \|^2 = \frac{1}{2} \sum_{S} \sum_{i} (y_i - d_i)^2$$

where for each exemplar in the set $S$, $i$ is the index over output units, and $e_i$ is calculated as the difference between the $i$th desired or target output $d_i$ and $i$th net output $y_i$, as shown in Figure 3. With backpropagation, one has the option of updating weights either after presentation of each exemplar in the data set which has the ad-
vantage of not requiring extra memory for saving derivatives. An alternative method is to accumulate $\partial E / \partial w$ over all input/output cases before changing weights. To avoid storing derivatives we chose to update weights after each exemplar is presented. Hence, the error measure for the $p$th exemplar is

$$E_p = \frac{1}{2} \sum_j (y_j - d_j)^2.$$  \hfill (2.9)

We wish to show that the change to each weight is proportional to the derivative of the error measure:

$$\Delta w_{ij}(p) \propto \frac{\partial E_p}{\partial w_{ij}} \hfill (2.10)$$

In the discussion of backpropagation that ensues, we follow the discussion of Rumelhart, et al. [56]. The derivative in Equation (2.10) is the product of two derivatives. One represents the change in error with respect to net input and one which is the effect the weight change has on the net input. Thus, for the $p$th exemplar,

$$\frac{\partial E_p}{\partial w_{ij}} = \frac{\partial E_p}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} \hfill (2.11)$$

Here we define $a_j$ as the activation level of the neuron from Figure 1,

$$a_j = \sum_i w_{ij} z_i \hfill (2.12)$$

Thus, for exemplar $p$,

$$\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{ik} z_k = z_j. \hfill (2.13)$$

Now if we define

$$\delta_j = \frac{\partial E_p}{\partial a_j} \hfill (2.14)$$
then we can write (2.11) as

\[ \frac{\partial E_p}{\partial w_{ij}} = \delta_j z_i \]  

(2.15)

For gradient descent, we use a learning coefficient \( \eta \in [0,1] \) so that the weight change for the \( p \)th exemplar is given by

\[ \Delta w_{ij}(p) = \eta \delta_j z_i, \]  

(2.16)

just as in the Delta rule of Equation (2.7). The difficulty for multilayer nets is to find \( \delta \) for each neuron in the network. The backpropagation algorithm recursively finds these \( \delta \) values and propagates them backwards through the network. To compute \( \delta_j \), we apply the chain rule to get

\[ \delta_j = \frac{\partial E_p}{\partial a_j} = \frac{\partial E_p}{\partial z_j} \frac{\partial z_j}{\partial a_j} \]  

(2.17)

The second factor of (2.17) is the derivative of the sigmoid nonlinearity,

\[ \frac{\partial z_j}{\partial a_j} = f'_j(a_j) \]  

(2.18)

For the first factor of (2.17), it is considered as two cases. The first case is simple; we assume that the neuron is an output node, then

\[ \frac{\partial E_p}{\partial z_j} = (y_j - d_j) \]  

(2.19)

Then from (2.17), we get

\[ \delta_j = (y_j - d_j)f'_j(a_j) \]  

(2.20)
If the neuron is a hidden unit, we use the chain rule on the first factor of (2.17) to obtain

\[
\sum_k \frac{\partial E_p}{\partial a_k} \frac{\partial a_k}{\partial z_j} = \sum_k \frac{\partial E_p}{\partial a_k} w_{kj} = \sum_k \delta_k w_{kj} \tag{2.21}
\]

A convenient feature of the sigmoid is that its derivative is easily calculated as a function of the output \( z, (z_j = y_j \text{ for output nodes}) \). Thus we can write,

\[
f_j'(a_j) = z_j(1 - z_j). \tag{2.22}
\]

To summarize, after presentation of the \( p \text{th exemplar} \), each weight is updated by

\[
w_{ij}(p) = w_{ij}(p - 1) + \eta \delta_j z_i \tag{2.23}
\]

where

\[
\delta_j = \begin{cases} 
  y_j(1 - y_j)(y_j - d_j) & (\text{output nodes}) \\
  z_j(1 - z_j) \sum_k \delta_k w_{kj} & (\text{hidden nodes}) 
\end{cases} \tag{2.24}
\]

There have been several attempts to improve the basic backpropagation rule described above, mainly in an attempt to improve the speed at which the weights converge. As mentioned previously, a fundamental limitation of the backpropagation algorithm is the weights sometimes take a very long time to converge to a constant or static net. We acknowledge this fact, but confine ourselves to using "standard backpropagation" since our emphasis is not on producing new learning rules, but on using neural nets in new control applications. The only modification made is the addition of a momentum term described below. Some of the more popular modifications of the backpropagation algorithm involve using combinations of an adjustable slope, horizontal offset, and vertical offset of the sigmoid nonlinearity. In [32] there
are promising results for speeding up learning using adaptive slopes. The adaptation of vertical and horizontal offsets however, often amounts to a simple variation of the scaling parameter $\beta$ in Equation (2.6).

True gradient descent requires that infinitesimal steps are taken. The constant of proportionality from (2.10) is the learning rate, $\eta$. By choosing $\eta$ large, we affect large changes in the weights. If $\eta$ is too large, then oscillation occurs and the weights do not converge to a constant value. On the other hand, if we choose $\eta$ to be too small, then learning takes a long time. To speed learning while also avoiding oscillation, we introduce a momentum term $\alpha \Delta w_{ij}(p - 1)$ to include the effect of past weight changes. With this term included, weights are updated by

$$w_{ij}(p) = w_{ij}(p - 1) + \eta \delta_j z_i + \alpha \Delta w_{ij}(p - 1)$$

(2.25)

The momentum is so named because of the physical connotation.

The relationship of the backpropagation algorithm to the least mean squares (LMS) algorithm is obvious to those familiar with LMS. Using the same error measure from Equation (2.8), we can easily show that for linear neurons with no hidden layers, the delta rule implements gradient descent in $E$. Following the same procedure as Equations (2.11)-(2.17), it is easy to show that

$$\frac{\partial E}{\partial w_{ij}} = \delta_j x_i$$

(2.26)

which is proportional to the weight change dictated by the Delta rule of (2.7). For linear units, we can calculate the two parts of the derivative in (2.11) as follows:

$$\frac{\partial E}{\partial z_j} = (y_j - d_j) = \delta_j$$

(2.27)
Substituting into Equation (2.11), Equation (2.26) results. We can then update weights using (2.23).

2.2 Designing Neural Net Experiments

A neural net experiment is an exercise to determine a mapping from an input space to an output space. Experiment design involves input data selection, choice of training algorithm, and evaluation of the resulting mapping. Further testing may be necessary to place the neural net into a control system, close the control loop, and validate the performance. This step will be addressed in the chapters pertaining to closed loop control. In the discussion that follows, we constrain ourselves to the issues of input data selection, determining the accuracy of the input/output mapping, and with the methods used to validate the mapping to ensure that the desired accuracy is obtained for more general inputs. Input data selection is a fundamental consideration for this process.

As in any scientific experiment, one must use good judgment to form a neural net experiment. The suggested procedure is:

1. Select the input and output variables for the neural net.

2. Select a representative input/output training data set that is informative.

3. Determine the net topology.

4. Determine the best fit of the neural net output data to the training set.
The signals used for the inputs and outputs of a neural net are sometimes obvious for the case of a functional mapping, or not so obvious for other applications. For example, in the case of a FIR filter, the number of inputs varies as the design evolves since the number of inputs is equal to the number of delays. Thus, for neural dynamical systems, the process of choosing the appropriate number of inputs and outputs is similar to choosing the model structure in system identification experiments. This point is elaborated in the following chapter with regard to neural net system modeling.

The second step from above is a fundamental issue since one must make wise choices with respect to the input data or else the mapping is completely invalid. The basic idea is to choose inputs to maximize the information about the mapping. An input sequence is informative if it contains a complete description of the input space. We address this issue further in the sequel. Furthermore, if the input sequence has time as a parameter, we can make more mathematical restrictions on the nature of the input sequence. This discussion is continued in Chapter III.

The net topology refers to the number of layers and nodes to be used in the neural net. The number of hidden nodes can be chosen using the result of [46] in which the author arrives at a formula for determining the number of hidden nodes needed based on the number of unique data points in the input data set. However, he only does so for nets with one hidden layer. Our results indicate that considerably better results can be obtained by using two or more hidden layers, so while the above result can give a lower bound for the number of hidden nodes to use, it is sometimes just as easy to proceed on a trial and error basis, using two hidden layers as a starting point.
The fourth issue is one that deals with validation of the proposed mapping neural network. There are several means of evaluating the accuracy with which a neural net performs the desired mapping that are discussed herein.

2.2.1 Input Data Selection for Accurate Functional Mappings

The input data selection process is the key to finding a good neural net functional mapping. The issue here is to select enough data points to fully describe the input space. This often amounts to a trade-off between size of the data set (number of exemplars) and the training time. In addition, if there is not enough data available, then the neural net implementation just will not work, since when implemented, the neural net will not be able to perform the required interpolation since the data points for which it was trained are too far apart. The problem, therefore, is to find a reasonable number of data points that will encompass every possible input case when the neural net is used for general mappings.

Once a data set is selected, it is generally a simple procedure to define a neural mapping from input space to output space, however, the generalization of this mapping to inputs from outside the training data set is more difficult. Many theoretical results have been published addressing the existence of neural net mappings for arbitrary functions. In [14], Cotter uses the Stone-Weierstrass Theorem, from classical real analysis, to show that certain neural network architectures are capable of approximating arbitrary functions. Furthermore, in [22], there is a proof that any $L_2$ function can be realized with any desired degree of accuracy with a neural net.
This proof is based on Fourier series representation of functions. A neural net can be trained to learn a sinusoid to a high degree of accuracy, then the appropriate number of these neural nets, based on the number of terms retained from the Fourier expansion, is summed with weights equal to the Fourier coefficients to achieve the desired result. Since a Fourier series can approximate a function to any desired degree of accuracy, a neural net will also do the same in this setting. Although this proof is an important result, it is not useful for the types of mapping functions that are of interest for the control applications discussed herein. This method works well when the functional mapping is exact, however, it does not allow for learning uncertainties and other phenomena common to the control paradigm, nor does it allow one to take advantage of the main benefit of neural nets: the ability to learn an internal representation of the functional mapping based on input/output data. Therefore, we address the methodologies germane to functional mappings using backpropagation as the learning rule for multilayer perceptrons and apply them to the control paradigm in later chapters. We begin with a formal definition of the mapping to be realized.

Definition 2.1 [22] A neural net mapping is an approximation of a bounded mapping \( f : S \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^m \), where \( f \) is a function which takes inputs from a compact set \( S \) in an \( n \)-dimensional Euclidean space and maps into a bounded subset \( f(S) \) in an \( m \)-dimensional Euclidean space.

The neural net uses examples (exemplars) \((x_1, y_1), (x_2, y_2), \ldots, (x_j, y_j), \ldots\) of the mapping \( y_j = f(x_j) \). Input vector \( x_j \) is chosen randomly from the set \( S \) in accordance with a fixed probability density function \( \rho_X(x) \), then scaled to the compact space
defined as the unit hypercube, as indicated by $D$ below. It is a standard result in real analysis that every closed and bounded set in $\mathbb{R}^n$ is compact. The unit hypercube is an example of such a compact space and is defined by the space $D$:

$$D = [0, 1]^n.$$  

(2.29)

Thus, each exemplar consists of a scaled input $x_j \in \mathcal{X} \subseteq \mathcal{S}$ which is a vector in $\mathbb{R}^n$ whose elements lie in the range $[0, 1]$.\footnote{It is sometimes suggested (e.g. [56]) that the data be scaled on the range $[0.1, 0.9]$ to avoid the asymptotes of the sigmoid nonlinearity. The effect of this scaling is to speed convergence of the backpropagation training rule since the change in weight is proportional to the derivative of the sigmoid nonlinearity, and that derivative is close to zero near the asymptotes.} Here, the symbol $\subseteq$ is used for subset notation.

Conditions can be imposed on the probability density function $\rho_x(x)$ based on the neural net application. In general, we wish to fully describe the space spanned by the inputs of the neural net. If the mapping is to be used for pattern recognition, then the input sequence should contain all possible patterns that will be encountered. If the inputs are a time sequence, then one may impose restrictions on the frequency content of the input sequence. By knowing \textit{a priori} the intention of the mapping, one can shape the probability density function to concentrate the input around a certain condition or well defined operating point.

The main reason that input choice is critical is that neural nets are very good at data interpolation but perform poorly at extrapolation. To better understand, one can think of a neural net as a simple curve fitting operation in a high dimensional space. Then, learning is the process of finding a best-fit surface in the high dimensional space to a finite set of points known as the training set. Thus, when using new data different from the training set, the net is interpolating the data on this fitting surface.
The extrapolation problem means that the data is too far from the fitting surface and the net cannot extend itself to give a reasonable value for the fit. By choosing inputs for training that span the entire working space of the neural net, the designer gives a good definition of the surface in the high dimensional space and thus ensures that the neural net always in interpolation mode when used with data not necessarily in the training set.

2.2.2 The Neural Net Convergence Problem

The neural net convergence problem is a hurdle that is the root of a great deal of skepticism toward neural nets. The difficulty stems from the non-uniformity of the surface in weight space for multilayer perceptrons with sigmoid activation functions. A desirable convergence criterion is to terminate the training algorithm when the error measure $E_T(\cdot)$ from Equation (2.8) has been sufficiently minimized. This may be accomplished by using a threshold on $E$, but it can lead to difficulties due to local minima in the weight surface. Specifically for backpropagation, a proof of convergence does not exist since, with a fixed learning rate, one cannot always guarantee that the error will decrease on each successive iteration. This violates a strict definition of convergence since the error measure does not continuously decrease. Furthermore, although there are means available for adjusting the step size, it cannot be done in such a way to guarantee convergence. Because the available algorithms cannot guarantee that the error is reduced by successive iterations, it cannot be proven to globally converge.

On the other hand, it has been shown in a plethora of examples that a neural net
with sufficient training performs admirably. With this in mind, we offer a definition for which conditions are put forth for knowing when to stop training the neural net. This definition is used for training in the examples contained in this dissertation, and satisfies the empirical nature of the training algorithm.

**Definition 2.2** *A neural net trained using backpropagation is trained to within bound epsilon (T.T.W.B.ε) if the following condition is met. Using the overall error measure $E_T$ defined in Equation (2.8), if*

$$\frac{1}{N} E_T < \epsilon$$

**(2.30)**

*then the neural net has a good representation of the data set and training can cease. The neural net is said to be fully parametrized in weight space, and it can now be used to interpolate data from outside the training data set.*

The neural net is trained until this condition is met which implies that for the training set $S$ with $N$ exemplars, the neural net has a sufficient internal representation of the data set. The neural net is said to converge to a set as opposed to converging to a local minima since even if the condition is met, it is not guaranteed that the net has converged even to a local minima. For example, in weight space, the surface could be the bottom of a bowl which has undulations such that several local minima exist even in the region where the above condition holds. Thus, it is conceivable that further training may affect a change in the weights that could force one to move in weight space from one global minima to another while not necessarily changing the overall performance of the neural net. This also implies that the weights may change significantly without actually changing the performance of the neural net.
Definition 2.2 alone is not sufficient for finding the desired parametrization of the neural net. The neural net must be tested using a data set different from the training data set. This process is often referred to as cross-validation and is discussed in the section to follow. It is noted the since the error process of the neural mapping ideally has a zero mean, then the value of $\epsilon$ is simply the expected value of the variance of the error process.

2.2.3 Validation of the Neural Mapping

So far, our only measure of neural net performance is from the error measure over the training data set. A more appropriate validation process for neural nets is one where data different from the training set (but selected according to the same rules) is used to evaluate performance. This process is referred to as the cross-validation process. The following definition is used to complete our requirements for trained neural networks.

**Definition 2.3** A neural net is said to be usable for extrapolation if the following condition holds. The overall error measure satisfies

$$\frac{1}{N} \mathcal{E}_T \sim \mathcal{O}(\epsilon)$$

for a data set $\tilde{S}$ different from the training data set $S$, where $\mathcal{O}(\epsilon)$ means that the right side of (2.31) is of order $\epsilon$.

Typically training is an iterative process. Cases exist where the neural net is trained for the condition of Definition 2.2, but the condition of Definition 2.3 is not met. The designer has several options: (1) use another training data set and begin
training again; (2) change the neural net topology and start over; or (3) continue
training until both Definition 2.2 and 2.3 hold. If several data sets $\mathcal{S}$ are used and the
condition of Definition 2.3 continues to hold such that the error measure for data sets
outside the training set is $O(\epsilon)$, then we ensure that the neural net is interpolating
(sometimes called generalizing) for all inputs in the allowable input class.

Given a neural network architecture, a learning rule, and a functional mapping,
$f(\cdot)$, a measure of the accuracy of the neural mapping is called an error surface. It
is a function of the weights $W$ of the neural network where $W$ is a vector containing
all weights of the neural net. A trained neural network is called a static net and is
parametrized by the weight vector $W$. Given a training data point from the $k$th trial
$(x_k, y_k)$ from the mapping $y_k = f(x_k)$, we present the neural net with the vector $x_k$
and receive as output $y_k'$ from the neural net. Let the square of the approximation
error for each exemplar be defined as

$$F_k = [f(x_k) - y_k'(x_k, W)]^2$$

since $y'$ is a function of both the net input and the weights of the neural net. Assuming
that $W$ is fixed, define $F(W)$ as

$$F(W) = \lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} F_k.$$  \hspace{1cm} (2.33)

In [23], it is argued that the limit in (2.33) exists. Since the mapping $f$ is bounded,
then all of the $y_k$'s are also bounded. Similarly, for a fixed $W$, $y_k'$ is a continuous
mapping from a compact set in $\mathbb{R}^n$ into $\mathbb{R}^m$, so $y_k'$ is bounded also. Thus, the
variance of the random variable $F_k$ is bounded. It is further argued that $F_k$ obeys
the strong law of large numbers and the sum in (2.33) must almost surely converge.
The error surface of a net is the surface defined by the equation $F = F(W)$ in the $(Q + 1)$-dimensional space of vectors $(W, F)$, where $Q$ is the number of weight connections in the network. The shape of the error surface of a single layer neural net with linear activation functions is concave upwards with only one minimum, so gradient descent guarantees that we will find the best set of weights. Conversely, the error surface of a multilayer neural net with sigmoid squashing functions is complex, with no guarantee of always being concave upwards, so the surface as described in [56] contains hills, valleys, flat areas and troughs with little slope. The learning rule has the property that given a starting point on the surface that is not a minimum, the learning rule will modify the weights so that $F(W)$ decreases. The momentum term is useful in spaces containing long ravines that are characterized by sharp curvature across the ravine and a gently sloping floor. The addition of a momentum term filters out the high curvature and allows the weight steps to be bigger, which speeds up learning when progressing across gently sloping surfaces. Interestingly, it has been found that if one makes $\eta$ small enough, one obtains identical results as using momentum and a larger value of $\eta$ at the expense of longer training times.

There exists an interesting phenomena observed during training of a neural net that is sometimes referred to as overtraining. This occurs when too many passes of the training data set are used in the training process. This is not unlike a similar occurrence when using a conventional parameter estimation technique called Recursive Least Squares (RLS), in which a forgetting factor is included to eliminate this problem. In the backpropagation case, the surface defined in weight space becomes
very sharply defined for each point in the training set, so that deviation from the training set is not tolerated. The result is poor interpolation of data points outside the training set. This is the prime motivation for using both Definitions 2.2 and 2.3 for the training criterion.

It was mentioned in [22] that local minima can exist on the error surface. If the total error is high, then the addition of a momentum term can move the weight away from the local minimum. If the total error is low enough, then the local minimum can allow the net to perform satisfactorily. The empirical contribution of this work and others like it demonstrate that the apparently fatal problem of local minima is not fatal in most applications. A similar phenomena is observed in standard numerical minimization problems, where for example, a quadratic cost functional is minimized to obtain an optimal control law. Although we can guarantee that for linear systems we can find a global minimum, for systems with nonlinear characteristics, this minimization will often result in a solution that works well even though the minimization is not the global minimum.

2.3 Common Structures for Neural Net Control

The feedforward neural nets presented in Section 2.1 possess no "memory" capabilities, so the concept of state variables is difficult to represent with neural nets. Some recent work using recurrent neural nets which allow for feedback between layers shows some promise, but these results are in their infancy. Therefore, for models of control systems one is restricted to use of a regression vector for inputs to a neural net to define a dynamical system. On the other hand, in a large number of control applications,
we are simply interested in a mapping from one space to another, which is precisely what a neural net is quite good at. In this section, several common structures are given for using neural nets in the control paradigm. In [71], Werbos refers to this area as neurocontrol; that is, the direct use of neural networks to control actions or to describe system dynamics. Also, in [5], Barto discusses several neural net control strategies under the heading of connectionist models.

2.3.1 Replacing Existing Controllers with a Neural Net

When designing experiments for neural net implementation, one is presented with the question of how to generate the training data for supervised learning. Often times, one may know the outputs of a system, but may not be able to find directly the inputs that produce those outputs, and vice versa. In this section, we use existing controllers to produce the training data for a neural net controller. A diagram of this structure is shown in Figure 4.

The input values of the neural net are the inputs of the controller, and the target values for the neural net are the control signal, so that the neural net is a mapping from output or state space to control space. One of the first applications of this was by Widrow [72] using the Adaline and Madaline networks to implement an inverted pendulum controller.

The question arises that if a controller already exists and works, why would we want to replace it with a neural net? A neural net controller is appropriate if the conventional control algorithm is not easily implemented in real time, or if there is a human in the loop to provide the control. Furthermore, the conventional controller
Figure 4: Existing Controller Replaced by a Neural Net Controller

does not necessarily have to be removed. It can work in concert with the neural net controller to provide a measure of reliability and robustness. This was done in [75] where a neural net controller and a variable structure controller are applied together to smooth the control signal. This implementation is discussed further in a later chapter.

2.3.2 Exact Modeling of Nonlinearities

In [47], Narendra gives four model structures for designing a neural net model of a linear system. Of the four model structures, there is one that covers the most general case, that is, a system model can be described by a difference equation that is a nonlinear function of the regression of past inputs and outputs. The other model structures make the assumption that the inputs and outputs can be separated and treated as individual nonlinear functions of previous data or even as linear functions.
In all but the most general case, the system nonlinearities are assumed to be known exactly. The general case is discussed in detail with respect to system modeling in the next chapter. In the more specific cases, one uses the known nonlinear function to provide the training data for a neural network, then replace this nonlinearity by the neural network. Although these systems allow for analytical studies, they are too restrictive for many control applications since the assumptions on linearity and separability are not always available in the general case, and exact knowledge of the plant is seldom valid. For this reason, we turn our attention to other methodologies.

2.3.3 Inverse Dynamics

Inverse dynamics is often used for control of robot systems in which the controller is given the present state and desired next state to produce the desired control action. For training a neural network, one can proceed using a two part process. As described in [33], generalized learning is done using the configuration of Figure 5(a) in which a set of desired actuator commands, \(U_d\) are used to drive the plant and produce a set of trajectories which are subsequently used as input to the neural net. The neural net is trained to produce a set of reference control signals \(U_r\) to match \(U_d\). Once the neural net is sufficiently trained, it can be used as shown in Figure 5(b) in a specialized learning configuration.

In the specialized learning configuration, the neural net is given the desired trajectories directly to produce the control signal used in the plant. Learning continues in this configuration by considering the plant as the \((L + 1)\)th layer of the neural
net. Although the backpropagation algorithm cannot affect this layer, it can be used to generate the partial derivatives used to propagate errors in the neural network to further train the net. This configuration allows one to limit the size of the training set for the generalized training. In addition, if new training patterns are added or if operating points change for the system, then the specialized training can most often be used to avoid long training times.
2.3.4 **Indirect Adaptive Control**

Of particular interest to control engineers is the ability of neural nets to capture highly nonlinear relationships, thus allowing the possibility of extending our knowledge of control of nonlinear systems. A model based adaptive controller can be implemented using neural networks for use in nonlinear systems in the following way. First the plant is identified using a neural network identification scheme. This neural model is then used in the controller as shown in Figure 6. The controller can either be an inverse plant model as described above, or one can use the identification model to compute the partial derivatives of a performance index with respect to the controller parameters as in [29].

![Figure 6: Indirect Adaptive Control of Nonlinear Plants using Neural Nets](image)

In [47] both direct and indirect adaptive control methods are discussed, but indirect methods are chosen since methods for directly adjusting the control parameters based on output error (between plant and reference model outputs) are not available for the nonlinear case. This is because the unknown nonlinear plant lies between the controller and the output error.
2.4 Chapter Summary

This chapter discusses neural nets in detail, beginning with computational aspects and continuing with a discussion of training algorithms, specifically focusing on the backpropagation training algorithm. Neural net functional mappings are a major emphasis of this dissertation, and are discussed in detail. Criterion for input data selection for the mappings is given so that the mapping continues to give valid results even when used for inputs outside of the training data set. The mapping problem is discussed in terms of a three-part process: data selection, convergence of the training algorithm and validation of the mapping. The chapter also includes some common paradigms for neural net control, giving block diagrams of some common structures.
CHAPTER III

Dynamic Modeling Using Neural Networks

This chapter addresses the problem of modeling and identification of both linear and nonlinear systems using neural nets. System identification is an experimental approach in which experiments are performed on a system, then a model is fitted to the recorded data by assigning suitable numerical values to its parameters. It is shown that neural net system modeling offers certain advantages over conventional techniques using "ARMA-type" models for identification. A general overview of system identification is given in Section 3.1, and in Section 3.2 the model structures for linear systems are compared for conventional and neural net approaches, followed by several examples in Section 3.3. Nonlinear system modeling and identification is addressed in Section 3.4, complete with an example and in Section 3.5, the effect of underparametrization is investigated using several different model structures. A brief summary concludes this chapter.

There are several contributions that can be found in this chapter. In general, the link between conventional modeling and neural net modeling is established, and a new black-box technique for identification of both linear and nonlinear systems is introduced. Furthermore, the input restrictions of Section 3.3 provide a systematic means for finding an accurate neural representation of the system.
3.1 System Identification Overview

Conventional techniques for model determination are embodied in the field of study called System Identification. System identification is the subject of constructing models of dynamical systems from experimental data. These models are subsequently used in a wide variety of applications: by control engineers for simulation, linear prediction, or controller synthesis; for use in the related areas of adaptive filtering and pattern recognition; and in non-engineering fields such as econometrics. System identification can be performed on-line or off-line, using both time domain and frequency domain techniques. We restrict our discussion to the off-line case and use time domain methods for parameter estimation.

A model is a mathematical means of describing a real world phenomenon such as a dynamical system. In most respects, it is desired that our model describes the properties of the real system sufficiently; it need not provide any physical insight into the plant as long as it is suitable for the desired purpose. To elucidate this notion further, in [41] Ljung states:

"In a sense, there is an impenetrable but transparent screen between our world of mathematical descriptions and the real world. We can look through this window and compare certain aspects of the physical system with its mathematical description but we can never establish any exact connection between them."

Thus, we can use techniques based on years of theoretical results from the so-called "conventional system identification" methods to apply new techniques for modeling
and prediction; the models are then validated on the basis of *usefulness* rather than *truth*. Our goals for modeling are to provide a "good" representation of a dynamical system based on prior input/output data. There are a variety of means available for describing a linear system, and in this chapter, several common methodologies are discussed for time domain descriptions for both linear time-invariant (LTI) systems and nonlinear systems. The motivated reader is encouraged to seek more detail in [41, 62] for system identification, in [20] for related topics in adaptive filtering and [9] for general time series analysis, to name a few.

The general procedure for conventional system identification is to (1) decide upon a model structure, (2) use input/output data recorded over a finite time interval to estimate model parameters using techniques from estimation theory, and (3) to validate the model by comparing various time or frequency responses. From the validation techniques, one can make statements of how well the identified model matches the real system. Just as in Section 2.2 on neural net experiment design, system identification is based on sound engineering judgment and here too, experimentation plays an important role. Thus, the above steps are often an iterative process, where various model structures are used to determine an appropriate representation of the system. Furthermore, input data selection is critical for finding accurate models. If the experiment is poorly planned, then the resulting models will not be very useful.

A shortcoming of the conventional techniques is that a large percentage of the theory is devoted to linear systems and, as such, nonlinear system identification is an area that has received little attention [7]. This is due in part to the difficulty of
deriving identification algorithms that can be applied to a reasonably large class of nonlinear systems. Nonlinear system identification is often done by making many restrictive assumptions regarding the nature of the nonlinearities and underlying differential equations. For example, the Hammerstein model assumes that the input contains polynomial nonlinearities, and coefficients of the polynomial are estimated using the conventional linear techniques. Black-box techniques for nonlinear systems have been surveyed in [7, 43, 18].

In this chapter, a unified discussion of the use of neural nets for modeling dynamical systems is presented. The procedures used for neural net modeling are similar to the conventional techniques for system identification, so parallel development is given, comparing conventional techniques to neural net techniques for both linear and nonlinear systems. The discussion is called a unified approach since if one views the system identification problem using "black-box" techniques, then the design process is identical, except for the parametrization process. For example, the first step toward identification is to choose a model structure to fit inside the black box. This means that one must make decisions regarding both the order of the system and the inputs to be injected into the black-box. For conventional systems, the model is parametrized using statistical techniques such as least mean squares (LMS) type algorithms. In the neural net case, the system order assumption fixes the number of inputs to the neural net. It is then up to the designer to choose the neural net topology. Finally, for parametrization of the neural network, we advocate the use of the backpropagation for finding the weights of the neural network.
The most attractive feature of the neural net approach is that neural net modeling requires very few assumptions on the underlying model structure. As we will see in the next section, if we assume a general structure for a SISO system, then numerous special cases exist based on the many alternate ways to parametrize a linear finite-order system. The structures such as ARMAX, ARX, etc. are simply special cases of the general structure. For neural nets, the different structures occur simply due to model order assumptions and whether or not exogenous inputs are included. Another major advantage of the neural net system approach is with respect to linear vs. nonlinear systems. Neural nets require no assumptions of linearity, so the techniques used for nonlinear system identification are the same as those used for linear systems. In the sections to follow, this is shown to be an attractive advantage.

The neural nets in this chapter are different from the usual applications requiring a static mapping due to implementations in the time domain. The neural nets described in Chapter II define a general mapping from input space to output space where a static functional mapping is well defined a priori. In this chapter, the inputs to the neural net are a moving average of prior inputs, outputs and disturbances, so that the neural net mapping can be thought of as a nonlinear dynamic mapping which can be used as a model representation.

3.2 Linear Model Structures: Conventional vs. Neural Net Descriptions

Consider the block diagram of Figure 7 with input sequence \{u(k)\}, output \{y(k)\}, and disturbance \{e(k)\}, where \{\cdot\} is used to denote a time sequence with normalized
time index $k$ ($t = kT$; $T$ is the interval between samples; $k = 1, 2, \ldots$). Here, we consider the disturbance $e(k)$ to be a sequence of independent and identically distributed white noise. The model is given by

$$y(k) = G(q^{-1}, \theta)u(k) + H(q^{-1}, \theta)e(k)$$

$$E[e(k)e^T(l)] = \Lambda(\theta)\delta_{k,l}$$

where $q^{-1}$ is the backward shift operator, $E[\cdot]$ denotes the expectation operator and $u(k)$ represents a deterministic input to the system. In (3.1) $G$ and $H$ can be considered as filters that most often are of finite order.

The filters $G(q^{-1}, \theta)$ and $H(q^{-1}, \theta)$ in addition to $\Lambda(\theta)$ are functions of $\theta$ which is a parameter vector. For the purposes of this work, the parameter vector $\theta$ is a member of the set $\mathcal{D}$, defined as

$$\mathcal{D} = \{\theta | H^{-1}(q^{-1}, \theta) \text{ and } H^{-1}(q^{-1}, \theta)G(q^{-1}, \theta) \text{ are asymptotically stable, }$$

$$G(0; \theta) = 0, H(0; \theta) = I, \Lambda \text{ is nonnegative definite}\}$$

The motivation behind the restrictions on the definition of $\mathcal{D}$ become apparent in the section on predictors, where it is shown that when $\theta \in \mathcal{D}$, then we can easily find a prediction of $y(k)$ based on previous inputs and outputs.
Equation (3.1) gives a general form of the linear model. In the discussion that follows, some common model structures are described to show how \( G, H \) and \( \Lambda \) depend upon \( \theta \).

### 3.2.1 Common Model Structures

Let the single-input single-output (SISO) system with input \( u(k) \), output \( y(k) \), and disturbance \( e(k) \) have the following model structure:

\[
A(q^{-1})y(k) = B(q^{-1})u(k) + C(q^{-1})e(k)
\]  

(3.3)

with

\[
A(q^{-1}) = 1 + a_1 q^{-1} + \cdots + a_n q^{-n_a}
\]  

(3.4)

\[
B(q^{-1}) = b_1 q^{-1} + \cdots + b_n q^{-n_b}
\]  

(3.5)

\[
C(q^{-1}) = 1 + c_1 q^{-1} + \cdots + c_n q^{-n_c}
\]  

(3.6)

and parameter vector \( \theta \)

\[
\theta = [a_1 \ldots a_n b_1 \ldots b_n c_1 \ldots c_n].
\]  

(3.7)

A system in this form is called an ARMAX model, which is short for ARMA model (autoregressive with moving average) with exogenous input. Equation (3.3) is called the polynomial form and can be written as a difference equation

\[
y(k) + a_1 y(k-1) + \cdots + a_n y(k-n_a) = b_1 u(k-1) + \cdots + b_n u(k-n_b) \\
+ e(k) + c_1 e(k-1) + \cdots + c_n e(k-n_c). 
\]  

(3.8)
Equation (3.3) can be rearranged to show the relationship to (3.1), which is also apparent from Figure 8.

\[
y(k) = \frac{B(q^{-1})}{A(q^{-1})}u(k) + \frac{C(q^{-1})}{A(q^{-1})}e(k)
\]  \hspace{1cm} (3.9)

where \(G\) and \(H\) are parametrized as

\[
G(q^{-1}, \theta) = \frac{B(q^{-1})}{A(q^{-1})}, \quad H(q^{-1}, \theta) = \frac{C(q^{-1})}{A(q^{-1})}.
\]  \hspace{1cm} (3.10)

Defining \(\lambda^2 = E[e^2(k)]\), then \(\Lambda(\theta) = \lambda^2\) and the set \(\mathcal{D}\) is found to be

\[
\mathcal{D} = \{\theta | \text{The polynomial } C(z) \text{ has all zeros outside the unit circle}\}
\]  \hspace{1cm} (3.11)

or equivalently, the reciprocal polynomial

\[
C^*(z) = z^{n_c} + c_1 z^{n_c-1} + \cdots + c_{n_c}
\]  \hspace{1cm} (3.12)

has all zeros inside the unit circle. Using this formulation, several special cases of common use are included below.
AR Model: The AR (autoregressive) model is obtained by setting $n_b = n_c = 0$, and it is assumed that no exogenous input is present. The resulting equation is

\[ A(q^{-1})y(k) = e(k) \]
\[ \theta = [a_1 \ldots a_{n_a}]^T. \]  

(3.13)

MA Model: The MA (moving average) model is obtained by setting $n_a = n_b = 0$. The resulting equation is

\[ y(k) = C(q^{-1})e(k) \]
\[ \theta = [c_1 \ldots c_{n_c}]^T. \]  

(3.14)

ARMA Model: The ARMA (autoregressive moving average) model is obtained when $n_c = 0$. Then

\[ A(q^{-1})y(k) = C(q^{-1})e(k) \]
\[ \theta = [a_1 \ldots a_{n_a} c_1 \ldots c_{n_c}]^T. \]  

(3.15)

FIR Model: The FIR (finite impulse response) model is obtained by setting $n_a = n_c = 0$. The resulting equation is

\[ y(k) = B(q^{-1})u(k) \]
\[ \theta = [b_1 \ldots b_{n_b}]^T. \]  

(3.16)

ARX Model: The ARX (controlled autoregressive) model is obtained when $n_c = 0$. Then

\[ A(q^{-1})y(k) = B(q^{-1})u(k) + e(k) \]
\[ \theta = [a_1 \ldots a_{n_a} b_1 \ldots b_{n_b}]^T. \]  

(3.17)
More general model structures exist so that the filters $G$ and $H$ have no common parameters, which give rise to still more special cases, but they are not considered here. In fact, we will restrict our attention to those structures of special interest to control engineers, namely those with the exogenous input $u(k)$ present. We now turn our attention to the predictor structures which use the model structures previously discussed.

### 3.2.2 Predictor Structures

A model obtained using identification methods can be used in many ways. Of particular interest to control engineers is the predictor model structure, in which the model parameter vector $\theta$ is chosen such that the prediction error

$$
\varepsilon(k, \theta) = y(k) - \hat{y}(k|k - 1; \theta)
$$

is small, where $\hat{y}(k|k - 1; \theta)$ is the one-step-ahead prediction of the output $y$ at time $k$ based on data available at time $k - 1$. For systems of a stochastic nature, the output at time $k$ cannot be exactly determined from the data available at time $k - 1$. A predictor is used to know at time $k - 1$ what the output is likely to be at time $k$ in order to take appropriate control action at time $k - 1$.

Construction of a predictor proceeds in the following way. One must first choose the model structure as discussed above, then, by fixing the model structure, the predictor is specified. Next, a criterion function is selected which is a scalar-valued function of all prediction errors available at the present time, sometimes referred to as a loss function. This function is used in the parametrization of the predictor, by
choosing the estimate \( \hat{\theta} \) which minimizes the loss function. Often times the predictor is chosen using optimal techniques, where one selects \( \hat{\theta} \) by forming the optimal mean square predictor, for example. The prediction error method is illustrated using the block diagram of Figure 9.

To more fully understand the process of designing a predictor, a first-order ARMAX model is used for illustration. The difference equation of the model is

\[
y(k) + ay(k-1) = bu(k-1) + e(k) + ce(k-1) \quad (3.19)
\]

where \( e(k) \) is a disturbance that is white noise with zero mean and variance \( \lambda^2 \). The parameter vector \( \theta \in \mathbb{R}^3 \) and is given by

\[
\theta = [a \ b \ c]^T. \quad (3.20)
\]

Assume that \( u(k) \) and \( e(l) \) are independent for \( k < l \). Then the model permits feedback from \( y(\cdot) \) to \( u(\cdot) \). The output at time \( k \) is given by

\[
y(k) = [-ay(k-1) + bu(k-1) + ce(k-1)] + [e(k)]. \quad (3.21)
\]
The two terms on the right side of (3.21) are independent, since \( e(k) \) is white noise. Let \( y^*(k) \) be an arbitrary prediction of \( y(k) \) based on data up to time \( k - 1 \), so that

\[
E[y(k) - y^*(k)]^2 = E[-ay(k - 1) + bu(k - 1) + ce(k - 1) - y^*(k)]^2 + \lambda^2 \geq \lambda^2
\]  

(3.22)

Thus, the lower bound on the prediction error variance is \( \lambda^2 \). The optimal predictor, \( \hat{y}(k|k-1;\theta) \) is one which minimizes this variance. At time \( k - 1 \), the output estimate at time \( k \) is given by

\[
\hat{y}(k|k-1;\theta) = -ay(k-1) + bu(k-1) + ce(k-1)
\]  

(3.23)

but this equation cannot be used directly since \( e(k - 1) \) is not a measurable quantity; however, it can be reconstructed from past data. Using (3.19) and successive substitutions into (3.23) back to time \( k = 0 \), we obtain

\[
\hat{y}(k|k-1;\theta) = \sum_{i=1}^{k-1} (c-a)(-c)^{i-1}y(k-i) - a(-c)^{-1}y(0)
\]

\[
+ b \sum_{i=1}^{k} (-c)^{i-1}u(k-i) - (-c)^k e(0) .
\]  

(3.24)

Assume that \( |c| < 1 \), which is the case for \( \theta \in \mathcal{D} \) from (3.2), so that the last term containing \( e(0) \) can be neglected for \( k \) sufficiently large. Implementation of (3.24) requires that all prior data is stored, which quickly becomes unrealistic for \( k \) large. To overcome this difficulty, a recursive formulation of (3.24) is used, given by

\[
\hat{y}(k|k-1;\theta) + cy(k-1|k-2;\theta) = (c-a)y(k-1) + bu(k-1)
\]  

(3.25)

or, written in terms of prediction error,

\[
e(k,\theta) + ce(k-1,\theta) = y(k) + ay(k-1) - bu(k-1) .
\]  

(3.26)
This equation illustrates the relationship between prediction error $e$ and disturbance $e$ by relating (3.26) to (3.19). The only difference is that $e(k)$ is replaced directly by $e(k, \theta)$. This is actually the method used for calculating the prediction error. In fact, Equation (3.25) can be written compactly in polynomial form using the fact that the prediction $\hat{y}(k)$ is simply the combination of the true value of $y(k)$ and a disturbance at time $k$, so that

$$\hat{y}(k|k - 1; \theta) = \frac{bq^{-1}}{1 + cq^{-1}} u(k) + \frac{(c-a)q^{-1}}{1 + cq^{-1}} y(k).$$

(3.27)

Consider now the following state-space model of a LTI discrete-time system.

$$x_{k+1} = \Phi x_k + \Gamma u_k + w_k, \quad w_k \sim N(0, Q)$$

$$y_k = H x_k + v_k, \quad v_k \sim N(0, R)$$

(3.28)

with initial conditions on the state and error covariance given by

$$\dot{x}_0 = E[x(0)] \quad \text{and} \quad P_0 = E[(x(0) - \dot{x}_0)(x(0) - \dot{x}_0)^T].$$

(3.29)

It is further assumed that $w$ and $v$ are uncorrelated.

We consider the optimal predictor for these systems by introducing the innovations model of the Kalman filter for linear systems (e.g. see [16]),

$$\dot{x}_{k+1} = \Phi \dot{x}_k + \Gamma u_k + K y_k$$

$$\hat{y}_k = H \dot{x}_k$$

(3.30)

where $K$ is the Kalman gain found from

$$K_k = P_k H^T [HP_k H^T + R]^{-1}$$

(3.31)

$$P_{k+1} = P_k - P_k H^T [HP_k H^T + R]^{-1} HP_k$$

(3.32)
The Kalman gain is chosen to minimize mean squared error, thus the choice for a cost functional is

\[ J_k = \text{trace}[P_k] \]  

so that the choice of \( K \) is equivalent to minimizing the length of the estimation error vector as seen by the definition of \( P \). The above mentioned equations for \( K_k \) can be found by taking the partial of (3.33) with respect to \( K \) and setting it equal to zero.

To continue with the previous illustration of an optimal predictor, again consider the first-order ARMAX model from (3.19) in state-space form

\[
\begin{align*}
    x(k+1) &= \begin{pmatrix} -a & 1 \\ 0 & 0 \end{pmatrix} x(k) + \begin{pmatrix} b \\ 0 \end{pmatrix} u(k) + \begin{pmatrix} 1 \\ c \end{pmatrix} e(k+1) \\
    y(k) &= \begin{pmatrix} 1 & 0 \end{pmatrix} x(k).
\end{align*}
\]  

The steady state solution to the Riccati equation yields a Kalman gain \( K \) of the form

\[ K = \begin{pmatrix} c - a \\ 0 \end{pmatrix} \]  

which results in the following state-space innovations form of the predictor

\[
\begin{align*}
    \hat{x}(k+1|k) &= \begin{pmatrix} -c & 1 \\ 0 & 0 \end{pmatrix} \hat{x}(k|k-1) + \begin{pmatrix} b \\ 0 \end{pmatrix} u(k) + \begin{pmatrix} c - a \\ 0 \end{pmatrix} y(k) \\
    \hat{y}(k|k-1) &= \begin{pmatrix} 1 & 0 \end{pmatrix} \hat{x}(k|k-1).
\end{align*}
\]  

After some manipulation, it follows that

\[
\hat{y}(k|k-1) = \frac{bq^{-1}}{1 + cq^{-1}} u(k) + \frac{(c - a)q^{-1}}{1 + cq^{-1}} y(k)
\]  

(3.37)
which is identical to the predictor of Equation (3.27) found previously. Furthermore, we can generalize by stating a result due to Rissanen and Barbosa [54] in which they showed that the ARMAX predictor that generates a linear least squares estimate of the output is equivalent to the state-space form of the Kalman filter. Having established the equivalence of the state-space methods with the ARMA-type input/output methods, we now turn our attention to the neural net implementations of the model structures and predictors.

### 3.2.3 Neural Net Model Structures

To construct neural net models of linear systems, the same procedure is used as for the conventional methods with a few notable exceptions. The procedure is (1) choose the model structure, (2) use input/output data recorded over a finite time interval to estimate model parameters using techniques from neural net training algorithms and Definition 2.2, and (3) validate the model using Definition 2.3. This procedure is illustrated using the ARMAX model in predictor form. The predictor form is most convenient for the neural net representation discussion since it is the manner in which they are used in the examples to follow.

Using the nth-order version of either (3.25) or (3.27), yields the ARMAX predictor

\[ \hat{y}(k|k-1; \theta) = B(q^{-1})u(k) + [1 - A(q^{-1})] y(k) + [C(q^{-1}) - 1] \varepsilon(k, \theta) \]

(3.38)

where \( \varepsilon \) defines the prediction error \( \varepsilon(k, \theta) = y(k) - \hat{y}(k|k-1; \theta) \). In compact form (3.38) becomes

\[ \hat{y}(k|k-1; \theta) = \theta^T \phi(k, \theta) \]

(3.39)
where $\theta$ is the parameter vector from (3.7) and $\phi(k, \theta)$ is a regression vector of past inputs, outputs and errors,

$$
\phi(k, \theta) = [y(k) \ y(k-1) \ ... \ y(k-n_a) \ u(k) \ u(k-1) \\
... \ u(k-n_b) \ e(k) \ e(k-1) \ ... \ e(k-n_c)]^T.
$$

(3.40)

The only change that is needed for a neural net implementation of the ARMAX predictor is in the parametrization of $\theta$. Now, Equation (3.38) becomes

$$
g(k|k-1; \theta_w) = \mathcal{Y}\{\phi, \theta_w\}
$$

(3.41)

where $\mathcal{Y}$ is an equation in the form of (2.6) and $\theta_w$ is a vector with elements $w_{ij}^p$, where $w$ is the weight connecting node $i$ in layer $p$ with node $j$ in layer $p+1$.

Using the FIR filter from adaptive signal processing applications, we can take a closer look at (3.41) to find a relationship between the neural net weights and the conventional parametrization. The conventional FIR filter is composed of former inputs $u$ and can be easily formed using a tapped delay line as shown in Figure 10. The equation describing this process is

$$
g(k) = B(q^{-1})u(k) = (b_1 q^{-1} + ... + b_n q^{-n_b})u(k)
$$

(3.42)

where the coefficients are a finite approximation of the impulse response. A neural net can be used to perform the same function as shown in Figure 11. The equation describing the neural FIR filter is

$$
g(k) = \beta f\{W_{23}[f(W_{12}[W_{01}\phi(k)])]\}
$$

(3.43)
where $f, \beta$ and $W_{ij}$ are described in the previous chapter. We now use an important result from [6] to establish a relationship between the two methods. Assuming that for a linear time-invariant system the neural net remains in the linear region of operation, the sigmoid nonlinearities can be replaced by linear functions to obtain

$$\hat{y}(k) = \hat{\beta} W_{32} W_{12} W_{01} \phi(k) = \hat{W}_{03} \phi(k).$$  \hspace{1cm} (3.44)

Thus, the neural net approximates the finite impulse response of a linear system, and

$$\hat{W}_{03} = [b_1 \ldots b_n]$$  \hspace{1cm} (3.45)

shows a relationship between neural net weights and the impulse response coefficients. More recently, it was found [63] that the approximation made for (3.44) was not necessary since for a linear system, expanding (3.43) analytically and forming a Volterra series, then the first order Volterra kernel is the neural net implementation of the FIR filter.

In the previous section, it was shown that if the states of a linear system are measurable, then the ARMAX and Kalman filter predictors are identical since they are both optimal mean square error estimators. To further establish this relationship for the neural net predictors, we can view the error function used in backpropagation as a loss function from which we can perform a minimization with respect to the weights of the neural net. In fact, when using the backpropagation algorithm to train the neural net, it was shown in Section 2.1.2 that for linear neurons, the backpropagation algorithm reduces to the LMS algorithm, which is the standard method for
Figure 10: A Conventional FIR Representation

Figure 11: A Neural FIR Filter
estimating the filter coefficients. In the actual nonlinear activation function case, we cannot establish optimality of the neural implementation since we cannot guarantee that a global minimum is reached. Therefore, we say that the neural net implementation yields a suboptimal predictor structure. However, if the neural net is trained to within bound epsilon (T.T.W.B.\(\varepsilon\)) as in Definition 2.2, and is generalizing well according to Definition 2.3, then the "degree of suboptimality" is often not too large and the neural net implementation almost always performs well. In fact, as we will investigate via example, the neural net has more degrees of freedom in terms of the number of weights to adjust, which could result in better performance when the system is underparametrized, which is the case when the system order is chosen too small.

### 3.3 Linear Model Structures: Comparative Performance Analysis

In this section, performance of the various modeling techniques is compared using several different examples. The conventional system identification was done using the Matlab System Identification Toolbox [40]. In each case a "best-effort" was made to perform the required model identification and parameter estimation. Similarly, for the neural net identification, several models were tried in each case using various topologies and training times. The figure of merit used to compare performance is error variance, where the error signal is the difference between the actual output data and the model output.

The backpropagation algorithm used for training for all of the simulations pre-
sented in this work is a version of the \textit{Neural Shell, V2.01} \cite{1} which is a program that was written first for use on Sun Computers and later optimized by Ahalt and his students for use on the Ohio Supercomputer Cray Y-MP8/864. Preliminary results were obtained using the Sun version on Sun SPARC Workstations. The results obtained from the Cray were found to be identical, and the Cray was used most often due to the fast turn-around time.

\subsection*{3.3.1 Input Data Requirements}

In Section 2.2, we discussed input data selection with respect to neural net static functional mappings. The discussion is continued here with the addition of time as a parameter in the input data. For the dynamic modeling applications of this chapter, the input data is a time sequence of sampled data. In this case, we are defining a dynamic mapping where again, the issue is to select enough data points so that the training data set fully describes the input space. When this is done correctly, we can ensure that the neural net interpolates well for data outside of the training data set. The notion of \textit{persistency of excitation} is used as a constraint to define a class of input functions. Roughly speaking, if the input is persistently exciting then all modes of the system will be excited during the identification process.

Consider the neural net predictor of Equation (3.41) with input in the form of (3.40). This is the general equation for the neural net predictor in ARMAX form where the system has inputs $u(k)$, outputs $y(k)$, and disturbance $e(k)$. Assume that the system to be identified can be written in the general form as in (3.1). Similarly, allow (3.41) to be written as a sum of two neural nets to decouple the stochastic dis-
turbance from the deterministic signals $u$ and $y$. Then the conventional and neural
net equations are written respectively as

$$\dot{y}(k) = G_1(q^{-1})u(k) + H_1(q^{-1})e(k) \quad (3.46)$$
$$\dot{y}(k) = \mathcal{Y}(q^{-1}, \theta_\omega)u(k) + H_2(q^{-1})e(k). \quad (3.47)$$

Now, suppose that $\varepsilon$ is a stationary process representing the prediction error and
that $E[\varepsilon(k)]^2 \equiv 0$ even though there is not necessarily an exact match between the
two models. Using the inverse Fourier transform, we have

$$E[\varepsilon(k)]^2 = \frac{1}{2\pi} \int_{-\pi}^{\pi} \Phi_\varepsilon(\omega) d\omega$$
$$= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left[ |G(e^{i\omega}) - \mathcal{Y}(\omega)|^2 \Phi_u(\omega) \right. \\
+ \left. |H_1(e^{i\omega}) - H_2(e^{i\omega})|^2 \Phi_\varepsilon(\omega) \right] d\omega \quad (3.48)$$

where $\Phi_u(\omega)$ and $\Phi_\varepsilon(\omega)$ are the power spectrums of the input and noise respectively.
Now since they are driven by the same noise process, the term on the right side of
the bracketed term is easily set to zero, so since $\Phi_\varepsilon(\omega) \neq 0$,

$$H_1(e^{i\omega}) - H_2(e^{i\omega}) = 0. \quad (3.49)$$

The term on the left side is

$$|G(e^{i\omega}) - \mathcal{Y}(\omega)|^2 \Phi_u(\omega). \quad (3.50)$$

In order for (3.48) to be zero then one of the two terms of (3.50) must be zero. This
allows one to place restrictions on the input signal $u(k)$ since if the quantity in (3.50)
implies that the models are equal then $|G(e^{i\omega}) - \mathcal{Y}(\omega)| = 0$ and the input data is
sufficiently informative.
From the above discussion we bring forth the concept of persistency of excitation.

**Definition 3.1** [41] A signal \( \{x(k)\} \) with spectrum \( \Phi_x(\omega) \) is said to be persistently exciting of order \( n \) if, for all filters of the form

\[
M_n(q) = m_1q^{-1} + \cdots + m_nq^{-n}
\]

the relation

\[
|M_n(e^{j\omega})|^2\Phi_x(\omega) \equiv 0
\]

implies that \( M_n(e^{j\omega}) \equiv 0 \).

This definition can have several interpretations. The function \( M_n(z)M_n(z^{-1}) \) can have at most \( n - 1 \) different zeros on the unit circle taking symmetry into account. A frequency domain interpretation that is convenient for our purposes is the following. We can say \( x(k) \) is persistently exciting of order \( n \) if \( \Phi_x(\omega) \) is different from zero at at least \( n \) points in the interval \(-\pi < \omega < \pi\). This follows directly from the definition. A detailed treatment of this definition can be found in [41, 16]. It is easily shown that a step input is persistently exciting of order one while a white noise input is persistently exciting of all orders.

For training a neural net, we wish to excite a system so that we have full knowledge of the system dynamics. Thus, for an \( n \)th-order system, we require that the input is persistently exciting of order \( n \). Furthermore, we require that the input signal amplitude reaches the permissible maximum and minimum values so that proper scaling parameters can be determined. Thus, we have a class of inputs for training with two requirements that must be met. If the system is trained using inputs that
meet the input class requirements and Definitions 2.2 and 2.3 hold, then we can almost always demonstrate a neural implementation that performs as an accurate representation of the system. The accuracy of the resulting mappings is typically $O(\varepsilon)$. This is demonstrated in the examples that follow.

### 3.3.2 FIR Filter Comparison

The first example is one which highlights some of the issues involved in the design of FIR filters and shows some of the benefits of neural implementations of these filters. The filter to be designed is a low pass filter. Our design choices are many, but for ease of implementation FIR filters are often chosen over IIR filters. However, one of the shortcomings of FIR filters is that the order of the filter is sometimes excessively high to achieve the desired response characteristics compared to the IIR case, and IIR filters can produce much sharper transitions in amplitude response with significantly fewer coefficients than the nonrecursive FIR filters. Furthermore, FIR filters often involve an iterative design while IIR filters can easily be specified. With these design considerations in mind, we propose a neural FIR filter design strategy that enjoys many of the advantageous characteristics of both high order FIR filters and IIR filters, while offering lower order and the ease of implementation of the conventional FIR filter.

This is the first example in which we examine the effect of underparametrization, which for this example means that the number of delays used is smaller than expected. It is shown that neural nets can perform well when underparametrized due to the internal representation of the data set in weight space. The conventional FIR filter
with \( N \) delays has \( N \) parameters to vary. A neural net FIR filter with the same number of delays can have many more free parameters which gives the designer more degrees of freedom when the weights are varied.

The design strategy is as follows. One can design either an IIR filter or a high order FIR filter with the specified characteristics, and use the filter to generate the training data set for the neural FIR design. The neural net FIR filter is designed with a reduced number of input delays and a desired topology, then trained with that data set. The neural net FIR is then easily implemented using a small number of delays and the static weights that result from training. A block diagram of the conventional and neural FIR filters is shown in Figures 10 and 11.

Consider a low pass filter with cutoff frequency at 100 Hz. In Figure 12, the frequency response of an FIR filter of order 30 is compared with a neural net FIR filter of order 10. Also included for comparison is an IIR filter of order 10. The neural net has the topology \( N_{10,20,10,1} \) and was trained with presentation of 50,000 epochs of a data set that is 256 records long. The neural net filter response is displayed with the dash-dotted line in Figure 12, where the performance is shown to lie somewhere between the IIR filter and the higher-order conventional FIR filter.

### 3.3.3 Second Order Model Example

We continue using a second order linear example, which can be considered as a model of a single mode of a flexible beam. The equations of motion are

\[
\dot{x} = \begin{bmatrix} 0 & 1 \\ -\omega_0^2 & -2\zeta\omega_0 \end{bmatrix} x + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u
\]  

(3.53)
Low Pass Filter Design Comparison

- Ideal
- IIR Design
- FIR Design
- Neural FIR

Figure 12: Frequency Response of Three Filter Types
\[ y = [1 \ 0] x \]  \hspace{1cm} (3.54)

or in transfer function form,

\[ G(s) = \frac{1}{s^2 + 2\zeta \omega_0 s + \omega_0^2} \]  \hspace{1cm} (3.55)

The output measurement equation is a sampled data system with output \( z_k \) corrupted by measurement noise \( v_k \) as shown in Figure 13.

![Figure 13: Second Order System Block Diagram](image)

In this example, we assume that the signal to noise ratio is relatively small (that is, the standard deviation of the noise is restricted to about 20% of the standard deviation of the output signal). The ARMAX model is compared with a neural net model for output estimation is the presence of noise. The results of trials for determining the neural net topology are not elaborated, but the final choice was a net with structure \( N_{6,12,8,1} \). The model was excited with a white noise signal and the output was recorded. The results of the identification are shown in Figures 14, where the top graph is a plot of the actual plant output, the ARMAX output and the neural net output; the bottom graph shows the plot of estimation error for the two modeling techniques. The error variances compare favorably to the variance of the injected noise, \( v_k \). This example shows that for a simple linear system, the ARMAX and neural net modeling techniques yield very similar results as expected.
Figure 14: Plots of ARMAX and Neural Net Models (top) and Error Plots of both Models (bottom)
3.3.4 The Random Walk

This example is a standard example used regularly in Kalman filter theory. The random walk is a linear time-invariant dynamical system and in this example it is used to demonstrate some basic ideas in system identification by comparing conventional techniques and neural net techniques. The random walk is also referred to as Brownian motion in statistics literature. Given a white noise process $w(t)$ as input, we wish to determine the output $x(t)$ by observing measurements $z_k$ at discrete time steps which are corrupted by noise $v_k$. It is assumed that $v$ is white, Gaussian and uncorrelated with $w$. The process is shown in Figure 15. The equations governing this process are

\[
\begin{align*}
\dot{x}(t) &= w(t), \quad w(t) \sim N(0, \sigma_w^2) \\
x_k &= x_k + v_k, \quad v_k \sim N(0, \sigma_v^2).
\end{align*}
\]

![Figure 15: The Random Walk Process](image)

A model based approach to output estimation uses the discrete-time Kalman filter. In this case, the discrete optimal filter is given by

\[
\hat{x}_{k+1} = \hat{x}_k + \frac{p_0/r}{1 + \frac{p_0}{T_k}} [z_k - \hat{x}_k]
\]

(3.57)
where $p_0$ is the initial value of the error covariance and $r = \sigma^2_e$, and it is assumed that model and the statistics of the noise is well known so that $q = 0$. Let the input $w(t)$ be a zero mean process with variance $\sigma^2_w$, and the measurement noise $v(t)$ has zero mean with variance $\sigma^2_v$, with $w, v$ uncorrelated. In Figure 16, it is shown that with $\sigma^2_w = 1$ and $\sigma^2_v = 0.1$, the filter does a good job of estimating the desired state. However, if an error is made in the model assumption, for example if we assume that we are trying to model a constant instead of a random walk, that is,

\[
\dot{x}(t) = 0 \\
z_k = x_k + v_k
\]

(3.58)

then the filter does a poor job of estimation and diverges rapidly. The output of a Kalman filter with incorrect model assumption is also shown in Figure 16. In addition, if the noise is not modeled properly, then divergence of the filter could occur. Perhaps divergence is not the correct word here since the filter is doing the best it can with the information supplied to it, so instead this is called a mismodeling effect.

This simple example shows the importance of making wise choices concerning the state-space model. Moreover, even though the Kalman filter has well known robustness properties, there are further assumptions that must be made regarding the noise characteristics. A recent study [58] shows how the Kalman filter performance can degrade when using improper noise assumptions. For these reasons, we move to an input/output approach for this example for determination of system models. Using the LMS algorithm for parameter estimation, the output due to several assumed model structures can be investigated.
Figure 16: Simulation of the Random Walk Process
The data used for both conventional model selection and neural net training was chosen using a Gaussian distribution for both \( w \) and \( v \), then the output was found via simulation of the random walk process and stored for use in training.

Model comparisons are made using a Kalman filter which is "perfectly tuned" for optimal performance as given above, an ARX filter representation, and the ARMAX model. The equations for these processes are given by

\[ y(k) = -ay(k - 1) + bu(k - 1) + ce(k - 1) + e(k) \quad \text{(ARMAX)} \]
\[ y(k) = -ay(k - 1) + bu(k - 1) + e(k) \quad \text{(ARX)} \quad (3.59) \]

The Kalman filter is perfectly tuned in the sense that the model is exact and the assumptions on noise are exact (Gaussian, known variance). These models are used to simulate the above system with the results shown in Figure 17 where plots of the output error are used for comparison, and the figure of merit used to determine the best "fit" is the output error variance. As expected, the optimal Kalman filter performs best as indicated by the variance of the error signal. The ARX and ARMAX perform well since they track the input without diverging, but the variance of the error signal is higher since they are given no a priori information about the characteristics of the noise. Several attempts were made at simulating the neural net in this example, with the "best" design chosen based on minimum output error.

The number of nodes in each hidden layer was varied with the training results displayed in Table 1. Each neural net was trained using 10,000 presentations of the training data. The "best" net is chosen for simulations and comparison to the conventional techniques based on output error variance. Although trial 1 appears to
be a good choice, it was shown to generalize poorly for inputs outside the training data set.

Table 1: Training Results for the Random Walk Neural Net

<table>
<thead>
<tr>
<th>Trial</th>
<th>Hidden Layer 1</th>
<th>Hidden Layer 2</th>
<th>Error Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>5</td>
<td>0.2366</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>10</td>
<td>0.2889</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>15</td>
<td>0.2957</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>0.2356</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>5</td>
<td>0.2348</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>10</td>
<td>0.2392</td>
</tr>
</tbody>
</table>

From Figure 17, it is shown that the neural design performs better than the ARX and ARMAX filters with respect to the variance of the output error. Perhaps the better performance is because the neural net learns an internal representation of both the system dynamics and the noise characteristics during training.

The results of Figure 17 are based on the data used for training the neural nets and for the parameter estimation routines in the ARX and ARMAX cases. The true test of a filter design is the cross-validation process, where data from outside the training set is used for comparison. The "best" neural net structure and the conventional models were used for the next set of simulations where the noise models were varied and new data is used to compare the three designs with the best filter chosen from minimum output error variance. In these examples, the Kalman filter was not "tuned" for the new statistics of the noise. It is obvious that the Kalman filter performs best under optimal conditions as expected, but its performance deteriorates as the filter moves away from optimal conditions. However, since the model is still
Figure 1.7: Error plots for several model structures of the Random Walk Process.
exact, the filter still performs well. The results are compiled in Table 2. The neural net and the ARMAX designs show similar performance which is supported by the fact that for linear systems, the LMS and BEP parameter adjustment algorithms are approximately equivalent.

Table 2: Cross-Validation Results for the Random Walk

<table>
<thead>
<tr>
<th>Model Type</th>
<th>Error Variance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kalman Filter</td>
<td>0.2186</td>
</tr>
<tr>
<td>Neural Net</td>
<td>0.2794</td>
</tr>
<tr>
<td>ARMAX</td>
<td>0.3129</td>
</tr>
<tr>
<td>ARX</td>
<td>0.3557</td>
</tr>
</tbody>
</table>

3.4 Nonlinear Model Structures

Unlike linear systems, there is no uniform approach to nonlinear system modeling and identification. This is due mainly to the inherent complexity of nonlinear systems and to the difficulty of deriving identification algorithms that can be applied to a reasonably large class of nonlinear systems. Approximate methods such as describing function approaches provide a means for modeling of systems with a known class of inputs. Further difficulties arise since it is often very difficult to quantify various sources of nonlinearities which complicates both the modeling problem and identification. These characteristics of nonlinear systems are also precisely the reason for difficulty in implementing a method for control such as feedback linearization [26], since a precise model is required for linearization.
3.4.1 Nonlinear Identification and Estimation Techniques

The nonlinear system identification problem using conventional methods is highly complex and can sometimes require excessive amounts of computations. Functional series methods such identification of Volterra kernels can be applied to a wide class of processes and work well for systems with mild nonlinearities, but are usually limited to second order expansions due to excessive computations. If a great deal of a priori information is known concerning the system structure and nonlinearities, then one can use parameter estimation methods which assume knowledge of the underlying differential equations or block-oriented methods in which a linear system is coupled with nonlinear blocks and analyzed. These methods, however, can be very restrictive in general and require many assumptions using physical insight as a guide.

On the other hand, neural networks offer an attractive alternative to conventional techniques, since the neural net predictor structures previously discussed for linear systems are equally applicable to nonlinear systems. One simply decides upon the inputs and outputs to be considered and the model structure (and subsequently the neural net topology) is fixed after the model order is determined. The key for implementation of a nonlinear predictor is to provide an adequate description of the reachable input and output spaces in the training data set using the results of Sections 2.2 and 3.3.1. These results ensure that the neural net remains in interpolation mode for the allowable class of inputs.

From the Stone-Weierstrass theorem [14], we know that a neural net can accurately describe any function, linear or nonlinear using only nonlinear blocks and sum-
mers. However, this theorem fails to provide a means for implementation using these nonlinear functions since an infinite number may be required. Thus, the problem of realizing a nonlinear function results in an approximation by limiting the number of nodes in the net to a reasonable finite number. This approximation is similar to the truncation made, for example on a Fourier series representation of a function. Similar to the Fourier methods, it is desired that the number of nodes remains small while ensuring that the mapping is adequate.

We now turn our attention to the Extended Kalman filter (EKF) which is a well known model-based technique for nonlinear minimum variance estimation. Given the following nonlinear system, with sampled nonlinear measurements,

\[
\dot{x}(t) = f(x(t), u(t), t) + w(t) \tag{3.60}
\]

\[
y_k = h_k(x(kT)) + v_k \tag{3.61}
\]

we wish to find the minimum variance estimator for \(x(t)\). As in the linear case, \(w(t) \sim N(0, Q)\) and \(v_k \sim N(0, R)\) are uncorrelated white noise processes with \(x(0) \sim N(x_0, P_0)\). The EKF is implemented by making approximations to \(f(\cdot)\) and \(h_k(\cdot)\) to obtain an algorithm that is computationally tractable for nonlinear systems. See [15, 37] for details.

There are many assumptions and approximations that must be done in order to design the EKF. First, it is assumed that the system dynamics equation \(f(\cdot)\) is exact and continuously differentiable so that we can expand it in a Taylor series about the current state estimate \(\hat{x}(t)\). For implementation, the Taylor series expansion of \(f\) is limited to a first order approximation. For the covariance update and measurement
equation, calculation of the Jacobian of both $f$ and $h$ is required. The equations that result are as follows:

\[ \dot{x} = f(\hat{x}, u, t) \quad (3.62) \]

\[ \dot{P} = F(\hat{x}, t)P + PF^T(\hat{x}, t) + Q \quad (3.63) \]

\[ K_k = P(kT)H^T(\hat{x})[H(\hat{x})P(kT)H^T(\hat{x}) + R]^\text{-1} \quad (3.64) \]

where the Jacobian in each case is calculated from

\[ F(x, t) = \frac{\partial f(x, u, t)}{\partial x} \quad (3.65) \]

\[ H(x) = \frac{\partial h(x, k)}{\partial x} \quad (3.66) \]

It is obvious that the EKF is computationally intensive. It works well if the model is well known for both the system dynamics and the measurement equation. There are other methods using higher order approximations that can also be used (see [27] or [15] for details), but will not be elaborated here. Because the error covariance matrix is calculated using an approximation, there is no guarantee that the estimate given by the EKF is indeed optimal. Performance is often verified via Monte Carlo simulation.

Due to the computational difficulties posed by the Kalman filter, especially in the nonlinear case, we propose use of the neural net Kalman filter. The EKF can take on many special cases. There can be linear system dynamics with nonlinear measurements, nonlinear dynamics with input added linearly, or the most general case is encompassed by Equations (3.60) and (3.61). This can be written in input/output
one-step ahead predictor form as a single nonlinear function $g(\cdot)$,

$$
\hat{y}_{k+1} = g(y_k, y_{k-1}, \cdots, u_k, u_{k-1}, \cdots, \hat{y}_k, \hat{y}_{k-1}, \cdots)
$$

(3.67)

where the regression of former inputs, outputs and errors is used to form the next output estimate. The nonlinear dynamics and/or measurements are all a part of $g$, and as such, a neural net is used to realize these nonlinear dynamics with the nonlinear function $g$ replaced by a neural network so that

$$
\mathcal{Y}(\phi, \theta_w) = g(\cdot)
$$

(3.68)

with $\phi$ and $\theta_w$ defined as in Equation (3.41).

As discussed in the previous chapter, the backpropagation algorithm is based on a cost functional that minimizes the mean square error of the output of the neural net over the training data set. While one cannot guarantee that the output estimate is optimal, it can be verified via Monte Carlo simulations, similar to the EKF. The main advantage of the neural net method is that there are no approximations needed such as a truncated Taylor series expansion needed for the nonlinearities of the system since the nonlinear system dynamics are embedded in the weighting functions of the neural net. This is especially appealing for systems whose nonlinear dynamics are not well known. An example follows which helps to illustrate some of the issues raised above.

### 3.4.2 Radar Tracking Example

The next example is a nonlinear system example used to compare the extended Kalman filter (EKF) with its neural net realization. This nonlinear example is for
A body falling vertically through the atmosphere has height $h$ above the earth's surface and velocity $\dot{h}$. In addition, the equations of motion use a ballistic coefficient $\beta$ to describe the motion. The equations of motion are

$$\frac{d\dot{h}}{dt} = D - g$$
$$\frac{d\beta}{dt} = w(t),$$

where $g = 9.8 \text{m/sec}^2$ is the acceleration due to gravity, $w(t) \sim (0, q)$ is white noise, and drag is given by

$$D = \frac{\rho \dot{h}^2}{2\beta}. \quad (3.71)$$

In Equation (3.71), $\rho$ is the atmospheric density given by

$$\rho = \rho_0 e^{-h/c} \quad (3.72)$$

with $\rho_0 = 1220 \text{g/m}^3$ the density at sea level and $c = 10263 \text{ m}$ a decay constant. Equation (3.72) is good to about 6000 m in the troposphere. A radar system is used to track the free-falling body, where range measurements are taken every second as shown in Figure 18. The measurement equation is

$$z_k = \sqrt{r_1^2 + (h_k - r_2)^2} + v_k \quad (3.73)$$

with $h_k \triangleq h(kT)$ and $v_k \sim (0, \sigma_r^2)$. 

radar system which tracks a body in free fall. This example was presented in [74], in which the authors presented a comparison of three nonlinear filters; the EKF, a second order EKF, and an iterated filter design. The numerical values for this example can also be found in [37] or [15].
The EKF and neural net nonlinear predictor are used to estimate the range $r$ from measurements $z_k$. Simulations were run using the above equations and the parameters of Table 3.

The neural net was trained using several trials, assuming noise free measurements. The neural net has topology $\mathcal{N}_{6,15,10,1}$ where the inputs to the neural net assume a model order $n = 3$, prior measurements and estimation error are used as inputs, resulting in an input vector of the form $[z_{k-1} \ z_{k-2} \ z_{k-3} \ \varepsilon_{k-1} \ \varepsilon_{k-2} \ \varepsilon_{k-3}]^T$ to estimate the range $\hat{r}_k$. 

<table>
<thead>
<tr>
<th>$r_1$</th>
<th>1000 m</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r_2$</td>
<td>500 m</td>
</tr>
<tr>
<td>$\sigma_r^2$</td>
<td>9 m²/Hz</td>
</tr>
<tr>
<td>$h(0)$</td>
<td>$\sim N(10000\text{m}, 500\text{m}^2)$</td>
</tr>
</tbody>
</table>

Figure 18: Tracking a Free-Falling Body
The plots of Figure 19 show the results of a Monte Carlo simulation where 100 trials were run. The top plot shows the error for a single Monte Carlo trial, and the bottom plot shows the RMS error over all trials. Since the model is exact, the conventional EKF performs well, especially for the initial part of the simulation, since the neural net is being run as a steady-state filter. However, in the event the process is not modeled well, the EKF performance degrades quickly, and the neural net predictor becomes a much more attractive method since the dynamics are learned from the input/output data.

3.5 Example using a Flexible Space Structure Testbed

The final example from this chapter is based on a flexible space structure experiment at the Jet Propulsion Lab. This example demonstrates the effect of underparametrization for both conventional and neural net models.

Flexible structures use finite element models to generate the dynamical equations describing their behavior. In this example, we show how a neural net can be used for modeling a subsystem of a flexible space structure. The structure under study is The JPL Flexible Ribbed Antenna Structure Testbed, used for validation of control and identification techniques. A detailed description of the physical structure is given in Appendix A, see also [67] for further details or [50] for details on controller design experiments. This example deals only with the boom-dish modes of the structure; those modes that are excited when the flexible boom moves in pendulous manner (see Figure 37).

A linear system model is used to describe the dynamical behavior, but nonlin-
earities exist if one considers the actuators as a part of the model. However, for these experiments, the motion is constrained to the linear range of operation, and we model the system using the first four most dominant modal frequencies. The subsystem model has the following linear state-space form:

\[
\dot{x}(t) = Ax(t) + Bu(t) \\
y(t) = Cx(t)
\]  

(3.74)
where $A$ has the form
\[
A = \begin{bmatrix} 0 & I \\ -\Omega^2 & -2\zeta\Omega \end{bmatrix},
\]
and $\Omega$ is a diagonal matrix with the modal frequencies as the diagonal elements.

For identification of the system model, we first excite the structure with a white noise input and determine the output, which is corrupted by sensor noise. An fourth-order ARMAX model and the equivalent neural net model were found with the results plotted in Figure 20. The neural net structure is of the form $N_{12,20,15,1}$. In this case, the neural net greatly outperforms the ARMAX model as can be seen from the error plots of Figure 20. At first look, this is a surprising result, since the ARMAX is known to be the best linear least squares estimator. However, since we are sampling a continuous time system, much of the error is due to discretization of the continuous time system. The neural net can better model the discretization error since it forms an "internal representation" of the data set to learn the system dynamics. Moreover, with the large number of hidden nodes, there are more parameters to set than in the ARMAX case which has a smaller number of coefficients as free parameters.

The benefits of neural net modeling can be further observed by looking at models that are underparametrized, that is, the filter order is less than the model order. We can compare the ARMAX and neural net models by using a second order model approximation for both cases, using a neural net with structure $N_{6,20,10,1}$. This effect is shown in Figure 21, with a plot of both the system output from the ARMAX and neural models, and the corresponding error plots when the input is a random noise input. The neural net shows fewer ill effects from underparametrization, since once
again it forms an internal representation of the system dynamics in weight space.

All four models were tested in the cross-validation process by exciting the flexible structure at the second modal frequency, and observing the resulting model outputs. In Figure 22, the four plots represent the results from the higher order ARMAX and neural models and the lower order ARMAX and neural models, respectively, where in each plot, the desired output is plotted along with the model output. Also included
Figure 21: Underparametrized ARMAX and Neural Net Model Outputs (top) and Error Plots (bottom)

in each case is the error variance, used as the figure of merit. It is obvious that the neural models perform much better than the ARMAX models, with few ill effects from underparametrization.
Figure 22: Plots of the Cross-Validation Process
3.6 Chapter Summary

In this chapter, it was shown that neural nets can be used in ARMA-form to model a physical system given input and output data. These models offer the control engineer the ability to use these neural models in place of conventional ARMA-type models as identified plants. Furthermore, the ability of neural nets to capture nonlinear dynamics allows one to use these models for control of nonlinear systems, thus giving the control engineer additional options with respect to control design and modeling of nonlinear systems.

Equivalences were drawn between state-space representations, ARMA models, and neural net ARMA models for linear systems. In addition, the neural net in ARMAX form is shown to give equivalent results when compared with the conventional formulations such as the Kalman filter and the Rissanen–Barbosa formulation for linear systems. For nonlinear systems the relationship between the Extended Kalman filter and the neural net equivalent filter were discussed. Several examples demonstrate the ability of neural nets to capture system dynamics for both linear and nonlinear systems.
CHAPTER IV

Neural Control of Flexible Systems with Partially Known Dynamics

In the previous chapter, neural networks were shown to model dynamic systems with a high degree of accuracy. For plants with well known dynamics, optimal control methods can be used to achieve the desired performance specifications. In this chapter, we investigate the use of neural nets for simultaneous modeling and control in an optimal control setting. A novel approach is given where neural nets are used in concert with conventional control methodologies. The class of plants considered in this chapter has only partially known dynamics, and a neural net-based model is used to estimate the unknown part of the dynamics. Similarly, for control, a unique control strategy is proposed using a two part control scheme: one part is model-based, using the known dynamics and conventional optimal control techniques, the second component of the control scheme is neural net-based. Furthermore, the neural net control scheme demonstrates by way of example, the ability of neural nets to adapt to continuously changing environments. One such example of a system with partially known dynamics is a flexible manipulator. Flexible link manipulators are of extreme interest to researchers interested in space-based robot applications and other robotic applications where one is faced with weight and power constraints. However, flexi-
bility leads to highly complex system models, resulting in more complication for the control designer. In this chapter, we show that for a one-link flexible manipulator a control design can be achieved that provides accurate slewing while minimizing vibration of the manipulator. It is further shown that the control design can adapt to changing environments presented by varying the manipulator payload, resulting in accurate slewing for a wide range of payload variations.

The chapter is organized as follows. Section 4.1 is used to motivate our new control strategy for control of the flexible robot system. In the section that follows, we divert our attention to a discussion of flexible manipulator modeling issues to better understand the control strategies implemented. In Section 4.3, the simulation model which is designed using the assumed modes method for distributed parameter systems is discussed. In addition, neural net modeling for the described class of systems is discussed simultaneously with the problem of controller design. Simulations are given in Section 4.4 for the flexible manipulator with fixed dynamics and a constant payload, then the problem is complicated by varying the payload, resulting in a system with variable dynamics. The ability of the controller to adapt to this disturbance is investigated via simulation. A brief summary completes this chapter.

4.1 Introduction

In this chapter, the use of neural nets for simultaneous modeling and control in an optimal control setting is investigated for systems with partially known dynamics. Specifically, we consider the case where the unknown portion is related to flexibility. Systems with a high degree of flexibility fall into the general class of distributed pa-
rameter systems whose motion is described by hybrid systems of integrodifferential equations. For the class of plants considered in this chapter, the known part of the dynamics is represented by a simple analytical model and a neural net-based model is used to estimate the unknown part of the dynamics. A novel approach is given where neural nets are used in concert with conventional control methodologies to achieve high performance for both trajectory tracking and vibration damping. The control strategy is a two-part control scheme: one part is model-based, using the known dynamics and conventional optimal control techniques, the second component of the control scheme is neural net-based. These strategies are applied to a one-link flexible manipulator. Flexible link manipulators are of extreme interest to researchers interested in space-based robot applications and other robotic applications with weight and power constraints. However, flexibility leads to highly complex system models, resulting in more complication for controller design. In this application, we show that for a one-link flexible manipulator, a control design can be achieved that provides accurate slewing while minimizing vibration of the manipulator. It is further shown that the control design can adapt to variations of the manipulator payload, resulting in accurate slewing and small tip vibrations for a wide range of payload variations.

There have been several examples of the use of neural nets for control. General control ideas were discussed in [38] for completely unknown nonlinear systems, and adaptive neural controllers were demonstrated in [35] for a simple linear plant. There have also been many examples of neural nets applied to robot control. In fact, in [36], there is an example of a neural net-based controller implemented on a real rigid-link
robot system, where a neural net is used for payload estimation and the payload estimate is used as a parameter in the feedforward controller. This scheme results in payload-invariant trajectory tracking. While there is a multitude of literature concerning rigid link robotic control using neural nets, there is relatively little concerning the control of flexible link manipulators.

Almost all real plants can be characterized as a system with partially known dynamics since one can never fully realize a real plant with a mathematical model. There exist a number of techniques available for control of linear systems with either unknown or partially known dynamics. Adaptive controllers [2] can be designed using a standard model structure with unknown parameters, but these systems are fraught with limitations. These types of systems assume a structured uncertainty, where the uncertainty is reduced using stability theory and parameter estimation techniques. However, systems of this type could become unstable due to excitation of unmodeled dynamics. Moreover, a plant is seldom completely linear, and there are few model structures that can accommodate a reasonably large class of nonlinear systems. For this application, we assume that the rigid dynamics are well known, but the nonlinear flexible dynamics are not known, and therefore must be learned from input/output data.

The control strategy proceeds as follows for slewing control of the one-link flexible robot. A model-based control is implemented based on the rigid dynamics of the system. The unmodeled flexible dynamics are learned using a neural network as a predictor, and a corrective control signal is synthesized using a second neural net for
control. This method was used by Iiguni, et al. [25] in which the authors presented a strategy for control of linear systems with a low degree of uncertainty and small additive nonlinearity using what was referred to as a nonlinear regulator. We extend this method to plants with a high degree of nonlinearity and apply it to the flexible robot problem.

4.2 General Aspects of Flexible Manipulator Models

Modeling of flexible robots is difficult due to the infinite dimensional nature of the distributed parameter systems and the difficulty in modeling the structural flexibility of the link. The dynamical equations of motion can be described mathematically using integro-partial differential equations (PDEs) with the appropriate boundary conditions. The methods known for modeling flexible robots are many and are addressed in a variety of works, using methods such as the Euler-Newton method with finite element models [59], the Euler-Lagrange method [8], or Hamilton's principle [11]. The model used for the simulations of this work is obtained from a distributed parameter model via the assumed modes method. The parameters are derived from the Ohio State University single-link flexible manipulator [4]. Experimental identification results obtained for this structure [31] demonstrate that only one mode is dominant in the identified model for transverse motion of the manipulator. Therefore, a one-mode expansion is used in the simulations.

In flexible robots, if the payload is allowed to be variable, then both the rigid dynamics and the flexible dynamics change as a function of payload. In addition, the boundary conditions assumed initially may not be valid as the payload varies. This
presents a difficult chore for the control designer as we shall see in the section that follows. Alternately, one can assume little or no knowledge of the plant and use robust control techniques and model the unknown dynamics as a disturbance. While this is a viable alternative, we have chosen to use model-based control methods that are between the two extremes, where we assume partial knowledge of system dynamics.

There has been much recent interest in the problem of slewing control for flexible manipulators. The resulting control strategies can be complex and difficult to implement. To ease the computational burden and implementation difficulties, we introduce the neural net-based composite control strategy described herein. Using a one-step-ahead prediction of the system outputs, we can form a control signal for predictive control. In addition, the estimation error in the identification neural net is used for training of the neural net for corrective control which is trained based on an unsupervised learning strategy.

Conventional system identification techniques have also been used to determine the model of a flexible link manipulator (see [64] for example). A potential problem with this approach, however is that a model structure must be assumed \textit{a priori}, and thus, modeling errors are built in for the infinite dimensional system being considered. Furthermore, most identification techniques are based on linear system models, introducing further difficulties for the highly nonlinear systems being considered. Further uncertainties exist due to the difficulties associated with modeling drive system mechanics, estimating physical parameters, in addition to the difficulty of accurately modeling the flexible dynamics. Our alternative approach offers many advantages
over conventional methods in this paradigm, since neural nets can be used to learn nonlinear system dynamics. The neural net not only learns the flexible dynamics, it also learns and thus compensates for modeling inaccuracies to ultimately obtain a more exact representation of the system dynamics.

4.3 Modeling and Control Strategy

In this section we discuss the simulation model in detail and address the problem of simultaneous neural net modeling and control applied to the flexible manipulator. The modeling problem is discussed first where a neural net is used to model the flexible dynamics of the plant, then the neural net output is combined with the rigid dynamics model output to form a one-step-ahead prediction of the plant output. The control signal to the plant is composed of two parts: a model-based control and a neural net-based control to correct for the unmodeled dynamics of the plant. It is assumed for this section, that the identification and control proceeds in a constant environment, that is, with a manipulator using a nominal payload at the tip. The problem of slewing control in a variable environment is investigated in the simulation section which follows.

4.3.1 Simulation Model for a Flexible One-Link Manipulator

Distributed parameter systems are characterized as having an infinite number of modes. For control applications, we describe the system using approximations that are finite-dimensional state-space representations. Starting with a Hamiltonian formula-
tion, we use an assumed modes representation of the flexure variable $\alpha$ for insertion into the system equations to obtain a finite-dimensional representation of the dynamics. The resulting equations are programmed on a computer to model the one-link flexible manipulator shown in Figure 23. In this case, the link is modeled as clamped at the hub and as a mass with an inertia is at the free end. The resulting mode shapes are referred to as CLTI mode shapes (cantilever with tip inertia). A detailed derivation of the equations can be found in [30] or in [60] for a two-link flexible manipulator. The terms used to obtain the dynamical equations of the manipulator are defined in Table 4.

<table>
<thead>
<tr>
<th>$I_h$</th>
<th>Hub inertia</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>Length of link</td>
</tr>
<tr>
<td>$EI$</td>
<td>Stiffness term</td>
</tr>
<tr>
<td>$\rho$</td>
<td>Mass density of link</td>
</tr>
<tr>
<td>$M$</td>
<td>Mass of hub</td>
</tr>
<tr>
<td>$\ell$</td>
<td>Spatial variable for link</td>
</tr>
<tr>
<td>$\alpha(\ell,t)$</td>
<td>Flexure of link at location $\ell$</td>
</tr>
<tr>
<td>$\theta$</td>
<td>rigid link angle of link</td>
</tr>
<tr>
<td>$u$</td>
<td>Input torque at hub</td>
</tr>
</tbody>
</table>

The equations for the flexible link manipulator can be written as

$$EI\alpha_{uu} + \rho\alpha_{u\ell} + \rho\ell\ddot{\theta} = \rho\alpha\dot{\theta}^2, \quad \ell \in [0,L], \quad t > 0$$  \hspace{1cm} (4.1)

$$[I_h + \frac{1}{3}\rho L^3 + \rho \int_0^L \alpha^2 d\ell] \ddot{\theta} + \rho \int_0^L \ell \alpha_{u\ell} d\ell = u$$  \hspace{1cm} (4.2)

The assumed modes method requires that the flexure be expanded as

$$\alpha = \sum_{j=1}^{N} \phi_j(\ell)q_j$$  \hspace{1cm} (4.3)
where \( j \) is the mode number, \( \phi \) is the mode shape, and \( q \) is the modal displacement.

The CLTI mode shapes have been used previously by [49] to model a link with a payload, and have also been used in [60] to model the first link of a two-link system as a link with an attached link as the payload. The boundary conditions are

\[
\alpha(0,t) = 0, \quad \alpha_t(0,t) = 0 \quad (4.4)
\]

\[
EI\alpha_{tt}(L,t) = -(M_pO_p^2 + I_p)\ddot{\alpha}(L,t) - M_pO_p\ddot{\alpha}(L,t) \quad (4.5)
\]

\[
EI\alpha_{ttt}(L,t) = M_p\dddot{\alpha}(L,t) + M_pO_p\dddot{\alpha}(L,t) \quad (4.6)
\]

where \( M_p \) is the mass of the payload, \( I_p \) is the mass moment of inertia of the payload, and \( O_p \) is the distance from the endpoint of the link to the center of mass of the payload. The first boundary condition represents the clamped boundary condition at the hub. The next two boundary conditions correspond to the mass plus inertia at the end of the link. In this paper, the above equations are simplified since the payload is modeled as a point mass at the tip, so that \( I_p \) and \( O_p \) are equal to zero. The CLTI mode shape equations can be found in [60].
The resulting model is of the form

\[ \mathcal{M}(X) \ddot{X} + KX + F(X, \dot{X}) = U \] (4.7)

where \( X = [\theta \ q_1 \ q_2 \ \cdots \ q_N]^T \), and \( N \) is the number of modes retained in the model from flexural effects of the link. The following expressions can be defined in terms of the variables of the above approximation:

\[ A_j \triangleq \int_0^L \rho \phi_j(\ell) d\ell \] (4.8)
\[ B_j \triangleq \int_0^L \rho \phi_j^2(\ell) d\ell \] (4.9)
\[ C_j \triangleq \phi(L) \] (4.10)
\[ \frac{\partial^4 \phi_j}{\partial \ell^4} \triangleq \lambda_j \phi_j . \] (4.11)

This allows for the calculation of the terms

\[ \int_0^L \alpha d\ell, \ \int_0^L \ell \alpha d\ell, \ \text{and} \ \int_0^L \alpha^2 d\ell . \] (4.12)

Substitution of (4.3) into (4.1) and (4.2) results in an ordinary differential equation for the rigid body motion motion and a partial differential equation for the flexure equation. To convert the flexure equation into an ordinary differential equation, we multiply through by \( \phi_k \) and integrate from 0 to \( L \). Since the \( \phi_k \)'s are orthogonal (the integral from 0 to \( L \) is an inner product, i.e., the \( \phi_k \)'s belong to the space \( \mathcal{L}_2[0, L] \)), all the terms drop out except the \( \phi_k \phi_k \) integral, so that \( N \) differential equations remain.

The resulting rigid body motion (\( \theta \)) and flexure (\( \alpha \)) equations are respectively:

\[ [I_k + \frac{1}{3} \rho L^3 + \sum_{j=1}^N B_j q_j^2] \ddot{\theta} + \sum_{j=1}^N A_j \ddot{q}_j = u \] (4.13)
\[ A_k \ddot{\theta} + B_k \ddot{q}_k + EI \frac{\lambda_k}{\rho} B_k q_k - B_k q_k \ddot{\theta}^2 = 0, \ k = 1, 2, \ldots, N \] (4.14)
Table 5: Physical Parameters of the OSU One-Link Manipulator

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>$6.8944 \times 10^{10}$ N/m$^2$</td>
</tr>
<tr>
<td>I</td>
<td>$3.3339 \times 10^{-11}$ m$^4$</td>
</tr>
<tr>
<td>A</td>
<td>$1.5875 \times 10^{-4}$ m$^2$</td>
</tr>
<tr>
<td>I_h</td>
<td>$1.6640 \times 10^{-5}$ kg-m$^2$</td>
</tr>
<tr>
<td>L</td>
<td>1.0 m</td>
</tr>
<tr>
<td>$\rho$</td>
<td>0.4847 kg/m</td>
</tr>
<tr>
<td>M</td>
<td>0.4847 kg</td>
</tr>
</tbody>
</table>

For the simulations, we let $N = 1$ and use the parameters of Table 5.

4.3.2 Modeling and Identification

To predict the flexible manipulator outputs, we use our knowledge of the rigid dynamics to form a model of the known part of the plant and use a neural net to learn the unmodeled flexible dynamics. Let $x^p$ be the output of the plant which we wish to estimate, $x^r$ the output of the known system, and $x^f$ be the output of the neural net, which when added to $x^r$ yields a prediction of the plant output. Thus for the one-step-ahead predictor, we have

$$\hat{x}^p_{k+1} = x^r_{k+1} + \hat{x}^f_{k+1}$$  \hspace{1cm} (4.15)

where $\hat{x}^p_{k+1}$ is the prediction of the plant output and $\hat{x}^f_{k+1}$ is the predicted state of the unknown dynamics at time $k + 1$. The known dynamics are a nonlinear function of prior states and inputs described by the function $f$,

$$x^r_{k+1} = f(x^r, u^p)$$  \hspace{1cm} (4.16)
and the unknown dynamics are a nonlinear function of the prior plant outputs, rigid
dynamic model outputs, and prior inputs, described by the function $g$,

$$\hat{x}_{k+1} = g(x^p, x^r, u^p)$$

(4.17)

where $u^p_k$ is the input to the plant at time $k$. Thus, we assume that the unknown
dynamics can be driven by the states of the known dynamics, but not vice-versa. We
impose the following system constraints in the form of assumptions on the structure
of the plant and the models.

**Assumption 4.1** *The plant to be controlled is completely stabilizable and detectable.*

**Assumption 4.2** *The dynamics given by $f(\cdot)$ is stabilizable and describes the plant
to a sufficient degree such that the controller designed to stabilize the known dynamics
also stabilizes the plant.*

**Assumption 4.3** *The unknown dynamics given by $g(\cdot)$ is stable.*

Assumption 4.1 gives us a starting point for the plants to be considered. Al­
though Assumption 4.2 is not a necessary condition for identification, it is provided
since identification proceeds much better with a stable plant. (In [68], an unstable
inverted pendulum was identified using a "human-in-the-loop" to provide stability
when necessary.) Assumption 4.3 is given since if the unmodeled dynamics are un­
stable, then plant states may blow up before an appropriate model can be found and
the subsequent controller designed.

The identification stage is a neural net training process using a supervised training
rule as shown in Figure 24. In this stage, the neural identifier produces the signal $\hat{x}^f$
representing the unmodeled flexible dynamics and uncertainties to produce a signal which "corrects" the output from the known dynamics to subsequently provide an accurate estimate of the plant outputs.

![Diagram](image)

Figure 24: Identification Stage Using Neural Nets

The predicted signal (4.15) is compared with the plant output to form the error signal used to train the neural net identifier. The neural net identifier receives as input both the reference signal for control and a regression of the output of the plant. It uses these signals to produce an output \( \hat{x}_{k+1} \), which is trained based on minimization of the error from the supervised training algorithm. The error function is

\[
E_{k+1} = ||x_{k+1}^p - \hat{x}_{k+1}^p||^2
\]  

(4.18)

where \( ||\cdot|| \) is the standard \( L_2 \) norm. The backpropagation algorithm [56] adjusts the weights at each time \( k \) by the update rule

\[
w_{i,j}(k + 1) = w_{i,j}(k) + \eta \frac{\partial E_{k+1}}{\partial w_{i,j}}
\]  

(4.19)
where \( \eta \) is the learning rate and \( w_{ij}^m \) is the weight connecting node \( i \) in layer \( m \) with node \( j \) of layer \( m+1 \). Differentiation of (4.18) yields

\[
\frac{\partial E_{k+1}}{\partial w_{ij}^m} = -(x_{k+1}^p - \hat{x}_{k+1}^p)^T \frac{\partial \delta_{k+1}^p}{\partial w_{ij}^m}
\]

\[= -(x_{k+1}^p - \hat{x}_{k+1}^p)^T \frac{\partial g(x_p^c, x^r, u^p)}{\partial w_{ij}^m} \tag{4.20}\]

To calculate \( \frac{\partial g(x_p^c, x^r, u^p)}{\partial w_{ij}^m} \), the backpropagation algorithm is used since this quantity is calculated during training. It is equal to the quantity \( \delta_j z_i \), where

\[
\delta_j = \begin{cases} 
  x_j^p(1 - x_j^p)(x_j^p - x_j^p) & \text{(output nodes)} \\
  z_j(1 - z_j) \sum_k \delta_kw_{kj} & \text{(hidden nodes)}
\end{cases}
\tag{4.21}
\]

where \( z_j \) is the output of a hidden node.

### 4.3.3 Controller Design

With the plant identified, we proceed to controller design. In Figure 25, the overall system architecture is given in terms of a control block and an identification block. It is seen that the control signal is composed of two parts: one part is based on conventional model-based control, so that any conventional controller design can be chosen for this part. The output of the conventional controller is given by \( u^r \). The second component comprising the control signal is the contribution due to the neural network for control, indicated by \( u^f \). Both \( u^f \) and \( u^r \) are formed using the error signal as input, calculated as the difference between the desired trajectory at time \( k+1 \) and the projected next state at time \( k+1 \). This is a predictive control scheme where it is desired to know at time \( k \) what the output is likely to be at time \( k+1 \) in order to take appropriate control action at time \( k \). The two components of the control are
combined for the composite plant input,

\[ u_k^p = u_k^r + u_k^l. \]  

(4.22)

Since the signal \( u_k^l \) is not known \textit{a priori}, it is generated using an unsupervised training rule for the neural net. Unsupervised learning rules exploit some key features of the backpropagation algorithm since backpropagation calculates partial derivatives during the training process. The output of the neural net under unsupervised training is found by adjusting the weights to minimize a performance function.

\[ \text{Figure 25: Identification and Control Using Neural Nets} \]

In the proposed configuration, the control strategy is derived by a model-based predictive controller where the model used for controller design is the rigid dynamics of the manipulator. The estimated next state is subtracted from the desired state to produce a prediction error which drives the system to the next state. A close look at the rigid dynamics equations reveals that in the absence of a gravity term, the system
is linear. Therefore, without loss of generality, we will assume that the gravity term is zero (which is the case for space-based robots). This is a valid assumption since we can always add a feedforward controller to perform gravity compensation separately. Now since the known dynamics is a linear system, we can use any conventional linear control technique to achieve the desired controller design. We will choose the linear quadratic regulator (LQR) and proceed with a brief discussion of the LQR.

Consider a linear time-invariant system described by

\[ x_{k+1} = Ax_k + Bu_k \]  \hspace{1cm} (4.23)

where \( x \in \mathbb{R}^n \) is the state, \( u \in \mathbb{R}^m \) is the control and \( A, B \) are known matrices. A linear control law of the form

\[ u = -Kx \]  \hspace{1cm} (4.24)

is sought. Instead of choosing \( K \) to achieve some prespecified closed loop poles, we choose \( K \) to minimize a cost function \( J \), which is quadratic in both state and input:

\[ J = \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k \]  \hspace{1cm} (4.25)

with \( Q, R \) symmetric. If we wish to follow a prespecified state trajectory \( x^d \), then Equation (4.25) becomes

\[ J = \sum_{k=0}^{\infty} (x_k - x_k^d)^T Q (x_k - x_k^d) + u_k^T R u_k. \]  \hspace{1cm} (4.26)

Without \( x^d \), the control objective is to bring the state to the origin.

The optimal control is given by minimizing (4.25) to obtain

\[ u_k = -Kx_k \]

\[ = - (R + B^T PB)^{-1} B^T P A x_k \]  \hspace{1cm} (4.27)
where $P$ is the steady state solution of the algebraic Riccati equation:

$$P = A^T PA - A^T PB(R + B^T PB)^{-1} B^T PA + Q \quad (4.28)$$

The unique solution to (4.28) exists and leads to a stable closed loop system if and only if $(A, B)$ is stabilizable (provided by Assumption 4.2) and $(Q^{1/2}, A)$ is detectable (see [3] for LQR details).

Although the LQR has well known robustness properties, one cannot expect it to work well when there is unmodeled dynamics not included in the mathematical description of the plant. Thus a corrective control, provided by the neural network is needed to improve the tracking of the system. A similar methodology was pursued in [25], in which the authors presented a strategy for control of linear systems with a low degree of uncertainty and small additive nonlinearity using what was referred to as a nonlinear regulator.

The problem is to find the corrective control signal that is used to compensate for system flexibility and other unmodeled dynamics. Since there is no signal for which comparison is valid, we use a neural net with an unsupervised training algorithm as in [29, 68]. Referring to Figure 25, it is obvious that when the linear system perfectly describes the plant, the control signal $u^r$ is the optimal control. Thus, we desire an error function for training that when minimized forces the neural net output to be zero when the linear model is an exact representation of the plant. A candidate error function is given by

$$E_c = (x_{k+1}^p - x_{k+1}^d)^T P(x_{k+1}^p - x_{k+1}^d) + u_k^p R u_k^p \quad (4.29)$$
since it is easily shown that the control produced by (4.27) minimizes this function at each time $k$. However, at time $k$, the plant state $x_{k+1}^p$ is not available, so we must use $\hat{x}_{k+1}^p$ which results in the following error function for training the neural net:

$$E_c = (\hat{x}_{k+1}^p - x_{k+1}^d)^T P (\hat{x}_{k+1}^p - x_{k+1}^d) + u_k^p R u_k^p. \tag{4.30}$$

This function is used to adjust the weights of the neural net in the following way. The weights of the neural net for control are indicated by $q$ to distinguish from the weights $w$ in the neural identifier and use the following update rule:

$$q_{i,j}^m(k+1) = q_{i,j}^m(k) + \eta \frac{\partial E_c}{\partial q_{i,j}^m} \tag{4.31}$$

To compute $\partial E_c/\partial q_{i,j}^m$ use Equations (4.15) and (4.22) to get

$$\frac{\partial E_c}{\partial q_{i,j}^m} = \hat{x}_{k+1}^p P \frac{\partial \hat{x}_{k+1}^p}{\partial q_{i,j}^m} + u_k^p R \frac{\partial u_k^p}{\partial q_{i,j}^m}$$

$$= \left[ \hat{x}_{k+1}^p P \left( B + \frac{\partial g(x^p, x^r, u^p)}{\partial u_k^p} \right) + u_k^p R \right] \frac{\partial h(x_k^p)}{\partial q_{i,j}^m} \tag{4.32}$$

where $g(\cdot)$ is the mapping performed by (4.17) in the neural identification stage, and $h(\cdot)$ represents the mapping performed by the control neural net. The quantity $\partial h(x_k^p)/\partial q_{i,j}^m$ is calculated from the backpropagation algorithm similar to the calculation for $\partial g(x^p, x^r, u^p)/\partial w_{i,j}^m$ since all of the necessary quantities are involved in the backpropagation training process. The quantity $\partial g(x^p, x^r, u^p)/\partial u_k^p$ is calculated from the neural network used for identification and from the weights $w_{i,j}^m$ in that network.

All of these quantities are easily extracted from the backpropagation training rule.
4.4 Neural Control Examples on a One-Link Flexible Robot

In this section, several examples are given to demonstrate the effectiveness of the proposed strategy. We first present both identification and control results for a system that operates in a constant environment using a nominal value for a payload added to the tip equal to 0.20 kg, which is about 41% of the link weight. Secondly, we present results for which we vary the payload over the range 0 to 70% of the link mass. Payload invariant slewing control is demonstrated and some of the difficulties encountered in implementation are discussed in detail.

4.4.1 Constant Environment

We begin with the identification stage for the system of Figure 24. The first step in the identification process is to collect data that can be used to train the neural network. The simulation model previously described is used in parallel with a model of the rigid dynamics, and the signals $x^p$ and $x^r$ are stored using several different inputs at $u^p$. The signals indicated by $x$ are vectors with elements of hub angle and hub angular velocity, so that $x \triangleq [\theta \ \dot{\theta}]^T$. In order to limit the amount of data for the neural network, the motion of the manipulator is limited to slews in the first quadrant, so $0 \leq \theta \leq 90$ degrees. Initially, a white noise input was used as an input to the system and data was collected and applied to the neural network for training. Then, the loop was closed using state feedback and data was collect for a variety of slews in the allowable range of inputs. This data was applied to the neural net which was previously trained using white noise inputs. Thus, the first stage of the
identification process can be thought of as a "coarse" training process, and the second stage can be thought of as a "fine" tuning process, where the neural net weights were adapted to learn the dynamics for slewing maneuvers.

For this application, using a model structure for the neural net of order two, we chose a neural net with five inputs and two outputs. Two hidden layers with 15 and 10 nodes respectively are used, and the notation $N_{5,15,10,2}$ is used for a shorthand means of stating the neural net with topology. The backpropagation algorithm used for training in the identification stage is a version of the *Neural Shell, V2.01* [1] which is a program that was written first for use on Sun Computers and later optimized by Ahalt and his students for use on the Ohio Supercomputer Cray Y-MP8/864. The results of the identification process are shown in Figure 26 for the neural net outputs of position and velocity. This plot is validation process where a 45 degree slew is commanded and the outputs of the plant and the prediction of the plant outputs are recorded. Also shown is the position without the neural net contribution. The benefit is obvious, as the variance of the estimation error is very small for both position and velocity ($\sigma^2 = [0.00196 \ 0.00854]$).

The control objective is to slew the arm along a desired trajectory as indicated by $x^d$, while minimizing arm vibrations. The model based controller in the block of Figure 25 is LQR control using hub angle state feedback. The weighting functions were chosen to put more emphasis on the output error term and less on control expenditure. However, we must limit the size of the feedback gains since large gains cause more flexure of the manipulator resulting in more vibrations of the tip. The
Figure 26: Identification Results for System with Nominal Payload

results for a slew from 0 degrees to 45 degrees are shown in the plots of Figure 27. The dashed line represents the manipulator *tip* position and the solid line is the hub angle position, both measured in degrees. The top plot is the output of the system with LQR state feedback control only, and the middle plot is the output of the system with the addition of neural control. The control signal follows in the bottom plot. The neural net achieves the desired effect, since an increase in damping reduces tip vibrations significantly. Note that the fundamental modal frequency changes to reflect the increase in damping of the flexible modes.
Figure 27: Flexible One-Link Manipulator: Position with LQR Control (top), Position with LQR and Neural Net Control (middle), and Control Signal (bottom)
4.4.2 Payload Variation

We now turn our attention to the problem where a payload at the tip of the manipulator is permitted to vary. The effect of adding a mass at the tip is that both the rigid and flexible dynamics change as a function of payload. Since the controller is designed with a fixed gain for the rigid dynamics, the control is no longer valid if the dynamics change. Again, to compensate for this effect, a neural net is added to the control loop, but now it is asked to perform more of the control effort. In addition, since the dynamics of the rigid body system has changed, the neural identifier must also be able to compensate for this variability.

Consider again the identification neural network, which performs the mapping \( g(\cdot) \) to compensate for the unmodeled flexible dynamics as

\[
\dot{x}_{i+1}^f = g(x^p, x^r, u^p)
\]  

(4.33)

where the output of the neural net, \( \hat{x}^f \), is a function of the plant output \( x^p \), the rigid dynamics \( x^r \) and the plant input, \( u^p \). It can be observed both analytically from the dynamical equations of motion and experimentally that by varying the tip mass, one varies the fundamental frequency of the flexible dynamics. Simulations were performed to determine the modal frequencies for various payloads. The results are displayed in Table 6.

The table shows that as the tip mass increases, the modal frequency of the manipulator decreases. The neural identifier in the constant payload case was trained to learn the dynamics for a single frequency. The variable payload case is much more challenging since the identifier must learn a continuously variable modal fre-
Table 6: Modal Frequency as a Function of Payload

<table>
<thead>
<tr>
<th>Payload Mass (kg)</th>
<th>0 kg</th>
<th>0.05 kg</th>
<th>0.10 kg</th>
<th>0.15 kg</th>
<th>0.20 kg</th>
<th>0.25 kg</th>
<th>0.30 kg</th>
<th>0.35 kg</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of Link Mass</td>
<td>0%</td>
<td>10.3%</td>
<td>21.6%</td>
<td>31.0%</td>
<td>41.3%</td>
<td>51.6%</td>
<td>61.9%</td>
<td>72.2%</td>
</tr>
<tr>
<td>Frequency (Hz)</td>
<td>1.3481</td>
<td>1.1323</td>
<td>0.9908</td>
<td>0.8968</td>
<td>0.8188</td>
<td>0.7592</td>
<td>0.7140</td>
<td>0.6705</td>
</tr>
</tbody>
</table>

frequency over the range defined in Table 6 which results in an identifier that effectively performs a “modal selection process”. A critical condition on the mapping process is that the mapping must be well conditioned. A well conditioned mapping is one where the mapping from input to output defines a unique one-to-one mapping. For example, if the manipulator output and the rigid dynamics output error exhibits a similar response to both a 0.10 kg payload and a 0.30 kg payload, then the neural net will not be able to discern the difference for the corrective response action since there will be ambiguities in the mapping. Fortunately this is not the case, since not only the modal frequency changes as a function of payload but also the error between the rigid dynamics output and the actual plant output varies in direct proportion to payload variation.

To achieve accurate identification, the neural net training data is selected using data from a variety of payload cases, then the neural net can interpolate the results to output the desired response. A new configuration is implemented for the identification stage to give the neural net more information, specifically with respect to the modal frequency. An accelerometer is included as an additional measurement at the tip of the manipulator and used as input to the neural net. For first bending mode vibrations, placement at the tip of the manipulator gives maximal output response [31]. Thus,
the neural identifier is changed to accept another input as shown in Figure 28, where
the accelerometer input at time $k$ is denoted by $\ddot{q}_k$.

Using experimental data from each of the payload cases of Table 6, the identification neural net was trained to learn the new flexible dynamics as a function of payload. The neural topology was changed to reflect the new information required for identification. There are four variables used for input to the identifier and assuming a model structure of order two, a delayed version of each input is also used to give the neural net a total of eight inputs. Since the neural net has more information to learn, a larger topology is used, with two hidden layers of size 30 and 20, respectively. The number of outputs remains at two, so that the neural net topology is $N_{8,30,20,2}$. The result of identification for a payload that is 23% of the link mass is identical to the results shown in Figure 26 where the estimation error variances in this case are $\sigma = [0.00375 \ 0.00986]$ for position and velocity respectively. Similar results were verified for several payload variations.
The neural net provides an excellent means of identification for both position and velocity. Note that the 23% value use for the payload is not part of the training data set, so the neural net performs the required interpolation with a high degree of accuracy. Thus, since the neural net can perform the required identification, the neural net controller is designed in the same manner as previously discussed. However, since the LQR feedback is designed for a plant with nominal payload, the neural net is asked to perform more of the task for control which results in slightly longer training time to achieve the desired tracking. This can be illustrated by way of example.

Consider the following experiment. The flexible link moves from a rest position to a desired position with the nominal value of 0.20 kg payload, and the neural net for control adjusts the weights in the control loop to achieve the desired tracking as in the previous example. In the new position, a mass is added to the link tip, and the link is then slewed to a new position during which time the neural net collects the data over the latest motion to adjust the weights to compensate for the new load. The manipulator does not perform well over the second slew, but it adjusts the weights such that during subsequent motion with the same payload, performance continues to improve until a desired performance specification is met.

This procedure is depicted in Figure 29. The manipulator starts at rest where it is assumed that training has been completed for the nominal case. The manipulator moves through a 45 degree slew and the tip mass is changed to 0.30 kg (61.9% of link mass). The manipulator moves in the same direction by 25 degrees to a rest position of 70 degrees. Although the manipulator did not dampen vibrations well,
subsequent maneuvers show that the control neural net is adjusting to compensate for the disturbance. The manipulator is moved back to 45 degrees and back again to 70 degrees to show how performance continues to improve. The control neural net trains itself on-line and improves its vibration damping characteristics with each slew. This can be seen when the manipulator moves through a slew of $-45$ degrees to the final position of 25 degrees. It was also found that if the mass is later changed back to the nominal value, the training time for the procedure is reduced slightly, but not enough to say that the neural net "remembers" the control for the original case, since the neural net adjusts itself to compensate for the most recent payload.

This plot shows that the neural net can achieve payload-invariant tracking with on-line training and a minimal amount of information known about the system dynamics or the payload. Real-time implementations of such an approach are dependent upon how much the payload changes from the nominal value. If there is a small change in the payload, then the neural net can easily adjust itself in one slew. If the payload size doubles, then several slews may be necessary to achieve the desired tracking specifications. To determine the ability of this control scheme to adjust to real situations, we investigated several different payload variations for a slew of 0 to 45 degrees (Trial 1), then back to 0 degrees (Trial 2). The maneuver was held constant at 45 degree slews and the payload was varied in order to make valid comparisons of training time. The results are shown in Table 7, where a $+20\%$ variation of the payload means that the payload changed from 0.20 kg to 0.24 kg. The first column is the percent variation of the payload, with the payload varied over the entire range of Table 6. The figure
The sum-squared value of the tip deflection is given by

$$J = \sum_{k=1}^{N} \alpha(L, k)^2.$$  \hspace{1cm} (4.34)\]

In Table 7 the value of $J$ is given for a slew from 0 to 45 degrees (Trial 1), then for a slew back to 0 degrees (Trial 2) in columns 2 and 3, respectively. Note that for large deviations of the payload in the positive direction, more than two slewing maneuvers might be necessary to control vibrations at the tip.
Table 7: Effect of Payload Variation on Training Time

<table>
<thead>
<tr>
<th>Payload (% change)</th>
<th>Trial 1</th>
<th>Trial 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0.0299</td>
<td>0.0285</td>
</tr>
<tr>
<td>+10%</td>
<td>0.0388</td>
<td>0.0351</td>
</tr>
<tr>
<td>-10%</td>
<td>0.0342</td>
<td>0.0325</td>
</tr>
<tr>
<td>+20%</td>
<td>0.0513</td>
<td>0.0378</td>
</tr>
<tr>
<td>-20%</td>
<td>0.0421</td>
<td>0.0352</td>
</tr>
<tr>
<td>+30%</td>
<td>0.1100</td>
<td>0.0391</td>
</tr>
<tr>
<td>-30%</td>
<td>0.0805</td>
<td>0.0380</td>
</tr>
<tr>
<td>+50%</td>
<td>0.4733</td>
<td>0.0823</td>
</tr>
<tr>
<td>-50%</td>
<td>0.1317</td>
<td>0.0420</td>
</tr>
<tr>
<td>+100%</td>
<td>1.2564</td>
<td>0.1100</td>
</tr>
<tr>
<td>-100%</td>
<td>0.1678</td>
<td>0.0422</td>
</tr>
</tbody>
</table>

4.5 Chapter Summary

The simulations clearly illustrate how we can take advantage of partial knowledge of plant dynamics to control a system using a combination of conventional controllers and neural-based control. The strategy outlined for the new control scheme was demonstrated on a flexible one-link manipulator in which the rigid body dynamics were assumed known and the flexible dynamics were learned by a neural identifier. A control scheme using conventional control was used for control of the rigid dynamics and a neural controller was used to provide a corrective control to compensate for the unmodeled dynamics. The overall effect is to minimize tip vibration of the manipulator. This is demonstrated first for the case of a constant payload chosen with a nominal value, then for the case where the payload was varied. On-line training of
the control neural net showed that control could be achieved in a minimum number of arm movements for reasonable changes of the payload.
This chapter addresses neural networks in a closed loop control setting, for use with plants for which only a nominal model is known. A fundamental issue in controller design is robustness, where a controller is expected to perform well in the presence of uncertainty. The uncertainty can arise from unknown plant parameters, unmodeled dynamics, or external disturbances. Further complications result when the plant is nonlinear. In this chapter we address robust control of nonlinear plants with parameter uncertainty, using variable structure control (VSC) as the robust control strategy. Neural nets are introduced to overcome some of the shortcomings that accompany the VSC technique.

This chapter is organized as follows. We begin in Section 5.1 with a brief overview of robust control, followed by a discussion in Section 5.2 on the properties of variable structure control. The use of neural networks to overcome the shortcomings of VSC is also discussed. We then move to a specific class of nonlinear plants with parametric uncertainty and discuss a control strategy that uses feedback linearization and variable structure control in Section 5.3. In Section 5.4, examples demonstrate the benefits of neural nets which synthesize the equivalent control signal to be used in concert with the VSC. A brief summary concludes this chapter.
5.1 Some Remarks on Robust Control

It is well known that mathematical descriptions of a dynamical system cannot match the output of a physical system exactly. Therefore, all control strategies must incorporate some amount of flexibility to compensate for the errors resulting from model uncertainty. Further uncertainties may arise as the complexity of the plant increases. For example, there can be uncertainty in the order of the plant, uncertainties in plant parameters, or poorly modeled dynamics can contribute to modeling errors, especially for nonlinear plants. Control methods are needed that can allow the system to perform well in the presence of these uncertainties.

Often, one only knows a nominal plant model from which a controller is designed to meet a desired performance specification. Performance of the controller design from classical (frequency domain) or modern (time domain) techniques is limited by the uncertainties that exist. Therefore, a robust control design is defined as a design which performs well in terms of stability and performance specifications even in the presence of uncertainty.

One popular approach to robust control is to place bounds on the allowable uncertainty, either in the form of structured perturbations in the state-space form or in terms of the magnitude of external disturbances. Then, bounds on stability margin can be found based on the magnitude of the perturbation such that stability is maintained. Examples of this approach can be found in [52, 76].

An alternative method in the time domain is to use the approach pioneered by Utkin [65, 66], under the name of variable structure systems (VSS). The resulting
control strategy, called variable structure control (VSC) is a high speed switching feedback control, characterized by a high gain property in the vicinity of the origin. For systems with uncertainties, we only require that the uncertainties appear in the range space of the control input, which allows one to achieve the desired insensitivity to system uncertainties. The VSC approach for robust control is discussed in the section which follows.

5.2 Variable Structure Control for Robustness

In this section, we discuss the the well known variable structure control (VSC) strategy in more detail, and give some of the common characteristics that are unique to VSC. This approach provides a useful tool for handling the system uncertainties resulting from unmodeled dynamics, parameter variations and disturbances. Thus, if an appropriate coordinate system can be found such that all the nonlinearities and uncertainties only appear in the range space of the control input, then robust control schemes such as the variable structure control approach can be applied to provide insensitivity to system uncertainties. A known characteristic of the VSC approach is that it is discontinuous about a switching surface due to the discontinuous control. A larger discontinuity in the control input usually guarantees the system robustness to a wide range of existing uncertainties if the bounds of those uncertainties can be covered by the high control gain. However, in practice, time delay in control switching will cause system states to zigzag along the switching surface. The larger the discontinuity amplitude of the switching control, the further the system states will be away from the switching surface each time the control is switched.
A general but passive way to alleviate the problem of system states moving about the switching surface is to insert a boundary layer about the switching surface and approximate the switching control by a continuous control inside the boundary layer [61]. An alternate and active way to deal with the problem is to form feedforward compensation to cancel the influence from system uncertainties as much as possible, provided partial information concerning the uncertainty influence can somehow be obtained from previous control actions. According to the theory of equivalent control [66], the equivalent control input is the input that forces the system states to be constrained to the given switching surface no matter what uncertainties the system may have. This equivalent control input can be achieved approximately by filtering the chattering control signal resulting in an average control input. Using the average control input as feedforward compensation, we can greatly reduce the control gain while the system remains robust to uncertainties.

5.2.1 Characteristics of VSC in Sliding Mode

There are several aspects of variable structure control that are unique to this method of control. Some of these characteristics make it an attractive control strategy for systems with disturbances, while others may present the designer with some problems. A few of these characteristics are highlighted below.

*High Gain in the Vicinity of the Switching Surface*

Consider the following continuous SISO system, with state $x(t) \in \mathbb{R}^n$, input $u(t)$ and output $y(t)$. We will assume that the system is in controller canonical form, with
the addition of a disturbance term as follows,

\[
\begin{align*}
\dot{z}_1 &= z_2 \\
\dot{z}_2 &= z_3 \\
&\vdots \\
\dot{z}_n &= -\sum_{i=1}^{n-1} \alpha_i z_i + g(z, t) + Bu(t) \\
y(t) &= z_1(t)
\end{align*}
\] (5.1)

where \( \alpha_i \) can be constants or time-varying parameters, and \( g(\cdot) \) represents a disturbance which may or may not be a function of the states. We assume for simplicity that the output is equal to the state, the control is added linearly, and that we know bounds on the disturbance \( g \),

\[
|g(x, t)| \leq g_{max}. 
\] (5.2)

We wish for the output to be regulated to zero and define a switching surface to be

\[
\sigma(t) \triangleq \sum_{i=1}^{n-1} \alpha_i z_i + z_n,
\] (5.3)

and therefore,

\[
\dot{\sigma}(t) = \sum_{i=1}^{n-1} \alpha_i \dot{z}_i + \dot{z}_n \\
= \sum_{i=1}^{n-1} \alpha_i z_{i+1} + g(z, t) + Bu. 
\] (5.4)

The existence of a sliding mode is guaranteed [66] if

\[
\sigma \dot{\sigma} < 0 
\] (5.5)
and based on this requirement, we can select the control law to be

\[ u(t) = \sum_{i=1}^{n-1} \alpha_i z_{i+1} - k \cdot \text{sgn}(\sigma(t)) \]  

(5.6)

where \( \text{sgn}(\sigma) \) is the signum function defined as

\[ \text{sgn}(\sigma) = \begin{cases} 
+1 & \text{if } \sigma > 0 \\
-1 & \text{if } \sigma < 0 
\end{cases} \]  

(5.7)

We choose the gain \( k \) in (5.6) to be large enough to satisfy the requirement \( \sigma \dot{\sigma} < 0 \) which means that the gain is chosen to cover the disturbance \( g \), to yield

\[ \sigma \dot{\sigma} = \sigma(g - k \cdot \text{sgn}(\sigma)) < 0 . \]  

(5.8)

The gain is chosen in accordance with the existence condition,

\[
\begin{align*}
\text{for } \sigma > 0, & \quad g - k < 0 \\
\text{for } \sigma < 0, & \quad g + k > 0 ,
\end{align*}
\]

(5.9)

so that in each case, \( k \) can be chosen to remove the disturbance,

\[ k > |g|_{\text{max}} . \]  

(5.10)

Comparing this strategy to a conventional feedback control law \( u(t) = k z(t) \) it can be seen that the condition of (5.5) needs the gain \( k \) to be infinite in the case that \( g \) is independent of the state variables. This demonstrates the high gain property of the VSC control law in the neighborhood of the switching surface.

Note that \( g(\cdot) \) may or may not depend on the system state. This implies that despite the existence of the disturbance, if switching control can be generated at an infinite rate, then an ideal sliding motion will take place once the switching surface is reached and performance is guaranteed in the presence of the disturbance.
**Chattering**

The control law for VSC requires an infinitely fast switching mechanism, which is most often impossible to realize for physical systems. The phenomenon of nonideal but fast switching is labeled as *chattering*. The high frequency components of chattering are undesirable because they may excite unmodeled high-frequency dynamics which could result in unforeseen instabilities. There have been many methods designed to minimize the amount of chattering in the control law. On the other hand, this chattering issue represents extra effort of the VSC with respect to the system uncertainties to achieve robustness. Therefore, it reflects unknown system dynamics indirectly and can be picked up through an averaging operation to improve control performance as will be addressed as follows.

**Concept of Equivalent Control**

A possible method for determining the motion of the system in sliding mode is the method of *equivalent control*. Once the system reaches the switching surface, then one can find an average control which tends to an "equivalent control" signal representing the average value of the chattering control signal as measured by the output of a first-order filter. The existence of a sliding mode implies that once the system is in sliding mode, then \( \sigma(z(t)) = 0 \) and \( \sigma(z(t)) = 0 \), and for the case previously discussed,

\[
\sum_{i=1}^{n-1} \alpha_i z_i + z_n = 0 \quad \text{or} \quad z_n = - \sum_{i=1}^{n-1} \alpha_i z_i.
\]

(5.11)

On the switching surface, the dynamics are linear and in the general case the system order is reduced by order \( m \) where \( m \) is the dimension of the input space.
Using the chain rule, and substituting for \( \dot{z} \), we have

\[
\frac{\partial \sigma}{\partial z} \dot{z} = \frac{\partial \sigma}{\partial z} \left[ \sum_{i=1}^{n-1} \alpha_i z_i + z_n + g(z, t) + B \cdot u_{eq} \right] = 0
\]  

(5.12)

where \( u_{eq} \) is the equivalent control which solves this equation. Notice that

\[
\frac{\partial \sigma}{\partial z} = [\alpha_1 \alpha_2 \cdots \alpha_{n-1}]^T \quad \text{and} \quad B = [0 \ 0 \cdots 1]^T
\]

(5.13)

so that

\[
\frac{\partial \sigma}{\partial z} B = 1.
\]

(5.14)

Solving for \( u_{eq} \), we obtain

\[
u_{eq} = - \sum_{i=1}^{n-1} \alpha_i z_i - z_n - g(z, t) .
\]

(5.15)

Thus, \( u_{eq} \) is composed of a signal that is linear in the state \( z(t) \), and a signal that is proportional to the disturbance. Now, assume that the disturbance \( g \) varies around an average value \( g_{av} \) such that

\[
g = g_{av} + \Delta g(t)
\]

(5.16)

If we further assume that \( g_{av} > 0 \), then the bound on the disturbance can be written as

\[
|g|_{max} = g_{av} + |\Delta g(t)|_{max} .
\]

(5.17)

Then, the control law (5.6) can be rewritten as

\[
u(t) = \sum_{i=1}^{n-1} \alpha_i z_{i+1} - g_{av} - k \cdot \text{sgn}(\sigma(t))
\]

(5.18)
and the switching gain can be chosen to cover the deviation of the disturbance from its average value,

$$k > |\Delta g|_{\text{max}}$$

resulting in a smaller value for the switching gain.

Thus, the notion of equivalent control can be used to minimize chattering by combining the equivalent control in a feedforward controller with the output of the VSC controller. As we will see, this has the effect of minimizing the chattering in the control signal, allowing for more accurate system performance.

### 5.2.2 Neural Nets for Synthesis of Equivalent Control

Neural nets are introduced for use with VSC to overcome some shortcomings of VSC. The neural nets of this applications are used to perform a nonlinear mapping from input to output space, and the output is used in concert with conventional techniques to improve the overall performance.

In order to implement the average control gain as a feedforward compensation, it is necessary for the compensator to remember a number of different response patterns with respect to several reference inputs. This can be considered as a nonlinear mapping from reference input and system states to the average control input. Since there may be an infinite number of possible reference inputs, it is more important for the compensator to give approximate feedforward compensation to any new reference input which is inside two neighboring learned references. Thus, an interpolation capability is required.
Neural networks possess many of the necessary qualifications for implementation of such a feedforward compensation. As shown in Chapter 2, neural nets can perform nonlinear static mapping from input space to output space while also performing the necessary nonlinear interpolation that is desired. Thus, for this problem, the neural network performs a static nonlinear mapping from reference input and system states to equivalent control. The approximation of equivalent control provides the motivation for using neural networks and the overall result is that the chattering of the VSC signal is greatly reduced.

5.3 Robust Control of Nonlinear Systems with Parameter Uncertainty

A common method for control of nonlinear plants is to use feedback linearization for the control law design. However, feedback linearization and with plant uncertainty are conflicting ideas, since the control strategy is to essentially cancel the nonlinearities of a plant to render it a linear plant for which we know a plethora of techniques for control. It is difficult to cancel nonlinearities when the plant is not well known. To compensate for the uncertainties, variable structure control is used to ensure stability of the overall control scheme.

A considerable amount of literature has been devoted to the topic of feedback linearization [12, 13, 26, 34], an approach which has the advantage of giving the control designer linear tools for control of nonlinear systems. Feedback linearization is accomplished by a cascade of blocks consisting of a nonlinear controller, a nonlinear plant, and a nonlinear coordinate transformation, such that the system is linear from
the input terminal of the nonlinear controller to the output of the nonlinear coordinate transformation. Then, any linear control strategy can be used between this input-output pair. The main drawback of feedback linearization is that the design procedure is highly dependent upon full knowledge of the plant being completely available.

In the subsections that follow, we give a discussion of feedback linearization for nonlinear systems with parameter uncertainty, followed by a discussion of the inclusion of VSC with feedback linearization. The drawbacks of using these approaches in this paradigm are highlighted via simulations on a second order nonlinear system. It is later shown how neural nets are used for synthesis of the equivalent control which greatly reduces the chattering of the control signal.

5.3.1 Feedback Linearization of Systems with Uncertainties

We begin by discussing input-state linearization for single-input systems represented by the state equations

$$\dot{x} = f(x, p) + g(x, p)u$$

(5.20)

where $f$ and $g$ are smooth vector fields, $p$ is a parameter vector, and $u$ is a scalar control input. In Equation (5.20), $x, f$ and $g \in \mathbb{R}^n$ and $p \in \mathbb{R}^p$. In general, when applying the feedback linearization approach, $f, g, x$ and $p$ of the nonlinear system (5.20) must be known exactly. In this paper we consider the effect of system uncertainty from unknown parameters $p$, which are bounded by a known hypercubic over the range $[p_{\text{min}}, p_{\text{max}}]$. To ensure that the nonlinear system (5.20) is feedback linearizable, the following assumption is needed.
Assumption 5.1 For any \( p \in [p_{\text{min}}, p_{\text{max}}], x \in \Omega \subset \mathbb{R}^n \) the system (5.20) satisfies the following properties:

1. the vector fields \( \{g, \text{adj}g, \ldots, \text{adj}^{n-1}g\} \) are linearly independent in \( \Omega \).
2. the set \( \{g, \text{adj}g, \ldots, \text{adj}^{n-2}g\} \) is involutive in \( \Omega \).

According to the assumption, there exists a nonzero scalar function \( T_1(x) \) satisfying

\[
L_gT_1 = L_{\text{adj}g}T_1 = \ldots = L_{\text{adj}^{n-1}g}T_1 = 0 \tag{5.21}
\]

\[
L_{\text{adj}^{n-1}g}T_1 \neq 0 \tag{5.22}
\]

where \( L_gT_1 \) is the Lie derivative of \( T_1 \) along \( g \) and \( \text{adj}^{i}g(i = 0, \ldots, n-1) \) is the iterated Lie bracket

\[
\text{adj}^{i}g = [f, \text{adj}^{i-1}g] \tag{5.23}
\]

with \( \text{adj}^{0}g = g \). \( T_1(x) \) can be derived from the \( n \) equations of (5.21) and (5.22).

If we choose a new set of state variables defined by

\[
z = [z_1, \ldots, z_n]^T = [T_1, L_fT_1, \ldots, L_f^{n-1}T_1]^T \tag{5.24}
\]

then the first \( n - 1 \) state equations satisfy

\[
\dot{z}_j = z_{j+1} \quad \text{for} \quad j = 1, \ldots, n - 1 \tag{5.25}
\]

while the last state equation is

\[
\dot{z}_n = L_f^nT_1 + L_gL_f^{n-1}T_1u = \alpha(z, p) + \beta(z, p)u \tag{5.26}
\]
where $\alpha(z, p) = L^T_0 T_1 [T^{-1}(z, p)]$, and $\beta(z, p) = L_0 T_0^{-1} T_1 [T^{-1}(z, p)]$. Therefore, the transformed dynamics are in companion form in which all the nonlinear and uncertain terms are in the range space of the control input $u$. When $p$ is known precisely a priori, a simple way of designing a controller is to realize the inverse dynamics of (5.26) and use a pole assignment strategy for the resulting linearized system, i.e.

$$u = \frac{1}{\beta}(v - \alpha) \quad (5.27)$$

$$v = h(s)(r - z) \quad (5.28)$$

where $h(s)$ is a Hurwitz polynomial, and $r$ is the desired reference input. This method, however, fails to work well when the nonlinear cancellation cannot be accomplished, specifically when system uncertainties exist.

Consider the following second order system:

$$\dot{x}_1 = \frac{5(e^{x_2} - 1)}{e^{x_2} + 1}$$

$$\dot{x}_2 = \frac{p_1 \cos(x_2)}{1 + p_2 x_1^2} + p_3 u \quad (5.29)$$

where $p = [p_1, p_2, p_3]^T$ are system parameters that can vary over a range of $\pm 50\%$ from the nominal value of $p_0 = [5, 2, 1]^T$, and $p_2 > 0$. For this system, we have

$$f = \begin{bmatrix} \frac{5(e^{x_2} - 1)}{e^{x_2} + 1} \\ \frac{p_1 \cos(x_2)}{1 + p_2 x_1^2} \end{bmatrix} \quad \text{and} \quad g = \begin{bmatrix} 0 \\ p_3 \end{bmatrix} \quad (5.30)$$

It is easy to check that for $\pm 50\%$ variations of the parameter vector $p$ and $p_3 \neq 0$,

$$\forall p \in \left[ \frac{1}{2} p_0, \frac{3}{2} p_0 \right], \quad \text{and} \quad x_1, x_2 \in (-\infty, \infty)$$

$$\text{rank}[g \ ad^Tg] = 2 \quad (5.31)$$
Therefore Assumption 5.1 is satisfied since a single vector $g$ is involutive. By solving

$$L_gT_1 = 0 \quad \text{and} \quad L_{adjg} \neq 0,$$

we obtain the coordinate transformation as $z_1 = T_1(x) = x_1$, then

$$z_2 = \dot{z}_1 = \frac{5(e^{x_2} - 1)}{e^{x_2} + 1}$$

and

$$\dot{z}_2 = \alpha(z, p) + \beta(z, p)u$$

where

$$\alpha(z, p) = \frac{5p_1}{2(1 + p_2 z_1^2)}(1 - (0.2z_2)^2) \cos \left( \ln \frac{1 + \frac{z_2}{5}}{1 - \frac{z_2}{5}} \right)$$

$$\beta(z, p) = 2.5p_3 \left( 1 - \left( \frac{z_2}{5} \right)^2 \right).$$

Since only the nominal value of $p$ is available, the inner-loop control is selected as

$$u = \beta_0^{-1}(z, p_0)(v - \alpha_0(z, p_0))$$

The outer-loop control is

$$v = k_1(r - z_1) + k_2(\dot{r} - z_2)$$

where $k_1 = 4, k_2 = 4$ are feedback gains chosen to place the poles of the linearized system at $\lambda = -2, -2$.

In Figure 30, the responses to a step input are shown for the system with no uncertainties present and for a system where $p_1 = 1.4p_0, p_2 = 0.5p_0, \text{and} p_3 = 0.5p_0$. It is obvious that the feedback linearization method works well for the nominal case,
but poorly for a system with uncertainties. In other words, the controller designed in terms of feedback linearization is sensitive to system uncertainties such as parameter variations. Therefore it is necessary to incorporate robust control into the system to reject system uncertainties, as will be discussed in the next section.

Figure 30: Step response of system with parameter uncertainty (dashed line) compared to nominal system response (solid line)
5.3.2 Synthesis of Variable Structure Control with Feedback Linearization

In this section, we consider VSC as a robust control approach to treat the uncertainty problem that exists in the feedback linearization. The VSC philosophy is to use a high gain feedback, which changes structure abruptly according to the location of the state vector with respect to a prescribed switching logic so as to achieve a desired overall behavior of the system. It is well known that the VSC is robust against parameter variations, disturbances or modeling errors when those uncertainties are reasonably bounded in a known range.

Now let us consider applying the VSC method to the system (5.29) that contains uncertain parameters. First we use (5.36) as the inner-loop control to cancel nonlinear and uncertain terms in (5.34) as much as possible. Then the system becomes

\[ \dot{z}_1 = z_2 \]
\[ \dot{z}_2 = \rho(z, p, p_0) + (1 + \delta(p, p_0))v \]
\[ (5.38) \]

where

\[ \rho = \alpha(z, p) - \frac{\beta(z, p)}{\beta(z, p_0)} \alpha(z, p_0) \]
\[ \delta = \frac{\beta(z, p) - \beta(z, p_0)}{\beta(z, p_0)} \]
\[ (5.39) \]

For the example, since \( p \) is bounded on the range \([\frac{1}{2}p_0, \frac{3}{2}p_0]\), we can easily find the bounds on \( \rho \) and \( \delta \).

\[ |\rho(z, p, p_0)| \leq \frac{5}{2} \left| 1 - \left( \frac{z_2}{2} \right)^2 \right| = \rho_{\text{max}}(z) \]
\[ |\delta(p, p_0)| \leq \delta_{\text{max}} = \frac{1}{2} \]
\[ (5.40) \]
Now choose the switching surface

$$\sigma = ce + \dot{e} \quad (5.41)$$

where $e \triangleq r - z$. The derivative of (5.41) is

$$\dot{\sigma} = c\dot{e} + \dot{r} - \rho(z, p, p_0) - (1 + \delta(z, p, p_0))v \quad (5.42)$$

Let $\sigma \dot{\sigma} < 0$, then we can find the outer-loop robust control input

$$v = \frac{\rho_{\text{max}}(z) + |c\dot{e} + \dot{r}|}{1 - \delta_{\text{max}}} \text{sgn}(\sigma) \quad (5.43)$$

where $\text{sgn}(\sigma)$ is the signum function defined in Equation (5.7). The outer-loop control $v$ guarantees that the system state $z$ converges to the switching surface $\sigma = 0$, which causes $e \to 0$.

The system response to a step reference input with VSC is shown in Figure 31. In spite of the existing parameter uncertainties, the state error is greatly reduced from about 140% in Figure 30 to about 1.7% in Figure 31. The static error can be further reduced to one tenth of the error displayed in Figure 31 by changing the sampling interval from 10 msec to 1 msec.

In practice, the implementation of VSC results in discontinuous control as seen in Figure 32. In this figure, the solid line shows the chattering control signal, which demonstrates the discontinuity property of the VSC approach. It is this discontinuity that realizes high gain control at the neighborhood of equilibrium to suppress system uncertainties. On the other hand, a large discontinuity in the control signal may cause severe chattering that leads to degradation in performance.
Figure 31: Step response of system with VSC

One approach to overcome the problem is to make a trade-off between robustness and smoothness by introducing what is known as a boundary layer around the switching surface and approximate the switching control by a continuous control inside the boundary layer.

Those chattering issues, however, reflect the effort of the control system to suppress system uncertainties. As can be observed in Figure 32, the dashed line, which is an
output of a first order low pass filter with input being the chattering control,

\[ \tau \dot{y} + y = v \]  \hspace{1cm} (5.44)

represents the average control effort of the VSC with respect to the required reference and system uncertainties. According to the equivalent control method, when the sliding mode exists, we get \( \sigma = 0, \dot{\sigma} = 0 \), and the equivalent control \( v_{eq} \) is

\[ v_{eq} = \frac{c \dot{e} + \ddot{e} - \rho(z, p, p_0)}{1 + \delta(z, p, p_0)} \]  \hspace{1cm} (5.45)
If applying $v_{eq}$ as an outer-loop control input to the system 5.29, by virtue of the sliding mode, the tracking error $e$ approaches zero asymptotically in spite of the existence of unknown $p$. This equivalent control issue, although it cannot be calculated directly from (5.45) due to the unknown $p$, can be approximated by the average control in (5.44) with any required precision provided the switching frequency can be sufficiently high and the time constant $\tau$ in Equation (5.44) can be sufficiently small. This interesting feature motivates us to use the average control signal as a feedforward compensation to cancel most of the system uncertainties and nonlinearities so that the control discontinuity can be greatly reduced without losing robustness.

In the next section, we will show how to implement the feedforward compensation with an artificial neural network (ANN).

### 5.4 Neural Net Compensation in Concert with VSC

Neural networks have many advantages for use in closed loop control system applications. In this particular application, we are concerned with static mappings for a known class of inputs, specifically step inputs, so we trained the neural nets using a series of step responses. The neural net has the ability to remember a large number of patterns, and in [46] there are guidelines given for minimal network size based on the number of data points presented to the net. An attractive feature of neural nets that is perhaps more important than the net memory capabilities is the ability to interpolate between several trained patterns.

In this application, a neural net is used to implement a feedforward input to compensate and thus reduce discontinuity in the variable structure control signal to the
plant by performing a static mapping from input space to output space. Specifically, a neural network is used to provide a nonlinear static mapping from the plant state variables and reference input to the input to the nonlinear controller. The input space $X$ is defined as

$$X_k \triangleq [x_1(k) \ x_2(k) \ r(k)]^T \subset \mathbb{R}^3 \text{ for } k = 1, \ldots, N$$

(5.46)

and the output space is

$$Y_k \triangleq v_{nn}(k) \subset \mathbb{R} \text{ for } k = 1, \ldots, N$$

(5.47)

where $N$ is the number of points in training data set. The mapping is performed by a nonlinear function $\mathcal{F}$ that maps the compact set $X_k$ to an element in $Y_k$ with the objective that we find $\mathcal{F}$ so that

$$\|\hat{v}_{nn} - v_{nn}\| = \|\mathcal{F}(x) - \mathcal{F}(x)\| \leq \epsilon$$

(5.48)

where $x$ is an element of the set $X$. In the case of backpropagation, the norm $\| \cdot \|$ is a standard $L_2$ norm so that we minimize the mean squared error at the output of the neural net. The output of the neural net is calculated by

$$\hat{v}_{nn}(k) = W_{23}\{f[W_{12} f(W_{01}x_k)]\}$$

(5.49)

where $f(\cdot)$ is the sigmoid nonlinearity given by

$$f(a) = \frac{1}{1 + e^{-a}}$$

(5.50)

and $W_{ij}$ is a matrix of weights from layer $i$ to layer $j$, with the 0th layer representing the input layer.
The neural net used in these experiments is a three layer net, with configuration $N_{3,10,5,1}$. The inputs are the two plant states and the reference input, and the output is an approximation of the equivalent control from the VSC. In the simulations that follow, the backpropagation learning rate $\eta$ is 0.2. A momentum term $\alpha = 0.8$ is also used. The neural net in closed-loop is shown in Figure 33.

For simplicity, the training data used for the neural net consists of two step responses with a magnitude of 1.0 and 1.5, and the results of closed-loop simulations for these cases can be seen in Figure 34. The step response in each case has nearly zero steady state error. Furthermore, for a step of magnitude 1.25, the system again has near zero steady error as shown in Figure 34. This response is not a part of the training data set and thus shows the ability of the ANN to perform nonlinear interpolation of the control signal. The control signal from the neural net for each case is shown in Figure 35. In this figure, the solid line is the output of the neural net for a step of 1.0, the dashed line is the output of the neural net for a step of 1.5,
and the dotted line is the output of the neural net for a step of 1.25. Aside from the advantage of small steady state error, the appeal of introducing the neural net approach is the capability of reducing the discontinuous control signal magnitude while the system remains robust. In fact, the chattering control outputs of the VSC block are scaled down to 40% of the original ones, yet there are no increments of steady error. Figure 36 compares the control inputs to the plant for the system with VSC only and for the system with both VSC and ANN compensation with respect to $r=1$. The discontinuity chattering of the control signal is reduced by about 40%.

Figure 34: Step response of system with VSC and Neural Net Compensation
Figure 35: Output of Neural Net controller with step reference input

Figure 36: System input with VSC only (dashed line) and with both VSC with ANN compensation (solid line); $r = 1.0$
5.5 Chapter Summary

In this application we investigated the synthesis of feedback linearization, variable structure control and artificial neural network based compensation methods. The main purpose of combining those different strategies is to provide an effective control approach to systems which contain both nonlinearities and uncertainties. For those nonlinear systems matching linearizable conditions, variable structure control is a satisfying robust control scheme to handle system uncertainties in the new coordinate system. With a pre-trained neural network as a feedforward compensator, the undesired discontinuity phenomena of VSC around steady state can be mitigated without losing robustness. Via simulation of a second order system, it was observed that the synthesized control system achieves both the robust property from the VSC and the well interpolated feedforward compensation property from the artificial neural network.
CHAPTER VI

Conclusions

This dissertation addressed the use of neural nets in control applications, focusing on both their benefits and limitations in a variety of paradigms for both system modeling and control. The focus of this work is toward the user of neural networks such that the machinery is developed for successful applications of mapping neural networks in control systems applications. Convergence of the neural nets was defined to satisfy the empirical nature of the experiments contained within. Requirements on the inputs was defined so that the neural net interpolates between the training data set with a high degree of precision.

The neural net modeling problem was addressed in Chapter III and comparisons were made between some of the standard ARMA-type linear system models and the corresponding neural net models. Dynamical systems were realized in predictor form by neural nets using a regression of prior inputs, outputs and errors as neural net inputs, allowing the network to learn an internal representation in weight space of the dynamics of the process under study. Common model structures for linear systems were presented, along with their neural net representations. A performance analysis was given for the linear structures using several examples where the conventional methods were compared with neural methods, with error variance as the figure of
merit. System modeling and identification was also investigated for nonlinear systems. There are few identification methods for nonlinear systems that are applicable to a wide range of problems, so the neural net methods offer the control designer a new means for modeling nonlinear systems. This chapter also included a section containing examples in which the neural nets were compared with conventional methods for system identification and output estimation using a large space structure example. The issue of underparametrization was investigated via example. In every case, the neural nets trained using the methods outlined in Chapter II were shown to have superior performance over the conventional methods with respect to output error.

Applications of neural nets in closed-loop control algorithms were given in Chapters IV and V. In Chapter IV, simulations on a flexible robot demonstrate that neural nets can be used for vibration damping and control of systems with partially known dynamics. In Chapter V, neural nets are used together with conventional robust control techniques to give a feedforward neural net compensator that shows improvement over the conventional methods for control of a nonlinear system with parameter uncertainty.

Although neural nets research has made great strides in the last five years, there remains much work to be done. The direct connection between the weight matrices neural nets used for modeling dynamical systems and known quantities such as poles and zeros or parameters of a transfer function has yet to be established. This limits our options for controller design. Also, there does not exist a global convergence theorem for the backpropagation algorithm, so any convergence proofs must appeal to
more empirical interpretations. Training algorithms are also needed that provide real-time identification schemes for system identification. Until algorithms and methods are developed that converge quickly, much of the work done with dynamic system modeling will have to be performed in part off-line.

In the applications area, we are also limited by the slow convergence of the back-propagation algorithm, but clever designs can eliminate this problem. For our specific examples, general methods for feedback linearization would be very helpful for the nonlinear system example of Chapter V. In fact, although we found that we could provide very accurate functional mappings for the nonlinear coordinate transformation part of feedback linearization, we could not use a neural implementation of feedback linearization because even very small errors can lead to instability. Further investigation of the neural feedback linearization problem is needed which would then extend applications of this technique for a wide range of nonlinear controller designs.

For neural control of flexible manipulators, there is a great deal of promise for further research. In fact, the methods outlined in the example of Chapter V could be extended to the two-link case. This could be done for trajectory tracking by finding several linear models and designing LQR controllers at each operating point in a manner similar to [59]. It is expected that neural nets would have a large advantage over conventional techniques due to their interpolation properties and computational efficiency.
Appendix A

The JPL/AFAL Flexible Structure Testbed

A project was initiated under the auspices of the guest investigator program at the Jet Propulsion Lab (JPL) for verification of several decentralized control approaches on the JPL/AFAL Flexible Structure Testbed. This facility was used for the example of Chapter III for system identification of an experimental flexible space structure. Included below is a description of the experimental testbed.

A.1 Description of the Experimental Facility

The facility can be described as a 3-D antenna-like structure which exhibits many characteristics of a typical large space structure. These characteristics include many low frequency modes, densely packed modes, low structural damping, and three-dimensional structural interaction among components. A brief description of the testbed follows. A detailed description can be found in [67].

The main component of the testbed facility consists of a central hub to which 12 flexible ribs are attached as shown in Figures 37 and 38. The ribs, which are 2.25 m in length, are coupled by two concentric rings of pretensioned wires. The hub is of radius 0.6 m, so that the entire structure is 5.7 m in diameter. Each flexible rib is supported at its 40% and 80% points by levitators, 0.9 m and 1.8 m from the rib root. Each
A levitator consists of a counterweight attached to the rib by a wire which passes over a low friction pulley. The levitator function is twofold: it prevents structural collapse due to gravity while also providing a means for measuring vertical displacement of the rib. A 1.0 m flexible boom is attached to the central axis of the hub and has a mass at its lower end to simulate the feed horn of an antenna. The feed mass is 4.5 kg. The hub is mounted to a backup structure via a two-axis gimbal which allows rotational freedom about two perpendicular axes in the horizontal plane. The gimbal bearings support approximately one quarter the weight of the ribs, the entire weight of the hub, boom and feed, and their respective sensing and actuation devices.

### A.1.1 Actuators

Each of the ribs can be excited dynamically by a single rib-root actuator with a lever arm of about 0.3 m from the hub attachment point. Each rib-root actuator consists of a speaker-coil type device which reacts against a mount rigidly attached to the hub. In addition, two speaker coil type actuators are mounted on the hub to provide controlled torquing about the two gimbal axes. These hub torquers apply linear forces to the hub at its outer circumference to yield the required torques about the axis of rotation. Together, these 14 actuators are capable of controlling all flexible modes of the structure. The location of these actuators are shown in Figure 38, together with the assignments of the rib numbers. As is, the axes of rotation are construed to be along rib #4 and rib #10, and rib #1 and rib #7, with respective rotations excited by hub actuators HA1 and HA10. For convenience, the axes corresponding to the 1–7 axis and 4–10 axis are called axis 1 and axis 2 in later references.
Figure 37: JPL/AFAL Flexible Structure Testbed

Figure 38: Rib Numbering Convention and Location of Sensors and Actuators
A.1.2 Sensors

Each of the 24 levitators is equipped with an incremental optical encoder which measures the relative angular rotation of the levitator pulley. These angular measurements are then translated into the vertical motion of the ribs at the levitator/rib attachment points, relative to the backup structure. Hub angular rotations about the two axes are measured by two rotary variable differential transformers (RVDT) mounted directly at the gimbal bearings. Additional linear variable differential transformer (LVDT) sensors are provided for rib displacement measurements at four evenly spaced rib root actuator locations but are not used for these experiments. The locations of these 30 sensors are also shown in Figure 38. As assigned, angular displacement about axis 1 (1-7 axis) are measured by sensor HS1, and that about axis 2 (4-10 axis) by sensor HS10.

A.2 The System Model

The system is modeled by considering the modes shown below in Table 8, where \( k \) represents the circular wave number, axis refers to the horizontal axis of rotation (1 or 2), and BG is the bending group for each rib. When \( k = 1 \), these modes are referred to as the boom-dish modes. All other modes are called dish modes. The symmetry of the dish structure makes it possible to separate variables and express a given mode shape as the product of a shape function, which is independent of the rib number, and a scalar function which depends on the rib number. This scalar function, which reflects the circular dependence of the given mode shape, can be written by
Table 8: Modes included in the system model.

<table>
<thead>
<tr>
<th>(k)</th>
<th>Axis</th>
<th>BG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1,2,3 and 4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1,2,3 and 4</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1,2</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1,2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1,2</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1,2</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>1,2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1,2</td>
</tr>
</tbody>
</table>

inspection in the following form:

\[ \sin \left( \frac{2\pi ik}{n} + \phi_k \right) \]  

where \(i\) is the number of the rib, \(n\) is the number of ribs in the dish structure (12 for this structure), and \(k\) is the circular wave number for a given mode ranging from \(k = 0\) to \(k = 6\), and \(\phi_k\) is the phase angle depending on the coordinate system transformation. A mode is completely specified by its frequency, circular wave number, phase angle, and the boom and rib shape functions. Due to the symmetry of the structure, the dish modes are both uncontrollable and unobservable from the hub. However, the converse is not true: the boom−dish modes are “weakly” controllable from the rib actuators. When considering the dish modes only, all reaction forces on the hub cancel, leaving the appearance that the hub is clamped.
The following state space equations describe the system $S$:

$$
\begin{align*}
\dot{x}(t) &= Ax(t) + \sum_{i=1}^{N} B_i u_i(t) \\
y_i(t) &= C_i x(t), \ i = 1, \ldots, 14
\end{align*}
$$

(A.2)

where $x \in \mathbb{R}^{56}$, $u \in \mathbb{R}^{14}$, and $y \in \mathbb{R}^{26}$. Thus, the structure has 14 actuators and 26 sensors. The hub sensors measure angular displacement and the levitator rib sensors measure vertical displacement. The actuators act on the structure to provide a torque at the point of application. All units are measured in MKS. The system matrix is in the standard modal coordinate form with open loop damping included for each mode. Damping was originally estimated at 2%, for all modes, but was later adjusted for each mode based on open loop simulation results.
BIBLIOGRAPHY


154


