SPARSE AND DEEP LEARNING-BASED NONLINEAR CONTROL DESIGN WITH HYPERSONIC FLIGHT APPLICATIONS

By

SCOTT A. NIVISON

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2017

I dedicate this dissertation to my wonderful wife, Ivy, and two amazing daughters, Rori and Cassia.

ACKNOWLEDGEMENTS

First and foremost, I would like to acknowledge my advisor Dr. Pramod Khargonekar. He has provided a tremendous amount of encouragement, support, and motivation during my graduate studies. I feel fortunate to have had the opportunity to learn from him. He has provided a lasting impact that I will not forget.

I would like to thank my committee members, Dr. Dapeng Wu and Dr. Warren Dixon, for their inspiring graduate courses which have contributed greatly to my research. I would also like to thank Dr. Norman Fitz-Coy for his thought-provoking questions, during the oral qualifiers, which increased the quality of the dissertation. I would also like to express my gratitude to Dr. Eugene Lavretsky and Dr. David Jeffcoat for their time, effort, and discussions.

I would also like to acknowledge the support given to me by the Air Force Research Lab (AFRL) and my many co-workers. Specifically, I would like to mention Mr. John K. O'Neal and Mrs. Sharon Stockbridge for their helpfulness and continued support during my graduate program. Lastly, I would like to thank the Department of Defense for its financial support through the Science, Mathematics, and Research for Transformation (SMART) Scholarship program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DLC	Deep Learning Controller
GRU	Gated Recurrent Unit
HSV	Hypersonic Vehicle
LQR	Linear Quadratic Regulator
NN	Neural Network
PE	Persistence of Excitation
ROA	Region of Attraction
RBF	Radial Basis Function
RNN	Recurrent Neural Network
SHL	Single Hidden Layer
S-DLC	Sparse and Deep Learning Controller

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

SPARSE AND DEEP LEARNING-BASED NONLINEAR CONTROL DESIGN WITH HYPERSONIC FLIGHT APPLICATIONS

By

Scott A. Nivison

December 2017

Chair: Pramod Khargonekar
Major: Electrical and Computer Engineering

The task of hypersonic vehicle (HSV) flight control is both intriguing and complicated. HSV control requires dealing with interactions between structural, aerodynamic, and propulsive effects that are typically ignored for conventional aircraft. Furthermore, due to the long distance and high-speed requirements of HSVs, the size of the flight envelope becomes quite expansive. This research focuses on the development of sparse and deep neural network-based control methods to solve HSV challenges.

The first aspect of the research develops a novel switched adaptive control architecture called sparse neural network (SNN) in order to improve transient performance of flight vehicles with persistent and significant region based uncertainties. The SNN is designed to operate with small to moderate learning rates in order to avoid high-frequency oscillations due to unmodeled dynamics in the control bandwidth. In addition, it utilizes only a small number of active neurons in the adaptive controller during operation in order to reduce the computational burden on the flight processor. We develop novel adaptive laws for the SNN and derive a dwell time condition to ensure safe switching. We demonstrate the effectiveness of the SNN approach by controlling a sophisticated HSV with flexible body effects and provide a detailed Lyapunov-based stability analysis of the controller.

The second aspect of the research develops a training procedure for a robust deep recurrent neural network (RNN) with gated recurrent unit (GRU) modules. This

procedure leverages ideas from robust nonlinear control to create a robust and high-performance controller that tracks time-varying trajectories. During optimization, the controller is trained to negate uncertainties in the system dynamics while establishing a set of stability margins. This leads to improved robustness compared to typical baseline controllers for flight systems. We leverage a recently developed region of attraction (ROA) estimation scheme to verify the stability margins of the flight system. Inspired by the SNN adaptive control research, we develop the concept of a sparse deep learning controller (S-DLC) in order to improve perimeter convergence and reduce the computational load on the processor. We demonstrate the effectiveness of each approach by controlling a hypersonic vehicle with flexible body effects.

CHAPTER 1
INTRODUCTION

1.1 Motivation and Literature Review

Hypersonic vehicle (HSV) research could provide a path for safe space exploration and space travel while improving the capability to launch satellites into low Earth orbit. Additionally, HSV research is applicable to numerous military capabilities including increased survivability of flight vehicles and the ability to respond quickly to long distance targets that pose a significant threat [1]. Recent success from hypersonic flight vehicles such as the X-51 relies on conventional aircraft such as the B-52 and a solid rocket to boost the vehicle to high altitude and velocities. Afterwards, the flight vehicle separates from the aircraft, discards the rocket, and uses its actively cooled air-breathing supersonic combustion ramjet (scramjet) engine to accelerate to hypersonic speeds [2, 3]. Other hypersonic flight vehicles, commonly called boost-glide vehicles, use re-entry into the Earth's atmosphere to gain enough speed to become hypersonic [4]. A less documented limitation of hypersonic control is the lack of processing power. For instance, it is estimated that in real time the hypersonic flight vehicle needs to process information an order of magnitude faster than a subsonic platform.

Regardless of the details of the operation, control of hypersonic flight vehicles is a challenging task due to extreme changes in aircraft dynamics during flight and the vastness of the encountered flight envelope [5]. Furthermore, these vehicles operate in environments with strong structural, propulsion, thermal, and control system interactions [6]. Moreover, the initial mass of the HSV can dramatically decrease during flight which significantly impacts the structural modes of the flight vehicle [7]. It is worth noting that flexible body dynamic modeling is also a significant area of interest for extremely large flexible subsonic flight vehicles such as X-HALE (high altitude long endurance). These vehicles possess low-frequency structural vibration modes which can cause large nonlinear body deformations [8]. In addition, these

endurance flight vehicles operate at high angles of attack where nonlinear effects become prominent uncertainties in the system dynamics. In addition, operating at extremely high temperatures can deteriorate sensors and have a major impact on the pressure distribution on the HSV which affects vehicle stiffness. Unpredictable errors stemming from ablation or thermal expansion can drastically affect model parameters. For example, traveling at such high speeds causes the HSV to be subjected to extreme temperatures due to shock wave radiation and aerodynamic friction. Typically, an ablative material is added to absorb the heat and protect the vehicle surfaces from melting. The ablative material is designed to decompose while keeping the surfaces of the flight vehicle relatively cool. Unfortunately, this ablation process is difficult to model due to the complex structure of the ablative material and the coupling between the ablative material and the flight vehicle surface [9]. Another troublesome source of error could come from elastic effects such as thermal expansion caused by spatially varying temperature distribution which can cause structural deformations [10].

Inspired by the vast flight envelope and computation limitations encountered by hypersonic vehicles, we developed the sparse neural network (SNN) concept for adaptive control (see Chapters 4 and 5). The SNN uses a segmented flight envelope approach to select a small number of active nodes based on operating conditions. This encourages region based learning while reducing the computational burden on the processor. Additionally, the SNN allows segments to share nodes with one another in order to smooth transition between segments and improve transient performance. Various advancements in the SNN architecture are developed throughout this document with main contributions stated in Section 1.3.

In order to take advantage of the sophisticated hypersonic models, we develop a training procedure for a deep recurrent neural network (see Chapters 6 and 7). Based on our previous work in adaptive control, we extend the SNN concept to a recurrent deep architecture. This results in a sparse deep learning controller (DLC)

which utilizes a stacked deep recurrent neural network (RNN) architecture with gated recurrent units (GRU). The sparse nature of the controller drastically limits the number of computations required at each time step while still reaping benefits of its deep architecture. Robustness metrics of the DLC are verified through simulation and region of attraction (ROA) estimation via forward reachable sets. We show the effectiveness of the sparse deep learning controller approach through simulation results.

The research included in this dissertation contributes to the field of HSV control by improving upon (and using) techniques from Model Reference Adaptive Control (MRAC), policy search and deep learning. Sections 1.1.1 and 1.1.2 below provide brief overviews of MRAC, deep learning and policy search. More detail on these topics can be found in the background chapters (Chapter 2 and Chapter 3).

1.1.1 Neural Network-Based Model Reference Adaptive Control

Adaptive control has revolutionized nonlinear control and has brought tremendous improvements to the field in terms of performance and the amount of uncertainty and disturbances that system can handle. Unfortunately, there are some drawbacks. For instance, typical adaptive nonlinear controllers are not designed seeking specified transient performance requirements (e.g. rise-time, settling-time, etc.) and do not seek to minimize energy expenditure optimally. In addition, it is challenging to design adaptive controllers to perform well when facing significant non-parametric uncertainties and adaptive systems frequently oscillate and possess slower convergence rates when encountering sizable parametric uncertainties.

In order to address the large parametric uncertainty drawback, there have been two key approaches. Gibson [11] suggested adding a Luenberger-like term to the reference model. Narendra, uses the multiple model approach. This approach creates multiple identification models that switch between one another depending on which model is working the “best,” which is based on some predefined performance criteria. More recently, Narendra [12] uses the previously mentioned indirect adaptive control approach

utilizing multiple models where the output of all the models is combined to create a more accurate estimate of the parameter vector which results in better tracking performance. In the coming chapters (see Chapter 4 and 5), we present a sparse adaptive control methodology which seeks to reduce oscillations by operating with small learning rates and encouraging region-based learning. Also, Chapters 6 and 7 focus on improving transient performance and minimizing control rates of vehicles with sophisticated dynamics by using a deep recurrent architecture.

As mentioned previously, a main focus of this research is to improve direct model reference adaptive control (MRAC) methods through the use of neural networks. Traditionally, there have been two dominant approaches to neural network based adaptive flight control: structured neural networks and unstructured neural networks. A structured (linear-in-the-parameters) neural network approach provides adaptive controllers with a universal basis by fixing the inner layer weights and activation functions while updating only the outer layer weights with an adaptive update law [13]. A typical structured neural network approach for flight control utilizes a Gaussian Radial Basis Function (RBF) Neural Network where RBFs are generally spread out across the input space with fixed centers and widths. The drawback of this approach is that the number of RBFs needs to increase significantly as the input space increases [14]. A single hidden layer (SHL) neural network is an unstructured approach (nonlinearly parameterized neural network) where both the inner layer weights and outer layer weights are updated concurrently. Although more complicated, SHL based adaptive controllers often have better transient performance and are easier to tune compared to the RBF networks [14, 15]. However, RBF neural networks contain local support that allows for more desirable learning structure [16]. In both approaches, in order to ensure uniformly ultimately boundedness of the tracking error and boundedness of the adaptive weights, robust adaptive control modifications must be applied (e.g. projection, dead-zone) to the update laws which ensure stability even if the persistence of excitation condition is not satisfied [17, 18]. In

order for an adaptive controller to cancel the uncertainty precisely and have the adaptive weights to converge to their ideal values, the states of the system must be persistently exciting (PE) [19]. Unfortunately, this condition is often not met in flight control and is difficult to verify [16, 20].

For both RBF and SHL adaptive systems, the selection of adaptive learning rates and the number of hidden nodes is paramount. Both selections have trade-offs that significantly affect tracking performance and are areas of active research [16]. Recently, a performance comparison between SHL and RBF based controllers in [15] showed that, for a specified constant learning rate, there is an optimal number of hidden nodes such that the norm of the tracking error does not significantly decrease by adding additional nodes to the system. Often for flight control applications, the number of nodes is less than ideal and selected based on computational constraints [21]. Another well-known trade-off in adaptive control, specifically MRAC, is the selection of the learning rates. Higher learning rates correspond to reducing the tracking error more rapidly [13, 17, 22]. However, high learning rates can cause high-frequency oscillations in systems with unmodeled dynamics [23]. Hence, there exists a significant trade-off between robustness and tracking performance.

In the adaptive control community, neural network-based adaptive controllers are predominately initialized to small random numbers then updated according to adaptive update laws discovered through Lyapunov analysis. In other words, we are handcuffed in designing weights for the neural network based on the stability analysis. This methodology is used by two of the most popular authors in neural network adaptive control of flight vehicles, Lavretsky [16] and Calise [24]. It is also worth noting that there has been some results which allow the neural network weights to be designed off-line under certain restrictions [25]. In addition, using indirect adaptive control and multiple model philosophy, Chen and Narendra proposed an alternate approach which switches

between a nonlinear neural network and a robust linear control design [26] which has more flexibility in design.

Although not the focus of this research, recently there have been many enhancements to the MRAC architecture that aim to improve transient performance without satisfying the PE condition. Concurrent learning is one developed approach which focuses on weight convergence by using current and recorded data during adaptation to improve learning performance and, under certain conditions, guarantees exponential tracking and ideal weight convergence even without satisfying the persistence of excitation (PE) condition [18]. Another approach called L_1 adaptive control focuses on instantaneously dominating uncertainty through fast adaptation by using high adaptive gains and employing a low pass filter at the controller output [27]. In addition, there has been much effort to establish stability and performance metrics for adaptive flight control through the use of verification methods [28].

Recently, there has been increased research interest in the area of switched nonlinear systems, fuzzy logic, intelligent control, and neural networks due to numerous breakthroughs in Lyapunov based stability methods for switched and hybrid systems [29,30]. For instance, neural network based fuzzy logic techniques have been developed for MRAC systems which aims to modify the reference model online in order to improve transient performance through the use of a supervisory loop [31]. Additionally, adaptive neural networks have recently been used in both SHL and RBF networks in order to augment switched robust baseline controllers for robot manipulator and unmanned aerial vehicles [32, 33]. Moreover, control methodologies have been developed for hypersonic and highly nonlinear flight vehicles using fuzzy logic and switched tracking controllers [32, 34]. Furthermore, the stability of adaptive RBF networks with dwell time conditions that dynamically add and remove nodes was investigated for systems that include switched dynamics [35, 36].

As summarized in Section 1.3, Chapters 4 and 5 develop a sparse neural network switched nonlinear control approach to MRAC. Additionally, Chapter 3 provides additional background and detail on neural network based MRAC.

1.1.2 Policy Search and Deep Learning

Over the past decade, there have been tremendous breakthroughs in machine learning. Many of these breakthroughs derive from deep learning methods. Deep architectures learn complex representations of data sets in multiple layers where each layer is composed of a number of processing nodes. Deep learning succeeds by discovering complex structure in high dimensional data where each layer strives to find hidden and low dimensional features [37]. In order to do so, deep learning architectures are typically, at least, three to five layers deep. The deeper the design, the more complex of a function the algorithm can learn. Deep learning has become prevalent in industry where several companies (e.g. Apple, Microsoft, Google, Facebook, Adobe) have obtained impressive performance and utility in speech recognition and face recognition tasks and applications (see [37, 38]).

Much of the literature surrounding the utilization of deep learning methods for control of dynamical systems is found in the reinforcement learning field within machine learning. The goal of reinforcement learning is to develop methods to sufficiently train an agent by maximizing a cost function through repeated interactions with its environment [39].

1.1.2.1 Reinforcement learning

The majority of reinforcement learning based methods focus on a dynamic programming based approach to control. These methods are either model-based or model-free. Model-free based methods are typically based on temporal difference learning algorithms such as Q-learning or SARSA where the value function is estimated from repeated interactions with the environment. Alternatively, model-based methods use a model of the system dynamics and dynamic programming to compute a control

policy that minimizes a value function. In both cases, the value function is pre-defined. In small discrete spaces these algorithms work well, but in more realistic scenarios (continuous and large) these methods suffer from the ”curse of dimensionality” [40]. That is, discretization scheme results grow exponentially in the dimension of space.

Policy search is a subfield of reinforcement learning that seeks to find the best possible parameters of a controller in a particular form (e.g. deep neural network) such that it optimizes a pre-defined cost function [39]. Optimization of the controller parameters is often performed using gradient-based policy updates where the gradients are either computed analytically or numerically (e.g. finite difference methods). In order to compute analytic gradients; the cost function, policy, and plant dynamics are required to be differentiable. In contrast to dynamic programming, policy search methods handle high-dimensional spaces well, and they do not suffer from the ”curse of dimensionality.” Similar to dynamic programming, there exist both model-based and model-free policy search methods. Model-free methods learn policy parameters by interacting with the environment through the use of sample trajectories. Model-based policy search methods typically learn the plant dynamics through repeated interactions with the environment and then subsequently use the learned dynamics to train the parameters of the controller internally. Similar to adaptive control literature, if the trained parameters of the controller are obtained based on internal simulations; the resulting controller often lacks robustness to modeling errors. By adding noise and uncertainties to the system through probabilistic models or direct injection leads to improved robustness and a smoother objective function which allows the parameters to avoid local minima [39].

The research described in this dissertation is inspired by the recent work of Levine and Koltun [41] and Sutskever [42]. Levine and Koltun create a ”Guided Policy Search” framework that uses trajectory optimization in order to assist policy learning and avoid parameter convergence to poor local minima. Levine recently explored training deep learning architectures using the methodology established in guided policy search [43]

and applied those controllers to sophisticated robotic tasks. Sutskever [42] also explored deep learning based policy search methods, but his research focused on the benefits of using Hessian-free optimization. He found that including disturbances and time delays in training samples during optimization of his deep recurrent network led to improved robustness for simple systems.

The idea of Lyapunov funnels has also been a main influence on our work. Funnels for control policies have been used in robotics and nonlinear control to provide certificates of convergence from a large set of initial conditions to a specified goal region [44]. Recently, Tedrake et al. [45] used tools from sum of squares (SOS) programming to estimate regions of attraction for randomized trees stabilized with LQR feedback. SOS programming leverages convex optimization to allow the control designer to check positivity and negativity of Lyapunov functions for polynomial systems [46]. SOS programming has strong connections to robustness analysis of nonlinear systems and has become a popular tool for analyzing stability for time-invariant systems [47]. In addition, a number of powerful computational tools for SOS programming have been developed including the direct computation of Lyapunov functions for smooth nonlinear systems [48]. Majumdar and Tedrake [49] used SOS programming to design sequential controllers along preplanned trajectories while explicitly aiming to maximize the size of the funnels during design. One shortcoming of these methods is they do not explicitly focus on improving the robustness of the controller. In fact, as discussed by the author, uncertainties and disturbances can cause stability and performance guarantees to be violated in practice [50].

Even though deep learning controllers have shown great promise in completing robotic control tasks, they often require a large number of computations at each time step, do not possess standard training procedures, and have few analysis tools for the resulting control design. In addition, deep learning controllers are often optimized without regard to robustness with the assumption that the optimal policy for the learned

model corresponds to the optimal policy for the true dynamics (i.e. the certainty-equivalence assumption) [39]. Chapters 6 and 7 aim to develop methods that combat these limitations.

1.1.2.2 Robust deep learning controller analysis

Traditionally in flight control, gain and phase margins have been required for linear time-invariant (LTI) based control laws for flight systems in order to provide sufficient robustness from uncertainties and unmodeled dynamics. For nonlinear systems, especially adaptive systems, control designers can utilize time delay margins and an alternative form of gain margins as important robustness metrics [28]. Time delay margin can be defined as the amount of time delay that the closed-loop system can handle without resulting in instability. We will assume that the time delay enters the system through the control signal. Although recent research has attempted to establish fundamental methods for computing time delay margin, it is still quite popular to compute time delay margin using simulations.

There have been a number of authors that have explored robust control synthesis based on ROA estimation. The vast majority of these authors rely on Lyapunov based methods for ROA estimation (e.g. sum of squares) with controllers in polynomial or linear form. For instance, Dorobantu et al. [47] have developed a methodology to train linear controllers to be robust to parametric uncertainties by iterating between using sum of squares (SOS) and nonlinear optimization. Theis [51] used SOS methods to find control Lyapunov functions (CLFs) for polynomial systems with bounded uncertainties. Kwon et al. [52] used SOS methods to determine how to gain schedule a boosted missile. Even though many popular and successful computational tools for SOS programming have been developed, including the direct computation of Lyapunov functions for smooth nonlinear systems [48], Lyapunov-based methods have a number of drawbacks. For example, Lyapunov based ROA estimates often lead to results that

are extremely conservative, restricted to polynomial models, and are negligent of system limitations (e.g. actuator saturation) [53].

In Chapter 7, we explore using reachable set analysis to estimate the ROA of an equilibrium point of our closed-loop system. Reachability analysis is used to find the exact or over-approximate set of states that a system can reach, given an initial set of states and inputs. It is well-known that the exact reachable set of hybrid and continuous systems can only be computed in special cases. Hence, typical methods over-approximate by using geometric methods. Popular in the safety assurance community, reachable set methods can be used to guarantee the avoidance of unsafe states while assuring convergence to desirable equilibrium points [54].

1.2 Overall Contribution

The goal of the research included in this dissertation was to develop sparse and deep learning-based nonlinear and adaptive control methods which directly target HSV control challenges. In order to do so, we used two separate control methodologies: model reference adaptive control and policy search-based deep learning. Using model reference adaptive control as the framework, we developed a sparse neural network-based adaptive controller which reduces the computational burden on the processor while drastically improving learning and tracking performance on control tasks with persistent region-based uncertainties. Alternatively, we developed an innovative off-line training procedure for a deep learning controller (DLC) that simultaneously trains for performance and robustness. By using a sparsely connected DLC, we show significant improvement in parameter convergence while also reducing the number of computations required at each time step. Both architectures were evaluated and analyzed using a sophisticated hypersonic control model.

1.3 Chapter Descriptions

Following this chapter, we provide two background chapters which aim to provide a sufficient background in deep learning and model reference adaptive control

(MRAC) (Chapter 2 and Chapter 3). An overview which includes key contributions of the remaining chapters are given below.

1.3.1 Chapter 4: Improving Learning of Model Reference Adaptive Controllers: A Sparse Neural Network Approach

The contribution of this chapter is a novel approach to adaptive control that uses sparse neural networks (SNN) to improve learning and control of flight vehicles under persistent and significant uncertainties. The SNN approach is proven to enhance long-term learning and tracking performance by selectively modifying a small portion of weights while operating in each portion of the flight envelope. This results in better controller performance due to the better initialization of the weights after repeated visits to a specific portion of the flight envelope. Flight control simulations show quite impressive results generated from the SNN approach against the traditional single hidden layer (SHL) and radial basis function (RBF) based adaptive controllers.

1.3.2 Chapter 5: A Sparse Neural Network Approach to Model Reference Adaptive Control with Hypersonic Flight Applications

This chapter expands on the progress made in the previous chapter. We provide three key contributions that lead to significantly improved tracking performance based on simulation results and a uniformly ultimately bounded (UUB) Lyapunov stability result with a dwell time condition. First, we develop adaptive control terms which mitigate the effect of an unknown control effectiveness matrix on the baseline, adaptive, and robust controllers. Secondly, we derive a robust control term which is used to calculate a dwell time condition for the switched system. The inclusion of the robust control term along with a newly derived dwell time condition is used to ensure safe switching between segments. In our work, the robust control term is only activated when the norm of the tracking error exceeds preset bounds. While inside the error bounds, we disable the robust controller in order to allow the SNN to maximize learning and control the vehicle more precisely. Lastly, we investigate neural network adaptive control laws produced by increasing the order of the Taylor series expansion around the hidden layer

of the matched uncertainty. In addition to the previously mentioned developments, we demonstrate the performance of the SNN adaptive controller using a hypersonic flight vehicle (HSV) model with flexible body effects. For comparison, we also include analysis results for the more conventional SHL approach.

1.3.3 Chapter 6: Development of a Robust Deep Recurrent Network Controller for Flight Applications

The main contribution of this chapter is the development of an optimization procedure to train the parameters of a deep recurrent network controller for control of a highly dynamic flight vehicle. We train our recurrent neural network controller using a set of sample trajectories that contain disturbances, aerodynamic uncertainties, and significant control attenuation and amplification in the plant dynamics during optimization. In addition, we define a piecewise cost function that allows the designer to capture both robustness and performance criteria simultaneously. Inspired by layer-wise training methods in deep learning, we utilize an incremental initialization training procedure for multi-layer recurrent neural networks. Next, we compare the performance of the deep RNN controller to a typical gain-scheduled linear quadratic regulator (LQR) design. Finally, we demonstrate the ability of the controller to negate significant uncertainties in the aerodynamic tables while remaining robust to disturbances and control amplification/attenuation.

1.3.4 Chapter 7: Development of a Deep and Sparse Recurrent Neural Network Hypersonic Flight Controller with Stability Margin Analysis

In this chapter, we present a novel approach for training and verifying a robust deep learning controller (DLC) for a highly dynamic hypersonic flight vehicle. We leverage a sample based trajectory training methodology established in Chapter 6 for training and optimization of the DLC weights. In order to design a sufficiently robust and high-performing controller, the controller utilizes a training set with a high number of training samples which contain varying commands, uncertainties (e.g. time delay, control amplification/attenuation), and disturbances (e.g. flexible body effects) in each

training sample. The training phase allows the designer to include disturbances and uncertainties that are not explicitly included in the dynamics of the flight vehicle but are anticipated based on region-specific models. Next, we extend the sparse neural network (SNN) adaptive controller architecture developed in Chapter 4 into a deep neural network framework. We use this innovative sparse deep learning controller (S-DLC) architecture to reduce the computation load on the processor and significantly improve parameter convergence. By recognizing connections of the GRU module to feed-forward networks, we develop a systematic training procedure which improves optimization of the DLC. Additionally, robustness metrics of the DLC are verified through simulation by using ROA estimation via forward reachable sets. Lastly, we analyze the results of the optimization and provide simulation results.

CHAPTER 2
BACKGROUND: DEEP NEURAL NETWORKS

Since the establishment of machine learning methods, there has been a group of techniques referred to as supervised learning. The idea behind supervised learning is to adjust a set of parameters (weights) in order to reduce some predefined error (cost function) based on labeled training data. That is, during the training phase, the supervised learning algorithm is fed input data (e.g. images) and known outputs (target values). The idea is to adjust the system weights in order to find a (local) minimum of the cost function using a chosen optimization method. Typically, researchers employ batch methods (e.g. L-BFGS) or stochastic gradient descent (SGD) to solve unconstrained optimization problems with deep architectures. During implementation, batch methods are far less common due to the fact that they require the entire training set of data in order to compute the value and gradient of the cost function. Another popular research area in the deep learning community is unsupervised learning. Unsupervised learning methods make use of unlabeled data in order to discover structure in the data [37, 55].

Neural networks is a branch of machine learning that became popular in the 1980s. A neural network is a biologically inspired mathematical architecture which can be described as an input-output map. This network contains a large number of neurons where each neuron is represented by an activation function (e.g. sigmoid, tanh, or linear rectifier). Each neuron (activation function) produces a single output based on a nonlinear transformation of the input. Neural networks became prevalent in machine learning after the discovery of backpropagation. Backpropagation is an extension of the chain rule and can be implemented on a multi-layer neural network to compute the gradients of that network [55]. In the 1990s, many researchers were led away from neural network research due to insignificant and inconsistent results while using multi-layer neural networks. Soon after, support vector machines (SVM) were discovered

and were employed more often due to several reasons. Most importantly, multi-layer networks often converged to saddle points and local minima while SVMs were typically formulated as convex optimization problems with global minimums. Secondly, neural networks often have issues over-fitting the data which leads to low errors during the training phase and large errors during testing or execution phase. At the time, that led to SVMs outperforming multi-layer neural networks on most tasks [37].

In the early 2000s, improvements in computational power, parallel computing, automatic gradient computation engines, and large databases of labeled training data led to the re-emergence of deep neural networks. Shortly after, in 2006, there was a breakthrough in deep learning research. A successful method for training the deep learning architectures was discovered, called greedy layer-wise training. The main idea behind this approach is simple; incrementally train each layer of the network; then, after all of the layers are trained, “fine-tune” the network by re-tuning all the weights concurrently. In the machine learning community, often many initial layers are “pre-trained” using unsupervised learning methods. After the network is “pre-trained”, the weights have a better initialization in the parameter space than if they had been randomly initialized. This typically results in the optimization algorithm converging to a “better” local minimum because it has some information about the system a priori [55]. For smaller labeled datasets, unsupervised pre-training helps to prevent over-fitting and leads to better generalization [56]. In practice, digital signal processing (DSP) techniques (e.g. Fourier transforms and cepstral analysis) are used to train lower layers of the deep learning network in an unsupervised fashion before end-to-end training. Due to the many breakthroughs in deep neural network training, Gaussian HMM-based statistical methods, which have been used for decades, are being replaced on the most popular commercial speech recognition systems (e.g. Apple, Microsoft, Google, Facebook, Adobe) [37, 57].

Sparse neural network methods have also contributed to the recent success of deep learning. One example is the widespread unsupervised learning method called the sparse autoencoder. For the sparse autoencoder, a neural network is trained to produce an output which is identical to the input while limiting the average activation value for each node in the network. By utilizing this technique, only a small percentage of neurons are impacted when optimizing weights for each training example. This leads to faster convergence and a better classifier for image and text tasks [58]. Another example of sparsity in deep learning can be found in the most successful activation functions. For instance, linear rectifiers as activation functions are more biologically plausible due to their natural sparsity characteristics. This leads to superior performance on learning tasks compared to their sigmoidal or hyperbolic tangent counterparts [58]. Furthermore, in 2015, recent research for initialization of ReLU based deep architectures was provided in [59]. After random initialization, linear rectifiers hidden units are active approximately 50% of the time. In addition, recently developed sparse techniques such as the maxout function [60] and closely related channel-out function [61] have found tremendous success on classification problems.

More recently, due to the popularity of deep learning, there has been significant research effort in applying deep learning to solve time-series problems, specifically, using recurrent neural networks (RNNs). RNNs are neural networks that process inputs sequentially where the hidden nodes keep a history of the past states. Recurrent neural networks are most frequently trained using backpropagation through time (BPTT) or real-time recurrent learning (RTRL). In comparison to standard feed-forward neural networks, recurrent neural networks are even more difficult to train and have large susceptibility to local minima and saddle points. This has been attributed to the difficulty that recurrent neural networks have with learning long-range temporal dependencies, also known as the well-studied vanishing gradient problem. Fortunately, there have been notable improvements in training recurrent neural networks. Firstly, long short-term

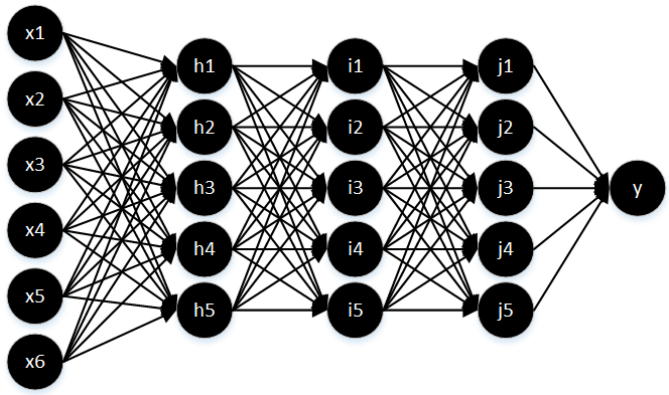

Figure 2-1. Example of a multi-layer feed-forward neural network.

memory (LSTM) and gated recurrent units (GRU) are modularized architectures for recurrent neural networks which have proven to provide significant benefits in terms of performance by addressing the vanishing gradient problem. Hessian-free optimization was created to address that same issue [42]. Echo-state machines is another alternative which side-steps the vanishing gradient problem.

The rest of this chapter is dedicated to providing a general background in neural networks and deep learning while providing references for the reader. Chapter 4 describes a sparse neural network (SNN) adaptive control architecture inspired by deep sparse neural network literature. Chapter 6 aims to extend deep learning based policy search methods to the complicated dynamics of high-speed flight while considering robustness properties of the controller during optimization. Chapter 7 extends the work in Chapter 6 to include hypersonic control and improves the controller's robustness and tracking performance by employing a sparse connectivity scheme.

2.1 Deep Multi-layer Neural Networks

Algorithm 2.1 Forward Propagation

1: $a_i = data$
2: **for** $(i = 1 : numHidden + 1)$ **do**
3: $\quad z_{i+1} = W * a_i + b_i$
4: $\quad a_{i+1} = f(z_{i+1})$
5: **end for**
6: $y = a_{i+1}$

Algorithm 2.2 Backpropagation

```
for (l = numLayers : −1 : 1) do
    if (i = lastLayer) then
        δ_l = −(y − labels) ∗ f′(z_l)
    else
        δ_l = ((W^T δ_{l+1}) ∗ f′(z_l)
    end if
    dJ/dW(l) = δ_{l+1}(a_l)^T
    dJ/db(l) = δ_{l+1}
end for
```

The simplest form of multi-layer network is illustrated in Figure 2-1 and is often referred to as a feed forward multi-layer neural network. The network is trained (see Algorithm 2.1) using an optimization algorithm (e.g. stochastic gradient descent or L-BFGS) and a method called backpropagation (see Algorithm 2.2). Backpropagation determines the derivative of the cost function, usually denoted J, with respect to the weights using the chain rule (i.e. dJ/dW). The network shown in Figure 2-1 has five layers: an input layer, three hidden layers, and an output layer. As stated in the introduction, the deeper the network the more complex of a function the neural network can represent. Generally for time-series problems, each input node represents one-step back in the time history of that signal, which is referred to as a time-delayed neural network. For each node added to the network, more weights are needed to connect to the internal nodes. This results in expanding memory for each time history sample added. Hence, time delay multi-layer feed-forward networks are generally restricted to a small amount of time history.

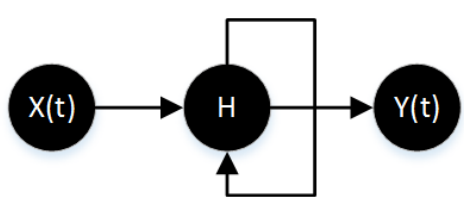

Figure 2-2. Example of a single hidden layer recurrent neural network.

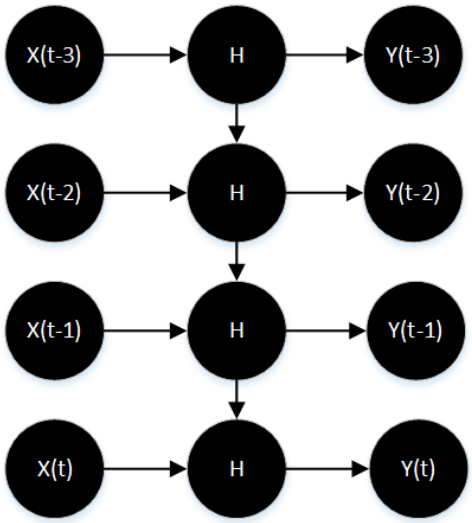

Figure 2-3. Example of an expanded recurrent neural network.

2.2 Deep Recurrent Neural Networks

Recurrent networks (RNN) have had outstanding success in addressing time series problems in recent history. An example diagram of an RNN can be seen in Figure 2-2. RNNs have a natural way of incorporating time history of states into the network structure. Recurrent networks also have the benefit of operating with a much smaller number of parameters than feed-forward networks. The expanded view of the RNN can be seen in Figure 2-3, where the current time-step is fed data from previous time steps. Generally, deep recurrent networks attach several hidden layers before resolving the output. Similar to feed-forward networks, the derivative of the cost function with respect to the weights (i.e. $\frac{dJ}{dW}$) is obtained by using the chain rule after running forward propagation to determine the cost (see Algorithm 2.3). For recurrent networks, this is called backpropagation through time (see Algorithm 2.4). Recently, RNNs have set new benchmarks in speech recognition by using deep bidirectional long short-term memory (LSTM) based networks [62].

Traditional recurrent neural networks often struggle with capturing information from long-term dependencies. This struggle is coined the ”vanishing gradient” problem and also occurs with very deep feed-forward networks [63]. The struggle originates

when calculating the derivative of the cost function with respect to the weights (e.g. BPTT). Regularization, RNN architecture (e.g. long short-term memory (LSTM) or gated recurrent unit (GRU)), and optimizers such as Hessian-free optimization have addressed the long-standing issue of vanishing gradients. A similar issue of exploding gradients can also occur during training, but recently a method of clipping the gradient has been proven to mitigate that issue [64].

2.2.1 Memory Modules for Recurrent Neural Networks

2.2.1.1 Long short-term memory (LSTM)

In standard recurrent neural networks, the hidden state is given by

$$s_t = f(Ux_t + Ws_{t-1}) \tag{2–1}$$

where x_t is the current state, s_{t-1} is the previous hidden state, and $(U,\ W)$ are tunable parameters of the network. As stated earlier, this structure has difficulty in learning long-term dependencies due to the vanishing gradients problem. In literature, LSTM is often mislabeled as a new architecture for recurrent neural networks. In fact, LSTM is a module which replaces how to update the hidden state using a gated mechanism [65]. See the equations stated in Algorithm 2.5, where each gate has corresponding weights that determine how much previous information should be kept and how much should be forgotten [63]. In addition to the gated mechanism, LSTM has an internal memory, c_t, and an output gate, o, which determines how much of the internal memory is provided to the next module. LSTM requires more memory than the traditional implementation and more weights to tune but is simple to implement and effective.

Algorithm 2.3 Forward Propagation (RNN)

```
for t = 1 : T_f do
    for i = 1 : numLayers do
        s_t^i = f(U^i * a_t^i + W^i * S_{t-1}^i + b^i)
    end for
    y_t = V * S_t^i + b^{i+1}
end for
```

Algorithm 2.4 Backpropagation through time (BPTT)

for $(i = T_f : -1 : 1)$ **do**
  $dJ/dU = dJ/dx * dx/du$
  $dU/dz = f'(u)$
  $dJ/dz_2 = dJ/dy * dy/dz$
  $dJ/ds = (V' * dJ/dz_2') + dJ/dS$
  $dJ/dV = (dJ/dz_2 * s_i) + dJ/dV$
  $dS/dz_1 = f'(s_i)$
  $dJ/dz_1 = dJ/dS * dS/dz$
  $dJ/dS^{i-1} = W^j * dJ/dz_1$
  for $(j = numHidden : -1 : 1)$ **do**
    $dJ/dU_j = dJ/dz_1 * s_i^{j+1} + dJ/dU_j$
    $dJ/dW_j = dJ/dz_1 * s_{i-1}^{j} + dJ/dW_j$
  end for
end for
$dJ/db_{j+1} = \Sigma\Sigma dJ/dz_2$
$dJ/dS0_{j+1} = \Sigma W'_{j+1} * (dJ/dS * dS/dz_{j+1})$
for $j = numHidden : -1 : 1$ **do**
  $dJ/dS0_j = \Sigma W'_j * (dJ/dS_j * dS/dz_{j+1})$
  $dJ/db_j = \Sigma\Sigma dJ/dS_j * dS/dz_{j+1}$
end for

2.2.1.2 Gated recurrent unit (GRU)

Gated Recurrent Units (GRU) were recently discovered modules used in a similar fashion to LSTM but require less memory and have different structure [66], see Algorithm 2.6. The two gates in the GRU structure determine the trade-off between new information and past information. Notice that the GRU does not have internal memory. We will utilize GRU modules in our deep architecture used for flight control, see Chapters 6 and 7.

2.2.2 Deep Recurrent Network Architectures

In addition to optimization algorithm research, memory modules, and layer-wise training, another significant area of research in the last few years lies in determining the most productive recurrent network architectures. Pascanu et al. [67] explored different recurrent neural network architectures and found that the deep stacked recurrent neural network worked best for the majority of the tasks (see Figure 2-4). Graves et al. [62]

determined deep bidirectional recurrent neural networks to be effective for speech recognition problems. In Chapters 6 and 7, we will provide a more detailed description of the stacked recurrent neural network.

2.3 Optimization

Optimization is a necessary tool for machine learning algorithms in order to minimize a predefined cost function that is specified by the user. A few popular optimization routines for unconstrained optimization problems found in deep learning are described below, but the reader is encouraged to see Ngiam et al. [68] and Sutskever [42] for a more detailed description and comparison.

2.3.1 Gradient Descent

One of the most fundamental and simple ways to solve a function minimization problem described previously is gradient descent. Gradient descent is a first-order optimization algorithm which is easily described by (2–2), where a small gain, α, is used to step in the direction of the negative gradient. The smaller the gain, the longer it will take to converge. If the gain is too large, the algorithm may diverge (overstep).

$$\Theta_t = \Theta_{t-1} - \alpha \nabla_{\Theta} E[J(\Theta)] \tag{2–2}$$

2.3.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is by far the most popular optimization method used for deep learning due to its speed and effortless implementation. SGD has the advantage over batch methods, like traditional gradient descent, because it does not

Algorithm 2.5 Long Short-Term Memory (LSTM)

$i = \sigma(x_t U^i + s_{t-1} W^i)$
$f = \sigma(x_t U^f + s_{t-1} W^f)$
$o = \sigma(x_t U^o + s_{t-1} W^o)$
$g = tanh(x_t U^g + s_{t-1} W^g)$
$c_t = c_{t-1} * f + g * i$
$s_t = tanh(c_t) * o$
Note: $*$ represents element-wise multiplication

Algorithm 2.6 Gated Recurrent Units (GRU)

$u = \sigma(x_t U^u + s_{t-1} W^u)$
$r = \sigma(x_t U^r + s_{t-1} W^r)$
$h = tanh(x_t U^h + (s_{t-1} * r) W^h)$
$s_t = (1 - z) * h + z * s_{t-1}$
Note: $*$ represents element-wise multiplication

require the entire training set to make parameter adjustments. For very large datasets, batch methods become slow. For SGD, the gradient of the parameters is updated using only a single training example, see (2–3). In practice, a subset of the original training set is chosen at random to perform the update. SGD has the reputation of leading to a stable convergence at a speedy pace. Unfortunately, SGD comes with drawbacks. For instance, the learning rate has to be chosen by the user and can be difficult to determine. Fortunately, there has been a tremendous amount of research for SGD to improve convergence. Methods include adaptively changing the learning rate or simply decreasing it based on some pre-defined schedule [55]. In addition, momentum methods can be applied in order to accelerate in direction of the gradient and consistently reduce the cost function more quickly. Traditional momentum equations are stated in Algorithm 2.7. Unfortunately, this introduces another tunable parameter (λ) which determines how much gradient information from the past is used on each update and is often referred to as the learning rate.

$$\Theta_t = \Theta_{t-1} - \alpha \nabla_\Theta J(\Theta; x_i, y_i) \tag{2–3}$$

Recently, Nesterov's accelerated gradient algorithm has been shown to improve convergence for deep recurrent neural networks [42]. Nesterov's algorithm [69], seen in Algorithm 2.8, additionally requires a momentum schedule (μ) and adaptive learning rate (ϵ).

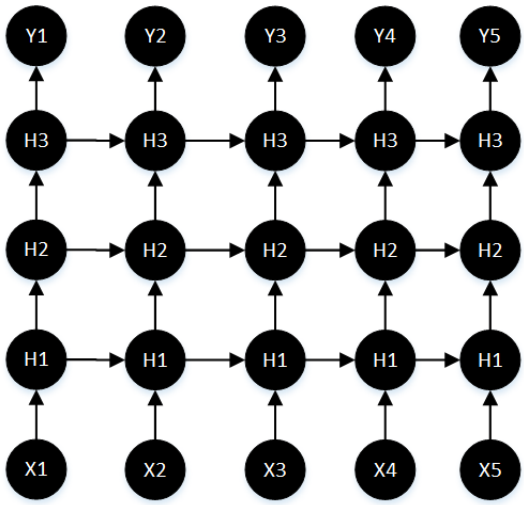

Figure 2-4. Simplified stacked recurrent neural network (S-RNN) architecture.

2.3.3 Broyden-Fletcher-Goldfarb-Shanno (BFGS)

Newton's method is an iterative method that uses the first few terms in the Taylor series expansion to find roots of a function (i.e. where $f(x) = 0$). In optimization, this concept is used to find the roots of the derivative of the function, f. That is, Newton's optimization method (see Algorithm 2.9) is used to find the stationary points of f which are also the local minima and maxima of the function. This is often described as fitting a quadratic function around point x and then taking steps toward the minimum of that quadratic function. The issue with Newton's method for optimization is that it requires an analytical expression for the Hessian which is often computationally expensive to compute and requires the Hessian to be invertible. For these reasons most second-order methods, called quasi-newton methods, solve unconstrained optimization problems by estimating the inverse Hessian matrix.

Broyden-Flectcher-Goldfarb-Shanno (BFGS) is a quasi-newton second-order batch method which can be used for finding extrema. BFGS is often implemented as L-BFGS which stands for limited memory BFGS. BFGS is one type of quasi-newton method that solves unconstrained nonlinear optimization problems by estimating the inverse Hessian (see Algorithms 2.10 and 2.11). A comparison of quasi-Newton methods, details of

implementation and test results can be seen in [70]. We will utilize L-BFGS optimization in Chapters 6 and 7.

Algorithm 2.7 Momentum

$v_t = \lambda v_{t-1} - \alpha \nabla_\Theta J(\Theta; x_i, y_i)$

$\Theta_t = \Theta_{t-1} + v_t$

Algorithm 2.8 Nesterov's Momentum

$v_t = \mu_{t-1} v_{t-1} - \epsilon_{t-1} \nabla f(\Theta_{t-1} + \mu_{t-1} v_{t-1})$

$\Theta_t = \Theta_{t-1} + v_t$

$u_t \cong 1 - (3/t + 5)$

Algorithm 2.9 Newton's Method for Optimization

$g = \nabla f(x_{t-1})$

$H = \nabla^2 f(x_{t-1})$

$x_t = x_{t-1} - H^{-1} g$

Algorithm 2.10 BFGS Update

Δg_t=$\nabla f(x_t) - \nabla f(x_{t-1})$

$\Delta x_t = x_t - x_{t-1}$

$\rho_t = (\Delta g_t^T \Delta x_t)^{-1}$

$H_{t+1}^{-1} = (I - \rho_t \Delta g_t \Delta x_t^T) H_t^{-1} (I - \rho_t \Delta x_t \Delta g_t^T) + \rho_t \Delta x_t \Delta x_t^T$

Algorithm 2.11 Broyden, Fletcher, Goldfarb, Shanno (BFGS) Minimization

$min_{H^{-1}} ||H_t^{-1} - H_{t-1}^{-1}||^2$

$s.t.\ H_t^{-1} \Delta g_t = \Delta x_t$

H_t^{-1} is symmetric

CHAPTER 3
BACKGROUND: MODEL REFERENCE ADAPTIVE CONTROL

Adaptive control was an early innovation in the development of flight control. Aircraft and other flight vehicles are often required to operate in dynamic flight envelopes that span vastly different speeds, altitudes, and dynamic pressures. In addition, the flight vehicle is subjected to numerous dynamic disturbances throughout its flight. In contrast to many robotic systems, flight controllers are often required to follow predetermined trajectories or guidance laws. Therefore, it is beneficial for that system to adhere to pre-defined transient performance metrics. For many types of control systems, these metrics are embedded in baseline controllers [16]. That is, baseline controllers are often required to possess certain performance and robustness properties for flight control applications. Often these controllers are designed using a gain-scheduled controller whose gains are adjusted based on the current operating condition of the flight vehicle. In order to determine controller gains across the flight envelope, the flight vehicle's model is linearized about selected trim points. At each trim point, a linear controller is developed based on transient and robustness criteria. Linear quadratic regulator (LQR) is a proven optimal control technique that gives the control designer independent variables (Q and R matrices) to tune and tweak performance. The designer usually looks at metrics based on "loop shaping," which is a mechanism for designing controllers in the frequency domain. These metrics include margins (gain and phase), singular values, rise-time, and sensitivity functions (e.g. "gang of six") [71]. Gain-scheduled LQR controllers often remain robust to time-state dependent nonlinear uncertainties that exist through the control channel (matched). Unfortunately, in the presence of these uncertainties and significant nonlinearities, the baseline performance of the system is degraded [16]. In most modern control systems, this degradation is overcome by the use of adaptive controllers.

Generally, adaptive controllers are designed by creating update laws based on the closed-loop errors of the system. Lyapunov-based stability analysis is used to make guarantees in terms of stability, boundedness of adaptive weights, and tracking convergence. Ubiquitous in the aerospace industry, a type of design called Model Reference Adaptive Control (MRAC) is used to improve the robust linear control baseline controller by adding an adaptive layer. Most commonly, MRAC is implemented so that the adaptive control portion of the control is only active when the baseline performance is degraded. Traditionally, there are two different types of MRAC: direct and indirect [19]. For direct adaptive control, control gains are adapted directly to enforce the closed-loop tracking performance. For indirect adaptive control, the controllers are designed to estimate the unknown plant parameters on-line then use their estimated values to calculate controller gains. As stated above, adaptive control is used to drive the defined system error to zero even when the parameters of a system vary. These parameters do not necessarily converge to their true value when the error is driven to zero. In order for convergence to occur, the persistence of excitation condition must be met. To explain this succinctly, the persistence of excitation requires the control input to exhibit a minimum level of spectrum variability in order for parameters to converge to their true value [18].

Traditionally, MRAC problems are conceived assuming the structure of the uncertainty is known and can be linearly parameterized using a collection of known nonlinear continuous functions (regression matrix) and unknown parameters. For problems where the structure of the uncertainty is unknown, universal approximation properties of neural networks can be exploited in adaptive controllers to mitigate the uncertainties of the system within certain tolerances over a compact set [16]. One of the most successful implementations of a neural network based adaptive controller in practice was implemented on several variants of the Joint Direct Attack Munitions (JDAM) which has been developed by Boeing and has had numerous successful flight tests [72]. This flight vehicle operates using a direct model reference adaptive control (MRAC) architecture

where robust modifications (e.g. sigma modification) are designed to keep the neural network weights within pre-specified bounds.

This chapter is dedicated to providing a background in model reference adaptive control (MRAC). Chapter 4 will develop a sparse neural network (SNN) architecture for model reference adaptive control (MRAC) that encourages long-term learning. Chapter 5 builds on Chapter 4 and provides a Lyapunov stability based result based on an enforced dwell time condition.

3.1 Baseline Control of a Flight Vehicle

The majority of this section is formulated based on preliminary work published in [73]. The work is based on methodology and derivations in [16] and applied to a high-speed flight vehicle. The approach is considered state-of-the-art the flight control community and is the foundation for the MRAC based research efforts in this dissertation. The output feedback nature of this section is ignored, as this is not the focus of the dissertation but is included in the paper cited above. The control design is based on linearized mathematical models of the aircraft dynamics. The baseline controller is designed to be robust to noise and disturbances. The robustness of the baseline design is augmented with an adaptive controller in the form of a model-reference adaptive controller (MRAC).

3.1.1 Linearized Flight Dynamics Model

We assume the flight dynamics can be described by a set of ordinary differential equations

$$\dot{x} = f(t, x), \qquad x(t_0) = x_0, \qquad t \geq t_0 \tag{3–1}$$

which are composed of position, kinematic, translational and rotational equations of motion (see for example, Stevens and Lewis [74]).

We are interested in designing a baseline controller using a gain-scheduled linear quadratic regulator (LQR) approach. Hence, it is necessary to linearize the system at various flight conditions based on the modeled dynamics of the vehicle and derive linear

short period plant matrices $A_p \in \mathbb{R}^{n_p \times n_p}$, $B_p \in \mathbb{R}^{n_p \times m}$, $C_p \in \mathbb{R}^{p \times n_p}$, and $D_p \in \mathbb{R}^{p \times m}$. The flight dynamics are numerically linearized with respect to states and control inputs around each flight condition which results in

$$\begin{aligned} A_p(i,\ j) &= \left.\frac{\partial f(i)}{\partial x(j)}\right|_{\substack{x=x^* \\ u=u^*}}, \quad B_p(i,\ k) = \left.\frac{\partial f(i)}{\partial u(k)}\right|_{\substack{x=x^* \\ u=u^*}}, \\ C_p(i,\ j) &= \left.\frac{\partial x(i)}{\partial x(j)}\right|_{\substack{x=x^* \\ u=u^*}}, \quad D_p(i,\ k) = \left.\frac{\partial x(i)}{\partial u(k)}\right|_{\substack{x=x^* \\ u=u^*}}, \end{aligned} \tag{3–2}$$

where i and j are indices of the state vector, trim conditions are denoted by asterisks as x^* and u^*, and k is the index of the control input. We can now write the augmented short period linearized model as

$$\begin{bmatrix} \dot{e}_I \\ \dot{x}_p \end{bmatrix} = \begin{bmatrix} 0 & C_{reg} \\ 0 & A_p \end{bmatrix} \begin{bmatrix} e_I \\ x_p \end{bmatrix} + \begin{bmatrix} 0 \\ B_p \end{bmatrix} u + \begin{bmatrix} -1 \\ 0 \end{bmatrix} y_{cmd} \tag{3–3}$$

where $C_{reg} \in \mathbb{R}^{m \times n_p}$ selects the regulated state and $y_{cmd} \in \mathbb{R}^m$ is the bounded external command.

3.1.2 Baseline Controller

We can rewrite (3–3) to include system uncertainties $\Lambda \in \mathbb{R}^{m \times m}$ and $f(x) \in \mathbb{R}^m$ as

$$\begin{bmatrix} \dot{e}_I \\ \dot{x}_p \end{bmatrix} = \begin{bmatrix} 0 & C_{reg} \\ 0 & A_p \end{bmatrix} \begin{bmatrix} e_I \\ x_p \end{bmatrix} + \begin{bmatrix} D_{reg} \\ B_p \end{bmatrix} \Lambda(u + f(x)) + \begin{bmatrix} -1 \\ 0 \end{bmatrix} y_{cmd} \tag{3–4}$$

where $y = C_{reg} x_p$ and the integral error is defined as $\dot{e}_I = y - y_{cmd}$. The goal is to design a control input, u, such that the system output y tracks the command y_{cmd}. We can generalize (3–4) as [16]

$$\dot{x} = Ax + B\Lambda(u + f(x)) + B_{ref} y_{cmd} \tag{3–5}$$

where optimal control can be used to produce asymptotic constant command tracking proportional plus integral (PI) baseline controller. We start the baseline controller design

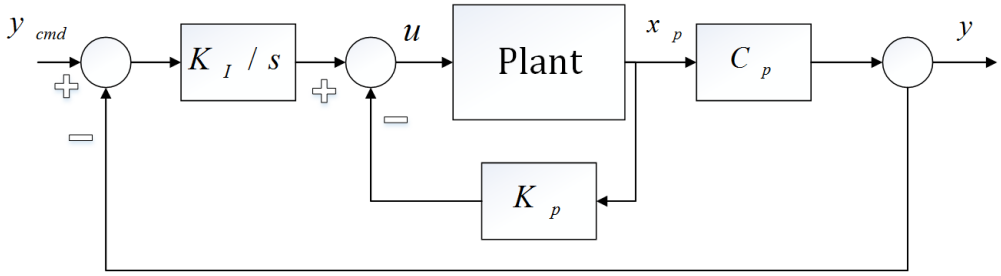

Figure 3-1. Standard LQR PI baseline controller architecture.

process by rewriting the equations above into servomechanism form given by [75]

$$\begin{bmatrix} \dot{e} \\ \ddot{x_p} \end{bmatrix} = \begin{bmatrix} 0 & C_c \\ 0 & A_p \end{bmatrix} \begin{bmatrix} e \\ \dot{x_p} \end{bmatrix} + \begin{bmatrix} D_c \\ B_p \end{bmatrix} \mu \tag{3–6}$$

where $\mu = \dot{u}$, $e = y_c - r$, c denotes the command tracking variables, and r is the command input. We can write the servomechanism design model in the form:

$$\dot{z} = \tilde{A}z + \tilde{B}u \tag{3–7}$$

where z, $\tilde{A}$, and $\tilde{B}$ correspond to the vectors in (3–6).

Optimal linear quadratic (LQ) methods enable a systematic approach to controller design. Linear quadratic regulation (LQR) can be applied in order to obtain a set of linear gains that minimize the quadratic cost defined by

$$J = \int_{t_0}^{\infty} z^T Q z + \mu^T R \mu \, dt \tag{3–8}$$

which is constrained by (3–6). Solving the algebraic Riccati equation (ARE) given by

$$\tilde{A}^T P + P\tilde{A} + Q - P\tilde{B}R^{-1}\tilde{B}^T P = 0$$

obtains the set of gains, K_c, that are used to define the control law

$$u(t) = -K_I \int_{t_0}^{t} e(\tau) d\tau - K_{x_p} x_p(t) \tag{3–9}$$

where $K_c = [K_I, \ K_{x_p}]$. The resulting controller is shown in Figure 3-1.

3.1.3 Iterative Design Loop

In order to develop a baseline controller that is robust to disturbances and achieves sufficient transient response, an iterative loop around the optimal controller is developed. Within this loop, the values within the Q and R matrices are tweaked to adjust performance. Specifically, increasing the penalty for integral error will increase the rise time. Increasing the penalty on angular rate will decrease oscillations in the step response. At each iteration, the controller increases its gains while minimizing oscillations in the time-domain. This loop iterates until the controller has to meet certain criteria. This includes evaluating the gains in a much more realistic model than the design model which includes sensors and actuators. If the design meets certain performance criteria (e.g. rise-time, overshoot, undershoot, crossover frequency, gain and phase margins) then the gains are saved and the loop exits. Margins in the MIMO sense are related to the return difference (RD) and stability robustness (SR), see [16] for more details.

It is important to note that singular value based stability margins are always more conservative than the single-loop classical margins. It is possible and can be insightful to compute the single-input single-output stability (SISO) margins of each channel of a MIMO transfer function while all other channels are closed. This analysis can provide the control designer with some information on where the weaknesses of the design are [73].

Typically, linear control designers look at designs using sensitivity analysis. Robustness analysis requires acceptable levels at both the plant input and output. The frequency responses of MIMO transfer functions are computed at each input and output channel with all other channels remaining closed. This includes the ”gang of six” transfer functions [71]. This gang is required to describe how the system reacts to disturbances, noise, and set point changes. The gang includes the return difference, stability robustness, sensitivity, and complementary sensitivity functions. Unlike MIMO systems, SISO systems have equivalent loop gains at the plant input and output where L denotes loop

gain. We define sensitivity (S), co-sensitivity (T), return difference (RD), and stability robustness (SR) in the SISO case by

$$S = \frac{I}{I+L} \tag{3–10}$$

$$T = \frac{L}{I+L} \tag{3–11}$$

$$RD = I + L \tag{3–12}$$

$$SR = I + L^{-1} \tag{3–13}$$

where I is the identity matrix of sufficient size. Generally, designers want to design $T(s) = 1$ at low frequencies for sufficient tracking performance and $T(s) = 0$ at high frequencies for sensor noise rejection. Since $T(s) + S(s) = 1$, designers want $S(s) = 0$ at low frequencies for plant disturbance rejection and $S(s) = 1$ at high frequencies. Without proper analysis, designs may lead to process disturbances and sensor noise being amplified in the closed-loop.

It is worth noting that most dynamics vary drastically depending on current position, velocity, or orientation changes. In order to account for these changes while still using a linear controller, gain-scheduling is performed [19]. Usually, different metrics are chosen for each trim point design (e.g. rise-time) where the designer attempts to identify trends based on the flight dynamics which enable a more automated design process over the operating regime.

3.2 Nonlinear and Adaptive Control of a Flight Vehicle

In order to illustrate the basis for direct adaptive control, a single-input single-output (SISO) example is shown below from [16, 76]. The method is expanded to multi-input multi-output (MIMO) systems while incorporating possible system uncertainties.

3.2.1 Single-Input Single-Output (SISO) Adaptive Control

Consider a dynamical system in the form:

$$\dot{x}_p(t) = a_p(t)\, x_p(t) + b_p(t)\, u(t) \tag{3–14}$$

$$\dot{x}_m(t) \quad = \quad a_m\, x_m(t) + b_m\, r(t) \tag{3–15}$$

where a_p and b_p are unknown scalar plant parameters, u is the plant input provided by the controller, r is the external command, and x_p is the state's current value. A reference model is described in (3–15) where a_m and k_m are known constants. The reference model represents the ideal closed-loop performance of the system under nominal conditions. Hence, x_m represents the desired reference model state of the system where we assume $a_m < 0$. It is necessary for the reference model to be stable in order to prove stability. The goal is to drive x_p to x_m while keeping all states and control inputs of the system bounded. We define the state tracking error and its derivative as

$$e(t) \quad = \quad x_p(t) - x_m(t) \tag{3–16}$$

$$\dot{e}(t) \quad = \quad \dot{x}_p(t) - \dot{x}_m(t). \tag{3–17}$$

A control solution to the SISO plant problem with corresponding adaptive laws take the form:

$$u(t) = \hat{k_p} x_p(t) + \hat{k_r} r(t) \tag{3–18}$$

where the existence of ideal controller gains k_p and k_r given by

$$k_p \quad = \quad \frac{a_m - a_p}{b_p} \tag{3–19}$$

$$k_r \quad = \quad \frac{b_m}{b_p} \tag{3–20}$$

is guaranteed for any controllable pair $(a_p,\, b_p)$. Using the ideal form of the controller gains, we can derive the closed-loop error dynamics in the form:

$$\dot{e} = a_m e(t) + b_p \tilde{k_p} x_p + b_p \tilde{k_r} r \tag{3–21}$$

where $\tilde{k_r} = \hat{k}_r - k_r$ and $\tilde{k_p} = \hat{k}_p - k_p$.

Generally, the Lyapunov function is chosen to represent the total kinetic energy of all the errors in the system. The goal of this control design is to design a controller

such that the energy of the system is guaranteed to dissipate with time, i.e. ideally the derivative of the Lyapunov function would be negative definite, see (3–26). The Lyapunov function candidate for our SISO problem is given by

$$V(e, \Delta k_p, \Delta k_r) = \frac{e^2}{2} + |b_p|(\frac{\tilde{k_p}^2}{2\Gamma_p} + \frac{\tilde{k_r}^2}{2\Gamma_r}) \tag{3–22}$$

where $|\cdot|$ denotes absolute value.

Next, we design the adaptive update laws in the form:

$$\dot{\hat{k}}_p(t) = -\Gamma_p\, x_p\, e\ \mathrm{sgn}(b_p) \tag{3–23}$$

$$\dot{\hat{k}}_r(t) = -\Gamma_r\, r\, e\ \mathrm{sgn}(b_p) \tag{3–24}$$

where (Γ_p, Γ_r) are positive definite learning rates and the sign of b_p is assumed to be known.

The time derivative of V computed along the trajectories of (3–16) results in

$$V(e,\ \Delta k_p,\ \Delta k_r) = \frac{e^2}{2} + |b_p| \left(\frac{\tilde{k_p}^2}{2\Gamma_p} + \frac{\tilde{k_r}^2}{2\Gamma_r} \right) \tag{3–25}$$

$$\dot{V}(e,\ \Delta k_p,\ \Delta k_r) = e\dot{e} + |b_p| \left(\frac{\tilde{k_p}\dot{\hat{k}}_p}{\Gamma_p} + \frac{\tilde{k_r}\dot{\hat{k}}_r}{\Gamma_r} \right) \tag{3–26}$$

$$\dot{V}(e,\ \Delta k_p,\ \Delta k_r) = a_m e^2 + |b_p|\tilde{k_p} \left(\mathrm{sgn}(b_p) pe + \frac{\dot{\hat{k}}_p}{\Gamma_p} \right) \tag{3–27}$$

$$+|b_p|\tilde{k_r} \left(\mathrm{sgn}(b_p) re + \frac{\dot{\hat{k}}_r}{\Gamma_r} \right) \tag{3–28}$$

$$\dot{V}(e,\ \Delta k_p,\ \Delta k_r) = a_m e^2 \leq 0. \tag{3–29}$$

By substituting the adaptive update laws shown in (3–23) into the time derivative of (3–25) results in a negative semi-definite $\dot{V}$. We assume the reference input is a bounded, which implies the control input is bounded. It follows that we can show that the states of the system and adaptive gains $(\hat{k}_p,\ \hat{k}_r)$ are bounded. Using Barbalat's Lemma (see [77]), we can prove that the error between the plant states and the designed

reference model states is driven to zero, shown in (3–30).

$$lim_{t\to\infty}e(t) = 0 \tag{3–30}$$

However, the adaptive parameters do not necessarily converge to their true unknown values unless the persistence of excitation criteria was met.

3.2.2 Direct Model Reference Adaptive Control (MRAC) with Uncertainties (MIMO)

We can extend the previous design to a MIMO system using the methodologies discussed in [16]. The goal is to design u such that y tracks y_{cmd} while operating with the uncertainties Λ and $f(x)$. The dynamical system and reference model can be written as:

$$\dot{x} = Ax + B\Lambda\left(u + f(x)\right) \tag{3–31}$$

$$\dot{x}_{ref} = A_{ref}x_{ref} + B_{ref}y_{cmd} \tag{3–32}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{p\times n}$ are known. The matrix $\Lambda \in \mathbb{R}^{m\times m}$ is a constant unknown diagonal matrix which we will refer to as the control effectiveness term. The matched uncertainty, $f(x)$, is an unknown continuously differentiable function which is often assumed to be linear-in-the-parameters. In other words, $f(x)$ takes the form

$$f(x) = \Sigma_{i=1}^{N}\Theta_i\phi_i(x) = \Theta^T\Phi(x) \tag{3–33}$$

where $\Phi(x) \in \mathbb{R}^n$ is the known regressor vector and Θ_i are unknown constants or slowly-varying parameters. The goal is to find adaptive update laws that drive the plant state dynamics x to reference state dynamics x_{ref}.

Recently it has been shown that adding a Luenberger-like term to the typical open-loop reference model can decrease oscillations and increase transient performance. The closed-loop reference model takes the form [11]:

$$\begin{aligned} \dot{x}_{ref} &= A_{ref}x_{ref} + L_v(x - x_{ref}) + B_{ref}y_{cmd} \\ y &= Cx_{ref}\ . \end{aligned} \tag{3–34}$$

where $A_{ref} = A - BK_{LQR}$ is designed to be Hurwitz and K_{LQR} are baseline controller gains that were designed to meet certain performance and robustness criteria.

Consider a direct adaptive controller in the following form:

$$u = \hat{k}_x^T x + \hat{k}_r^T y_{cmd} - \hat{\Theta}^T \Phi(x) \tag{3–35}$$

where we can use the form of the controller along with the known form of the dynamics and the reference model to obtain the matching conditions for the model reference adaptive controller. The matching conditions are given by

$$A + B\Lambda K_x^T = A_{ref} \tag{3–36}$$

$$B\Lambda K_r^T = B_{ref} \tag{3–37}$$

where, for MRAC systems, the ideal gains K_x and K_r must exist for the controller to remain stable. For aerospace systems, this is a typical assumption that is satisfied due to the known form of the system dynamics and the carefully chosen reference model.

After applying Lyapunov stability analysis to the previously mentioned system, we retrieve the following adaptive update laws:

$$\dot{\hat{\Theta}} = \Gamma_\Theta \Phi(x) e^T P B \tag{3–38}$$

$$\dot{\hat{k}}_x = -\Gamma_x x e^T P B \tag{3–39}$$

$$\dot{\hat{k}}_r = -\Gamma_r r e^T P B \tag{3–40}$$

where Θ's update law is a function of P which is the unique symmetric positive-definite solution of the algebraic Lyapunov equation ($PA + A^T P = Q$) where A is typically A_{ref}. This term exists because $e^T P e$ is a term that is included in the Lyapunov function.

3.2.3 Robust Adaptive Control Tools

Robust adaptive control provides additional tools to the designer which increase performance and stability in the presence of uncertainties. Unmatched uncertainties (e.g. environmental disturbances) are especially troublesome to adaptive control laws.

For instance, estimated parameters can drift slowly due to non-parametric uncertainties when the persistence of excitation condition is not met [16]. This has been shown to lead to sudden divergence and failure.

In order to improve adaptive controllers in the presence of such uncertainties, Ioannou and Kokotovic [78] proposed the σ-modification which combats this phenomenon by providing a damping term, see (3–41). Unfortunately, when errors are small, the damping term dominates the adaptive update law and causes the parameter estimates to drift. If persistence of excitation is not met, the errors will increase immediately. Note that the σ-modification based adaptive law often takes the form:

$$\dot{\hat{\Theta}} = \Gamma_{\Theta}(\Phi(x)e^{T}PB - \sigma\hat{\Theta}) \tag{3–41}$$

where σ is a positive constant.

Narendra and Annaswamy [79] proposed e modification which mitigates this issue by adding the tracking error to the modification term, see (3–42).

$$\dot{\hat{\Theta}} = \Gamma_{\Theta}(\Phi(x)e^{T}PB - \sigma||e^{T}PB||\hat{\Theta}) \tag{3–42}$$

Dead-zone is another robust adaptive control modification which keeps the parameters from drifting due to noise. Dead-zone performs the desired update, stated in (3–38), unless the error falls below a certain threshold. In this case, the adaptive parameters are frozen.

Unfortunately, σ and e modification can negatively impact tracking performance by slowing adaptation. The projection operator is an alternative method which bounds adaptation parameters (gains) and protects against integrator windup while still allowing fast adaptation. We will utilize and describe the projection operator in more detail in Chapters 4 and 5.

3.2.4 Adaptive Augmentation-Based Controller

In the previous sections, linear optimal control theory was used to create designs at certain flight conditions which determine the reference model for which the adaptive error was driven to. In this section, we show how the baseline controller can be augmented with an adaptive controller to form the overall controller.

The baseline design is the core of the robust and adaptive control framework. It defines the target performance of the controller which was designed with built-in robustness attributes. Adaptive augmentation is needed to complement the baseline robustness properties. Additionally, the adaptive laws are capable of restoring baseline control performance as changes to the plant take place (i.e. matched uncertainties). For additional details on derivation see [16].

The adaptive portion of the controller is used to deal with the system matched uncertainties. When augmenting the baseline controller with an adaptive component with no uncertainties, the adaptive controller's contribution to the controller will be zero. This is simply because the controller is tracking the reference model perfectly.

For this section we assume the nonlinear plant is in the form similar to (3–5). We re-write the dynamics as

$$\dot{x} = Ax + B\Lambda\left(u + \Theta^T\Phi(x)\right) + B_{ref}y_{cmd} \tag{3–43}$$

where $\Phi(x) \in \mathbb{R}^n$ is the known regressor vector with components $\phi_i(x)$ which are Lipschitz in x. We also note that θ_i are unknown constants. Notice that previous techniques could only compensate for linear dependence among unknown parameters (parametric uncertainty).

Neural networks can extend this design to non-linear in the parameter functions. For example, $f(x) = W^T\sigma(V^Tx) + \epsilon$ represents a neural network with ideal weights W and V and reconstruction error ϵ. When weights of the inner layer, V, are fixed, then Lyapunov-based stability analysis would result in the following adaptive update law for

the outer-layer weights W [17]:

$$\dot{\hat{W}} = \mathrm{Proj}(\Gamma_W \sigma(V^T x) e^T P B). \tag{3–44}$$

The adaptive law is derived by assuming that the uncertainties can be approximated on a bounded closed set within a small tolerance using the universal approximation theorem. The universal approximation theorem only holds if and only if the regressor $\sigma(V^T x)$ provides a basis for the system. Commonly for aerospace applications and structured neural network approaches, radial basis functions are chosen to define the regressor.

Radial basis functions (RBFs) can be written in the following form:

$$\phi_i(x,\ x_c) = e^{-\frac{\|(x-x_c)\|^2}{2w_i}} \tag{3–45}$$

where $x_c \in \mathbb{R}$ defines the center of the RBF and $w_i \in \mathbb{R}^+$ denotes the width of the RBF. The number of RBFs and the spacing is chosen to cover the task space. In the case of flight control, the RBFs would fill the flight envelope. Typically, the weights of the neural network are updated based on the following update law

$$\dot{\hat{\Theta}} = \mathrm{Proj}\left(\Gamma_\Theta \Phi(x) e^T P B\right) \tag{3–46}$$

which was formulated using Lyapunov-based stability analysis where $e = x - x_{ref}$. Notice the notation where Θ replaces W and $\Phi(x)$ replaces $\sigma(V^T x)$. The RBF-based adaptive controller is elaborated on in Chapter 4.

The single hidden layer (SHL) neural network is an alternative approach which adaptively updates both the inner layer weights along with the outer layer weights:

$$\dot{\hat{W}} = \Gamma_W((\sigma(\widetilde{V}^T \mu) - \dot{\sigma}(\hat{V}^T \mu)\hat{V}^T \mu) e^T P B) \tag{3–47}$$

$$\dot{\hat{V}} = \Gamma_V \mu e^T P B \dot{\hat{W}}^T \dot{\sigma}(\hat{V}^T \mu) \tag{3–48}$$

where $\mu = [x^T\ 1]^T$, $\sigma = [\sigma^T\ 1]^T$, and $\dot{\sigma}$ is the derivative of the sigmoid function. We will directly compare the performance of the RBF and SHL adaptive controllers in Chapter 4.

3.2.5 Structure of the Adaptive Controller

The final form of the controller uses any of the above neural network based adaptive controller methodologies along with a baseline controller. In addition to the projection operator, we use dead-zone to combat noise. For a review of Lyapunov-based stability results see [76] and [77]. For this specific proof of stability see [16]. The overall control signal is determined from the sum of adaptive and baseline components and is given by

$$u = u_{bl} + u_{ad} \tag{3–49}$$

where $u_{bl} = -K_{LQR}x$ and

$$u_{ad} = -\hat{\Theta}^T\bar{\Phi}(x) - \hat{K_{BL}}^T u_{bl} \tag{3–50}$$

where the adaptive update laws are given by

$$\dot{\hat{K_{BL}}} = \mathrm{Proj}(\Gamma_u u_{bl} e^T PB) \tag{3–51}$$

$$\dot{\hat{\Theta}} = \mathrm{Proj}(\Gamma_\Theta \Phi(x) e^T PB). \tag{3–52}$$

CHAPTER 4
IMPROVING LEARNING OF MODEL REFERENCE ADAPTIVE CONTROLLERS: A SPARSE NEURAL NETWORK APPROACH

Over the last decade neural network based model reference adaptive controllers (MRAC) have become ubiquitous in the flight control community due to their ability to adapt quickly for complex maneuvers while undergoing significant uncertainties. Single hidden layer (SHL) and radial basis function (RBF) based neural networks are the most effective and common in adaptive control. Recent machine learning breakthroughs have shown the advantages of using sparse networks for learning based identification tasks. We show that using sparse networks for adaptive control can reap similar benefits including improved long-term learning and better tracking while reducing the computational burden on the controller. Simulations results demonstrate the effectiveness of the proposed controller.

This section is based on the published paper [80].

4.1 Model Reference Adaptive Control Formulation

MRAC is an adaptive architecture used to ensure plant states, x, successfully track the chosen reference model states, x_{ref}. The reference model is designed to specify the desired closed-loop tracking performance of the system under nominal conditions. We will augment a baseline controller with a neural network based adaptive controller. The adaptive controller aims to restore degraded baseline closed-loop tracking performance in the presence of matched uncertainties [20]. More specifically, the goal is to design a control input, u, such that y tracks y_{cmd} while operating under uncertainties Λ and $f(x)$ [73]. The dynamical system has the form:

$$\begin{aligned} \dot{x} &= Ax + B\Lambda\left(u + f(x)\right) + B_{ref}y_{cmd} \\ y &= Cx \end{aligned} \tag{4–1}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $B_{ref} \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{p\times n}$ are known matrices. The matrix $\Lambda \in \mathbb{R}^{m\times m}$ is an uncertainty in the form of a constant unknown diagonal matrix. The

diagonal elements of Λ are assumed to be strictly positive. The matched uncertainty, $f(x) \in \mathbb{R}^m$, is an unknown continuously differentiable function. We assume an external bounded time-varying command $y_{cmd} \in \mathbb{R}^m$ where the $B_{ref} y_{cmd}$ term in the open-loop dynamics is included when the state vector is augmented with the integrated tracking error, i.e. $x = (e_I \; x_p) \in \mathbb{R}^n$ where x_p represents the original plant states and $\dot{e}_I = y - y_{cmd}$. The reader is referred to Lavretsky [16] for a derivation of this model.

Consider a closed loop reference model in the form:

$$\begin{aligned} \dot{x}_{ref} &= A_{ref} x_{ref} + B_{ref} y_{cmd} \\ y_{ref} &= C_{ref} x_{ref} \end{aligned} \tag{4–2}$$

where $A_{ref} \in \mathbb{R}^{n \times n}$ and $C_{ref} \in \mathbb{R}^{p \times n}$ are known matrices. We assume the pair $(A_{ref,} \, B\Lambda)$ is controllable and A_{ref} is designed to be Hurwitz where

$$A_{ref} = A - BK_{LQR}^T. \tag{4–3}$$

The overall controller takes the following form:

$$u = u_{BL} + u_{AD} \tag{4–4}$$

where u_{AD} represents the adaptive control signal and u_{BL} is designed using linear optimal regulation (LQR) to obtain a proportional-integral (PI) baseline given by

$$u_{BL} = -K_{LQR} x \tag{4–5}$$

$$= -e_I \, K_I - x_p \, K_p \tag{4–6}$$

which takes into consideration rise-time, undershoot, overshoot, and robustness metrics which are useful for applications such as flight control. Optimal linear quadratic (LQ) control methods provide a systematic approach to design baseline controller gains, K_{LQR}. A block diagram of the overall controller is shown in Figure 4-1.

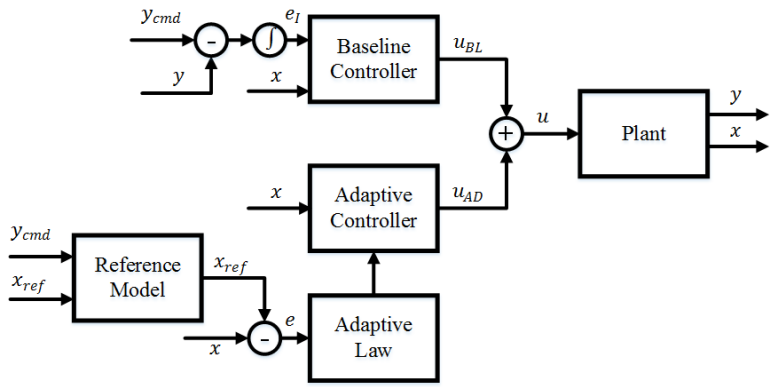

Figure 4-1. Model reference adaptive control augmentation control architecture.

The state tracking error is defined as

$$e = x - x_{ref} \tag{4–7}$$

which is used in the Lyapunov analysis to prove a uniformly ultimately bounded result.

By substituting the form of the baseline controller in (4–5) into the dynamics in (4–1) and utilizing the form of the state tracking error in (4–7), the open-loop tracking error dynamics can be written as

$$\dot{e} = A_{ref}e + B\Lambda(u_{AD} + f(x) + (u_{BL} - \Lambda^{-1}u_{BL})) \tag{4–8}$$

where the adaptive control portion of the controller, u_{AD}, takes the following form:

$$u_{AD} = -\hat{\Theta}^T\Phi(x) - \hat{K}_{BL}u_{BL} \tag{4–9}$$

where $\hat{K}_{BL}u_{BL}$ term accounts for the matched uncertainty, $u_{BL} - \Lambda^{-1}u_{BL}$. For this chapter, we will focus entirely on the neural network contribution to the controller, i.e. $\hat{\Theta}^T\Phi(x)$. For the structured neural network approach, the regression matrix, $\Phi(x)$, contains a predefined set of known nonlinear basis functions (e.g. radial basis functions). For the unstructured approach, $\Phi(x)$ is a vector that holds N activation functions (e.g. sigmoid) along with associated estimates of the inner-layer weights. For both approaches, $\hat{\Theta}$ is a matrix that contains the estimates of outer-layer weights of the neural network.

For each neural network based approach in Section 4.2, we employ robust adaptive control techniques to ensure boundedness of the weights without requiring the PE condition. The projection operator is one such technique that bounds adaptation parameters/gains and protects against integrator windup while still allowing fast adaptation [16]. Robust adaptive control provides additional tools to the designer which improves performance and stability in the presence of uncertainties. For instance, estimated parameters can drift slowly due to non-parametric uncertainties when the persistence of excitation condition is not met. This has been shown to lead to sudden divergence and failure in flight vehicles [16].

The update laws presented in Section 4.2 are stated as a function of P, which is the unique symmetric positive definite solution of the algebraic Lyapunov equation

$$PA_{ref} + A_{ref}^T P = Q \tag{4–10}$$

where Q is a tunable positive definite matrix. That is, for any positive definite matrix $Q \in \mathbb{R}^{n\times n}$ there exists a positive definite solution $P \in \mathbb{R}^{n\times n}$ to the Lyapunov Equation. Since $e^T Pe$ is a term that is included in the Lyapunov function; this term is necessary for the adaptive update law.

Each adaptive update law in Section 4.2 also contains a tunable symmetric positive definite matrix, Γ, that include rates of adaptation. The larger the values, the faster the system changes the weights in order to adapt for the uncertainties in the system [16]. As mentioned in the introduction, the goal is to set the learning rates to a reasonable value to limit high actuator rates and avoid high-frequency unmodeled disturbances. For this chapter, we hold the learning rates constant.

Full stability proofs and analysis for the neural network based MRAC schemes can be seen in Hovakimyan [14, 22] or Lewis [13, 17, 81]. For results on applications of previously defined methods in flight control see Shin [82], Lavretsky [16], Mcfarland [83], or Chowdhary [18, 84].

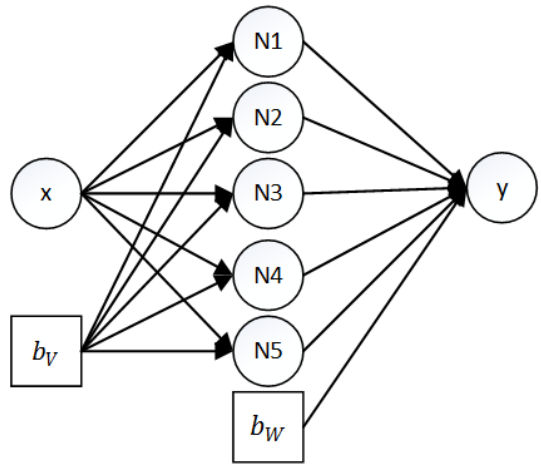

Figure 4-2. Example single hidden layer (SHL) neural network.

In the following sections, single hidden layer (SHL), radial basis function (RBF) and sparse neural network (SNN) approaches to neural network based adaptive control are presented and analyzed.

4.2 Neural Network-Based Adaptive Control

The most popular neural network based adaptive control approaches (SHL and RBF) are presented in this section, as well as the novel SNN approach.

A neural network can be described as an input-output map $NN(x) : \mathbb{R}^n \to \mathbb{R}^m$ composed of N neurons where m is the number of output layer neurons and n is the number of input layer neurons. A conventional feed-forward neural network takes the form:

$$NN(x) = W^T f(V^T x + b_V) + b_W \tag{4–11}$$

where $x \in \mathbb{R}^n$ denotes the input vector and $W \in \mathbb{R}^{N \times m}$, $V \in \mathbb{R}^{n \times N}$, $b_v \in \mathbb{R}^N$, and $b_W \in \mathbb{R}^m$ represents the ideal weights and biases of the neural network. Each neuron is composed of an activation function (e.g. sigmoid, tanh, or linear rectifier), $f(x) : \mathbb{R} \to \mathbb{R}$, which produces a single output based on a nonlinear transformation of the input. An example of a neural network can be seen in Figure 4-2, where each line connecting a

node has a single weight associated with it. The output of the first node of the hidden layer can be mathematically expressed as $N_1 = f(V_1 x + b_V)$.

The objective of the neural network is to adjust parameters W, V, b_v, and b_W in order to approximate a smooth nonlinear function within specified thresholds. Neural networks satisfy the universal approximation theorem which has been proved for sigmoidal and radial basis function networks [85]. The universal approximation theorem states that any smooth function, $f(x)$, can be approximated over a compact domain, $x \in X \subset \mathbb{R}^n$, by a single hidden layer neural network. It implies that given a compact set, X, and for any $\epsilon^* > 0$, there exist neural network weights such that

$$f(x) = W^T \sigma(V^T x + b_V) + b_W + \epsilon, \ ||\epsilon|| < \epsilon^\star \tag{4–12}$$

where ϵ is referred to as the reconstruction error. For structured neural networks, the reconstruction error bound, ϵ^*, can be made arbitrarily small by increasing the number of hidden layer nodes. For the structured RBF network approach, this corresponds to increasing the number of radial basis functions.

For this chapter, in order to be concise and computationally efficient, we will redefine the neural network weight matrices (W and V) to include the bias terms (b_v and b_W) of the neural network [16]. That is, $V = [V^T \ b_V]^T \in \mathbb{R}^{(n+1)\times N}$ and $W = [W^T \ b_W]^T \in \mathbb{R}^{(N+1)\times m}$.

4.2.1 Radial Basis Function (RBF) Adaptive Control

The following equation defines a radial basis function (RBF):

$$\phi(x, \ x_c) = e^{-\frac{||x - x_c||^2}{2\sigma^2}} \tag{4–13}$$

where x is the input of the activation function, x_c is the center of the RBF, and σ denotes the RBF width [16]. For adaptive control applications, a radial basis function (RBF) is a popular choice of an activation function used for a structured neural network based adaptive control approach where the centers of each RBF are predefined to fill the

operational envelope. The adaptive controller takes the following form:

$$u_{AD} = -\Theta^T \Phi(x) = -\hat{W}^T \phi(x) \tag{4–14}$$

where $\phi(x) = [\phi_1(x),\ \phi_2(x),\ \phi_3(x),\ ...,\ 1]^T \in \mathbb{R}^{N+1}$ is a vector of N radial basis functions [84] with distinct fixed centers and $\hat{W} \in \mathbb{R}^{(N+1)\times m}$ are the outer layer weight and bias estimates that are adjusted using the Lyapunov based adaptive update law shown in (4–15).

In other words, structured neural network controllers have weights of the inner layer that are fixed, and standard Lyapunov analysis provides the adaptive update law for the outer-layer weights, $\hat{W}$. When using RBF neural network based controller paired with a robust adaptive control tool such as the projection operator, it is well known that the following adaptive update law results in uniform ultimate boundedness of the tracking error:

$$\dot{\hat{W}} = \mathrm{Proj}(\Gamma_W \phi(x) e^T P B). \tag{4–15}$$

The RBF based approach leverages the universal approximation theorem for Radial Basis Functions if the output of the radial basis function vector, $\phi(x)$, provides a basis [13]. It is important to note that the persistence of excitation (PE) condition is still required to ensure that the adapted weights, $\hat{W}$, converge to their ideal weights, W.

An example of an RBF based regressor matrix used in a structured neural network based adaptive control approach is seen in Figure 4-3, which is designed for a flight control system that the operating regime is known a priori (e.g. in Figure 4-3 between -6 and 6 degrees angle of attack). The elements in this design are considered to be parametric elements with centers and widths that do not change during operation. One obvious disadvantage is that the controller is semi-global in the sense that representation of the input vector is significantly reduced outside of the designed envelope. As the width of the RBF vectors is increased, the hidden layer sparsity is reduced, but the richness of representation of the input is increased. A disadvantage of RBF based

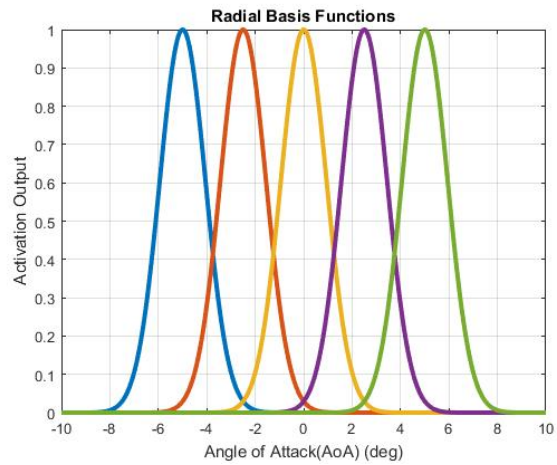

Figure 4-3. Example radial basis function (RBF) network distributed across angle of attack.

representations is that the richness and entanglement of each representation changes based on the input. For instance, consider if the input lies directly between two RBFs versus directly on one RBF. By increasing the number of radial basis functions, this problem is mitigated.

4.2.2 Single Hidden Layer (SHL) Adaptive Control

We will use the following compact notation for the unstructured neural network based adaptive controller

$$u_{AD} = -\hat{W}^T f(\hat{V}^T \mu) \tag{4–16}$$

where $\mu = [x \quad 1] \in \mathbb{R}^{n+1}$ is the input to the neural network.

For the SHL approach, the stability analysis holds for any chosen “squashing” function [14]. In this chapter, a sigmoidal activation function will be chosen and is defined as follows

$$\begin{aligned} \sigma(x) &= \tfrac{1}{1+e^{(-x)}} \\ \sigma'(x) &= \sigma(x)(1-\sigma(x)) \end{aligned} \tag{4–17}$$

where $\sigma'(x)$ represents the derivative of the sigmoid function in terms of itself.

The following update law is derived from the Lyapunov stability analysis

$$\begin{aligned} \dot{\hat{W}} &= \mathrm{Proj}(2\Gamma_W((\sigma(\hat{V}^T\mu) - \dot{\sigma}(\hat{V}^T\mu)\hat{V}^T\mu)\hat{e}^T PB)) \\ \dot{\hat{V}} &= \mathrm{Proj}(2\Gamma_V \mu \hat{e}^T PB\hat{W}^T \dot{\sigma}(\hat{V}^T\mu)) \end{aligned} \tag{4–18}$$

Similar to the RBF based approach, these update laws ensure uniform ultimate boundedness of the tracking error. In addition, the single hidden layer (SHL) based approach leverages the universal approximation theorem for sigmoidal functions. RBF based adaptive controllers are often chosen over SHL networks for flight control applications due to their simplicity and ability to learn. This topic will be further explored in the results section of this chapter.

4.2.3 Sparse Neural Network (SNN) Adaptive Control

As mentioned in the introduction, ideas in this chapter were constructed based on inspiration from the machine learning community. More specifically, we aim to exploit the benefits of distributed sparse representations of neural networks discovered in the deep learning literature. In contrast to dense representations, sparse representations result in the activation of only a small percentage of the neurons for a variety of input data. During optimization, this prevents against un-learning weights for tasks with dissimilar input data. Dense distributed networks have the richest representations (i.e. highest precision), while sparsely distributed networks hold the enticing properties of rich representations and non-entangled data [58]. We will explore these trade-offs by creating an approach that segments the flight envelope and enables us to vary the number of neurons in the system.

For flight control problems, we typically are limited in the number of neurons (nodes) based on onboard computational processing capabilities [21]. The proposed adaptive architecture will be able to increase the total number of nodes by keeping a high percentage of these nodes inactive. We do this by segmenting the flight envelope into regions and distributing a certain amount of nodes to each region. Compared to a

typical fully-connected neural network based control schemes, our approach allows for more nodes while reducing the computational burden on the processor by only using a small percentage of the nodes for control at each point in the operating envelope. This strategy will allow the network to "remember" weights that were obtained while operating in each region and recall those weights when the region is revisited. The overall goals of this approach are to improve long-term learning performance of the controller, especially for repeated flight maneuvers, while avoiding high-frequency oscillations and actuator overuse by operating with small to moderate learning rates.

For the sparse neural network (SNN) approach, we begin by creating a predefined N_D-dimensional grid that spans the flight envelope. We define a set of points, $P = \{p_1, ..., p_T\}$, and segments, $S = \{s_1, ..., s_T\}$, where each point, $p_{i \in I} \in P$, is uniformly spaced on that grid, and $T \in \mathbb{N}$ represents the total number of segments. We let $I = \{1, ..., T\}$ be the index set of the sets S and P. We allow each dimension of the grid to be represented by a distinct flight condition (e.g. angle of attack, Mach, altitude). We can visualize this concept using a figure similar to a Voronoi diagram in Figures 4-4 and 4-5 where the Voronoi space (operating envelope) is divided into Voronoi regions (segments) [86]. Each segment, $s_{i \in I} \in S$, defines a convex polytope that surrounds each pre-defined point, $p_{i \in I}$, on the grid

$$s_i = \{x_{op} \in X : \ D(x_{op}, p_i) \leq D(x_{op}, p_j) \ \forall i \neq j\} \tag{4–19}$$

where x_{op} is an arbitrary point in the Voronoi space and (X, D) defines the metric space where $X \subset \mathbb{R}^{N_D}$ is the non-empty set composed of points that define inputs to a function $D : X \times X \to \mathbb{R}$. In our case, X is comprised of all the possible points in the operating envelope. The function $D(\cdot, \cdot)$ calculates the Euclidean distance between two points. We define the total number of nodes available to the system as $N \in \mathbb{N}$ where the complete set of nodes is included in the set $Y = \{e_{1,...,} e_N\}$ and $B = \{1, ..., N\}$ is the index set for the set Y. All nodes in Y are then distributed equally amongst the segments where we

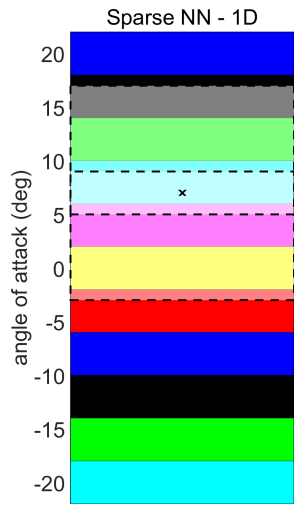

Figure 4-4. Adaptive sparse neural network (SNN) segmented flight envelope in one dimension.

define the number of nodes per segment as $Q \in \mathbb{N}$, where $Q = \frac{N}{T}$. We establish a set of nodes for each index, i, denoted by $E_{i \in I} = \{e_{Q(i-1)+1}, \ldots, e_{iQ}\}$ where the complete set of nodes can be stated as $Y = \bigcup_{i \in I} E_i$.

At each point in time, t, the SNN determines the set of nodes and associated weights, ($\hat{W}^i$, $\hat{V}^i$), used in the adaptive controller and adaptive update laws based on the segment number, i, that the flight vehicle is currently operating, s_i. That is, every segment, s_i, has a pre-defined set of nodes that is active when operating in that region. The set of active nodes is denoted by E_{A_i} where $\forall i \in I : \quad E_i \subseteq E_{A_i}$. We assume the number of active nodes is restricted due to processing constraints and denoted $R \in \mathbb{N}$ where $R \geq Q$. Notice if $R = Q = N$ then we are simply using the traditional SHL approach. The simplest implementation of the SNN is achieved by assuming that while operating in the segment s_i only the nodes that were allocated to that segment are active (i.e. $\forall i \in I : \quad E_{A_i} = E_i$). We refer to this extreme case as the pure sparse approach. Notice for the pure sparse approach, nodes are only activated and updated by the adaptive update laws while operating within a specific region and can not be used for control or modified outside that segment. For a blended approach where $R > Q$, the adaptive controller will use all nodes allocated to its current segment along with

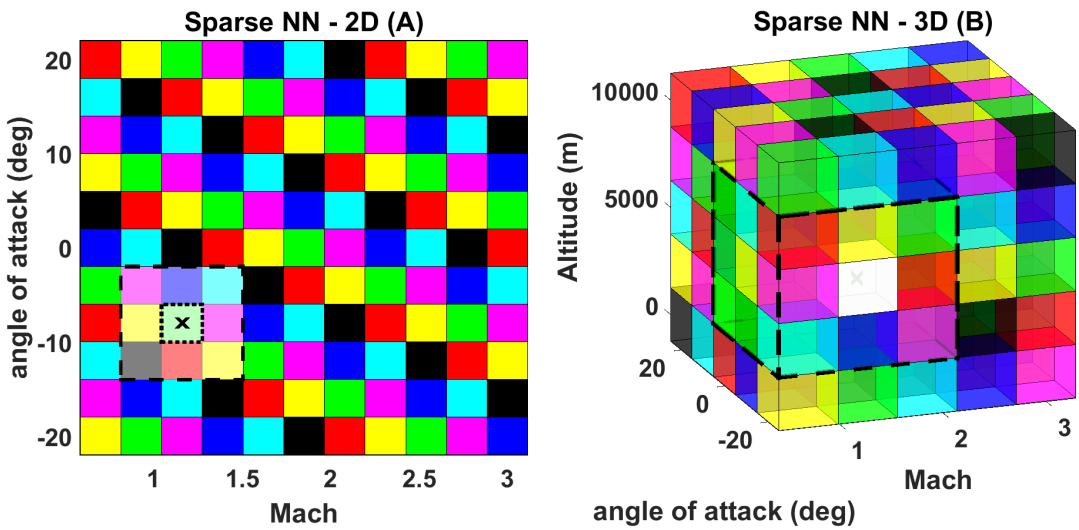

Figure 4-5. Adaptive sparse neural network (SNN) segmented flight envelope in A) two and B) three dimensions.

additional nodes from nearby segments. For this approach, the active node list for each segment i, $E_{A_{i \in I}}$, is determined by selecting the closest R nodes to each segment's center point, p_i. In order to accomplish this task, we uniformly distribute Q points within each segment and assign each node to a distinct point within its assigned region. Next, for each index $i \in I$ we compute the distance between the center point of that segment, p_i, and every node in the flight envelope. The indices for the closest R nodes to each center point, $p_{i \in I}$, are then stored in set $C_{i \in I}$ for later use. Since all the parameters of the SNN approach are pre-defined, the list of active indices is created before run-time.

Algorithm 4.1 shows the step-by-step SNN approach for each time the controller is called.

Possible segmented flight space for 1, 2, and 3-dimensional cases can be seen in Figure 4-4 and Figure 4-5. In each figure, an X represents the current operating point of the flight vehicle within the specified flight envelope. In the 1-D case, Figure 4-4, the neural network will determine the active nodes based on a single state, angle of attack (AoA), which is divided into 11 segments. The segments contained by dotted and

Algorithm 4.1 Sparse Neural Network Execution

1: receive x(t) and the corresponding location in the operating envelope, x_{op}
2: determine the index, $i \in I$, of point p_i in the set P where the distance between x_{op} and p_i is minimum (i.e. $argmin_{i \in I}\ D(x_{op}, p_i)$)
3: retrieve the set of indices, C_i, corresponding to the index i
4: form a single hidden layer neural network using the weights, $\hat{W}^i$ and $\hat{V}^i$, associated with the active nodes stored in E_{A_i}
5: use the neural network to form an adaptive control signal following (4–20)
6: create an overall control signal following (4–4)
7: apply the overall control signal to the system dynamics (4–1)
8: update the weights ($\hat{W}^i$, $\hat{V}^i$) of each node in E_{A_i} according to the adaptive update laws in (4–21) and (4–22)

dashed white regions show possible active regions by varying the number of shared nodes between segments in the blended approach.

An example of a typical neural network (RBF and SHL) used for adaptive control is shown in Figure 4-6. In RBF and SHL networks, full connectivity between nodes is assumed at all points in the flight envelope. In contrast, Figure 4-6 also shows a pure sparse network approach where the state domain is divided into five segments with each segment containing exactly two nodes and each segment not sharing any nodes with adjacent segments. For each type of network in Figure 4-6, segments are represented by rectangles while active nodes for each segment are represented with distinct colors. The active nodes for segment 1-5 are highlighted in red, yellow, green, blue, and purple, respectively. We also include a blended network in the figure. This architecture has the same number of total nodes as the sparse approach, but segments now share 1 node with each neighboring segment. For instance, (N0, N1, N2, N3) are all active when operating in the first segment (red) and (N2, N3, N4, N5) are all active in the second segment (yellow). The node N4 is colored yellow and green to signify that the weights of that node are used in the adaptive controller when operating in segments 3 and 4. The sparsity of the network can be adjusted by varying the number of nodes that adjacent segments share, the total number of nodes in the network, and the total number of active nodes.

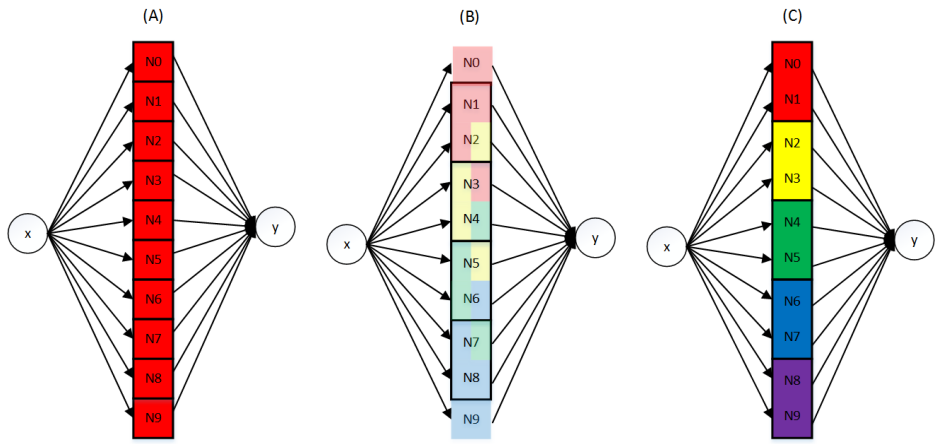

Figure 4-6. Single hidden layer with A) typical connectivity, B) blended connectivity, and C) spare connectivity.

Note that the discrete update for the selection of the nodes used for control causes switching in the closed-loop system. For every point in time, t, the SNN controller is operating within a single segment, s_i, using a pre-defined set of active nodes, $E_{A_{i \in I}}$, for control. Using compact neural network notation, the form of the adaptive controller while operating in the ith segment can be stated as

$$u_{AD} = \hat{W^i}^T \sigma(\hat{V^i}^T \mu) \tag{4–20}$$

where $\hat{W}^i$ and $\hat{V}^i$ denote the estimates of the outer and inner layer weights updated by specified adaptive update laws.

The adaptive update laws used in the ith interval were derived using Lyapunov stability analysis and are given by

$$\dot{\hat{W}}^i = \mathrm{Proj}(2\Gamma_W((\sigma(\hat{V^i}^T \mu) - \dot{\sigma}(\hat{V^i}^T \mu)\hat{V^i}^T \mu)\hat{e}^T PB)) \tag{4–21}$$

$$\dot{\hat{V}}^i = \mathrm{Proj}(2\Gamma_V \mu \hat{e}^T PB \hat{W^i}^T \dot{\sigma}(\hat{V^i}^T \mu)) \tag{4–22}$$

where the adaptive update laws ensure the tracking error, e, and the SNN weights errors, ($\tilde{W}^i$, $\tilde{V}^i$), remain bounded while operating in every segment, $\forall i \in I$.

The benefits of this approach are analyzed in Section 4.4.3.

4.3 Nonlinear flight dynamics based Simulation

In this section, we compare the performance of the neural network based MRAC controllers described previously. We are interested in a flight vehicle with complicated and dominant disturbances and uncertainties. A linear model of the short-period longitudinal dynamics was extracted from a generic flight vehicle at the specific flight condition stated in Table 4-1 and was implemented in the fashion described in Section 4.1. The extended open-loop dynamics can be stated as

$$\begin{pmatrix} \dot{e}_I \\ \dot{\alpha} \\ \dot{q} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & \frac{Z_\alpha}{V} & 1+\frac{Z_q}{V} \\ 0 & M_\alpha & M_q \end{pmatrix} \begin{pmatrix} e_I \\ \alpha \\ q \end{pmatrix} + \begin{pmatrix} 0 \\ Z_{\delta_e} \\ M_{\delta_e} \end{pmatrix} \Lambda(\delta_e + f(x)) + \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix} y_{cmd} \quad (4\text{–}23)$$

where α is the angle of attack (AoA), q is the pitch rate, δ_e is the control input (elevator deflection), Λ is a positive constant set to reduce control effectiveness, $f(x)$ represents the matched uncertainty, and $(Z_\alpha,\ Z_q,\ M_\alpha,\ M_q,\ Z_{\delta_e},\ M_{\delta_e})$ are the flight vehicles stability derivatives [16].

Before the neural network based adaptive controllers could be implemented, a well-designed LQR baseline controller with servomechanism structure was created using an iterative loop described in Dickinson et al. [73] The servomechanism structure allows for the addition of the integral error of AoA as a state, which provides asymptotic constant command tracking with desirable and predictable robustness properties [16]. The linear optimal controller was used in the model reference adaptive controller (MRAC) to form the closed loop reference model. The combination of the adaptive controller and baseline controller was used to provide the elevator deflection command to the flight vehicle, displayed in (4–4).

Table 4-1. Flight condition to analyze

Mach	Altitude (m)	AoA (deg)	AoS (deg)	$\bar{q}$(kPa)	$Motor$
0.7	7000	0	0	14.1	Off

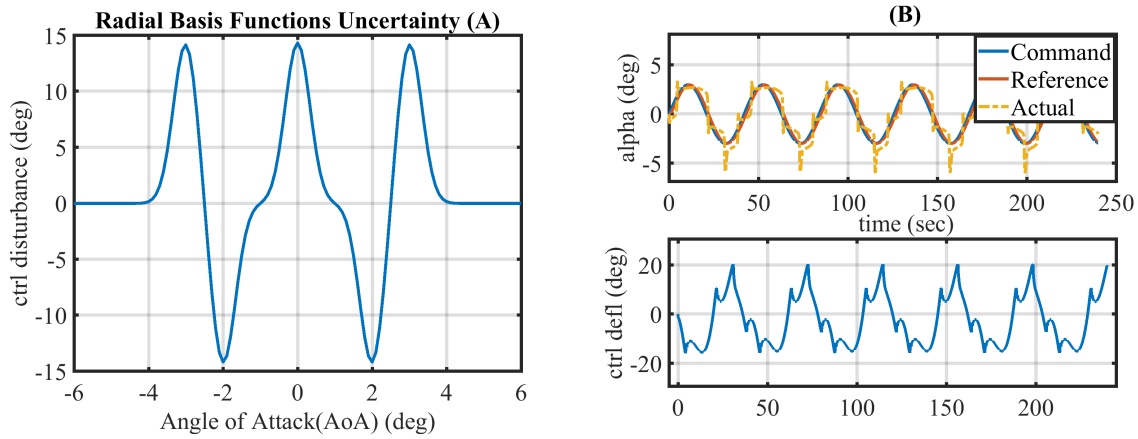

Figure 4-7. Baseline control A) transient performance when subjected to B) radial basis function matched uncertainty.

In order to provide a highly nonlinear and significant uncertainty which dominates the dynamics of the system, we created an uncertainty term, $f(x)$, that is the result of the summation of several radial basis functions centered at different angles of attack (AoA). To demonstrate the effectiveness of this disturbance, it was first tested against the well-tuned LQR baseline controller. The disturbances and tracking performance of the LQR baseline controller is shown in Figure 4-7. It is worth noting that the Λ uncertainty, which essentially is a control effectiveness term, is quickly mitigated in each controllers approach. Hence, it is not analyzed in the results section.

For the simulation, each type of neural network based adaptive controller was tested against a repeated angle of attack maneuver. Each maneuver required the vehicle to increase and decrease in AoA in a sinusoidal fashion. This required the vehicle to spend similar amounts of time in each section of the flight envelope and provided a good learning comparison. In addition to the tracking performance, uncertainty estimates were computed based on the final weights that were obtained after finishing the simulation. Typically, these controllers are only interested in determining the uncertainty at the current operating point, but by fixing the weights and sweeping the input across the whole flight envelope, this enables a comparison in the ability of the adaptive controller to learn and remember estimates of uncertainty from previously visited regions in the

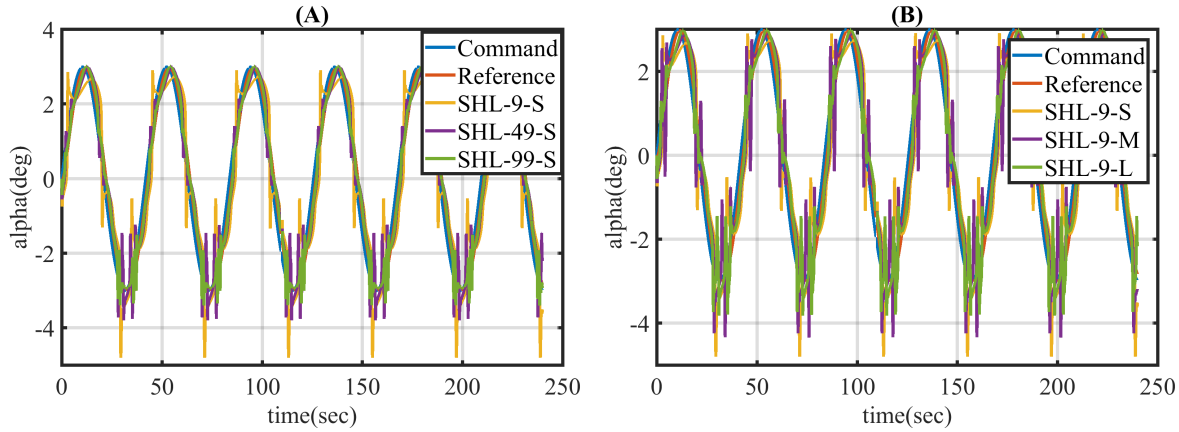

Figure 4-8. Single hidden layer transient analysis by varying the A) number of total nodes and the B) the learning rate.

flight envelope. Each maneuver will be regarded as a "pass" in the results section of this chapter.

For all simulations, a constant $dt = 0.01\ sec$ was used along with a second-order integration method (AB-2). We found that the most effective initial weights of the neural network were similar to those stated in Lewis [13, 17]. That is, $\hat{V}_0$ is initialized to small random numbers and each element in $\hat{W}_0$ is initialized to zero.

4.4 Results

In order to understand the advantages of the sparse learning-based approach, SHL and RBF based approaches were implemented and analyzed for comparison using the disturbances generated in the previous section. For these experiments, the value for the constant learning rate, Γ, was denoted in the following figures either S ($\Gamma_W = 0.025 \times I,\ \Gamma_V = 0.0125 \times I$), M ($\Gamma_W = 0.05 \times I,\ \Gamma_V = 0.025 \times I$), or L ($\Gamma_W = 0.10 \times I,\ \Gamma_V = 0.05 \times I$), where I denotes an identity matrix of appropriate size. The number of total nodes in the network tested varied from $N = 10\ to\ 100$ and is also identified in the subsequent figures.

4.4.1 Single Hidden Layer (SHL)

The results from implementing a SHL based MRAC using the form of the controller specified in (4–16) was consistent with literature results. For instance, increasing the

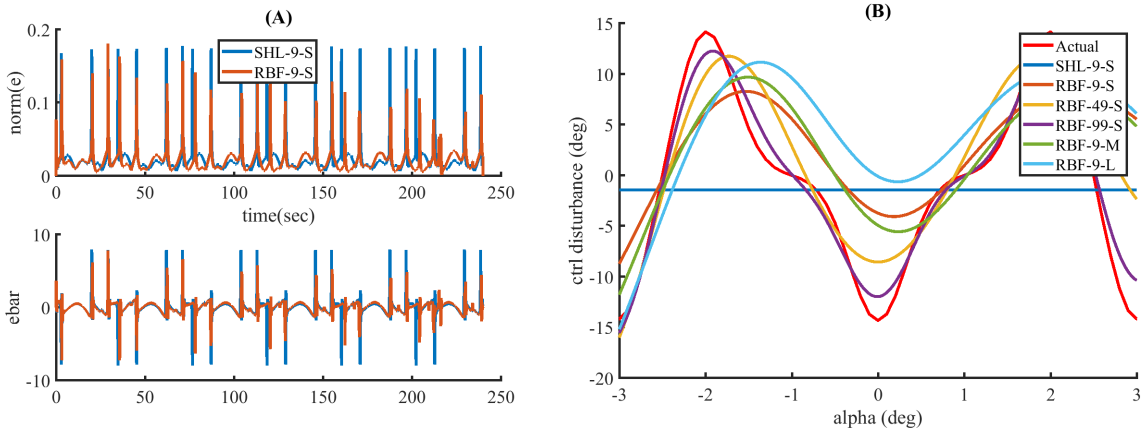

Figure 4-9. Radial basis function (RBF) versus single hidden layer (SHL) analysis plots.

number of nodes of the network or increasing the learning rates greatly improves system tracking performance (see Figure 4-8).

In most cases, the simulation showed that the SHL based adaptive controllers reacted faster than their structured neural network counterparts with the same number of active nodes and similar learning rates. But their disadvantage is that they did not learn or remember the weights that they used for an angle of attack that was previously visited. Hence, traditional SHL adaptive controllers do not improve in tracking performance with multiple passes.

4.4.2 Radial Basis Function (RBF)

In this section, we will consider the problem of using a set of radial basis functions to define our regression matrix for a structured neural network approach.

Similar to the SHL, the RBF based approach performed better as the number of nodes in the network increased. The larger learning rate drastically improved transient performance but seemed to have a detrimental effect on the uncertainty estimate for repeated flight maneuvers. This could be due to the fact that when the learning rate is increased, the weights with the lowest impact end of changing significantly. Using a small learning rate with several repeated passes over the same region resulted in the best uncertainty estimate. This point will be elaborated on in the successive section.

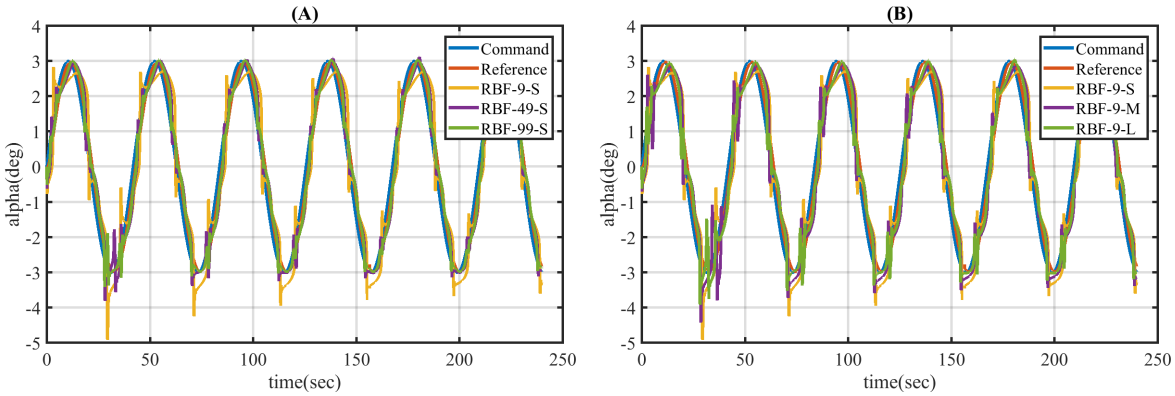

Figure 4-10. Radial basis function (RBF) transient analysis by varying A) the number of nodes and the B) the learning rate.

The uncertainty estimate at any given alpha is based on the weights obtained at the end of the simulation (i.e. approximately -3 degrees alpha) and is displayed in Figure 4-9. As mentioned previously, SHL control schemes adapt well for flight control systems but do not possess the architecture to "remember" weights or estimates of uncertainty for previously visited regions in the flight regime. In contrast, RBF based control schemes do hold that valuable ability to keep reasonable estimates of the uncertainty for all different sectors of the state vector around the neighborhood of the estimate even when operating outside of that region. Unfortunately, it is difficult to determine the ideal placement of the RBF centers, and there exists a clear trade-off between increasing the size of the RBF widths for transient improvement or decreasing the widths for better learning capabilities. This result is demonstrated more clearly in Figure 4-10, where norm(e) is defined as $||e||_2$ where $e = x - x_{ref}$ and ebar is defined by the following equation, $\bar{e} = e'PB$. This gives a different perspective of the simulation results shown previously. It clearly shows how the RBF controller improves with repeated maneuvers, while SHL has consistent performance with each pass.

4.4.3 Sparse Neural Network (SNN)

Following the approach presented in Section 4.2.3, several different sparse networks were designed and tested using the same dynamics and disturbances described

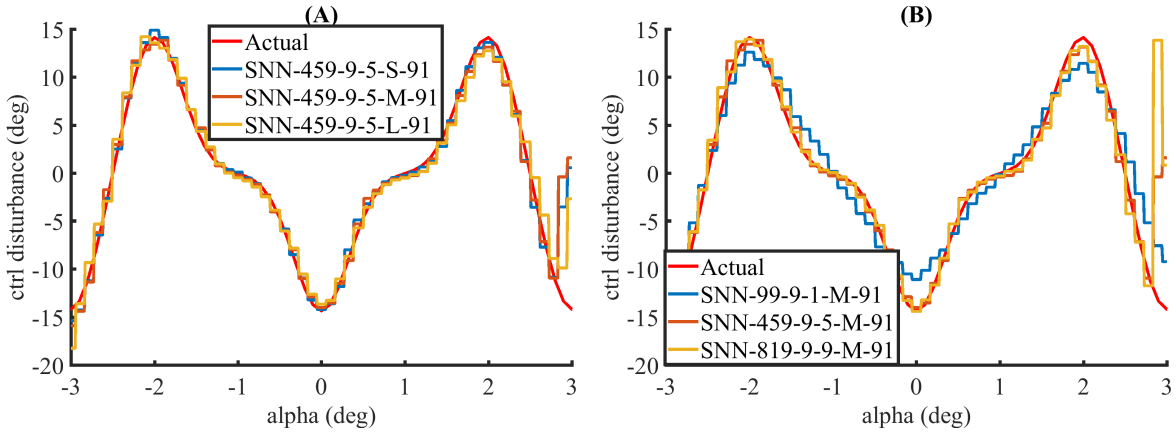

Figure 4-11. Sparse neural network matched uncertainty comparison by A) varying the learning rate and the B) number of shared nodes.

in the previous sections. Since the disturbance, $f(x)$, was designed based on a single input variable, α, only the 1-D SNN architecture was employed for simulation results. That is, each SNN partitioned the flight envelope, which is only represented by α in the 1-D case, into T segments. Each sparse network was created with a different amount of sparsity while holding the total number of segments and number of active nodes constant. As stated previously, the sparseness of each network is determined based on the ratio of the number of active nodes, R, to the total number of nodes in the network, N. In order to compare the new sparse architecture results with the RBF and SHL results, we set the number of active nodes to be equivalent (i.e. $R = 9$) in each case. In other words, the processing load was held constant. We describe the parameters of each SNN architecture in the legend of each figure in the following format: total number of nodes (N) - number of active nodes (R) - number of nodes per segment (Q) - learning rates (Γ) - total number of segments in the flight envelope (T), i.e. $(N - R - Q - \Gamma - T)$.

After running numerous simulations, it became apparent that the SNN architecture is superior to the legacy architecture in many ways. For instance, Figure 4-11 shows the estimates of the uncertainty after the run is complete. Clearly, this is a much better estimate than obtained with the RBF controller using the same learning rates.

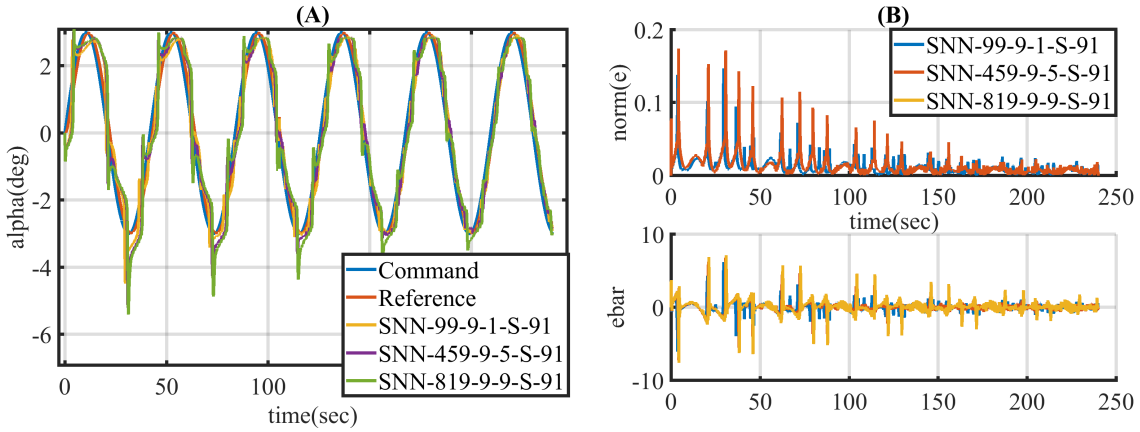

Figure 4-12. Sparse neural network transient analysis by A) sinusoidal commands with B) error comparison results.

Comparing Figure 4-12 and Figure 4-13 reveals the improved tracking performance with the sparse neural network (SNN) controller especially after repeating maneuvers.

Results from Figure 4-13 show that an increased learning rate has beneficial results in terms of tracking and uncertainty estimates. The most sparse SNN controller (819-9-9-M-91) which dedicated nine nodes to each segment and did not share nodes with adjacent segments resulted in the best uncertainty estimate but performed poorly in the initial stages of the runs (e.g. Figure 4-12) and had the worst overall tracking performance compared to the rest of the SNN controllers. The blended SNN (99-9-1-M-91) that acquired almost all (i.e. ~88.8%) its active nodes from other segments had the worst uncertainty estimate but performed better in the initial stages of the runs than the other controllers. The best overall SNN controller took the best aspects from each extreme. For example, the blended controller (i.e. 459-9-5-M-91) which acquired exactly four of its active nodes from adjacent segments, had the best overall tracking performance and a terrific memory of uncertainty estimates. Taking a closer look at this SNN's architecture reveal why it was successful. Firstly, if the controller entered into an adjacent segment (e.g. higher angle of attack), four out of the nine active nodes for the new segment were updated while operating in the previous segment, which would allow it to perform better than having randomly initialized nodes. The other five nodes

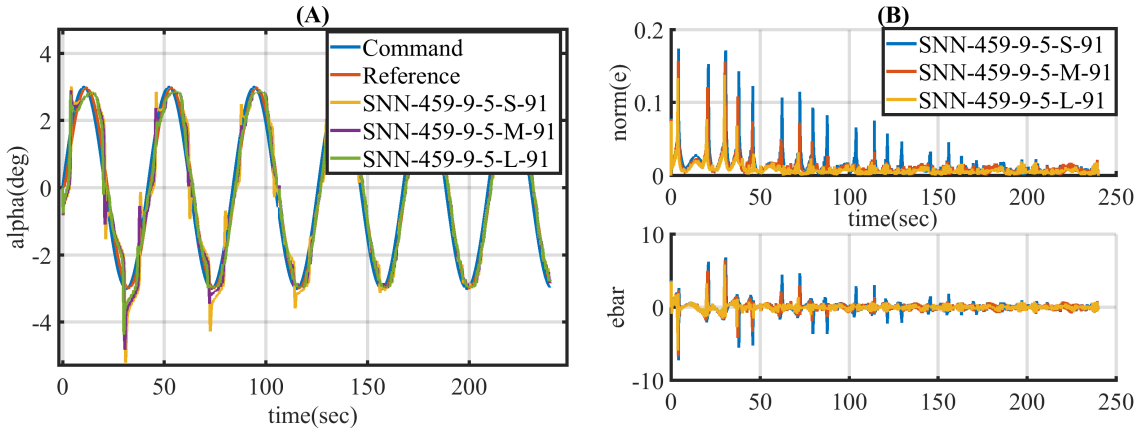

Figure 4-13. Learning rate comparison using A) sinusoidal commands with B) transient results.

that were not acquired in the previous segment will be updated while operating in the current segment and will be available for other adjacent segments to use. Notice that one node allocated to each segment is only active for one segment, while the other four are active for exactly two adjacent segments. This allows this SNN (459-9-5-M-91) to retain better uncertainty estimates than the SNN (99-9-1-M-91) whose nodes span several segments.

In order to quantifiably demonstrate the superior performance of the sparse neural network over the traditional architectures, the following tables were constructed based on the data obtained previously. Rather than comparing every controller previously tested, we selected the best controller from each category with moderate learning rates which used only 9 active nodes, i.e. RBF controller RBF-9-M, SHL controller SHL-9-M, and SNN controller 459-9-5-M-91. Table 4-2 compares the norm of the error in tracking each maneuver using the following equation:

$$e_{TE} = \sum_{t=0}^{T_F} ||e(t)||_2 \tag{4–24}$$

where T_F is defined as the final time of each pass and $e(t) = x(t) - x_{ref}(t)$.

Table 4-2. Tracking error comparison table of RBF, SHL, and SNN

Tracking Error	Pass 1	Pass 2	Pass 3	Pass 4	Pass 5	Pass 6
RBF	88.99	75.55	65.24	64.69	62.32	61.11
SHL	92.68	88.81	85.25	89.21	83.82	85.26
SNN	89.26	42.07	29.84	25.17	24.74	23.98

Table 4-3 shows an uncertainty estimation comparison using the summation of the squared estimation error (SSE) over the whole flight envelope, as seen in

$$e_{UE} = \sum_{\alpha} (f(\alpha) - \hat{f}(\alpha))^2 \tag{4–25}$$

where $f(\alpha)$ is the true uncertainty in the system and $\hat{f}(\alpha)$ is the estimated uncertainty using the frozen weights obtained at the end of the simulation.

Clearly, the SNN is competitive in the first pass and dominates in performance in the following 5 passes. The SHL does not change in performance with each pass. The SNN has a superior uncertainty estimate which demonstrates that it does a better job learning state dependent matched uncertainties than the other architectures.

4.5 Summary

Traditional neural network based adaptive controllers (RBF and SHL) update their weight estimates based solely on the current state vector as input and utilize all nodes for control during every portion of the flight envelope. This leads to poor long-term learning and only slight improvements in transient performance when tracking a repeated command sequence or flight maneuver.

Sparse adaptive controllers only update a small portion of neurons at each point in the flight envelope which results in the SNNs ability to remember estimates and weights from previously visited sectors. The blended SNN also has the ability to share nodes

Table 4-3. Estimation comparison table of RBF, SHL, and SNN

Estimation Error	Total SSE
RBF	4.90
SHL	29.89
SNN	0.11

with other segments, which improves the tracking performance of initial passes and smooths transitions between segments. The SNN has superior performance in terms of tracking and estimating uncertainties than the single hidden layer (SHL) and radial basis function (RBF) systems on tasks that have consistent uncertainties and disturbances over regions of the flight envelope. Sparse neural networks have the added advantage of having only a small number of computations at each point in time due to the high percentage of inactive neurons.

CHAPTER 5
A SPARSE NEURAL NETWORK APPROACH TO MODEL REFERENCE ADAPTIVE CONTROL WITH HYPERSONIC FLIGHT APPLICATIONS

In this chapter, we ensure stability of the closed-loop system using a dwell time condition requirement and demonstrate the sparse neural network (SNN) adaptive control capabilities on a hypersonic flight vehicle (HSV). In addition, we develop a number of improvements to the SNN control scheme, including an adaptive control term used to counteract control degradation. We also explore including higher order Taylor series expansion terms in our adaptive error derivation for additional benefits. Hypersonic control simulation results are used to properly compare the SNN to the more conventional single hidden layer (SHL) approach.

5.1 Augmented Model Reference Adaptive Control Formulation

We start by considering a class of n dimensional multiple input multiple output (MIMO) system dynamics with m inputs in the form:

$$\begin{aligned} \dot{x} &= Ax + B\Lambda\left(u + f(x)\right) + B_{ref}y_{cmd} \\ y &= Cx \end{aligned} \tag{5–1}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{p\times n}$ are known matrices. We assume an external bounded time-varying command as $y_{cmd} \in \mathbb{R}^m$. The control effectiveness matrix $\Lambda \in \mathbb{R}^{m\times m}$ is an unknown constant diagonal matrix with uncertain diagonal elements, denoted Λ_i, which are assumed to be strictly positive. We assume the control effectiveness matrix (Λ) can be upper bounded in by a constant term which we denote $\bar{\Lambda}$. The matched uncertainty in the system is represented by the continuously differentiable function $f(x) \in \mathbb{R}^m$. We define the state tracking error as $e = x - x_{ref}$ and the output tracking error as $e_y = y - y_{cmd}$. The system state vector, x, which appears in (5–1), contains the traditional plant state vector, x_p, along with the integral tracking error as a system state, i.e. $x = (e_I\ x_p) \in \mathbb{R}^n$. The inclusion of this augmented state vector

creates the $B_{ref}y_{cmd}$ term in the extended open-loop dynamics, (5–1). See [16] for more details and derivations of this model.

We now discuss the reference model for the MRAC architecture which defines the ideal closed-loop behavior of the system under nominal conditions. To begin design, we remove the uncertainties from the system dynamics stated in (5–1). This results in the following ideal extended open-loop dynamics

$$\begin{aligned} \dot{x} &= Ax + Bu + B_{ref}y_{cmd} \\ y &= Cx. \end{aligned} \tag{5–2}$$

Next, we assume the baseline controller takes the following form:

$$u_{BL} = -K_{LQR}x \tag{5–3}$$

$$= -e_I\, K_I - x_p\, K_P \tag{5–4}$$

where K_{LQR} are fixed baseline controller gains typically designed using systematic optimal linear quadratic (LQ) control methods. The systematic approach to design LQ gains allow the control designer to take into consideration robust (e.g. margins, singular values, and loop shaping) and performance metrics (e.g. rise-time, overshoot, and undershoot) while utilizing traditional analysis tools (e.g. root locus, bode plots). For this paper, we assume the baseline controller is in proportional-integral (PI) form with gains (K_{LQR}) consisting of integral (K_I) and proportional gains (K_P).

By substituting the form of the baseline controller defined in (5–3) into the ideal extended open-loop dynamics shown in (5–2), we derive the form of the closed-loop reference model dynamics as

$$\begin{aligned} \dot{x}_{ref} &= A_{ref}x_{ref} + B_{ref}y_{cmd} \\ y_{ref} &= C_{ref}x_{ref} \end{aligned} \tag{5–5}$$

where we assume the pair $(A,\ B)$ is controllable, $C_{ref} = C \in \mathbb{R}^{p\times n}$ is known, and $A_{ref} = A - BK_{LQR}^T \in \mathbb{R}^{n\times n}$ is designed to be Hurwitz, see (5–5). Note that the proper

design of the baseline controller gains in (5–3) results in a robust linear controller with ideal transient characteristics.

The model matching conditions for the previously defined MRAC system can be described as follows. Given a constant unknown positive definite matrix Λ and a Hurwitz matrix A_{ref}, there exists a constant unknown gain matrix $K_{\Lambda_{BL}}$ such that

$$A_{ref} = A - B\Lambda K_{\Lambda_{BL}}^{T}. \tag{5–6}$$

In order to satisfy both the model matching conditions and the closed-loop reference model dynamics in (5–5), we define the ideal gain matrix, $K_{\Lambda_{BL}}$, as

$$K_{\Lambda_{BL}} = K_{LQR}\Lambda^{-1} \tag{5–7}$$

where $K_{\Lambda_{BL}}$ has been proven to exist for any nonsingular matrix Λ and controllable pair $(A,\ B)$ [16].

The overall MRAC design goal is to achieve reasonable bounded tracking of the external time-varying command (y_{cmd}) and reference states (x_{ref}) in the presence of the nonlinear matched uncertainty $(f(x))$ and the control effectiveness term (Λ). However, since the baseline controller is a fixed gain controller, unexpected changes in flight dynamics and unmodeled effects create uncertainties which degrades the performance of the controller. The role of the adaptive controller in the MRAC architecture is to reduce or ideally cancel the effect that the uncertainties have on the overall dynamics of the system and restore baseline tracking performance. That is, for any bounded time-varying command (y_{cmd}) the adaptive controller is designed to drive the selected states (x) to track the reference model states (x_{ref}) within bounds while keeping the remaining signals bounded. This goal is achieved through an incremental adaptive control architecture in the following form:

$$u = u_{BL} + u_{AD} + u_{RB} \tag{5–8}$$

where u_{AD} represents the adaptive control signal, u_{BL} is the optimally designed proportional-integral (PI) baseline controller, and u_{RB} is the robust term used to ensure quick convergence to a specified error region for safe switching. The subsequent sections will provide architectural details of the adaptive controller along with Lyapunov based stability analysis results.

We now state a common Lyapunov-like theorem that will be used to verify uniform and ultimate boundedness of the previously defined system and control scheme. It is worth noting that many of the symbols for the variables in the theorem were chosen to match those in the stability analysis.

Theorem 5.1. *Let γ_1 and γ_2 be class κ functions, V be a continuously differentiable function, and $X \subset \mathbb{R}^n$ be a domain that contains the origin and V. Consider a system in the form:*

$$\dot{\zeta} = f(t, \zeta) \tag{5–9}$$

where f is piecewise continuous in t and locally Lipschitz in the associated state vector $\zeta \in X$. Suppose that

$$\begin{aligned} \gamma_1(||\zeta||) &\leq V(t, \zeta) \leq \gamma_2(||\zeta||) \\ \frac{\delta V}{\delta t} + \frac{\delta V}{\delta \zeta} f(t, \zeta) &\leq -W_3(\zeta), \quad \forall ||\zeta|| \geq r_1 > 0 \end{aligned} \tag{5–10}$$

for $\forall t \geq 0$ and $W_3(\zeta)$ is a continuous positive definite function. Suppose that

$$r_1 < \gamma_2^{-1}(\gamma_1(r)) \tag{5–11}$$

where $r > 0$. For every initial state $\zeta(t_0)$ satisfying

$$||\zeta(t_0)|| \leq \gamma_2^{-1}(\gamma_1(r)) \tag{5–12}$$

there exists a $T \geq 0$ such that the solution of (5–9) satisfies

$$||\zeta(t)|| \leq \gamma_1^{-1}(\gamma_2(r_1)) = r_2, \quad \forall t \geq t_0 + T \tag{5–13}$$

where we refer to the ultimate bound as r_2.

Proof. See [16, 77, 82].

5.2 Sparse Neural Network Architecture

The sparse neural network (SNN) architecture is created by segmenting the flight envelope into regions where each region is assigned a certain number of neurons with associated neural network (NN) weights based on user selections. During operation within that region, the adaptive controller only utilizes and updates weights belonging to neurons that are active in that region while the remaining regions and neuron weights are frozen. The following sections aim to expound on the SNN architecture developed in Chapter 4 by providing a detailed Lyapunov stability analysis with simulation results.

Notice that the traditional single hidden layer (SHL) neural network approach can be viewed as a special case of the more extensive SNN concept where the entire operating domain is considered one (passive) segment and the adaptive update laws are established using first-order Taylor series expansion of the matched uncertainty.

5.2.1 Sparse Neural Network Control Concept

In order to introduce the SNN concept, we create a pre-defined N_D-dimensional grid that spans the flight envelope. The user selects the dimensions of the grid based on known operating conditions (e.g. velocity, Mach, and altitude). Next, we define a set of segments $S = \{s_1, ..., s_T\}$ with center points $P = \{p_1, ..., p_T\}$ where $T \in \mathbb{N}$ denotes the total number of segments. We let $I = \{1, ..., T\}$ be the index set of the sets S and P. Similar to the grid, the center points are provided by the user and do not require any spacing requirements. For best results, the number of center points should be chosen to be dense in regions with significantly varying dynamics or regions which contain significant or unknown uncertainties (e.g. high angle of attack). We define the total number of nodes in the neural network as $N \in \mathbb{N}$ where every node is assigned to a specific segment. We define the number of nodes per segment as $Q \in \mathbb{N}$ where $Q = \frac{N}{T}$ and $E_{i \in I}$ is the set of nodes allocated to segment s_i. Within each segment, the

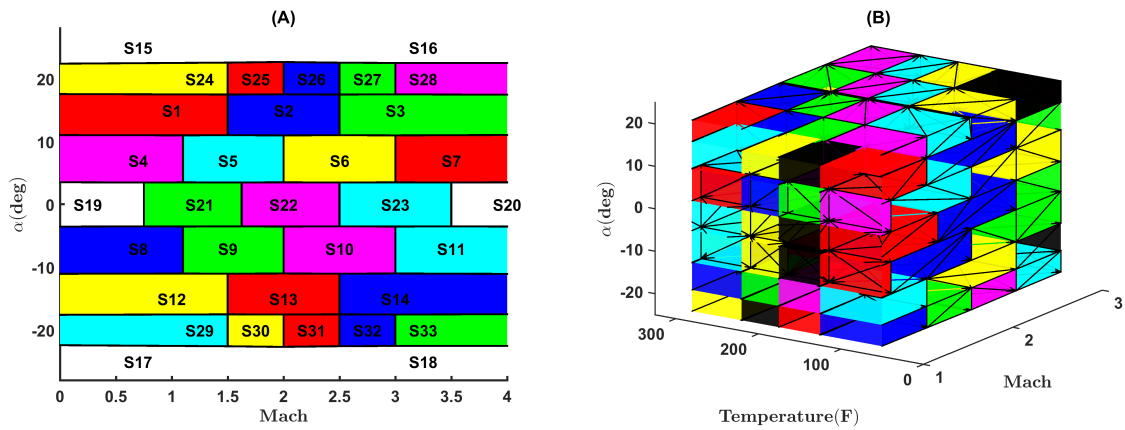

Figure 5-1. Sparse neural network segmented flight envelope for hypersonic control in A) two and B) three dimensions.

user determines the spacing of the nodes (e.g. uniform distributed) where every node is allocated to a specific position within its assigned segment. Finally, we define the set of active nodes for each segment s_i as E_{A_i}.

We use a Voronoi diagram to create the T segments using convex polygons where we denote the index of segment s_i as $i \in I$ [86]. In a Voronoi diagram, each segment is defined by a region surrounding a center point which encloses all points in the N_D space that are closer to that center point than any other center point. The Voronoi diagram also allows for an efficient and simple way to partition the flight envelope into segments while still allowing flexibility in selecting the locations of the center points. In addition, by using the nearest neighbor graph generated from Delaunay triangulation, we can efficiently locate the region for which the current operating point lies within the flight space during flight time.

Consider the examples of static segmented flight envelopes in the $N_D = 2$ and $N_D = 3$ dimensional cases. Voronoi and Delaunay diagrams for a 2-D SNN flight controller utilizing angle of attack (α) and altitude as operating conditions are shown in Figure 5-1 and Figure 5-2. For the 3-D case, the geometric diagrams are shown in Figure 5-1 and Figure 5-2. The chosen center points for each segment are labeled for the 2-D case. For each case, the neural network will determine the set of active nodes

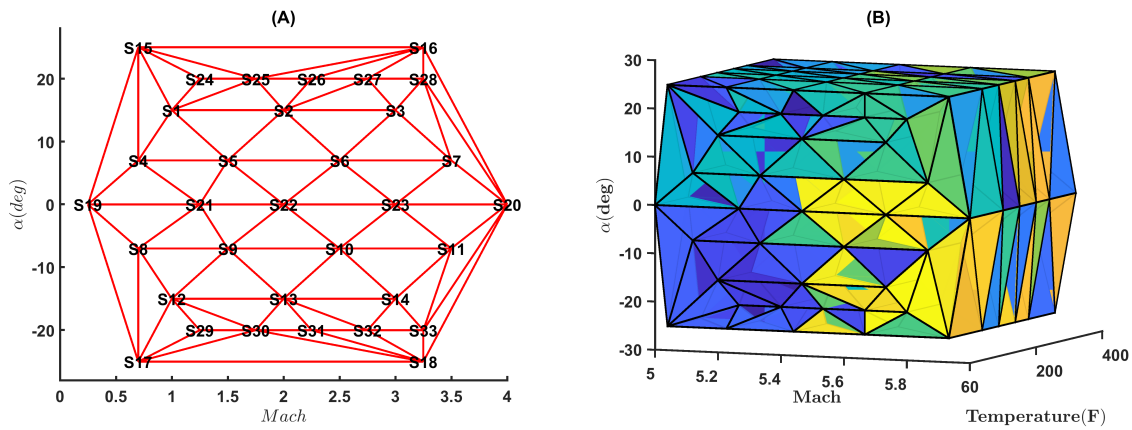

Figure 5-2. Delaunay diagrams for sparse neural network hypersonic control in A) two and B) three dimensions.

(E_{A_i}) used in the adaptive controller based on the index (i) of the current segment (s_i) of operation. Each enclosed colored region of the Voronoi diagrams in Figure 5-1 represents the domain of a single segment (s_i).

In terms of implementation, consider the feed-forward neural network diagrams in Figure 5-3. Each diagram contains an arbitrary number (e.g. $N = 10$) of nodes where the nodes are denoted by $N0 - N9$. Each color indicates a different segment number. All nodes inside a colored segment belong to that segment. For instance, nodes N2 and N3 are allocated to the segment represented by the color yellow. An example of the traditional fully connected neural network architecture used for SHL and RBF adaptive control schemes is also shown in Figure 5-3 where all nodes belong to the same segment. An example of the SNN architecture is shown in Figure 5-3 where there are $N = 10$ nodes with $T = 5$ segments and $Q = 2$ nodes per segment.

For the sake of brevity, the reader is referred to Chapter 4 for more intimate details on the sparse neural network (SNN) adaptive control concept with mathematical notation and original implementation results. The next subsection describes the procedure for which the SNN adaptive controller executes during flight.

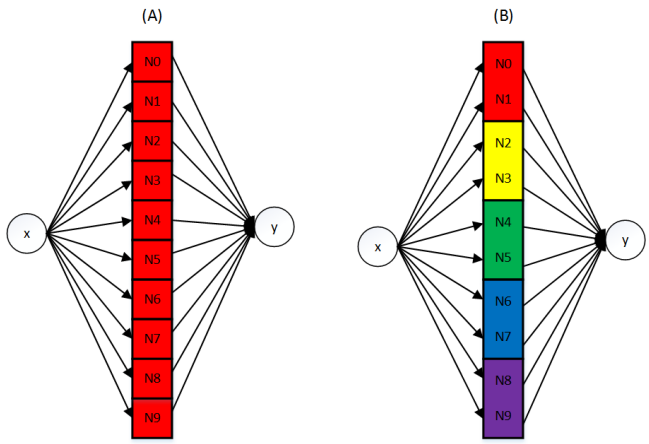

Figure 5-3. Neural network controller for hypersonic control with A) full connectivity and B) sparse connectivity.

5.2.2 Sparse Neural Network Algorithm

At each point in time, t, the SNN determines the set of active nodes (E_{A_i}) with associated weights, ($\hat{W}_i,\ \hat{V}_i$) to be used in the adaptive controller and adaptive update laws based on the segment number, i, that the flight vehicle is currently operating. Rather than finding the index of the current segment number by brute force, we instead use the previous operating segment index and the nearest neighbor graph, generated from the Delaunay diagram, to determine the current segment number. That is, we use the nearest neighbor graph to generate a table which stores a list of neighbors for each segment. At each time the controller is called, that list is used to calculate the closest center point. We have found that this approach significantly reduces the burden on the processor.

We now define the number of active nodes, denoted $N_{act} \in \mathbb{N}$. Active nodes define the exact number of neurons that are being used for control at all times. The number of active nodes is a parameter can be varied by the user based on processing constraints. We refer to the case where N_{act} is selected to be equal to Q as the pure sparse approach, where only the set of nodes allocated to the segment (E_i) currently in operation are used in the adaptive controller. We now consider the case where the number of active nodes is $N_{act} > Q$. That is, the adaptive controller operating in

Algorithm 5.1 Sparse Neural Network Execution

```
receive x(t) and the corresponding location in the operating envelope
use previous segment index (j) and Delaunay diagram to determine the index, i, of
  the closest operating segment s_i
if dwell time condition is met OR inside the error bounds then
    retrieve the set of nodes, E_{A_i}, corresponding to the index i
    form a SHL NN using the weights (Ŵ_i, V̂_i) associated with the segment i
else
    store current operating segment index (i)
    use previous segment index (j) to form a SHL NN using the weights (Ŵ_j, V̂_j)
      associated with the segment j
end if
use the neural network to form an adaptive control signal following (5–18)
create an overall control signal following (5–8)
apply the overall control signal to the system dynamics (5–1)
update the adaptive weights (Ŵ_i, V̂_i, K̂_Λ) according to the adaptive update laws
  stated in (5–24, 5–25, 5–27)
```

segment s_i must utilize its nodes along with nodes from nearby segments for control. For this blended approach, the active node list for each segment, $E_{A_{i\in I}}$, is determined by selecting the closest N_{act} nodes to each segment's center point, p_i. Notice if $N_{act} = Q = N$ then we are resorting to the traditional SHL approach. For the SNN, all the parameters are pre-defined based on user selections and the list of active nodes is created before run-time.

Algorithm 5.1 shows the step-by-step SNN approach for each time the controller is called.

5.3 Adaptive Control Formulation

In order to develop a neural network approximation of the matched uncertainty, $f(x)$, our stability analysis for the sparse neural network (SNN) adaptive controller leverages the universal neural network approximation theorem [13, 17]. We consider a single hidden layer (SHL) feed-forward neural network which takes the form:

$$NN(x) = W^T \sigma(V^T x + b_V) + b_W \tag{5–14}$$

where $x \in \mathbb{R}^{n\times 1}$ is the input vector and $W \in \mathbb{R}^{N\times m}$, $V \in \mathbb{R}^{n\times N}$, $b_V \in \mathbb{R}^N$, and $b_W \in \mathbb{R}^m$ represent the ideal weights ($W,\ V$) and biases ($b_V,\ b_W$) of the neural network. We also define $\hat{W},\ \hat{V},\ \hat{b}_V$, and $\hat{b}_W$ as the estimates of the ideal weights and biases with error terms defined as $\tilde{W} = \hat{W} - W$, $\tilde{V} = \hat{V} - V$. Below is the neural network approximation theorem that will be utilized in subsequent sections.

Theorem 5.2. *Any smooth function, $f(x)$, and can be approximated over a compact domain, $x \in X$, by a single hidden layer neural network with a bounded monotonically increasing continuous activation function, σ. That is, for any $\epsilon^* > 0$, there exist neural network weights W, V, b_V, and b_W with N neurons such that*

$$f(x) = W^T\sigma(V^T x + b_V) + b_W + \epsilon, \quad ||\epsilon|| < \epsilon^\star \tag{5–15}$$

where ϵ is called the reconstruction error.

Proof. See [16] or [82].

The objective of the adaptive neural network controller is to adjust the neural network parameters (i.e. $\hat{W},\ \hat{V},\ \hat{b}_V$, and $\hat{b}_W$) in order to approximate a smooth nonlinear function within specified thresholds. By redefining the neural network weight matrices ($W = [W^T\ b_W]^T \in \mathbb{R}^{(N+1)\times m}$ and $V = [V^T\ b_V]^T \in \mathbb{R}^{(n+1)\times N}$) and the input vector ($\mu \in \mathbb{R}^{(n+1)\times 1}$), we can simplify notation and be more computationally efficient during run-time [16, 80]. For the stability analysis, we will assume that an ideal neural network approximation exists within a known constant tolerance while operating within a compact set, X, with known bounds. That is, we define the compact set

$$X = \{x \in \mathbb{R}^n :\ ||x|| \leq R\} \tag{5–16}$$

and approximation bound as

$$||\epsilon|| \leq \bar{\epsilon}, \quad \forall \epsilon \in X. \tag{5–17}$$

For our stability analysis, the SNN controller requires a discrete update for the selection of the nodes used for control which causes switching in the closed-loop

system. That is, for every point in time, t, the SNN controller is operating within a single segment, s_i, using a pre-defined set of active nodes, $E_{A_{i\in I}}$, for control. Using compact neural network notation, the form of the adaptive controller while operating in the ith segment can be stated as

$$u_{NN} = -\hat{W}_i^T \sigma(\hat{V}_i^T \mu) \tag{5–18}$$

where $\hat{W}_i \in \mathbb{R}^{(N_{act}+1)\times m}$ and $\hat{V}_i \in \mathbb{R}^{(n+1)\times N_{act}}$ denote the estimates of the outer and inner layer weights of the ideal neural network. The total adaptive control signal is given by

$$\begin{aligned} u_{AD} &= u_{NN} + u_{K_\Lambda} && (5–19) \\ &= -\hat{W}_i^T \sigma(\hat{V}_i^T \mu) - \hat{K}_\Lambda(u_{BL} + u_{NN} + u_{RB}) && (5–20) \end{aligned}$$

where u_{K_Λ} is designed to negate the control degradation term, Λ. We assume the input vector, μ, is uniformly bounded and stated by

$$||\mu|| \leq \bar{\mu}, \qquad \bar{\mu} > 0$$

where $\bar{\mu}$ is an upper bound.

The open-loop dynamics can be derived by using the form of the baseline controller in (5–3) along with the chosen reference model in (5–5) and can be stated as

$$\dot{x} = A_{ref}x + B(u_{AD} + u_{RB}) + B\Lambda\left((I - \Lambda^{-1})u + f(x)\right) + B_{ref}y_{cmd} \tag{5–21}$$

where $I \in \mathbb{R}^{m\times m}$ is an identity matrix.

Recall that the definition of the state tracking error as

$$e = x - x_{ref}, \tag{5–22}$$

then the state tracking error dynamics can be stated as

$$\dot{e} = A_{ref}e + B(u_{AD} + f_\Lambda(x)) + Bu_{RB} + B\Lambda K_\Lambda u. \tag{5–23}$$

This is obtained by utilizing the form of the open-loop dynamics in (5–21) and the state tracking error in (5–22) where we define $K_\Lambda = (I - \Lambda^{-1})$ and $f_\Lambda(x) = \Lambda f(x)$.

The neural network weights are updated according to the following adaptive update laws used in the ith segment:

$$\dot{\hat{W}}_i = Proj(\Gamma_W(2(\sigma(\hat{V}_i^T\mu) - \dot{\sigma}(\hat{V}_i^T\mu)\hat{V}_i^T\mu) + \ddot{\sigma}(\hat{V}_i^T\mu)\,\mathrm{diag}(\hat{V}_i^T\mu)\hat{V}_i^T\mu)e^TPB) \quad (5\text{–}24)$$

$$\dot{\hat{V}}_i = Proj(\Gamma_V\mu e^TPB\hat{W}_i^T(2\dot{\sigma}(\hat{V}_i^T\mu) - \ddot{\sigma}(\hat{V}_i^T\mu)\,\mathrm{diag}(\hat{V}_i^T\mu))) \quad (5\text{–}25)$$

where the positive definite matrix $P = P^T > 0$ satisfies the algebraic Lyapunov equation

$$PA_{ref} + A_{ref}^TP = -Q \quad (5\text{–}26)$$

for any positive definite $Q = Q^T > 0$. The index for the estimated neural network weights $(\hat{V}_i,\ \hat{W}_i)$ is determined based on the flight vehicle's location within the flight envelope. The adaptive law used for canceling the effect of Λ takes the following form [16]:

$$\dot{\hat{K}}_\Lambda = Proj(2\Gamma_K(u_{BL} + u_{NN} + u_{RB})e^TPB). \quad (5\text{–}27)$$

Note that $\Gamma_{W,}\ \Gamma_V$, and Γ_K are diagonal matrices of positive constant learning rates.

5.3.1 Neural Network Adaptive Control Law

In this subsection, we formulate the adaptive update laws stated in (5–24) to (5–25) through the use of Taylor Series expansion and determine an upper bound on the adaptive error.

Recall from Section 5.2 that T refers to the total number of segments in the sparse neural network architecture and N_{act} denotes the number of active nodes per segment which is set based on processing constraints. During operation in segment s_i, we form a SHL neural network with the following notation. Let $\hat{V}_i = [\hat{V}_i(1)\ \cdots\ \hat{V}_i(N_{act}+1)]^T$ and $\hat{V}_i(a) = [\hat{V}_i(a,\,1)\ \cdots\ \hat{V}_i(a,\,m)]^T$ where $\hat{V}_i(a,\,b)$ is the inner layer weight from the ath hidden node to the mth output for the ith segment. Similarly, let $\hat{W}_i = [\hat{W}_i(1)\ \cdots\ \hat{W}_i(n+$

$1)]^T$ and $\hat{W}_i(c) = [\hat{W}_i(c,\, 1)\ \cdots\ \hat{W}_i(c,\, N_{act}+1)]^T$ where $\hat{W}_i(c,\, d)$ is the outer layer neural network weight from the cth input node to the dth hidden layer node for the ith segment.

Now, consider the tracking error dynamics given by (5–22) and the update laws in (5–24) and (5–25). Recall, that the neural network portion of the adaptive controller is given by

$$u_{NN} = -\hat{W}_i^T \sigma(\hat{V}_i^T \mu) \tag{5–28}$$

while the neural network approximation error for the ith segment takes the form:

$$f_\Lambda(x) = W_i^T \sigma(V_i^T \mu) + \epsilon_i. \tag{5–29}$$

We define upper bounds $\bar{W}_i$ and $\bar{V}_i$ on the ideal neural network weights for the ith interval as

$$||W_i|| < \bar{W}_i,\ ||V_i|| < \bar{V}_i \tag{5–30}$$

where they satisfy the following inequalities [14]:

$$||\hat{W}_i|| < ||\tilde{W}_i|| + \bar{W}_i,\ ||\hat{V}_i|| < ||\tilde{V}_i|| + \bar{V}_i. \tag{5–31}$$

We now compute the second-order Taylor series expansion of $\sigma(V_i^T \mu)$ around $\hat{V}_i^T \mu$ which yields

$$\begin{aligned} \sigma(V_i^T \mu) &= \sigma(\hat{V}_i^T \mu) + \dot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu - \hat{V}_i^T \mu) \\ &\quad + \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu - \hat{V}_i^T \mu)^2 + O(\tilde{V}_i^T \mu)^3 \\ &= \sigma(\hat{V}_i^T \mu) + \dot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu - \hat{V}_i^T \mu) \\ &\quad - \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)((V_i^T \mu - \hat{V}_i^T \mu) \odot \tilde{V}_i^T \mu) + O(\tilde{V}_i^T \mu)^3 \end{aligned} \tag{5–32}$$

where $\odot$ is used to denote component-wise multiplication and $O(\tilde{V}_i^T x)^3$ represents the sum of the higher order terms greater than two. The activation function is denoted by σ and is stored as a diagonal matrix with Jacobian $\dot{\sigma}$ and Hessian $\ddot{\sigma}$. For this paper, we

use the sigmoid activation function and its derivatives which can be stated as

$$\begin{aligned}
\sigma &= \frac{1}{(1+e^{-x})} \\
\dot{\sigma} &= \frac{e^x}{(e^x+1)^2} = \sigma(1-\sigma) \\
\ddot{\sigma} &= \frac{-(e^x(e^x-1))}{(e^x+1)^3} = \sigma(1-\sigma)(1-2\sigma)
\end{aligned} \tag{5–33}$$

with bounds

$$\begin{aligned}
0 &\leq \sigma \leq 1 \\
0 &\leq \dot{\sigma} \leq \frac{1}{4} \\
\frac{-1}{6\sqrt{3}} &\leq \ddot{\sigma} \leq \frac{1}{6\sqrt{3}}.
\end{aligned} \tag{5–34}$$

Rearranging terms from (5–32) yields

$$\begin{aligned}
O(\tilde{V}_i^T\mu)^3 &= \dot{\sigma}(\hat{V}_i^T\mu)\tilde{V}_i^T\mu - (\sigma(\hat{V}_i^T\mu) - \sigma(V_i^T\mu)) \\
&\quad + \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T\mu)((V_i^T\mu - \hat{V}_i^T\mu) \odot \tilde{V}_i^T\mu).
\end{aligned} \tag{5–35}$$

As derived in Appendix A, this directly leads into the bounds on the adaptive error given by

$$\begin{aligned}
-u_{NN} - f_\Lambda(x) &= \hat{W}_i^T\sigma(\hat{V}_i^T\mu) - W_i^T\sigma(V_i^T\mu) - \epsilon_i \\
&= \tilde{W}_i^T(\sigma(\hat{V}_i^T\mu) - \dot{\sigma}(\hat{V}_i^T\mu)(\hat{V}_i^T\mu) + \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T\mu)(\hat{V}_i^T\mu \odot \hat{V}_i^T\mu)) \\
&\quad + \hat{W}_i^T\dot{\sigma}(\hat{V}_i^T\mu)\tilde{V}_i^T\mu - \frac{1}{2}\hat{W}_i^T\ddot{\sigma}(\hat{V}_i^T\mu)(\hat{V}_i^T\mu \odot \tilde{V}_i^T\mu) + h_i - \epsilon_i.
\end{aligned} \tag{5–36}$$

where we define h_i as

$$\begin{aligned}
h_i &= \tilde{W}_i^T(\dot{\sigma}(\hat{V}_i^T\mu)V_i^T\mu - \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T\mu)(\hat{V}_i^T\mu \odot V_i^T\mu)) \\
&\quad + \frac{1}{2}W_i^T\ddot{\sigma}(\hat{V}_i^T\mu)(V_i^T\mu \odot \tilde{V}_i^T\mu) - W_i^T O(\tilde{V}_i^T\mu)^3
\end{aligned} \tag{5–37}$$

which was chosen to include terms containing unknown coefficients (e.g. $W_i,\ V_i$) and higher order terms. The terms that are linear in $\tilde{W}_i,\ \tilde{V}_i$ with known coefficients can be

adapted for [17]. As we show in Appendix A, an upper bound can be established using the definition of h_i from (5–37) and the higher order terms definition in (5–35) [87, 88]:

$$
\begin{aligned}
||h_i - \epsilon_i|| &= ||\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu) V_i^T \mu - W_i^T \dot{\sigma}(\hat{V}_i^T \mu) \hat{V}_i^T \mu \\
&\quad + \frac{1}{2} W_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \hat{V}_i^T \mu) \\
&\quad - \frac{1}{2} \hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot V_i^T \mu) + W_i^T \sigma(\tilde{V}_i^T \mu) - \epsilon_i|| \qquad (5\text{–}38) \\
&\leq \zeta_i \psi_i. \qquad (5\text{–}39)
\end{aligned}
$$

We define the Frobenius norm of a matrix X as $||X||_F = \sqrt{trace(X^T X)}$ and use the relation $||XY||_F \leq ||X||_F \, ||Y||_F$ to form the following [14]:

$$
\begin{aligned}
\zeta_i(\bar{V}_i,\ \bar{W}_i) &= \max\{\bar{V}_i,\ \bar{W}_i,\ \bar{\epsilon}\} \qquad (5\text{–}40) \\
\psi_i(\hat{V}_i,\ \hat{W}_i,\ \mu) &= ||\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu)||_F \, ||\mu||_F + ||\dot{\sigma}(\hat{V}_i^T \mu) \hat{V}_i^T \mu||_F \qquad (5\text{–}41) \\
&\quad + \frac{1}{2} ||\ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \hat{V}_i^T \mu)||_F \\
&\quad + \frac{1}{2} ||\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu) \operatorname{diag}(\hat{V}_i^T \mu)||_F \, ||\mu||_F + 2.
\end{aligned}
$$

Notice that $\zeta_i(\bar{V}_i,\ \bar{W}_i)$ is a matrix of norms for unknown coefficients and $\psi_i(\hat{V}_i,\ \hat{W}_i,\ \mu)$ is matrix of known terms. We denote an upper bound on $||h_i - \epsilon_i||$ as $U_i \in \mathbb{R}^{m \times 1}$. The reader is referred to [88] and [16] for detailed derivations of upper bounds for neural network based adaptive controllers.

5.3.2 Robust Adaptive Control

Next, we discuss the adaptive laws stated in (5–24) to (5–27) which will be used in the stability analysis. Each adaptive law includes the projection operator (see [16] and [19]) as the chosen robust adaptive control technique which forces the weight estimates to remain within a known convex bounded set (bounded). It is worth noting that we chose not to use the $e - modification$ or $\sigma - modification$ operators in our adaptive laws due to their adverse effects on adaptation when operating with large

tracking errors [16]. Details regarding the projection operator, including definitions used in this paper, is provided in Appendix B.

The projection operator ensures that a system starting with any initial weights ($\hat{\Theta}(t_0)$) within the set Ω_0 will evolve with weights ($\hat{\Theta}(t)$) within the set Ω_1 for all $t \geq t_0$. In our neural network adaptive control set-up, we can use the properties of the projection operator to define upper-bounds for the weight matrices ($\hat{W}_i$ and $\hat{V}_i$). That is, we assume that the initial conditions of $\hat{W}_i$ and $\hat{V}_i$ lie in the compact sets Ω_{W_0} and Ω_{V_0} which are defined in the same manner as Ω_0 in Appendix B. Then, we can define the maximum values for the norm of the weight matrices $\hat{W}_i$ and $\hat{V}_i$ as [88]

$$\bar{\hat{W}}_i = \max_{\hat{W}_i \in \Omega_{W_1}} ||\hat{W}_i(t)|| \qquad \forall t \geq t_0 \tag{5–42}$$

$$\bar{\hat{V}}_i = \max_{\hat{V}_i \in \Omega_{V_1}} ||\hat{V}_i(t)||. \tag{5–43}$$

Similarly, the bound of the adaptive control effectiveness term can be stated as

$$\bar{\hat{K}}_\Lambda = \max_{\hat{K}_\Lambda \in \Omega_{K_{\Lambda_1}}} ||\hat{K}_\Lambda(t)|| \qquad \forall t \geq t_0. \tag{5–44}$$

5.4 Stability Analysis

Using the multiple Lyapunov function approach [30], consider the following Lyapunov candidate for each segment, s_i, as

$$\begin{aligned} V &= e^T P e + \frac{1}{2}\,\mathrm{trace}(\tilde{V}_i^T \Gamma_V^{-1} \tilde{V}_i) + \frac{1}{2}\,\mathrm{trace}(\tilde{W}_i^T \Gamma_W^{-1} \tilde{W}_i) \\ &\quad + \frac{1}{2}\,\mathrm{trace}((\tilde{K}_\Lambda \Lambda^{\frac{1}{2}})^T \Gamma_K^{-1} (\tilde{K}_\Lambda \Lambda^{\frac{1}{2}})) \end{aligned} \tag{5–45}$$

where time differentiating along the trajectories of (5–23) results in

$$\begin{aligned} \dot{V} &= (\dot{e}^T P e + e^T P \dot{e}) + \mathrm{trace}(\tilde{V}_i^T \Gamma_V^{-1} \dot{\tilde{V}}_i) + \mathrm{trace}(\tilde{W}_i^T \Gamma_V^{-1} \dot{\tilde{W}}_i) + \mathrm{trace}(\tilde{K}_\Lambda^T \Gamma_K^{-1} \dot{\tilde{K}}_\Lambda \Lambda) \\ &= -e^T Q_{ref} e + 2e^T P B (u_{AD} + f_\Lambda(x)) + 2e^T P B \Lambda (K_\Lambda (u_{AD} + u_{BL}) + u_{RB}) \\ &\quad + \mathrm{trace}(\tilde{K}_\Lambda^T \Gamma_K^{-1} \dot{\tilde{K}}_\Lambda \Lambda) + \mathrm{trace}(\tilde{V}_i^T \Gamma_V^{-1} \dot{\hat{V}}_i) + \mathrm{trace}(\tilde{W}_i^T \Gamma_W^{-1} \dot{\hat{W}}_i). \end{aligned}$$

By substituting the previous result into (5–36), we have

$$\begin{aligned}\dot{V} &= -e^T Q_{ref} e + \text{trace}(\tilde{V}_i^T \Gamma_V^{-1} \dot{\hat{V}}_i) + \text{trace}(\tilde{W}_i^T \Gamma_W^{-1} \dot{\hat{W}}_i) + \text{trace}(\tilde{K}_\Lambda^T \Gamma_K^{-1} \dot{\hat{K}}_\Lambda \Lambda) \\ &\quad -2e^T PB(\tilde{W}_i^T(\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu) + \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \hat{V}_i^T \mu)) \\ &\quad +\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu)\tilde{V}_i^T \mu - \frac{1}{2}\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \tilde{V}_i^T \mu) + h_i - \epsilon_i) \\ &\quad -2e^T PB\Lambda(\tilde{K}_\Lambda(u_{BL} + u_{NN} + u_{RB})) + 2e^T PBu_{RB}.\end{aligned} \tag{5–46}$$

By using the form of the adaptive update laws stated in (5–24) to (5–27), the projection operator property stated in (B–8), the vector property $\text{diag}(a)b = a \odot b$, and the trace property $\text{trace}(a + b) = \text{trace}(a) + \text{trace}(b)$, we can establish a simplified upper bound of (5–46) which is given by

$$\dot{V} \leq -e^T Q_{ref} e - 2e^T PB(h_i - \epsilon_i) + 2e^T PBu_{RB}. \tag{5–47}$$

Note that the derivations and details are provided in Appendix C.

Since we know that for all $e \in \mathbb{R}^n$,

$$\lambda_{min}(Q_{ref})\, ||e||^2 \leq e^T Q_{ref} e \leq \lambda_{max}(Q_{ref})\, ||e||^2 \tag{5–48}$$

then we can rewrite (5–47) using the upper bounds stated in (A–3) as

$$\dot{V} \leq -\lambda_{min}(Q_{ref})||e||^2 + 2||e||\lambda_{max}(P)||B||(U_i) + 2e^T PBu_{RB}. \tag{5–49}$$

5.4.1 Robust Control for Safe Switching

For our problem, switching in the system dynamics is introduced through the use of the sparse neural network adaptive controller. By using a robust control term and a multiple Lyapunov function approach with strict dwell time condition, we will ensure that switching between different segments in the adaptive controller does not result in instability. Consider the open-loop dynamics stated in (5–1) in the form of a family of dynamic systems [29]:

$$x = f_i(x,\, u) \tag{5–50}$$

where $i \in I$ is the segment number and f_i is locally Lipschitz.

Now consider the switched system generated from (5–50) [89]:

$$\dot{x} = f_S(x,\, u) \tag{5–51}$$

where a switching signal $S:\, \mathbb{R}^+ \to I$ specifies the index of the active segment at time t. For the remainder of this work, we will assume that the switching signal, S, is piecewise constant and right continuous.

Prevalent in switched system and hybrid literature, the average dwell time condition of a switching signal, S, can be stated as

$$N_{S(T,\,t)} \leq N_o + \frac{T-t}{\tau_\alpha} \tag{5–52}$$

where the switching signal, S, has an average dwell time of τ_α if there exist two numbers $N_o \in \mathbb{N}$ and $\tau_\alpha \in \mathbb{R}^+$ that satisfy the average dwell time condition stated in (5–52) where $N_{S(T,\,t)}$ denotes the number of switches on the time interval from $[t,\, T)$.

For the sparse neural network controller, we are interested in calculating a strict dwell time condition for the controller to follow. The dwell time condition holds if there exists a dwell time, $T_{dwell} \in \mathbb{R}^+$, such that

$$t_{S+1} - t_S \geq T_{dwell}, \qquad \forall s \in S \tag{5–53}$$

where t_S denotes the time of the Sth switch with dwelling interval $[t_S,\, t_{S+1})$. It can easily be shown that the strict dwell time condition is a special case of the average dwell time condition where $N_o = 1$ [29, 30]. In the case of the average dwell time condition, some switching intervals can be less than the specified average dwell time, τ_α.

In general, the use of the robust control can lead to high gain control and limit the learning performance of adaptive controllers. However, robust control terms can be used effectively in order to ensure safe switching between intervals. This is accomplished by

enabling a robust control term when the system error becomes larger than a predetermined threshold. While the error remains larger than the threshold, the robust control term remains active, and the sparse neural network is required to satisfy a dwelling time, T_{dwell}, requirement before switching to the next segment. This set-up ensures convergence to the predetermined error bound where the robust control term is deactivated and the controller performance is then determined based on the sparse adaptive neural network and the baseline controller.

Suppose u_{RB} takes the form:

$$u_{RB} = -(1 - f_{RB})k_{RB}\,\mathrm{sgn}(e^T PB) \tag{5–54}$$

where $k_{RB} > 0$ is selected to be the robust control gain and f_{RB} is used to fade out the effect of this control term. We define the fade out function, f_{RB}, by

$$f_{RB} = \begin{cases} 0 & ||e|| \geq r_{0_E} + \Delta_{RB} \\ 1 & ||e|| \leq r_{0_E} \end{cases} \tag{5–55}$$

where r_{0_E} is a design parameter used to define error bounds for the active region of the robust control term and Δ_{RB} and is selected to be the length of the fade out region.

Consider when $||e|| \geq r_{0_E} + \Delta_{RB}$ and the robust control term is active. By substituting (5–54) into (5–47), we derive the following inequality:

$$\dot{V} \leq -e^T Q_{ref} e - 2e^T PB((h_i - \epsilon_i) + k_{RB}\,\mathrm{sgn}(e^T PB)). \tag{5–56}$$

Using the equation [16]

$$e^T PBu_{RB} = -k_{RB} \sum_{j=1}^{m} |e^T PB|_j \tag{5–57}$$

and (5–56), where $|\cdot|$ denotes absolute value, results in the following

$$\dot{V} \leq -e^T Q_{ref} e + 2\sum_{j=1}^{m} |e^T PB|_j (||h_i - \epsilon_i|| - k_{RB}). \tag{5–58}$$

If we select the robust controller gain, k_{RB}, to satisfy the following inequality:

$$k_{RB} \geq U_i \tag{5–59}$$

then (5–58) becomes

$$\dot{V} \leq -e^T Q_{ref} e. \tag{5–60}$$

Consider the sphere set, $S_{r_0} \subset X$:

$$S_{r_0} = \left\{ \{e,\, \tilde{V}_i,\, \tilde{W}_i,\, \tilde{K}_\Lambda\} \;:\; ||e|| \leq r_{0_E} + \Delta_{RB} \right\} \tag{5–61}$$

where we define the radius associated with the largest Lyapunov function value inside S_{r_0} as

$$r_0 = \left(r_{0_E} + \Delta_{RB},\, \bar{\tilde{V}}_i,\, \bar{\tilde{W}}_i,\, \bar{\tilde{K}}_\Lambda \right)^T \tag{5–62}$$

which will be referenced in the later sections.

Without loss of generality, we define the upper bounds for the adaptive error terms in the Lyapunov candidate in (5–45) for the ith segment to be:

$$\bar{k}_{\tilde{V}_i} = \max_{\hat{V}_i \in \Omega_{V_1}} \frac{1}{2} \operatorname{trace}(\tilde{V}_i^T \Gamma_V^{-1} \tilde{V}_i) \tag{5–63}$$

$$\bar{k}_{\tilde{W}_i} = \max_{\hat{W}_i \in \Omega_{W_1}} \frac{1}{2} \operatorname{trace}(\tilde{W}_i^T \Gamma_W^{-1} \tilde{W}_i) \tag{5–64}$$

$$\bar{k}_{\tilde{K}_\Lambda} = \max_{K_\Lambda \in \Omega_{K_{\Lambda_1}}} \frac{1}{2} \operatorname{trace}((\tilde{K}_\Lambda \Lambda^{\frac{1}{2}})^T \Gamma_K^{-1} (\tilde{K}_\Lambda \Lambda^{\frac{1}{2}})) \tag{5–65}$$

where we denote the upper bounds of the adaptive errors as $\bar{\tilde{V}}_i$, $\bar{\tilde{W}}_i$, and $\bar{\tilde{K}}_\Lambda$.

Rewriting (5–58) in terms of the Lyapunov candidate in (5–45), we obtain:

$$\dot{V} \leq -c_V V + c_V \bar{k}_T \tag{5–66}$$

where $\bar{k}_T = \bar{k}_{\tilde{V}_i} + \bar{k}_{\tilde{W}_i} + \bar{k}_{\tilde{K}_\Lambda}$ and $c_V = \frac{\lambda_{min}(Q_{ref})}{\lambda_{max}(P)}$. Let t_0 and t_F be the initial time and final time while operating in a single segment, i, then (5–66) implies that

$$V(t) \leq \bar{k}_T + (V(t_0) - \bar{k}_T) e^{-c_V t} \tag{5–67}$$

for $\forall t \in [t_0,\ t_F]$.

We now calculate an upper bound for the dwell time of our adaptive system, T_{dwell}, based on the previous equations. Recall that we use dwell time to define the minimum time for the system to wait before switching to a different segment which ensures safe switching between segments.

Theorem 5.3. *Suppose that there exists continuously differentiable positive definite Lyapunov candidate functions $V_{i \in I}$. Let $t = [t_0,\ t_F]$ represent the complete time segment for which the robust control term is active, i.e. $||e|| > r_{0_E} + \Delta_{RB}$, where t_F is the final time and t_0 is the starting time. Suppose that the complete time segment (t) can be broken into a finite number of time segments (N_S) denoted Δt_S where the subscript S denotes the time segment number of the time segment defined by $\Delta t_S = [t_{S-1},\ t_S)$. Consider a switching sequence $\lambda = \{(\Delta t_1),\ (\Delta t_2),\ ...(\Delta t_{N_S})\}$ where exactly one set of nodes (i) with corresponding neural network weights ($\hat{V}_i,\ \hat{W}_i$) is active for each time segment in λ. If we assume the dwelling time, T_{dwell}, of the SNN adaptive controller is chosen to satisfy*

$$T_{dwell} \geq \frac{1}{c_V} ln\,(2) \tag{5–68}$$

then the system is guaranteed to enter the sphere set S_{r_0} in finite time with all closed-loop signals bounded.

Proof. Suppose the system is transitioning from segment Δt_S into segment Δt_{S+1} at time t_S. We imagine the set of active nodes for segment Δt_S is p while the set of active nodes for the segment Δt_{S+1} is c. Also let $V_p(t_S)$ and $V_c(t_S)$ denote the Lyapunov candidate values of the segments p and c, respectively, at the time instant t_S. Our goal is to show that by choosing a dwell time, T_{dwell}, that satisfies (5–68) then the Lyapunov candidate values will satisfy $V_p(t_{S-1})\ >\ V_c(t_S)$ and $V_p(t_S)\ >\ V_c(t_{S+1})$. Hence, this guarantees that the system will enter the sphere set S_{r_0} in finite time.

Consider the time instant, t_S, where the switch occurs, the tracking error $e_p(t_S)$ will initially be equivalent to $e_c(t_S)$ but the new set of (bounded) neural network weights

$(\tilde{V}_c, \tilde{W}_c)$ in the adaptive controller will cause a different Lyapunov result. That is, we define the contribution of the neural network weight errors to the Lyapunov candidates for segments c and p in the form stated in (5–45) as

$$k_c = \frac{1}{2}\,\mathrm{trace}(\tilde{V}_c^T \Gamma_V^{-1} \tilde{V}_c) + \frac{1}{2}\,\mathrm{trace}(\tilde{W}_c^{\,T} \Gamma_V^{-1} \tilde{W}_c) \tag{5–69}$$

$$k_p = \frac{1}{2}\,\mathrm{trace}(\tilde{V}_p^T \Gamma_V^{-1} \tilde{V}_p) + \frac{1}{2}\,\mathrm{trace}(\tilde{W}_p^{\,T} \Gamma_V^{-1} \tilde{W}_p) \tag{5–70}$$

which implies the instantaneous change in the Lyapunov value is upper bounded by

$$\begin{aligned} \epsilon_V &= V_c(t_S) - V_p(t_S) \\ &= k_c - k_p \\ &\leq \bar{k}_{NN} \leq \bar{k}_T \end{aligned} \tag{5–71}$$

where $\bar{k}_{NN} = \bar{k}_{\tilde{V}_i} + \bar{k}_{\tilde{W}_i}$ is defined in (5–66).

If we assume $||e|| > r_{0_E} + \Delta_{RB}$ while operating in the cth segment during the time interval Δt_{S+1}, then (5–67) becomes

$$V_c(t_{S+1}) \leq \bar{k}_T + (V_c(t_S) - \bar{k}_T) e^{-c_V \Delta t_{S+1}} \tag{5–72}$$

By forcing the system to abide by the dwell time requirement, i.e. $\forall \Delta t_S \geq T_{dwell}$, then (5–72) becomes

$$V_c(t_{S+1}) \leq \bar{k}_T + (V_c(t_S) - \bar{k}_T) e^{-c_V T_{dwell}} \tag{5–73}$$

Using the inequality in (5–71), we obtain

$$V_c(t_{S+1}) \leq \bar{k}_T + (V_p(t_S) + \bar{k}_{NN} - \bar{k}_T) e^{-c_V T_{dwell}} \tag{5–74}$$

Since our goal is to find a T_{dwell} such that $V_c(t_{S+1}) < V_p(t_S)$, then it is sufficient to prove that

$$\bar{k}_T + (V_p(t_S) + \bar{k}_{NN} - \bar{k}_T) e^{-c_V T_{dwell}} < V_p(t_S) \tag{5–75}$$

which implies that

$$T_{dwell} > \frac{1}{c_V} ln \left(\frac{V_p(t_S) + \bar{k}_{NN} - \bar{k}_T}{V_p(t_S) - \bar{k}_T} \right). \tag{5–76}$$

By assuming that $V_p(t_S) > \bar{k}_T + \bar{k}_{NN} + \bar{k}_\Lambda = 2\bar{k}_T$ where $\bar{k}_T \geq \bar{k}_{NN}$, then it follows that

$$\frac{1}{c_V} ln\,(2) \geq \frac{1}{c_V} ln \left(\frac{\bar{k}_{NN} + \bar{k}_T}{\bar{k}_T} \right) > \frac{1}{c_V} ln \left(\frac{V_p(t_S) + \bar{k}_{NN} - \bar{k}_T}{V_p(t_S) - \bar{k}_T} \right). \tag{5–77}$$

Hence, if T_{dwell} is selected to satisfy (5–68) then (5–76) will also be satisfied. It is worth noting that a larger lower bound assumption on $V_p(t_S)$ would result in a smaller dwell time requirement.

Next, let us assume that $||e|| > r_{0_E} + \Delta_{RB}$ while operating in the segment p during the time interval Δt_S, then (5–67) becomes

$$V_p(t_S) \leq \bar{k}_T + (V_p(t_{S-1}) - \bar{k}_T)e^{-c_V \Delta t_S} \leq \bar{k}_T + (V_p(t_{S-1}) - \bar{k}_T)e^{-c_V T_{dwell}} \tag{5–78}$$

and after rearranging terms results in

$$V_p(t_{S-1}) \geq \bar{k}_T + (V_p(t_S) - \bar{k}_T)e^{c_V T_{dwell}}. \tag{5–79}$$

By plugging the lower bound derived in (5–75) into (5–79) results in the following inequality:

$$V_p(t_{S-1}) > V_p(t_S) + \bar{k}_{NN} \tag{5–80}$$

which implies that $V_c(t_S) < V_p(t_{S-1})$. The previous inequality also suggests that the previous assumption used to derive a dwell time requirement, i.e. $V_p(t_S) > 2\bar{k}_T$, implies $V_p(t_{S-1}) > 2\bar{k}_T + \bar{k}_{NN}$. The proof is complete.

Notice that we are interested in finding an error bound, $\bar{e}_R$, such that the tracking error is guaranteed to enter the sphere set, S_{r_0}, in finite time, t_F, where

$$||e(t)|| = ||x(t) - x_{ref}(t)|| \leq \bar{e}_R. \tag{5–81}$$

By using (5–45), (5–67), and the upper bound

$$V(t) \geq e^T P e \geq \lambda_{min}(P)||e||^2 \tag{5–82}$$

we find the following relationship

$$||e(t)||^2 \leq \left(\frac{V(t_0) - \bar{k}_T}{\lambda_{min}(P)}\right) e^{-c_V t} + \frac{\bar{k}_T}{\lambda_{min}(P)} \tag{5–83}$$

for $\forall t$. By using the assumption $V(t_0) > 2\bar{k}_T + \bar{k}_{NN}$ and the dwell time requirement, T_{dwell}, from (5–77), results in

$$||e(t)|| \leq \sqrt{\frac{2\bar{k}_T}{\lambda_{min}(P)}} = \bar{e}_R \tag{5–84}$$

where $r_{0_E} + \Delta_{RB}$ must be selected to satisfy $r_{0_E} + \Delta_{RB} > \bar{e}_R$.

By the design of the SNN discussed in Section 5.3, it is clear that only one set of nodes is active for each discrete time segment for which the controller is active. In practice, time segments are not designed to be equivalent in length due to the possible variations in segment sizes. Also, the varying processing speed of the on-board processors along with fast switching could result in instability. However, by using the robust control term with error threshold and enforcing the dwelling time condition of (5–68), then $V_c(t_{S+1}) < V_p(t_S)$ holds for $\forall S$ for which $||e|| > r_{0_E} + \Delta_{RB}$. Notice that if $||e|| \leq r_{0_E} + \Delta_{RB}$ while operating in time segment Δt_S, then the system already belongs to the sphere set S_{r_0} and the robust control term is not active. This process will continue until the end of flight time. It can be shown that all signals in the closed loop system remain uniformly bounded based on (5–3), (5–19), (5–45), and (5–66).

Notice that we can relate the strict dwell time condition derived in (5–76) to common derivations in switched systems control by considering the generic form of the switched system shown in (5–51) [29, 30, 89]. This derivation is given in Appendix D.

5.4.2 Sparse Neural Network Control

Now consider when the system enters the sphere set S_{r_0} for which $||e|| \leq b_{RB} + \Delta_{RB}$ and the robust control term is not active ($i.e.\ u_{RB} = 0$). Starting from (5–49), we can write

$$\dot{V} \leq -\lambda_{min}(Q_{ref})||e||^2 + 2||PB||(||e||)[U_i]. \tag{5–85}$$

After rearranging, the Lyapunov derivative becomes

$$\dot{V} \leq -\lambda_{min}(Q_{ref})\left(||e|| - \frac{||PB||[U_i]}{\lambda_{min}(Q_{ref})}\right)^2 + \frac{(||PB||[U_i])^2}{\lambda_{min}(Q_{ref})}. \tag{5–86}$$

Hence, $\dot{V}(e,\, \tilde{V}_i,\, \tilde{W}_i,\, \tilde{K}_\Lambda) < 0$ if

$$||e|| > \frac{2||PB||[U_i]}{\lambda_{min}(Q_{ref})} \tag{5–87}$$

and $\dot{V}(e,\, \tilde{V}_i,\, \tilde{W}_i,\, \tilde{K}_\Lambda) < 0$ outside the sphere set, $S_{r_1} \subset X$:

$$S_{r_1} = \left\{ e \in \mathbb{R}^n \ : \ ||e|| \leq \frac{2||PB||[U_i]}{\lambda_{min}(Q_{ref})} = r_{1_E} \right\} \tag{5–88}$$

Using the result of (5–88), we define the radius associated with the largest Lyapunov function value inside S_{r_1} as

$$r_1 = \left(\frac{2||PB||[U_i]}{\lambda_{min}(Q_{ref})},\, \bar{\tilde{V}}_i,\, \bar{\tilde{W}}_i,\, \bar{\tilde{K}}_\Lambda \right)^T. \tag{5–89}$$

The next portion of this analysis details the derivation of an ultimate bound for the previously defined adaptive controller. See Figure 5-4 as a reference to the sets referred to in the stability analysis.

Notice, we can rewrite the Lyapunov function candidate in (5–45) as

$$V = \zeta^T M \zeta \tag{5–90}$$

where we define the Lyapunov vector as $\zeta = [e,\, \tilde{W}_i,\, \tilde{V}_i,\, (\tilde{K}_\Lambda \Lambda^{\frac{1}{2}})]^T$ and M is defined as

$$M \triangleq \begin{bmatrix} P & 0 & 0 & 0 \\ 0 & \Gamma_W^{-1} & 0 & 0 \\ 0 & 0 & \Gamma_V^{-1} & 0 \\ 0 & 0 & 0 & \Gamma_K^{-1} \end{bmatrix} \tag{5–91}$$

We define the largest sphere set, S_R, contained in the domain, X, as

$$S_R = \{\zeta \in X : \ ||\zeta|| \leq R\} \tag{5–92}$$

with the assumption that $R > r_0 > r_2 > r_1$.

Using the Lyapunov function candidate in (5–45), we can define two class κ functions γ_1 and γ_2 which define the bounds:

$$\begin{aligned} \gamma_1(\zeta) &= \lambda_{min}(P)||e||^2 + \lambda_{min}(\Gamma_W^{-1})(\tilde{W})^2 + \\ &\quad \lambda_{min}(\Gamma_V^{-1})(\tilde{V})^2 + \lambda_{min}(\Gamma_K^{-1})(\tilde{K}_\Lambda \Lambda^{\frac{1}{2}})^2 \end{aligned} \tag{5–93}$$

$$\begin{aligned} \gamma_2(\zeta) &= \lambda_{max}(P)||e||^2 + \lambda_{max}(\Gamma_W^{-1})(\tilde{W}) + \\ &\quad \lambda_{max}(\Gamma_V^{-1})(\tilde{V})^2 + \lambda_{max}(\Gamma_K^{-1})(\tilde{K}_\Lambda \Lambda^{\frac{1}{2}})^2 \end{aligned} \tag{5–94}$$

where

$$\gamma_1(\zeta) \leq V(\zeta) \leq \gamma_2(\zeta) \tag{5–95}$$

and more generically

$$\lambda_{min}(M)||\zeta||^2 \leq V(\zeta) \leq \lambda_{max}(M)||\zeta||^2. \tag{5–96}$$

Next, we define compact sets Ω_o and Ω_i as [88]:

$$\Omega_o = \{\zeta \in S_R : \ V \leq \omega_o \triangleq \min_{||\zeta||=R} V = R^2 \lambda_{min}(M)\} \tag{5–97}$$

$$\Omega_i = \{\zeta \in S_R : \ V \leq \omega_i \triangleq \max_{||\zeta||=r_1} V = r_1^2 \lambda_{max}(M)\} \tag{5–98}$$

where w_i denotes the maximum value of the Lyapunov function, V, on the edge of S_{r_1} and w_o is the minimum value of the Lyapunov function, V, on the edge of S_R.

Let us also introduce the smallest sphere set, S_{r_2}, that contains Ω_i as

$$S_{r_2} = \{\zeta \in S_R : \ ||\zeta|| \leq r_2\}. \tag{5–99}$$

Next, we create an annulus set Λ given by

$$\Lambda = \{\zeta \in X : \ \omega_i \leq V(\zeta) \leq \omega_o\} \tag{5–100}$$

where the time derivative of $V(\zeta)$ is strictly negative definite inside Λ. For all $\zeta \in \Omega_i$,

$$\lambda_{min}(M)||\zeta||^2 \leq \zeta^T M \zeta \leq \lambda_{max}(M) r_1^2 \tag{5–101}$$

which implies

$$||\zeta||^2 \leq \frac{\lambda_{max}(M)}{\lambda_{min}(M)} r_1^2 = r_2^2. \tag{5–102}$$

Using the relationship derived in (5–102) and the definition of r_1 in (5–89), we can derive an ultimate bound, r_2, given by

$$r_2 = \sqrt{\frac{\lambda_{max}(M)}{\lambda_{min}(M)}} r_1 \tag{5–103}$$

where

$$||\zeta(t)|| \leq r_2, \qquad \forall t \geq t_0 + T \tag{5–104}$$

which is equivalent to applying the formula shown in (5–13) of Theorem 5.1. That is, $||\zeta|| \leq \gamma_1^{-1}(\gamma_2(r_1)) = r_2$ [77, 82].

Theorem 5.4. *Consider the system in (5–1) with control effectiveness term Λ and matched uncertainty $f(x)$, the control law stated in (5–8), and the sparse neural network (SNN) switching scheme. Then, the adaptive switching weight update laws in (5–24), (5–25), and (5–27) ensure that the tracking error, e, of (5–22) and the SNN weight errors, ($\tilde{W}_i$, $\tilde{V}_i$) remain uniformly ultimately bounded (UUB) while operating within each segment, $\forall i$.*

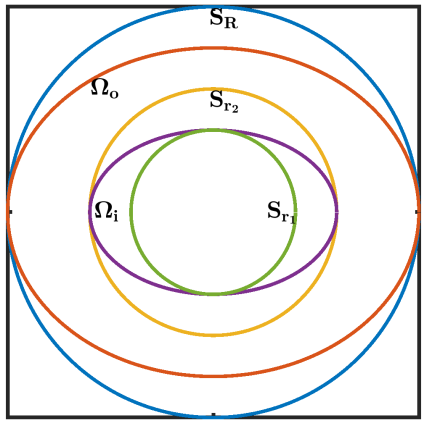

Figure 5-4. Visual Lyapunov function used for stability analysis.

Proof. By Theorem 5.3, the robust controller ensures safe switching between segments if the dwell time condition stated in (5–68) is met when operating above the error threshold specified in (5–55). Through the use of robust control (Section 5.4.1), the projection operator (Section 5.3.2), and the assumption that all trajectories, $\zeta(t_0)$, start in Ω_o ensures the convergence to the sphere set S_{r_0} in finite time with bounded weight errors.

While operating within a single segment inside the sphere set S_{r_0}, (5–86) reveals that the tracking error is $e \in L_\infty$ and the neural network weight errors in the operating region are $(\tilde{V}_i,\ \tilde{W}_i) \in L_\infty$. As stated previously, the input command is required to be $y_{cmd} \in L_\infty$ and A_{ref} is designed to be Hurwitz which implies $(x_{ref},\ x) \in L_\infty$. Since the matrices of the ideal neural network parameters for each region $(W_i,\ V_i)$ are bounded, then the neural network weight errors and estimates are also $(\hat{V}_i,\ \hat{W}_i) \in L_\infty$. Since the baseline controller is designed to be $u_{BL} \in L_\infty$ and the adaptive controller consists of bounded neural network weight estimates, then the overall control signal is $u \in L_\infty$. Hence, the closed-loop error system is $\dot{e} \in L_\infty$ which implies that $\ddot{V} \in L_\infty$. Since $\dot{V}$ is strictly negative in the annulus set Λ and V is lower bounded, then V decreases monotonically until the solution enters the set Ω_i in finite time with ultimate bound, r_2.

Hence, this analysis and the previously derived equations establish that if the initial conditions of the state vector, $\zeta(t_0)$, lie in Ω_o defined by (5–97), then the control law given by (5–19), (5–8), and (5–3) and the adaptation laws stated in (5–24), (5–25), and (5–27) ensure that the state tracking error, e, and neural network weight errors for each segment, $\tilde{V}_i$ and $\tilde{W}_i$, in the closed-loop system are uniformly ultimately bounded (UUB).

5.5 Hypersonic Flight Vehicle Dynamics with Flexible Body Effects

In typical subsonic flight control systems the stiffness of the aircraft body, the benign operating conditions, and the natural frequency separation between the rigid body modes and the flexible body modes allow flexible body effects to be ignored in modeling due to their negligible effect during flight. Since hypersonic flight vehicles operate at extreme temperatures and high speeds, the vehicle's flexibility and state coupling can create dramatic changes in the overall flow field which causes variations in the pressure distribution on the flight vehicle [6, 90]. In addition, the flexing of the fuselage can cause unexpected control moment effects from the control surfaces. For these reasons, we investigate the performance of the SNN controller versus the traditional SHL approach on a hypersonic vehicle with flexible body effects.

We consider a highly nonlinear hypersonic flight vehicle model with four independent servo-controlled fins $(\delta_1,\ \delta_2,\ \delta_3,\ \delta_4)$ oriented in an X-configuration [73]. For convenience, we created virtual fins (aileron δ_a, elevator δ_e, and rudder δ_r) used for control design in the conventional autopilot reference frame. We created a static mapping from the virtual fins to actual fin displacement in order to calculate forces and moments of the flight vehicle (see [73]).

In this research we consider only longitudinal dynamics of the flight vehicle, which are assumed to be entirely decoupled from the lateral dynamics. We can write the longitudinal 3-DoF equations of motion for a hypersonic flight vehicle in following form

(see [74] and [6]):

$$
\begin{aligned}
\dot{V}_T &= \frac{1}{m}(T\,\cos(\alpha) - D) - g\,\sin(\theta - \alpha) \\
\dot{\alpha} &= \frac{1}{mV_T}(-T\,\sin(\alpha) - L) + q + \frac{g}{V_T}\cos(\theta - \alpha) \\
\dot{\Theta} &= q \\
\dot{q} &= \frac{M}{I_{YY}} \\
\dot{h} &= V_T\,\sin(\theta - \alpha) \\
\ddot{\eta}_i &= -2\zeta_i\omega_i\dot{\eta}_i - \omega_i^2\eta_i + N_i, \qquad i = 1, 2, .., n_\eta
\end{aligned}
\tag{5–105}
$$

where m is the mass of the vehicle, Θ is the pitch angle, q is pitch rate, I_{YY} is the moment of inertia, and g is gravity. The equations for the ith structural mode of the flight vehicle are defined by the natural frequency (w_i), the damping ratio (ζ_i), and the generalized force (N_i). The natural frequencies of the hypersonic vehicle's body modes vary significantly based on temperature changes experiences throughout flight [7]. Hence, we will consider w_i a function of temperature, T. The forces and moments acting on the flight vehicle consist of thrust (T), drag (D), lift (L), and pitch moment (M). If we assume three elastic modes of the flight vehicle are active, the state vector, $x \in \mathbb{R}^{11}$, is given by

$$
x = [V_T,\, \alpha,\, \Theta,\, q,\, h,\, \eta_1,\, \dot{\eta}_1, \eta_2,\, \dot{\eta}_2,\, \eta_3,\, \dot{\eta}_3] \tag{5–106}
$$

where V_T is the true airspeed, α is angle of attack, and h is the altitude of the flight vehicle.

The axial and normal body forces ($A,\, N$) and pitching moment (M) can be approximated by (see [74] or [73]):

$$
A \approx \frac{1}{2}\rho V_T^2 S C_A \tag{5–107}
$$

$$
N \approx \frac{1}{2}\rho V_T^2 S C_N \tag{5–108}
$$

$$
M \approx \frac{1}{2}\rho V_T^2 S c_{ref} C_m \tag{5–109}
$$

$$N_i \approx \frac{1}{2}\rho V_T^2 S C_{N_i} \qquad i = 1, 2, .., n_\eta \tag{5–110}$$

where ρ denotes air density, S is the reference area, c_{ref} is the mean aerodynamic chord, and we assume zero thrust (i.e. $T = 0$). We assume the following mapping from axial and normal (A, N) forces to lift and drag forces (L, D) used in (5–105):

$$L = N\,\cos(\alpha) - A\,\sin(\alpha) \tag{5–111}$$

$$D = N\,\sin(\alpha) + A\,\cos(\alpha). \tag{5–112}$$

The axial force coefficient (C_A), normal force coefficient (C_N), pitch moment coefficient (C_m), and the generalized force (N_i) appearing in (5–107) to (5–109) take the following form:

$$C_A = C_{A_{ALT}}(h,\,\mathrm{Mach}) + C_{A_{AB}}(\alpha,\,\mathrm{Mach}) + \sum_{j=1}^{4} C_{A_{\delta_j}}(\alpha,\,\mathrm{Mach},\,\delta_j) \tag{5–113}$$

$$C_N = C_{N_0}(\alpha,\,\mathrm{Mach}) + \sum_{j=1}^{4} C_{N_{\delta_j}}(\alpha,\,\mathrm{Mach},\,\delta_j) + C_{N_q}(\alpha,\,\mathrm{Mach},\,q) \tag{5–114}$$

$$C_m = C_{m_0}(\alpha,\,\mathrm{Mach}) + \sum_{j=1}^{4} C_{m_{\delta_j}}(\alpha,\,\mathrm{Mach},\,\delta_j) + C_{m_q}(\alpha,\,\mathrm{Mach},\,q) \tag{5–115}$$

$$C_{N_i} = N_{i_{\alpha^2}}(\alpha^2) + N_{i_\alpha}(\alpha) + \sum_{j=1}^{4} N_{i_{\delta_j}}(\delta_j) + \sum_{k=1}^{3} N_{i_{\eta_k}}(\eta_k) \qquad i = 1, 2, .., n_\eta \tag{5–116}$$

where the aerodynamic coefficients can be computed using a look-up table based on the flight condition $(\alpha,\,\mathrm{Mach},\,h,\,q)$, the control inputs $(\delta_1,\,\delta_2,\,\delta_3,\,\delta_4)$, and the flexible body states (η_1, η_2, η_3). The total lift force, drag force, and pitch moment equations can be stated as

$$L_T = L + L_{flex} \tag{5–117}$$

$$D_T = D + D_{flex} \tag{5–118}$$

$$M_T = M + M_{flex} \tag{5–119}$$

where the contributions due to the flexible modes take the form:

$$L_{flex} = \frac{1}{2}\rho V_T^2 S(c_{L1}\eta_1 + c_{L2}\eta_2 + c_{L3}\eta_3) \tag{5–120}$$

$$D_{flex} = \frac{1}{2}\rho V_T^2 S(c_{D1}\eta_1 + c_{D2}\eta_2 + c_{D3}\eta_3) \tag{5–121}$$

$$M_{flex} = \frac{1}{2}\rho V_T^2 S(c_{M1}\eta_1 + c_{M2}\eta_2 + c_{M3}\eta_3) \tag{5–122}$$

where c_{L1}, c_{L2}, c_{L3}, c_{D1}, c_{D2}, c_{D3}, c_{M1}, c_{M2}, and c_{M3} are constants.

In order to determine control-oriented models suitable for MRAC, we select a dense set of trim points to suitably fill the potential flight envelope. The flight conditions of the trim points are selected to accurately represent the variations of the modeled dynamics of the flight vehicle throughout the flight envelope. For each flight condition for which a trim point is located, we are interested in a determining a longitudinal linear vehicle model which includes the short-period modes and the structural bending modes. In order to accomplish this task, we decouple the short period and phugoid modes of the vehicle by substituting trim values for V_T, h and Θ into (5–105).

Next, by representing the dynamical equations in (5–105) as a set of ordinary differential equations given by

$$\dot{x} = f(t, x), \qquad x(t_0) = x_0, \qquad t \geq t_0 \tag{5–123}$$

allows us to compute the linear short period plant matrices $A_p \in \mathbb{R}^{n_p \times n_p}$, $B_p \in \mathbb{R}^{n_p \times m}$, $C_p \in \mathbb{R}^{p \times n_p}$, and $D_p \in \mathbb{R}^{p \times m}$ [73]. That is, the flight model is numerically linearized with respect to the states and control inputs around each flight condition where the short period matrices take the form:

$$\begin{aligned} A_p(i,\, j) &= \left.\frac{\partial f(i)}{\partial x(j)}\right|_{\substack{x=x^* \\ u=u^*}}, \quad B_p(i,\, k) = \left.\frac{\partial f(i)}{\partial u(k)}\right|_{\substack{x=x^* \\ u=u^*}}, \\ C_p(i,\, j) &= \left.\frac{\partial x(i)}{\partial x(j)}\right|_{\substack{x=x^* \\ u=u^*}}, \quad D_p(i,\, k) = \left.\frac{\partial x(i)}{\partial u(k)}\right|_{\substack{x=x^* \\ u=u^*}}, \end{aligned} \tag{5–124}$$

where trim conditions are denoted by asterisks as x^* and u^*, i and j are indices of the state vector and k is the index of the control input. For control purposes, we create an additional integral error of tracking (e_I) state to include in the linear model. The resulting augmented short period linearized model takes the form:

$$\begin{bmatrix} \dot{e}_I \\ \dot{x}_p \end{bmatrix} = \begin{bmatrix} 0 & C_{reg} \\ 0 & A_p \end{bmatrix} \begin{bmatrix} e_I \\ x_p \end{bmatrix} + \begin{bmatrix} 0 \\ B_p \end{bmatrix} u + \begin{bmatrix} -1 \\ 0 \end{bmatrix} y_{cmd} \tag{5–125}$$

where $C_{reg} \in \mathbb{R}^{m \times n_p}$ selects the regulated state, the state vector is given by

$$x = [e_I,\, \alpha,\, q,\, \eta_1,\, \dot{\eta}_1\,, \eta_2,\, \dot{\eta}_2,\, \eta_3,\, \dot{\eta}_3] \in \mathbb{R}^9 \tag{5–126}$$

which includes the integral error of tracking, angle of attack, pitch rate, flexible mode positions, and flexible mode rates. We assume the flexible modal coordinates are not measured or used for feedback. The controller output elevator deflection ($u = \delta_e$) is produced an actuator. Notice by introducing the form of the uncertainties in the system (Λ and $f(x)$), (5–125) takes the general form of the MRAC problem shown in (5–1). For each flight condition, we then determine fixed baseline controller gains using LQ methods as discussed in Section 5.1. For implementation, the complete baseline control signal of the nonlinear flight vehicle is determined by gain-scheduling the linear controller gains ($K_I,\, K_P$) by Mach, angle of attack (α), and altitude (h).

5.6 Adaptive Control Results

In this section, we reveal the simulation results using various adaptive controllers (SHL and SNN) on the hypersonic vehicle model while tracking a simple sinusoidal command with relatively slow frequency ($f \approx 0.24\,Hz$) between -3 and 3 degrees angle of attack (α) for $t_f = 250\,sec$. The command requires the vehicle to spend similar amounts of time in each region of the flight envelope while repeating each sinusoidal maneuver approximately six times. We will refer to one complete (period) sinusoidal maneuver as a “pass”. This simulation was designed to provide an environment for

Table 5-1. Range of flexible mode frequencies and temperature of HSV

	MIN	MAX
$\mathrm{Temp}(F)$	-100	2000
$\omega_1(rad/sec)$	15	20
$\omega_2(rad/sec)$	35	45
$\omega_3(rad/sec)$	50	60

adaptive control long-term learning and tracking comparison. During testing, we assumed constant flexible mode damping terms where $\zeta_1 = \zeta_2 = \zeta_3 = 0.02$ and learning rate matrices for the adaptive controllers were set to either small (S) ($\Gamma_W = 2.5e^{-5} \times I$ and $\Gamma_V = 1.25e^{-5} \times I$) or moderate (M) ($\Gamma_W = 5e^{-5} \times I$ and $\Gamma_V = 2.5e^{-5} \times I$) where I denotes the identity matrix of appropriate size. In order to provide a proper comparison of the SHL and SNN architectures, the same number of active nodes were used in each test case $(i.e.\ N_{act} = 16)$. Also, note that a constant $\Delta t = 0.01\, sec$ time step with a second order integration method (AB-2) was used for the simulation.

For the sake of brevity, we selected a region surrounding a specific trim point in the flight envelope to analyze (i.e. $\mathrm{Mach} = 6.0$, $\Theta = 0,$ and $h = 14km$). In addition to varying angle of attack (α) and pitch rate (q) in the nonlinear model during simulation, we also assume that temperature varies due to angle of attack (α) (assuming constant velocity) where we use the following relationship between temperature and angle of attack (α):

$$Temp = k_{1t}\alpha^2 + k_{2t}\alpha + k_{3t} \tag{5–127}$$

where k_{1t}, k_{2t}, and k_{3t} are constants. We also assume a similar relationship between temperature and natural frequency (ω_i) of each flexible mode:

$$\omega_1 = k_{1a}\,\mathrm{Temp}^2 + k_{1b}\,\mathrm{Temp} + k_{1c} \tag{5–128}$$

$$\omega_2 = k_{2a}\,\mathrm{Temp}^2 + k_{2b}\,\mathrm{Temp} + k_{2c} \tag{5–129}$$

$$\omega_3 = k_{3a}\,\mathrm{Temp}^2 + k_{3b}\,\mathrm{Temp} + k_{3c} \tag{5–130}$$

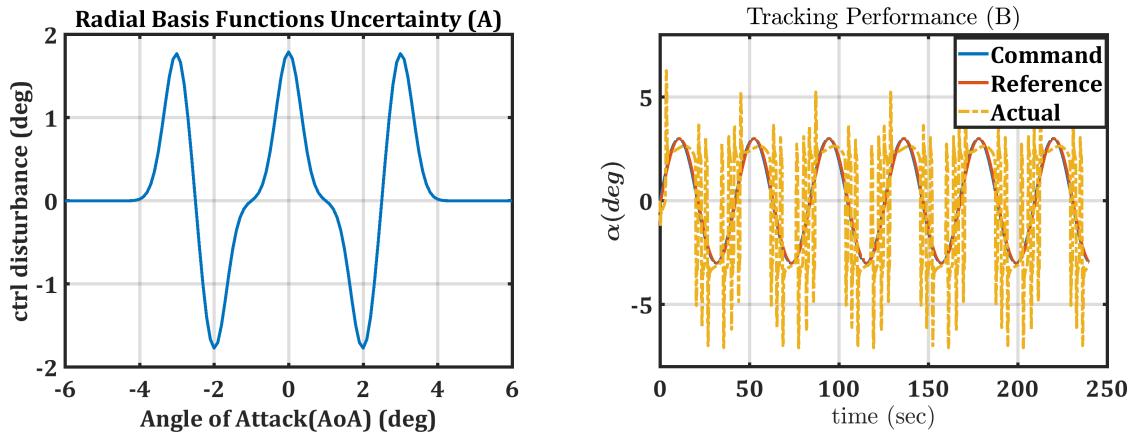

Figure 5-5. Hypersonic baseline controller under A) significant RBF based matched uncertainty with B) resulting tracking performance.

where k_{1a}, k_{1b}, k_{1c}, k_{2a}, k_{2b}, k_{2c}, k_{3a}, k_{3b}, and k_{3c} are constants and the range of temperatures and mode frequencies are shown in Table 5-1.

The uncertainty term, $f(x)$, that was used during the simulation was created using the summation of several radial basis functions (RBFs) centered at various angles of attack (α). The magnitudes for the RBFs were set to bring the baseline controller to the brink of instability when operating without adaptive control. The spacing of the RBFs was chosen to significantly vary the uncertainty based on angle of attack (α). The uncertainty term and baseline tracking performance can be seen in Figure 5-5.

5.6.1 Single Hidden Layer (SHL) Neural Network Adaptive Control

We first provide the results of the traditional SHL MRAC adaptive controller with adaptive update laws in the following form [17, 22]:

$$\dot{\hat{W}} = \text{Proj}(2\Gamma_W((\sigma(\hat{V}^T\mu) - \dot{\sigma}(\hat{V}^T\mu)\hat{V}^T\mu)e^TPB) \tag{5–131}$$

$$\dot{\hat{V}} = \text{Proj}(2\Gamma_V\mu e^TPB\hat{W}^T\dot{\sigma}(\hat{V}^T\mu)) \tag{5–132}$$

where we will refer to this controller as SHL-TS1. For this approach, the number of active nodes (N_{act}) is equal to the number of total nodes (N). The adaptive laws in the traditional case were derived using a first-order Taylor expansion.

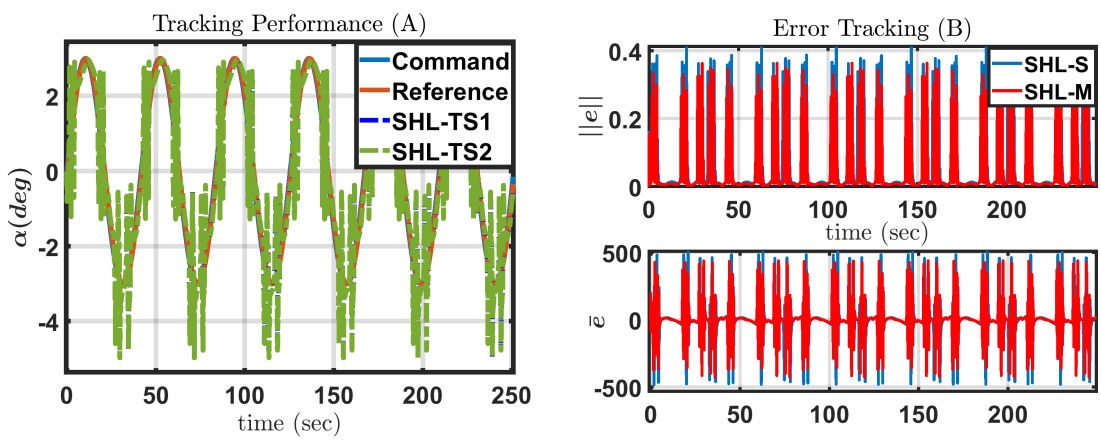

Figure 5-6. Single hidden layer (SHL) hypersonic A) tracking performance and B) error tracking.

We also consider the case of using the adaptive update laws:

$$\dot{\hat{W}} = \mathrm{Proj}(\Gamma_W(2(\sigma(\hat{V}^T\mu) - \dot{\sigma}(\hat{V}^T\mu)\hat{V}^T\mu) + \ddot{\sigma}(\hat{V}^T\mu)\,\mathrm{diag}(\hat{V}^T\mu)\hat{V}^T\mu)e^TPB) \tag{5–133}$$

$$\dot{\hat{V}} = \mathrm{Proj}(\Gamma_V\mu e^TPB\hat{W}^T(2\dot{\sigma}(\hat{V}^T\mu) - \ddot{\sigma}(\hat{V}^T\mu)\,\mathrm{diag}(\hat{V}^T\mu))) \tag{5–134}$$

derived using the second-order Taylor series expansion where we will refer to this adaptive controller as SHL-TS2.

As we see in the tracking plots of Figure 5-6, the SHL adaptive controllers achieve similar performance for each sinusoidal maneuver and do not improve with repeated passes due to the lack of learning capability in the traditional SHL adaptive architecture. Also, note that the SHL-TS2 performs similarly to the SHL-TS1 while using the same learning rates and active nodes during the simulation. A tracking comparison performance of the SHL-TS1 while varying the adaptive learning rates is shown in the error plots of Figure 5-6, where $||e||$ is defined as the 2-norm where $e = x - x_{ref}$ and $\bar{e}$ is defined by the following equation, $\bar{e} = e^TPB$.

5.6.2 Sparse Neural Network (SNN) Adaptive Control

Since the frequencies of the flexible modes explicitly depend on temperature and the flexible modes impact the moment and force equations, we will utilize a 2-D SNN

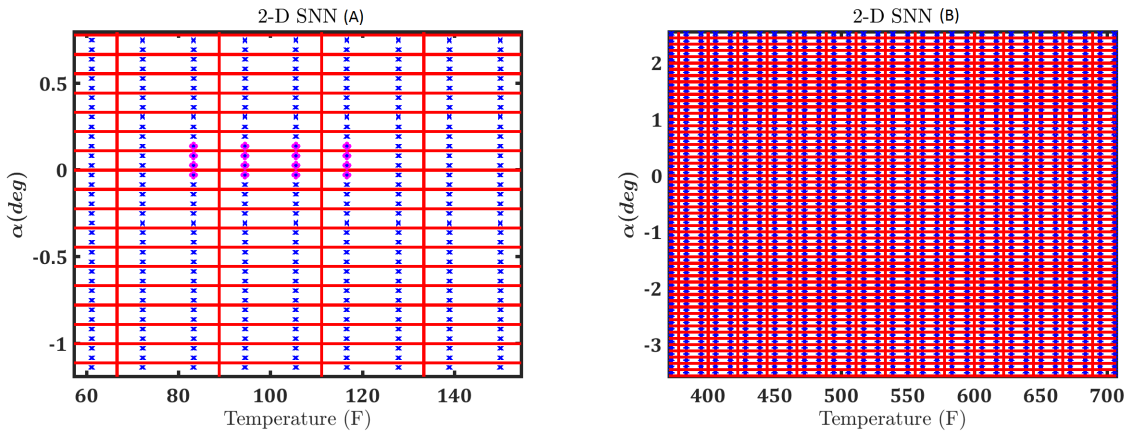

Figure 5-7. Hypersonic two dimensional flight envelope partition in A) zoomed in and B) bird's eye form.

architecture shown in Figure 5-7 where each red rectangle signifies the border of a segment in the flight envelope and each blue X indicates a particular location of a node in the SNN architecture. We will utilize the same values for learning rates and number of active nodes is previously specified. We chose to divide the flight envelope into $T = 4050$ segments with $Q = 4$ nodes allocated to each segment where each segment operates using 12 nodes. A close-up view of an arbitrary operating segment can be seen in Figure 5-7 where the magenta circles denote the active nodes of the current segment.

Similar to the previous subsection, we will compare the performance of two adaptive controllers utilizing the same architecture ($i.e.$ SNN), learning rates, commands, reference model, and architecture parameters ($i.e.$ number of active nodes (R), number of total nodes (N), number of nodes per segment (Q), and number of total segments (T)) but vary the form of adaptive update laws. First we consider an adaptive controller with adaptive update laws exercised in Chapter 4, in the following form:

$$\dot{\hat{W}}_i = \mathrm{Proj}(2\Gamma_W((\sigma(\hat{V}_i^T\mu) - \dot{\sigma}(\hat{V}_i^T\mu)\hat{V}_i^T\mu)e^T PB) \tag{5–135}$$

$$\dot{\hat{V}}_i = \mathrm{Proj}(2\Gamma_V \mu e^T PB\hat{W}_i^T\dot{\sigma}(\hat{V}_i^T\mu)) \tag{5–136}$$

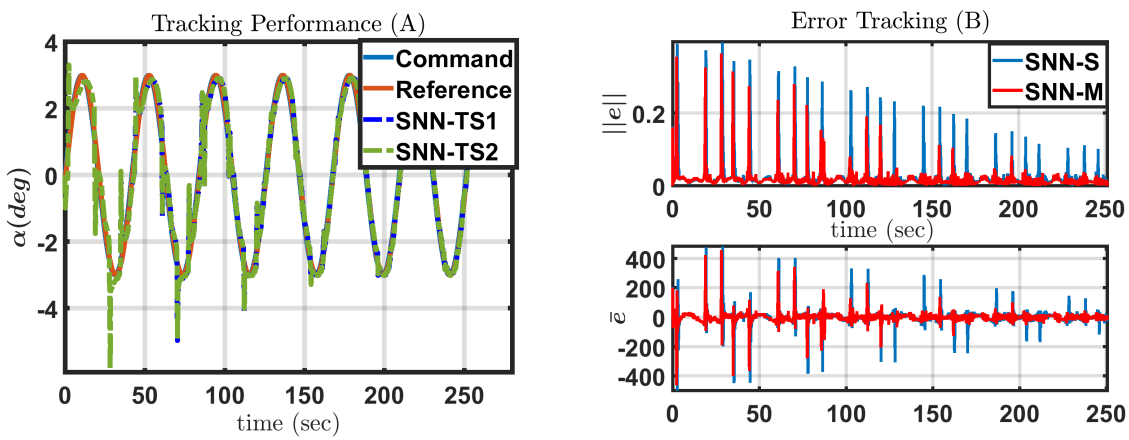

Figure 5-8. Hypersonic sparse neural network transient performance including A) tracking performance and B) error tracking.

where we refer to this controller as SNN-TS1. The second type of controller, which we refer to as SNN-TS2, uses the adaptive control laws specified in (5–24) to (5–25).

The tracking plots of Figure 5-8 show the superior transient performance of the SNN over the SHL approach due to the improved learning architecture. It also demonstrates the improved learning performance when the learning rates are increased. The controller clearly improves in performance with each repeated maneuver. In addition to the excellent tracking of the SNN controller, the SNN has the ability to retain reasonable estimates of the matched system uncertainty throughout the flight envelope. See Figure 5-9 where the uncertainty estimate for the SNN was obtained after the simulation by sweeping the neural network input ($e_I,\ \alpha,\ q$) across the flight envelope. In this case, we set $e_I = q = 0$ and varied angle of attack (α). It is worth noting that we chose $r_{0_E} = 1$, $\Delta_{RB} = 0.05$, and $U_i = 5$ for parameters of the robust control term. However for this analysis, the robust control term never becomes necessary due to the norm of the tracking error not eclipsing the robust control error threshold.

In order to properly compare the performance of the adaptive controllers with various learning rates, denoted SHL-S, SHL-M, SNN-S, and SNN-M, we created a table which quantifies the tracking performance of each controller over each pass. That is, for

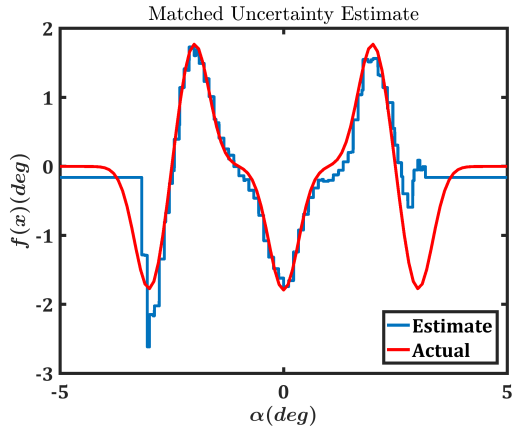

Figure 5-9. Hypersonic sparse neural network (SNN) matched uncertainty estimation.

each complete sinusoidal maneuver (i.e. pass), we calculate the norm of the error for that time interval (TE_p) using the following equation:

$$TE_p = \sum_{t=Ts_p}^{Tf_p} ||e_p(t)||_2 \quad p = 1, .., 6 \tag{5–137}$$

where Ts_p and Tf_p define the bounds of the time interval and e_p is the tracking error at time t where $e_p(t) = x(t) - x_{ref}(t)$. The results are shown in Table 5-2.

5.7 Summary

By using small learning rates and a relatively small number of neurons, we were able to control a sophisticated HSV model with flexible body effects using the SHL and SNN architectures. The SNN architecture provided superior performance in tracking and learning due to its sparse architecture. The innovative adaptive control effectiveness term was proven to nullify the effect of the control degradation on each portion of control. Through the use of a robust control term and dwell time condition, we were able

Table 5-2. Tracking performance comparison of SHL versus SNN

	TE_1	TE_2	TE_3	TE_4	TE_5	TE_6
SHL-S	145.07	141.14	141.10	141.11	147.05	141.00
SHL-M	136.03	134.68	134.68	134.65	134.65	134.71
SNN-S	114.09	91.79	83.66	78.01	70.13	61.92
SNN-M	100.79	81.29	74.69	50.48	46.09	42.38

to provide a complete Lyapunov stability analysis for the adaptive switching laws of the SNN. Future work includes implementing various forms of the sparse adaptive controller on a variety of vehicle dynamics while utilizing different activation functions and learning rates in the adaptive controller.

CHAPTER 6
DEVELOPMENT OF A ROBUST DEEP RECURRENT NEURAL NETWORK CONTROLLER FOR FLIGHT APPLICATIONS

This chapter presents a novel approach for training a deep recurrent neural network (RNN) to control a highly dynamic nonlinear flight vehicle. We analyze the performance of the deep RNN using a time and phase portrait analysis. The superior performance of the deep RNN is demonstrated again a well-tuned gain-scheduled LQR design.

6.1 Deep Learning-Based Flight Control Design

We are interested in designing a deep recurrent neural network controller to ensure the selected plant output state, $y_{sel} \in y$, successfully tracks the reference output, y_{ref}. The reference output is produced by an oracle which is designed to specify the desired closed-loop tracking performance of the system under nominal conditions [16]. In other words, the oracle will transform y_{cmd}, which is typically generated by a guidance law, into a reference command, y_{ref}. By using the oracle to produce desired ideal behavior, we can transform the control problem into a supervised learning problem. The goal is to design the deep RNN controller to achieve closed-loop tracking performance that is close to the oracle, regardless of numerous aerodynamic uncertainties and nonlinear effects while remaining robust to disturbances and noise.

For this chapter, a gain-scheduled LQR controller with servomechanism action was designed to provide oracle trajectories. For further details, the reader is referred to [16], [73], and [80].

For a system level view, we now consider the block diagram shown in Figure 6-1. The flight vehicle (plant dynamics) is controlled by a deep recurrent neural network (RNN) which takes a vector of states and commands and produces a control command, u. The control input to the plant, u_{act}, is produced by a second order actuator with input u, natural frequency (w_n), and damping factor (ζ).

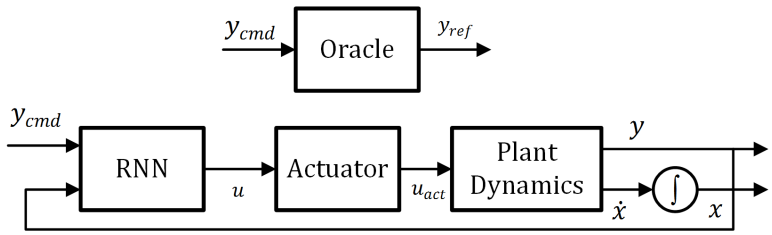

Figure 6-1. Deep learning control block diagram for RNN/GRU.

6.2 Flight Vehicle Model

This section details the plant dynamics block shown in Figure 6-1. We consider a highly nonlinear vehicle model composed of a cylindrical body with four independent servo-controlled fins $(\delta_1,\ \delta_2,\ \delta_3,\ \delta_4)$ oriented in an X-configuration [73]. The flight model was obtained from a traditional aerodynamic design tool by sweeping over a large range of flight conditions. Aerodynamic coefficients were computed from simulation data and indexed by angle of attack (α), angle of side-slip (β), Mach, and altitude (h). For convenience, we created virtual fins (aileron δ_a, elevator δ_e, and rudder δ_r) used for control design in the conventional autopilot reference frame. We created a static mapping from the virtual fins to actual fin displacement in order to calculate forces and moments of the flight vehicle (see [73]).

In this study, we consider only longitudinal dynamics of the flight vehicle, which are assumed to be decoupled from the lateral dynamics. The longitudinal 3-DoF equations of motion for a flight vehicle takes the following form (see [6, 74]):

$$\dot{V}_T = \frac{1}{m}(T\ \cos(\alpha) - D) - g\ \sin(\theta - \alpha) \tag{6–1}$$

$$\dot{\alpha} = \frac{1}{mV_T}(-T\ \sin(\alpha) - L) + q + \frac{g}{V_T}\cos(\theta - \alpha) \tag{6–2}$$

$$\dot{\Theta} = q \tag{6–3}$$

$$\dot{q} = \frac{M}{I_{YY}} \tag{6–4}$$

$$\dot{h} = V_T\ \sin(\theta - \alpha) \tag{6–5}$$

where m is the mass of the vehicle, I_{YY} is the moment of inertia, and T, D, L, and M are forces and moments. The state vector, $x \in \mathbb{R}^5$, is given by

$$x = [V_T,\ \alpha,\ \Theta,\ q,\ h] \tag{6–6}$$

where V_T is the true airspeed, α is angle of attack, Θ is the pitch angle, q is pitch rate, and h is the altitude of the flight vehicle. We define the output vector as

$$y = [\alpha,\ q,\ A_z,\ \bar{q}] \tag{6–7}$$

where $\bar{q}$ is dynamic pressure and A_z is vertical acceleration.

We assume the following mapping from axial and normal (A, N) forces to lift and drag forces (L, D) used in (6–1) to (6–5):

$$L = N\ \cos(\alpha) - A\ \sin(\alpha) \tag{6–8}$$

$$D = N\ \sin(\alpha) + A\ \cos(\alpha) \tag{6–9}$$

where the body forces (A, N) and pitching moment (M) can be approximated by (see [73, 74])

$$A \approx \frac{1}{2}\rho V_T^2 S C_A \tag{6–10}$$

$$N \approx \frac{1}{2}\rho V_T^2 S C_N \tag{6–11}$$

$$M \approx \frac{1}{2}\rho V_T^2 S c_{ref} C_m \tag{6–12}$$

where ρ denotes air density, S is the reference area, c_{ref} is the mean aerodynamic chord, and we assume zero thrust (i.e. $T = 0$).

The axial force coefficient (C_A), normal force coefficient (C_N), and pitch moment coefficient (C_m) appearing in (6–10) to (6–12) take the following form:

$$\begin{aligned} C_A = {} & C_{A_{ALT}}(h,\ M) + C_{A_{AB}}(\alpha,\ M) \\ & + \sum_{i=1}^{4} C_{A_{\delta_i}}(\alpha,\ M,\ \delta_i) \end{aligned} \tag{6–13}$$

$$C_N = C_{N_0}(\alpha,\, M) + \sum_{i=1}^{4} C_{N_{\delta_i}}(\alpha,\, M,\, \delta_i) \tag{6–14}$$

$$\begin{aligned} C_m &= C_{m_0}(\alpha,\, M) + \sum_{i=1}^{4} C_{m_{\delta_i}}(\alpha,\, M,\, \delta_i) \\ &\quad + C_{m_q}(\alpha,\, M,\, q) + q\rho_q + \alpha\rho_\alpha \end{aligned} \tag{6–15}$$

where each aerodynamic coefficient component can be computed using a look-up table based on the flight condition $(\alpha,\, M,\, h,\, q)$ and control inputs $(\delta_1,\, \delta_2,\, \delta_3,\, \delta_4)$. We introduce constant uncertainty parameters (ρ_q, ρ_α) in (6–15) that will be utilized during the control design process and elaborated on in Section 6.3.1.

In order to reduce the computational costs of evaluating each aerodynamic look-up table during run-time, polynomial approximations were produced. The terms selected for approximation include atmospheric variables (e.g. speed of sound and air density), inverse airspeed ($\frac{1}{V_T}$), and each aerodynamic coefficient component. For each term, the order of the polynomial fit was adjusted by evaluating accuracy and smoothness of the approximation. In addition, the trigonometric functions (e.g. $sin(\theta - \alpha)$) were approximated using Taylor series expansion. In order to increase the accuracy of the model while reducing the order of the polynomials, we created several polynomial models throughout the flight envelope. The reader is referred to [91] for more details regarding the use of polynomial approximations of aircraft dynamics. An example component of an aerodynamic coefficient that was fitted with a 3rd order polynomial is shown in Figure 6-2. The mesh represents the value of the coefficients after evaluating the polynomial while the circles denote the value of the coefficient evaluated at the breakpoints of the look-up tables.

The complete polynomial model for the longitudinal dynamics of the flight vehicle is given by

$$\begin{aligned} \dot{x} &= f(x,\, (\lambda_u(u_{act} + \rho_u)),\, \rho_\alpha,\, \rho_q) \\ y &= f(x,\, (\lambda_u(u_{act} + \rho_u)),\, \rho_\alpha,\, \rho_q) \end{aligned} \tag{6–16}$$

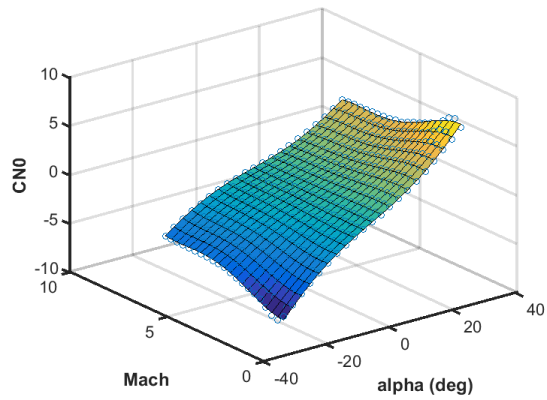

Figure 6-2. Example Coefficient Polynomial Fit

where elevator deflection (δ_e) is the control input (u_{act}), the state vector (x) is composed of the five states listed in (6–6), λ_u is a multiplicative control effectiveness term, ρ_u is a constant additive control uncertainty, and $(\rho_\alpha,\ \rho_q)$ represent additive uncertainty parameters that directly affect aerodynamic coefficients. The constant uncertainty parameters ($\rho_\alpha,\ \rho_q,\ \rho_u,\ \lambda_u$) are included as inputs to the flight vehicle model for control design purposes and will be elaborated on in Section 6.3.

For clarity of the control design process, we transform the general form of the flight vehicle model, shown in (6–16), into discrete time form [42]

$$\begin{aligned} x_{t+1} &= f(x_t, (\lambda_u(u_t^{act} + \rho_u)) + d_u, \rho_\alpha, \rho_q) + \zeta_p \\ y_t &= f(x_t, (\lambda_u(u_t^{act} + \rho_u)) + d_u, \rho_\alpha, \rho_q) + \zeta_p \end{aligned} \tag{6–17}$$

where x_t are the states of the plant at time t, u_t^{act} is the actuator output at time t, and y_t is the output vector at time t. Notice we also include ζ_p and d_u as plant noise and input disturbance terms, respectively.

6.3 Deep Recurrent Neural Network Controller

For this chapter, the RNN controller block shown in Figure 6-1 will utilize a stacked recurrent neural network (S-RNN) architecture [67]. A diagram of the S-RNN architecture is shown in Figure 6-3 where u_t denotes the control command at time t and H_t^i

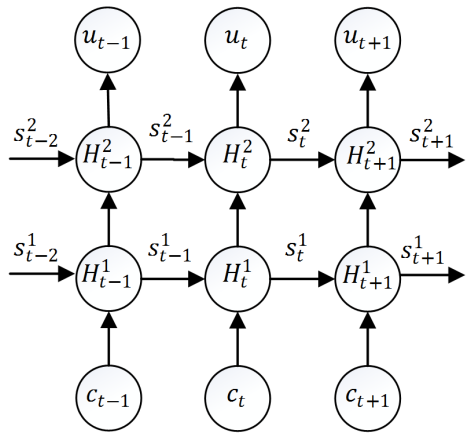

Figure 6-3. Two layer stacked recurrent neural network (S-RNN).

signifies a hidden node in the ith layer at time t. Each hidden node (H^i) is a function or set of functions that produce a hidden state, s_t. For instance, when using the traditional form of a recurrent neural network, the hidden state $s_t = f(Uc_t + Ws_{t-1})$ is calculated at each hidden node using the current input (c_t), the previous hidden state (s_{t-1}), and the RNN controller parameters $\Theta = [U, W]$. These hidden states are then passed to adjacent nodes as indicated in Figure 6-3.

For this chapter, the input to the RNN controller at time t, c_t, takes the following form:

$$c_t = [e_I, \alpha, q, \bar{q}] \tag{6–18}$$

where q is the pitch rate, $\bar{q}$ is the dynamic pressure of the vehicle, and e_I is the integral error of tracking ($e_I = \int_0^{t_f} y_{sel} - y_{cmd}\, dt$).

As mentioned in the introduction, the traditional RNN structure has difficulty in learning long-term dependencies due to the vanishing gradient problem (see [64, 92]). In order to combat this problem and encourage long-term learning, we will utilize gated recurrent (GRU) modules at the hidden nodes [66]. Algorithm 6.1 states the equations by which the GRU modules operate, where $\Theta_i = [U^u, W^u, U^r, W^r, U^h, W^h, b_1, b_2, b_3]$ is a matrix of parameters for each GRU module indexed by i where (z, r, h) are internal states of the module [63].

Algorithm 6.1 Gated Recurrent Units (GRU)

1: $z = \sigma(c_t U^u + s_{t-1} W^u + b_1)$
2: $r = \sigma(c_t U^r + s_{t-1} W^r + b_2)$
3: $h = tanh(c_t U^h + (s_{t-1} * r) W^h + b_3)$
4: $s_t = (1 - z) * h + z * s_{t-1}$
5: $*$ represents element-wise multiplication

The output nodes in Figure 6-3 of the RNN produce control commands (u_t) issued to the actuators at time t. The controller output is calculated by

$$u_t = s_t V + b_o \tag{6–19}$$

where s_t is acquired from the output of the last hidden layer's GRU module and $(V,\, b_o)$ are parameters of the controller. We can now define $\Theta = [\Theta_i,\, \Theta_{i+1}, ...,\, \Theta_M,\, V\, , b_o]$ to contain all the parameters of the controller where M denotes the total number of hidden layers.

6.3.1 Controller Optimization Procedure

The goal of the control design is to find a set of parameters, Θ, of a RNN/GRU controller that minimizes a predefined cost function in the following form [39]:

$$J_\Theta = \sum_{t=0}^{t_f} \gamma^t E[\chi(x_t, u_t)|u_\Theta)] \tag{6–20}$$

where χ is the immediate cost, u_Θ is the policy (controller) parameterized by Θ, Θ^* is the ideal set of parameters, t_f is the time duration, and γ is a constant discount term.

We define an estimate of the overall cost function with a diverse range of initial conditions, disturbances, uncertainties, and commands as

$$\tilde{J}(\Theta) = \frac{1}{N} \sum_{i=1}^{N} J_i(\Theta) \tag{6–21}$$

where N is the total number of sample trajectories. For each trajectory, we introduce uncertainty in the system through a set of constant parameters $(\rho^i_{\alpha,}\, \rho^i_q,\, \rho^i_u,\, \lambda^i_u)$ where i refers to the ith trajectory. The non-zero constant parameters $(\rho_\alpha,\, \rho_q)$ create additive

aerodynamic uncertainties shown explicitly in (6–15). The constant parameter λ_u acts as a control effectiveness term, and parameter ρ_u is an additive matched uncertainty, shown in (6–17).

In order to create the sample trajectories, we define $(R_\alpha,\, R_q,\, R_u,\, \Lambda_u) \sim U[min, max]$ to be a vector of uniformly distributed random variables with fixed min/max values that are determined a priori based on system requirements. We then draw N samples from each random variable and assign each sample to a particular trajectory. More precisely, $\rho_\alpha = \{\rho_\alpha^1, \rho_\alpha^2, ..., \rho_\alpha^N\}$, $\rho_q = \{\rho_q^1, \rho_q^2, ..., \rho_q^N\}$, $\rho_u = \{\rho_u^1, \rho_u^2, ..., \rho_u^N\}$, and $\lambda_u = \{\lambda_u^1, \lambda_u^2, ...\lambda_u^N\}$ are the sets of samples drawn from the uniform random variables $R_\alpha,\, R_q,\, R_u$, and Λ_u, respectively.

Since the goal is to satisfy robustness and performance constraints simultaneously, we create two types of sample trajectories with distinct labels. Each trajectory's label (performance (P) or robustness (R)) determines which elements (e.g. noise, uncertainties, and disturbances) are active during optimization and the form of the cost function for which we are minimizing. For each performance trajectory (P), the RNN/GRU controller parameters will be optimized in order to track the oracle's nominal trajectory from a variety of initial conditions with penalties on high control rates and tracking error while including small to moderate aerodynamic uncertainties in the nonlinear plant. For each robust trajectory (R), the RNN/GRU controller will use a variant of time-varying funnels to track the oracle's nominal trajectory within predefined bounds while including large aerodynamic uncertainties, noise, and disturbances in the plant dynamics during optimization. We define a set of time-varying funnels by

$$\beta_{\dot{u}}(t) = \{x \in \mathbb{R}^n | U(x,t) \leq b_{\dot{u}}(t)\} \tag{6–22}$$

$$\beta_e(t) = \{x \in \mathbb{R}^n | E(x,t) \leq b_e(t)\} \tag{6–23}$$

where $(b_{\dot{u}}(t),\, b_e(t))$ define the boundaries of the funnels at time t. In this chapter, we set $E(x,t) = |e_t|$ and $U(x,t) = |\dot{u}_t|$ where $|\cdot|$ denotes absolute value, the optimization

tracking error is defined by $e_t = y_{sel} - y_{ref}$, and the estimated control rate is calculated by $\dot{u}_t = \frac{u_t - u_{t-1}}{\Delta t}$. We also choose to select the time-varying funnel bounds $(b_{\dot{u}}(t),\, b_e(t))$ as constants (i.e. $b_{\dot{u}}(t) = b_{\dot{u}}$ and $b_e(t) = b_e$).

We now define the cost function for each trajectory, i, in the following form:

$$J_i(\Theta) = \sum_{t=0}^{t_f} \gamma^t \chi(x_t, u_t) \tag{6–24}$$

where the form of the instantaneous performance measurement calculation, $\chi(x_t, u_t)$, is determined based on each trajectory's label. That is, the instantaneous performance measurement is given by

$$\chi(x_t, u_t) = \begin{cases} k_1 {e_t}^2 + k_2 f_{\dot{u}}^2 & if\ label = P \\ k_3 f_e^2 + k_4 f_{\dot{u}}^2 & if\ label = R \end{cases} \tag{6–25}$$

where $k_1,\, k_2,\, k_3,$ and k_4 are positive constants. The funnel tracking error (f_e) and funnel control rate error ($f_{\dot{u}}$) at time t are defined by

$$f_e = \max(|e_t| - b_e, 0) \tag{6–26}$$

$$f_{\dot{\mu}} = \max(|\dot{u}_t| - b_{\dot{u}}, 0) \tag{6–27}$$

where $(b_{\dot{u}},\, b_e)$ are the constant bounds of the funnels.

As an additional goal of the control design, we seek to establish nonlinear stability margins through the use of the control effectiveness term, λ_u. We define the stability margin criteria to define the amount of amplification/attenuation that a controller can handle before the system becomes unstable. That is, a nonlinear system with a asymptotic stabilizing control law, $u = u_{BL}$, possesses stability margins (SM_{min}, SM_{max}) where $-1 \leq SM_{min} \leq SM_{max} \leq \infty$, if for every $SM \in [SM_{min}, SM_{max}]$ the control input $u = (1 + SM)u_{BL}$ also asymptotically stabilizes the system [93]. For sample-based optimization, we will relax the asymptotic tracking condition. Instead, we will consider a system to have acceptable stability margins, $SM \in [SM_{min}, SM_{max}]$, if each

Algorithm 6.2 Incremental Training Procedure

1: Randomly initialize controller parameters (Θ)
2: STEP 1: Optimize Θ for RNN/GRU using cost function (6–21) with linear dynamics and labels=P
3: STEP 2: Re-optimize Θ using nonlinear dynamics, labels=P, with small uncertainties in aerodynamics
4: STEP 3: Re-optimize Θ using nonlinear dynamics, labels=(R,P), with uncertainties, disturbances, and noise

trajectory generated during training remains between pre-defined funnel bounds, for all $t \in [0, t_f]$. That is, the control effectiveness term, λ_u, was sampled in the range of $(1 + SM_{min}) \leq \lambda_u \leq (1 + SM_{max})$ in order to establish stability margins (SM_{min}, SM_{max}).

6.3.2 Incremental Training Procedure

Since the optimization problem that we are solving is non-convex, the solution is only guaranteed to converge to a local minimum. For this reason, verification is necessary to guarantee the system adheres to all the previously specified constraints.

In order to assist policy learning and avoid parameter convergence to poor local minima, we developed an incremental training procedure detailed in Algorithm 6.2. This procedure was inspired by layer-wise training methods in deep learning [55] and resulted in the best overall results using the RNN/GRU controller architecture. Note, before executing STEP 1, the parameters (Θ) were initialized randomly based on methods discussed in [55]. Analysis for the optimized controllers is provided in Section 6.4.

6.4 Flight Control Simulation Results

Our goal for this section is to describe the training details of the RNN/GRU controller and provide a comparison of performance and robustness of the RNN/GRU controller to that of a typical gain-scheduled (GS) baseline controller.

6.4.1 RNN/GRU Controller Optimization

The RNN/GRU controller parameter optimization was performed using 2,970 sample trajectories. For best results, the sample trajectories were divided into 540

Table 6-1. Range of initial conditions and uncertainty variables

	MIN	MAX
α_0[deg]	-25	25
q_0[deg/sec]	-75	75
Mach_0	1.0	2.0
alt_0[km]	7	14

performance trajectories and 2,430 robust trajectories. Depending on the label, each trajectory was subjected to different disturbances, uncertainties, and noise while optimizing gains of the controller according to a pre-defined cost function described in Section 6.3.1. The initial conditions for each trajectory were selected to lie in the range displayed in Table 6-1 where α_0 is the initial angle of attack, q_0 is the initial pitch rate, Mach_0 is the initial Mach number, and alt_0 is the initial altitude. Each trajectory used in training was $t_f = 2.5$ sec in duration where the plant dynamics were discretized with $\Delta t = 0.01$ time steps using an AB-2 integration scheme. The controller optimization was performed using angle of attack step command tracking (i.e. $y_{sel} = \alpha,\ y_{cmd} = \alpha_{cmd}$) where we set the bounds of funnels to $b_e = 0.05$ and $b_{\dot{u}} = 1.74$.

Numerous control architectures were explored using the plant dynamics described in the previous sections. We found that 2 hidden layer network with 25 hidden nodes was sufficient when using the RNN/GRU architecture. Using three hidden layers slightly increased performance, but required significantly more optimization time and additional sample trajectories to meet requirements. It is interesting to note that several different architectures (e.g. time-delayed feed-forward network, RNN) succeeded in certain aspects of optimization but clearly lacked the sophistication necessary to complete this task in its entirety. In fact, training the RNN/GRU controller architecture with the

Table 6-2. Cumulative error (CTE), control rate (CCR), and final cost

	CTE	CCR	Cost
3-Layer RNN/GRU	339.15	100.16	1.0438
2-Layer RNN	359.28	313.64	2.4970
GS	1000.21	1180.04	-

sequential initialization scheme presented in Section 6.3.2 produced the best results. This experiment used L-BFGS optimization method for training the deep recurrent network. L-BFGS is a quasi-newton second order batch method which has produced state-of-the-art results on both regression and classification tasks [55]. Table 6-2 shows the value of the final estimated cost function after completing parameter optimization for several controllers.

As mentioned previously, the minimum and maximum for each uniformly distributed variable $(R_\alpha,\, R_q,\, R_u,\, \Lambda_u)$ used to produce sample trajectories for controller optimization was pre-defined and shown in Table 6-3. The bounds of the control effectiveness variable (Λ_u) were set to establish stability margins of $(-0.75,\, 2.0)$.

6.4.2 Analysis and Results

In order to compare the GS and RNN/GRU controllers, we are interested in a determining a nonlinear short period model of the longitudinal dynamics for the flight vehicle described previously. In order to accomplish this task, we decoupled the short period and phugoid modes of the vehicle by substituting trim values for $V_T,\, h$ and Θ into (6–16). The augmented polynomial short period model is given by

$$\begin{aligned} \dot{e}_I &= \alpha - \alpha_{cmd} \\ \dot{\alpha} &= f(\alpha,\, q,\, \delta_e,\, \rho_\alpha,\, \rho_q,\, \rho_u,\, \lambda_u) \\ \dot{q} &= f(\alpha,\, q,\, \delta_e,\, \rho_\alpha,\, \rho_q,\, \rho_u,\, \lambda_u) \end{aligned} \tag{6–28}$$

where the state vector includes the integral error of tracking, angle of attack, and pitch rate (i.e. $x = [e_I,\, \alpha,\, q]$). The input to the model is elevator deflection ($u_{act} = \delta_e$) produced

Table 6-3. Range of initial conditions and uncertainty variables

	MIN	MAX
R_α	-0.5	0.1
R_q	-7	5
R_u	-5	5
Λ_u	0.25	3.0

by the actuator, and the constant uncertainty parameters ($\rho_\alpha,\ \rho_q,\ \rho_u,\ \lambda_u$) are included to assess robustness. For the sake of brevity, we selected a region surrounding a specific trim point in the flight envelope to analyze (i.e. $\mathrm{Mach} = 1.7$, $\Theta = 0$, and $h = 14km$). For this study, we set the commanded angle of attack (α_{cmd}) to 0.

Next, we obtained various analysis models by selecting different values for the constant uncertainty parameters (i.e. $\rho_\alpha,\ \rho_q,\ \rho_u,\ \lambda_u$) that lie in the range shown in Table 6-3 and substituted them directly into (6–28). This analysis was performed by simulating 32 trajectories for each analysis model from various initial conditions. The initial conditions were chosen to span a range that slightly exceeded the range used for training (i.e. $\alpha_0 = [-30,\ 30]$ degrees and $q_0 = [-100,\ 100]$ degrees per second). In order to visualize a large number of system trajectories for each analysis model, we employ the use of phase portraits and tracking performance plots shown in Figure 6-4 through 6-7. The analysis models used to generate the plots were chosen to demonstrate the GRU/RNN improvement over the GS controller in extreme operating conditions.

In order to quantify the tracking performance of each controller, we calculate the average tracking error ($ATE = (\sum_{i=1}^{N}\sum_{t=1}^{t_f}|e_t|)/N$) and average control rate ($ACR = (\sum_{i=1}^{N}\sum_{t=1}^{t_f}|\dot{u}_t|)/N$) for each analysis model used to generate Figures 6-4 through 6-7. We define the cumulative tracking error (CTE) as the summation of the ATEs and cumulative control rate (CCR) as the summation of the ACRs (Table 6-2).

Traditionally, when training a deep neural network for control tasks, the goal is simply to drive the selected state, y_{sel}, to the commanded state, y_{cmd}. We found by using performance trajectories (P) and robustness trajectories (R) during training, the deep RNN controller performed significantly better than trying to solve the typical performance problem (P) for each trajectory regardless of the circumstances. For instance, the optimization sequence was not able to converge to an acceptable solution when using a traditional form of the cost function while including noise and wind gusts in the sample

trajectories. However, controller parameter convergence significantly improved by using the optimization procedure discussed in Section 6.3.1.

One drawback to deep learning is the execution time. We implemented the deep learning program in MATLAB using a 3.1 GHz PC with 256 GB RAM and 10 cores. It took approximately 1.45×10^5 seconds or 40 hours on average for completing Algorithm 6.2 in its entirety. Fortunately, after the first step, each successive step was significantly shorter in execution time. This drawback results in fewer test cases due to the large computation requirement. However, using a platform that is specifically designed for deep learning and utilizing the GPU is likely to decrease the average computation time.

6.5 Summary

We have presented a novel approach for training deep recurrent neural networks for flight control applications. We satisfied robustness and performance metrics concurrently by training the RNN/GRU controller using sample trajectories that contain disturbances, additive aerodynamic uncertainties, and control degradation in the plant dynamics. We demonstrated the performance of the trained RNN/GRU controller on a highly non-linear agile flight vehicle against a traditional gain-scheduled LQR design.

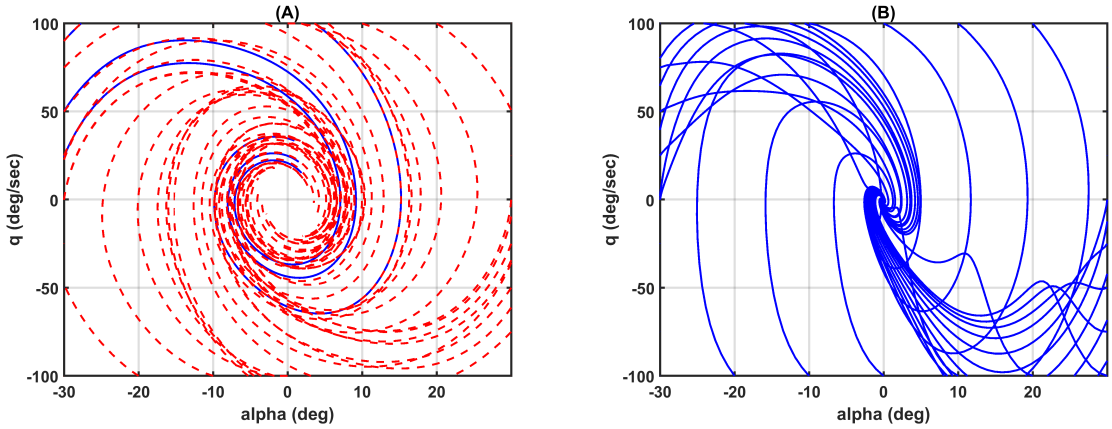

Figure 6-4. Phase portrait analysis for A) GS and B) RNN/GRU with uncertainty values $\lambda_u = 0.25$, $\rho_u = 0$, $\rho_\alpha = 0$, and $\rho_q = 0$.

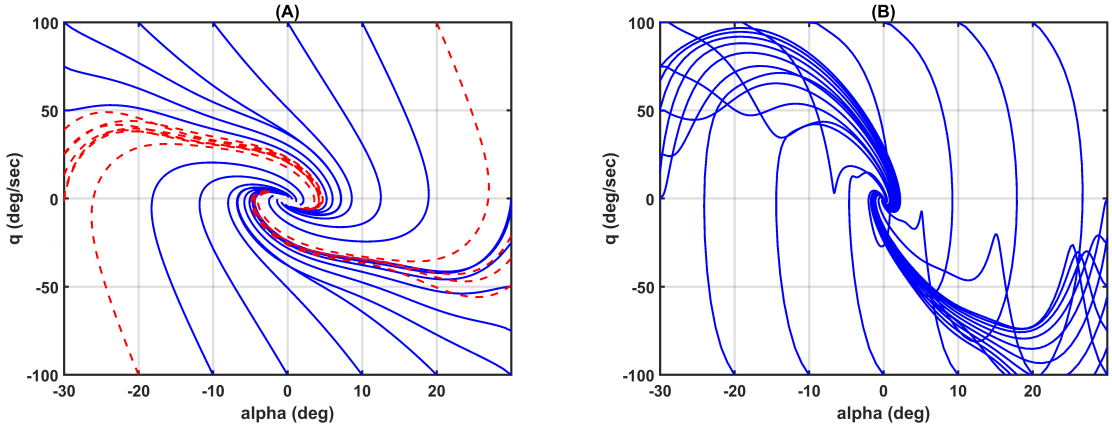

Figure 6-5. Phase portrait analysis for A) GS and B) RNN/GRU with uncertainty values $\lambda_u = 0.75$, $\rho_u = 0$, $\rho_\alpha = 0.025$, and $\rho_q = 5$.

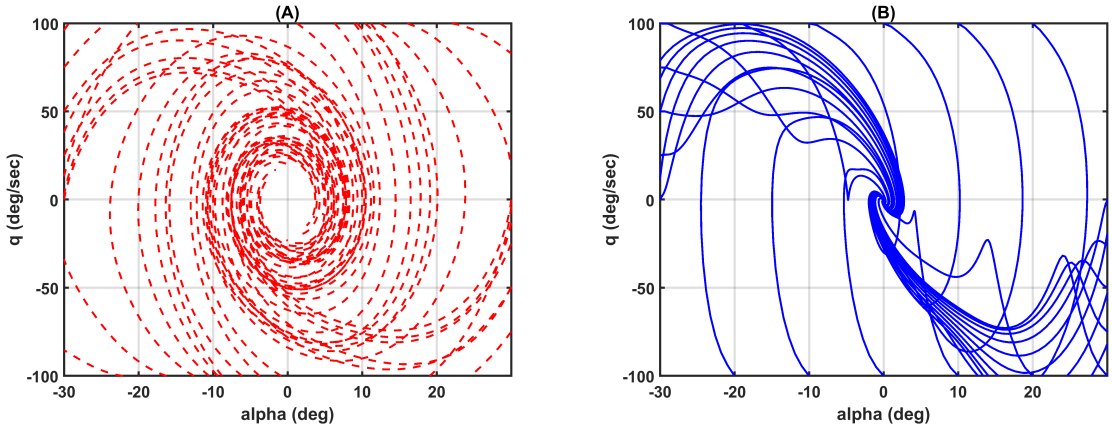

Figure 6-6. Phase portrait analysis for A) GS and B) RNN/GRU with uncertainty values $\lambda_u = 0.5$, $\rho_u = 0$, $\rho_\alpha = 0.05$, and $\rho_q = 2.5$.

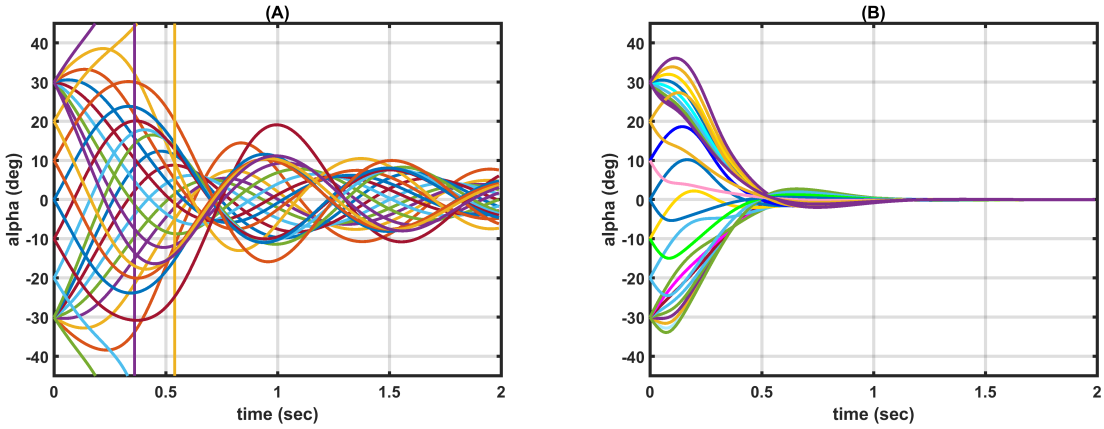

Figure 6-7. Traditional step responses for A) GS and B) RNN/GRU with uncertainty values $\lambda_u = 0.5$, $\rho_u = 0$, $\rho_\alpha = 0.05$, and $\rho_q = 2.5$.

CHAPTER 7
DEVELOPMENT OF A DEEP AND SPARSE RECURRENT NEURAL NETWORK HYPERSONIC FLIGHT CONTROLLER WITH STABILITY MARGIN ANALYSIS

In this chapter, we extend the sparse neural network (SNN) concept for adaptive control to a deep recurrent neural network. The sparse deep learning controller (S-DLC) is trained using a variant of the optimization procedure described in the previous chapter. In this chapter, we aim to train the S-DLC to establish stability and time delay margins. The stability margins are analyzed around an equilibrium point using region of attraction (ROA) estimation via forward reachable sets. Simulation results demonstrate the effectiveness of the control approach.

7.1 Deep Learning Controller

The deep learning controller described in the subsequent section is trained to ensure the selected plant output state, $y_{sel} \in y$, successfully tracks the reference output, y_{ref}. The reference output is produced by an oracle which is designed to specify the closed-loop tracking performance of the system under ideal conditions, i.e. no uncertainties or disturbances. Inspired from model reference adaptive control methods, we use a series of linear quadratic regulator (LQR) controllers to provide oracle trajectories [16].

The block diagram of the closed-loop system is shown in Figure 7-1. The diagram shows that the flight vehicle dynamics are controlled by a deep learning controller (DLC) which passes a control command, u, to the actuator. The actuator produces control deflections denoted u_{act} which acts on the flight vehicle dynamics.

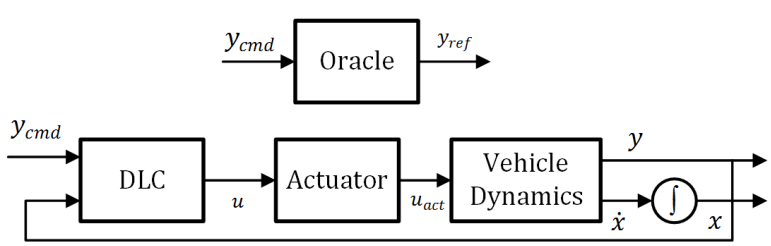

Figure 7-1. Deep learning control closed-loop block diagram.

7.1.1 Controller Architecture

Similar to our previous work, we will use a stacked recurrent neural network (S-RNN) for the architecture of the DLC (see [67, 94]). In order to combat the well-known vanishing gradient problem and encourage long-term learning, we will employ gated recurrent (GRU) modules at the hidden nodes [66].

For the reader's convenience, an example of a two-layer S-RNN architecture is shown in Figure 7-2 where u_t denotes the control command at time t and H_t^i signifies a hidden node in the ith layer at time t. Each hidden layer, H^i, is a set of functions that output a hidden state, s_t, at each discrete time t. The input of each hidden node, denoted i_t in the equations, is received from the previous layer. For instance, the inputs of H_t^1 and H_t^2 of Figure 7-2 are c_t and s_t^1, respectively.

As mentioned previously, each hidden node (i.e. GRU module) acts according to the following equations:

$$z_{GRU} = \sigma(i_t U^u + s_{t-1} W^u + b_1) \tag{7–1}$$

$$r_{GRU} = \sigma(i_t U^r + s_{t-1} W^r + b_2) \tag{7–2}$$

$$h_{GRU} = \tanh(i_t U^h + (s_{t-1} * r_{GRU}) W^h + b_3) \tag{7–3}$$

$$s_t = (1 - z_{GRU}) * h_{GRU} + z_{GRU} * s_{t-1} \tag{7–4}$$

where $(z_{GRU}, r_{GRU}, h_{GRU})$ are internal states of the module and $*$ denotes element-wise multiplication. Notice that the hidden states are sent to the next layer for use as inputs, i_t, as well as feedback to itself at the next time step, s_{t-1}. We define $\Theta_i = [U^u, W^u, U^r, W^r, U^h, W^h, b_1, b_2, b_3]$ as the matrix of weights for each GRU module indexed by i where $L \in \mathbb{N}$ denotes the total number of hidden layers [63].

The controller output is calculated from

$$u_t = s_t^L V + b_o \tag{7–5}$$

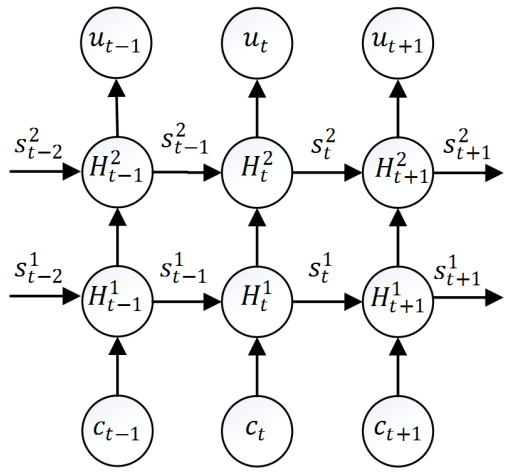

Figure 7-2. Stacked deep learning controller (DLC) architecture with two hidden layers

where $(V,\, b_o)$ are weights of the controller and s_t^L is acquired from the output of the last hidden layer's GRU module. Lastly, we define $\Theta = [\Theta_i,\, \Theta_{i+1}, ...,\, \Theta_L,\, V\,,\, b_o]$ as the matrix that contains all the weights of the controller.

7.1.2 Extension to Sparse Neural Network

The sparse neural network (SNN) was originally created as a switched control system approach in model reference adaptive control (MRAC). It was inspired by recent results in distributed sparse learning methods in the deep learning community. The approach aims to utilize controller memory while reducing the computational load on the processor in order to improve the tracking performance of flight vehicles with persistent and significant uncertainties. The reader is referred to Chapter 4 for more details and description of the SNN [80].

The basic idea behind the SNN is to segment the flight envelope into regions and activate only a small percentage of neurons while operating in that region. The remaining unused neuron weights are frozen until activated by the controller. We consider a N_D dimensional flight envelope where the dimension of the flight envelope is determined based on flight conditions which impact the performance of the controller (e.g. Mach and angle of attack). We denote the total number of segments in the flight envelope as $T \in \mathbb{N}$ where $S = \{s_1,\, ...,\, s_T\}$ and $P = \{p_1,\, ...,\, p_T\}$ are the set of segments

Algorithm 7.1 Sparse Neural Network Execution

1: receive x_t and the corresponding location in the operating envelope (x_{op})
2: recall the previous segment index (j) and the nearest neighbor graph
3: determine the current operating segment (i.e. $argmin_{\forall i \in I}\ dist(x_{op},\ p_i)$)
4: retrieve the indices for the set of active nodes, E_{A_i}, corresponding to i
5: use E_{A_i} to select the appropriate neural weights, Θ, for use in control

and center points, respectively. For future use, we let $I = \{1,\ ...,\ T\}$ be the index set of the sets S and P.

Let $N \in \mathbb{N}$ be the number of hidden layer nodes for each layer where we allocate $Q = \frac{N}{T}$ nodes to each segment. The spacing of the nodes within each segment is determined by the user. In this research, we assume the spacing of nodes is the same for each layer. Hence, the active node indices for each layer are the same. We can now establish the set of indices of the hidden layer nodes for each segment, s_i, denoted by $E_{i \in I} = \{Q(i-1)+1,\ ...,\ iQ\}$. In addition, we define a particular position in the N_D space for each hidden node where we let $D_{i \in I} = \{d_1,\ ..., d_N\}$ be the set of Euclidean distances from the ith center point to each hidden node location in the flight space. This calculation is performed a priori based on the location of the center points provided by the user. Finally, we can define the set of indices for the active nodes for each segment s_i as E_{A_i} where $E_i \subseteq E_{A_i}$. We define the number of active nodes as $N_{act} \in \mathbb{N}$ where $N_{act} \geq Q$. For each index of a center point, i, we can determine E_{A_i} by finding the closest N_{act} hidden nodes to the center point p_i. This can be easily accomplished before run-time using the set of Euclidean distances stored in $D_{i \in I}$. In terms of implementation, the DLC operates according to the process described in Algorithm 7.1 where $dist(\cdot)$ calculates the Euclidean distance between two N_D vectors.

For this chapter, we chose a $N_D = 2$ dimensional SNN flight space to train the DLC. We assume the spacing of nodes is the same for each layer. Hence, the active node indices for each layer are equivalent. The Voronoi diagram of the flight space is shown in Figure 7-3 and the simulation results can be seen in subsequent sections.

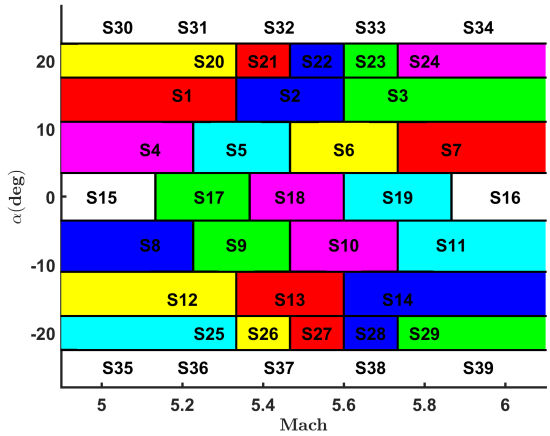

Figure 7-3. Two dimensional SNN segmented flight space for deep learning control.

Each enclosed colored region of the Voronoi diagram represents the domain of a single segment (s_i) where the predetermined center points are labeled.

7.1.3 Optimization Procedure

The goal of the optimization procedure is to find a set of weights, Θ, of a L layer DLC in order to minimize a cost function. For our research, the cost function is given by

$$J_\Theta = \sum_{t=0}^{t_f} \gamma^t E[\chi(x_t, u_t)|u_\Theta)] \tag{7–6}$$

where $E[\cdot]$ denotes expected value, t_f is the time duration, γ is a constant discount term, u_Θ represents the controller parameterized by Θ, and χ is the immediate cost.

Using concepts established in Chapter 6, we can estimate the overall cost function using a sample trajectory-based approach [39]. The estimated cost function takes the following form:

$$\tilde{J}(\Theta) = \frac{1}{N_T} \sum_{i=1}^{N_T} J_i(\Theta) \tag{7–7}$$

where N_T denotes the total number of sample trajectories used during training and $J_i(\Theta)$ represents the cost associated with the ith trajectory. The sample trajectories are created using a diverse set of initial conditions, uncertain parameters, and time-varying disturbances. In this research, we define the set of uncertain parameters for the ith

trajectory as $P_c^i = (\lambda_u^i, \rho_u^i, \tau_d^i)$ where λ_u^i, ρ_u^i, and τ_d^i represent control effectiveness, additive uncertainty, and time delay, respectively. Similarly, we define the set of time varying disturbances for the ith trajectory as $P_v = (d_u^i, \zeta_p^i)$ where d_u^i and ζ_p^i denote plant noise and input disturbance terms, respectively. We assume for each sample trajectory, the set of uncertain parameters is held constant. The disturbances stemmed from a variety of sources (e.g. wind, vibration) and were activated randomly for each sample trajectory. See Section 7.2 for a complete description of the dynamics of the hypersonic model.

To begin the optimization procedure, we define the sets of uncertainty parameters for all sample trajectories as $\Lambda_u = \{\lambda_u^1, \lambda_u^2, ..., \lambda_u^{N_T}\}$, $P_u = \{\rho_u^1, \rho_u^2, ..., \rho_u^{N_T}\}$, and $T_d = \{\tau_d^1, \tau_d^2, ..., \tau_d^{N_T}\}$. The sets of uncertainty parameters are obtained by sampling a set of uniformly distributed random variables. The range of each random variable used in training will be described in Section 7.3.

We utilize the robust training process for recurrent neural networks (RNNs) with GRU modules established in Chapter 4, which is summarized here for convenience [94].

The cost function for the ith trajectory is given by

$$J_i(\Theta) = \sum_{t=0}^{t_f} \gamma^t \chi(x_t, u_t) \tag{7–8}$$

where $\gamma \in [1, 2]$ is used emphasize the importance of convergence near the final time, t_f. We now introduce the form the instantaneous cost function

$$\chi(x_t, u_t) = \begin{cases} k_1 {e_t}^2 + k_2 f_{\dot{u}}^2 & if\ label = P \\ k_3 f_e^2 + k_4 f_{\dot{u}}^2 & if\ label = R \end{cases} \tag{7–9}$$

which is determined based on the ith trajectory's label where k_1, k_2, k_3 and k_4 are user-defined positive constants. The funnel tracking error (f_e) and funnel control rate error

($f_{\dot{u}}$) at time t are defined by

$$f_e = max(|e_t| - b_e, 0) \tag{7–10}$$

$$f_{\dot{\mu}} = max(|\dot{u}_t| - b_{\dot{u}}, 0) \tag{7–11}$$

where $b_{\dot{u}}$ and b_e are the constant bounds of the funnels. We define instantaneous tracking error as $e_t = y_{sel} - y_{ref}$ and the estimated control rate as $\dot{u}_t = \frac{u_t - u_{t-1}}{\Delta t}$ where Δt denotes the time step duration. The use of trajectory (performance (P) or robust (R)) labels are used to balance robustness and performance goals simultaneously. Robust trajectories are designed to track the oracle's reference trajectory within predefined bounds while including significantly large uncertainties, disturbances, and noise in the nonlinear plant during optimization. Additionally, performance trajectories are optimized to track the oracle's reference trajectories while including small aerodynamic uncertainties in the plant dynamics. Both sets of trajectories are penalized for high control rates during operation. For more details regarding the training procedure, see [94].

Inspired by research in safety assurance for flight systems, we seek to establish a time delay margin and nonlinear stability margin. This goal was indirectly targeted through the use of control effectiveness, λ_u, and time delay, τ_d, uncertainties used during DLC training. We say a nonlinear system with an asymptotic stabilizing control law, $u = u_{DLC}$, possesses stability margins (SM_{min}, SM_{max}) where $-1 \leq SM_{min} \leq SM_{max} \leq \infty$, if for every $\lambda_u \in [SM_{min},\, SM_{max}]$ the control input $u = (1 + \lambda_u)u_{DLC}$ also asymptotically stabilizes the system [93]. Similarly, we say a discrete nonlinear system has time delay margins (TD_{min}, TD_{max}) if for every $\tau_d \in [TD_{min},\, TD_{max}]$ the control input $u_t = u_{t-\tau_d}$ also asymptotically stabilizes the system. Notice, τ_d denotes the number of units of time delay in the closed-loop system. After optimization of DLC weights, the closed-loop system's stability margins will be verified for particular equilibrium points using ROA

estimation via forward reachable sets in Section 7.3. Time delay margin will be verified using extensive simulations.

7.1.4 Systematic Procedure for Weight Convergence

We introduce a systematic procedure for training a DLC for improved weight convergence. We begin by redefining the weight matrix for the ith layer into two separate matrices where $\Theta^i_{FF} = [U^u, U^r, U^h]$ and $\Theta^i_{FB} = [W^u, W^r, W^h]$. Notice that Θ_{FF} includes weights of the deep network primarily associated with the input vector, i_t, and Θ_{FB} contains neural network weights associated with the recurrent hidden state, s_{t-1}. Then, we define a matrix of neural weights as $\Theta_{FF} = [\Theta^i_{FF}, \Theta^{i+1}_{FF}, ..., \Theta^L_{FF}, V, b_o, b_1, b_2, b_3]$ where Θ_{FF} does not involve weights associated with the recurrent hidden state. By eliminating the recurrent hidden state in the GRU module equations stated in (7–1) to (7–4), the DLC becomes a feed-forward network with all weights contained in Θ_{FF}. Hence, the training procedure first optimizes the set of weights in Θ_{FF} by disregarding the recurrent hidden state input. Then, the optimization is re-run to optimize the feedback weights, Θ_{FB}, using frozen feed-forward weights from the previous step. Finally, we again re-optimize the total weights in the system together. For each step of the training procedure described above, we utilize the layer-wise training procedure described in Chapter 6, see [94].

7.2 Hypersonic Flight Vehicle Model

In this research, we consider a highly nonlinear hypersonic flight vehicle model with flexible body effects. For flight vehicles operating at lower speeds, these flexible effects are typically ignored due to the natural frequency separation between the rigid body modes and the flexible body modes [16]. We found it desirable to include such effects due to the unexpected control moments that can be generated from the flexing of the fuselage at such high speeds. In addition, operating at hypersonic speeds with a flexible vehicle can cause changes in the pressure distribution on the flight vehicle resulting in changes in the overall flow field [6, 90].

The hypersonic flight vehicle model was modeled with four independently controller surfaces $(\delta_1,\ \delta_2,\ \delta_3,\ \delta_4)$ [94]. For control design purposes, we created virtual control surfaces (aileron δ_a, elevator δ_e, and rudder δ_r) which are operating in the conventional autopilot reference frame, see [73] for details. We assume the form of a static matrix which describes the mapping from the virtual control surfaces to the actual displacement. This is used inside the model in order to calculate forces and moments on the flight vehicle [73].

We consider only longitudinal dynamics of the flight vehicle which are assumed to be decoupled from the lateral dynamics. The well-studied longitudinal 3 degrees of freedom (DoF) equations of motion for a hypersonic flight vehicle can be written in the following form [6, 74]:

$$\begin{aligned}
\dot{V}_T &= \frac{1}{m}(T\ \cos(\alpha) - D) - g\ \sin(\theta - \alpha) \\
\dot{\alpha} &= \frac{1}{mV_T}(-T\ \sin(\alpha) - L) + q + \frac{g}{V_T}\cos(\theta - \alpha) \\
\dot{\Theta} &= q \\
\dot{q} &= \frac{M}{I_{YY}} \\
\dot{h} &= V_T\ \sin(\theta - \alpha) \\
\ddot{\eta}_i &= -2\zeta_i\omega_i\dot{\eta}_i - \omega_i^2\eta_i + N_i, \qquad i = 1, 2, .., n_\eta
\end{aligned} \tag{7–12}$$

where m is the mass of the vehicle, T is the thrust, V_T is the true airspeed, α is angle of attack, h is the height (i.e. altitude) of the flight vehicle, Θ is the pitch angle, q is pitch rate, I_{YY} is the moment of inertia, and g is gravity. For this problem, we consider no engine or rocket booster which results in zero thrust (i.e. $T = 0$). The ith structural mode of the flight vehicle is defined by the natural frequency (ω_i), the damping ratio (ζ_i), and the generalized force (N_i). The forces and moments acting on the longitudinal dynamics of the flight vehicle include thrust (T), drag (D), lift (L), and pitching moment (M).

For this work, we assume the vehicle operates at a constant temperature. However, it is worth noting that in many cases the natural frequencies of the hypersonic vehicle's body modes can vary based on temperature [7]. We define the state vector, $x \in \mathbb{R}^{11}$, by

$$x = [V_T,\, \alpha_,\Theta,\, q,\, h,\, \eta_1,\, \dot{\eta}_1, \eta_2,\, \dot{\eta}_2,\, \eta_3,\, \dot{\eta}_3] \tag{7–13}$$

where we assume that only three elastic modes of the flight vehicle are active. We also assume the following relationship between axial and normal $(A,\, N)$ forces and lift and drag forces $(L,\, D)$:

$$L = Ncos(\alpha) - Asin(\alpha) \tag{7–14}$$

$$D = Nsin(\alpha) + Acos(\alpha) \tag{7–15}$$

where L and D are the lift and drag forces used in (7–12).

Like many recent works in hypersonic control, we estimate the axial and normal body forces $(A,\, N)$ and pitching moment (M) by [73, 74]

$$A \approx \frac{1}{2}\rho V_T^2 S C_A \tag{7–16}$$

$$N \approx \frac{1}{2}\rho V_T^2 S C_N \tag{7–17}$$

$$M \approx \frac{1}{2}\rho V_T^2 S c_{ref} C_m \tag{7–18}$$

$$N_i \approx \frac{1}{2}\rho V_T^2 S C_{N_i} \qquad i = 1, 2, .., n_\eta \tag{7–19}$$

where ρ denotes air density, S is the reference area, c_{ref} is the mean aerodynamic chord, C_A is the axial force coefficient, C_N is the normal force coefficient, and C_m is the pitch moment coefficient.

The force and moment coefficients are composed of smaller coefficient components (e.g. $C_{A_{AB}}$and C_{m_0}) which are stored in the form of a look-up table (LUT). The inputs of the LUT consist of the flight condition $(\alpha,\, \mathrm{Mach},\, h,\, q)$, the control inputs $(\delta_1,\, \delta_2,\, \delta_3,\, \delta_4)$, and the flexible body states (η_1, η_2, η_3). We assume the coefficients take the following

form:

$$
\begin{aligned}
C_A &= C_{A_{ALT}}(h,\,\text{Mach}) + C_{A_{AB}}(\alpha,\,\text{Mach}) \qquad (7\text{–}20)\\
&\quad + \sum_{j=1}^{4} C_{A_{\delta_j}}(\alpha,\,\text{Mach},\,\delta_j) + \sum_{k=1}^{3} C_{A_{\eta_k}}(\eta_k)\\
C_N &= C_{N_0}(\alpha,\,\text{Mach}) + C_{N_q}(\alpha,\,\text{Mach},\,q) \qquad (7\text{–}21)\\
&\quad + \sum_{j=1}^{4} C_{N_{\delta_j}}(\alpha,\,\text{Mach},\,\delta_j) + \sum_{k=1}^{3} C_{N_{\eta_k}}(\eta_k)\\
C_m &= C_{m_0}(\alpha,\,\text{Mach}) + C_{m_q}(\alpha,\,\text{Mach},\,q) \qquad (7\text{–}22)\\
&\quad + \sum_{j=1}^{4} C_{m_{\delta_j}}(\alpha,\,\text{Mach},\,\delta_j) + \sum_{k=1}^{3} C_{m_{\eta_k}}(\eta_k)\\
C_{N_i} &= N_{i_{\alpha^2}}(\alpha^2) + N_{i_\alpha}(\alpha) \qquad (7\text{–}23)\\
&\quad + \sum_{j=1}^{4} N_{i_{\delta_j}}(\delta_j) + \sum_{k=1}^{3} N_{i_{\eta_k}}(\eta_k)
\end{aligned}
$$

Since our plant dynamics are called at each iteration of the DLC optimization, it is beneficial to use polynomial approximations of each aerodynamic coefficient stated previously. The lowest order of the polynomial fit was selected based on evaluating accuracy and smoothness metrics. As discussed in the sparse neural network case, we created separate polynomial models for different regions throughout the flight envelope. This allows the accuracy of the model to increase while reducing the order of the polynomials. We refer the reader to our previous work in Chapter 6 and [91] for more details regarding the use of polynomial approximations of aircraft dynamics.

The discrete time form of the polynomial model of the longitudinal dynamics of the flight vehicle used for optimization of the DLC is given by

$$
\begin{aligned}
x_{t+1} &= f(x_t, (\lambda_u(u^{act}_{t-\tau_d} + \rho_u)) + d_u) + \zeta_p \qquad (7\text{–}24)\\
y_t &= f(x_t, (\lambda_u(u^{act}_{t-\tau_d} + \rho_u)) + d_u) + \zeta_p
\end{aligned}
$$

where u^{act} is the control deflection from the actuator, $x_t = [V_T, \alpha, \Theta, q, h]$ is the state vector which disregards the flexible modes, y_t is the set of output states, λ_u is a multiplicative control effectiveness term, ρ_u is a constant additive control uncertainty, and ζ_p and d_u are plant noise and input disturbance terms, respectively, at time t. We also include a time delay parameter $\tau_d \in \mathbb{N}$ which is used during optimization to produce a time delay margin.

7.3 Results

7.3.1 Deep Learning Optimization

We explored several variations of the DLC by varying the number of layers, hidden nodes, and shared nodes (for the S-DLC case). Similar to previous work, we found that by using $L = 2$ hidden layers and $N = 12$ hidden nodes resulted in reasonable tracking performance. Unfortunately, by adding an additional layer, the DLC only performed slightly better. However, we did see significant improvement in parameter convergence by using the S-DLC with segmented flight space shown in Figure 7-3. The S-DLC contained $T = 39$ segments with $Q = 6$ nodes allocated to each segment. The S-DLC operated with $N_{act} = 12$, the same number as the DLC, which aimed to keep the computational burden on both systems equivalent. The S-DLC, with its expanded memory, used $N = 234$ total nodes in the system. We used the well-studied L-BFGS optimization method in order to train the deep recurrent network with GRU modules. L-BFGS is a quasi-newton second order batch method which has produced state-of-the-art results on both regression and classification tasks [55].

Table 7-1. Range of initial conditions and uncertainty variables

	MIN	MAX
α_0(deg)	-25	25
q_0(deg/sec)	-75	75
$Mach_0$	5	6
alt_0(km)	12	16
P_u(deg)	-5	5
T_d(samples)	0	5
Λ_u	0.5	2

For our research, the controller optimization was performed using angle of attack command tracking (i.e. $y_{ref} = \alpha_{ref},\ y_{cmd} = \alpha_{cmd}$, and $y_{sel} = \alpha$) where we set the bounds of funnels to $b_e = 0.005$ and $b_{\dot{u}} = 0.9$. The input to the DLC controller at time t, c_t, takes the following form:

$$c_t = [e_I,\ \alpha,\ q,\ \mathrm{Mach},\ h] \tag{7–25}$$

where α is the angle of attack of the vehicle, q is the pitch rate, M is the Mach number of the vehicle, h is the height (altitude), and e_I is the integral error of tracking ($e_I = \int_0^{t_f} y_{sel} - y_{cmd}\, dt$). For the S-DLC case, we reduce the set of states used as inputs to $c_t = [e_I,\ \alpha,\ q]$.

The RNN/GRU controller parameter optimization was performed using $N_T = 5,760$ sample trajectories. The sample trajectories were divided into 420 performance trajectories and 5,250 robust trajectories. The optimization procedure and associated cost function are described in Section 7.1.3. The plant dynamics were discretized with $\Delta t = 0.01$ second time steps using AB-2 integration. Each trajectory used in training was $t_f = 2.5$ seconds in duration.

As mentioned in Section 7.1.3, each trajectory is defined by its predetermined initial conditions and uncertainty parameters. Individual samples were determined by sampling uniformly distributed random variables between bounds established based on system requirements. For instance, Table 7-1 shows the range of initial conditions used during training where α_0 is the initial angle of attack, q_0 is the initial pitch rate, Mach_0 is the initial Mach number, and alt_0 is the initial altitude. Similarly, the table also shows the range of each uncertainty parameter in P_c where each parameter is held constant for each sample trajectory. Note that the control effectiveness term, λ_u, was sampled in the range of $(1 + SM_{min}) \leq \lambda_u \leq (1 + SM_{max})$ in order to establish stability margins (SM_{min}, SM_{max}). While the time delay term, τ_d, was sampled to establish time delay margins of (TD_{min}, TD_{max}). In order to reduce the dimensionality of the polynomial system during optimization, the flexible effects were included as disturbances in our

Table 7-2. Average tracking error (ATE), average control rate (ACR), and final cost

	ATE	ACR	Cost
3-Layer S-DLC (S)	185.6	193.4	98.79
2-Layer DLC (S)	568.4	79.23	572.5
2-Layer DLC	592.6	88.34	633.1
GS	1450.1	71.85	-

Note: (S) denotes systematic training procedure

system and the weights of the controller were not optimized to counter-act them. We will verify the stability margins using ROA estimation in Section 7.3.3.

7.3.2 Hypersonic Flight Control Simulation

For simulation results and clarity, we developed an analysis model for the longitudinal dynamics of a flight vehicle. This analysis model was obtained by substituting particular trim values of true airspeed (V_T), altitude (h), and pitch angle (Θ) into (7–12). The analysis model is given by

$$\dot{e}_I = \alpha - \alpha_{cmd} \tag{7–26}$$

$$\dot{\alpha} = f(\alpha,\, q,\, \delta_e,\, \lambda_u,\, \rho_u,\, \tau_d) \tag{7–27}$$

$$\dot{q} = f(\alpha,\, q,\, \delta_e,\, \lambda_u,\, \rho_u,\, \tau_d) \tag{7–28}$$

where $P_c = (\lambda_u,\, \rho_u,\, \tau_d)$ is the set of constant system uncertainties, $u_{act} = \delta_e$ is the elevator deflection produced by the actuators, and the state vector includes the integral error of tracking and takes the form $x = [e_I,\, \alpha,\, q]$. For the analysis in this chapter, we selected $\mathrm{Mach} = 5.5$, $\Theta = 0$, $h = 14$ km, and $\alpha_{cmd} = 0$ as the trim condition to analyze.

We analyzed a set of $N_S = 64$ trajectories with various initial conditions and uncertainty values. Similar to the procedure used during optimization, we obtained sample vectors for each uncertainty parameter and initial condition state by sampling a uniformly sampled random variable with ranges that slightly exceeded the ranges used during optimization (Table 7-1). For simplicity, we used two previously developed metrics, the average tracking error (ATE) and the average control rate (ACR), to compare the performance of the S-DLC, DLC, and gain-scheduled controller [94]. The metrics are defined

by

$$ATE = \frac{k_{TE}}{N_S} \sum_{i=1}^{N_S} \sum_{t=1}^{t_f} |e_t| \tag{7–29}$$

$$ACR = \frac{k_{CR}}{N_S} \sum_{i=1}^{N_S} \sum_{t=1}^{t_f} |\dot{u}_t| \tag{7–30}$$

where and e_t and $\dot{u}_t$ are the instantaneous tracking error and control rate defined previously. We used the constants k_{TE} and k_{CR} to scale ATE and ACR for comparison purposes. The optimization results are shown in Table 7-2. Time delay margins and closed-loop performance under additive uncertainties were verified for the analysis model described previously using extensive simulations. Phase portrait and tracking performance plots are provided in Figures 7-4 to 7-6 in order to compare the controller's performance under harsh conditions visually.

7.3.3 Region of Attraction Estimation via Forward Reachable Sets

We are interested in leveraging very recent breakthroughs in ROA estimation of an equilibrium point for the closed-loop flight system via forward reachable sets. This estimation provides a less conservative and non-polynomial approach which is provable and accurate. We use this approach to verify the stability margins for our closed-loop system.

Consider a generic dynamic system in the form:

$$\dot{x} = f(x(t),\, u(t)) \tag{7–31}$$

where u is the input, x is the state vector, t is the time, and f is locally Lipschitz continuous.

The stability region that we are interested in estimating can be defined by

$$S_{ROA}(x_{eq}) = \{x(0) \in \mathbb{R}^n : \lim_{t\to\infty} \nu(x,\, t) = x_{eq}\} \tag{7–32}$$

where $\nu(x,\, t)$ is a system trajectory of (7–31) which starts from the initial states $x(0)$. A reachable set is defined as the set of states that satisfies (7–32) within a given period of time ($\tau \in [0, t_f]$) starting from a set of initial states ($x(0) \in \mathbb{R}^n$).

The algorithm leveraged in this chapter finds an over-approximation of reachable sets using linear approximation methods and recursive partitioning of the state space. The vast majority of research over the last decade on reachability set methods stem from linear time-invariant (LTI) systems with uncertain inputs [54]. By linearizing the system dynamics about the current operating point and considering the linearization error as an uncertain input to the linearized system, we can apply results in reachability analysis to general nonlinear and hybrid systems, see [53, 54]. This approach has the added benefit of allowing us to use well-studied tools from linear systems theory such as the super position principle, linear maps, and linear geometric methods. We will not provide intimate details of ROA estimation or reachable set theory. Instead, the reader is referred to [53, 54].

In this chapter, we use a feed-forward only version of the DLC to analyze the stability margins. Hence, we assume the recurrent hidden state is disconnected, and the DLC operates with only the weights associated with feed-forward connections, i.e. Θ_{FF}. The resulting controller takes the following form:

$$\delta_e = f(e_I,\, \alpha,\, q) \tag{7–33}$$

where there is no longer a dependence on time. Due to memory constraints, we obtained a high order polynomial approximation of the feed-forward controller using methods described previously which is used in this analysis. Consider the 4-dimensional system defined by

$$\dot{e}_I = \alpha - \alpha_{cmd} \tag{7–34}$$

$$\dot{\alpha} = f(\alpha,\, q,\, \delta_e,\, \lambda_u) \tag{7–35}$$

$$\dot{q} = f(\alpha, q, \delta_e, \lambda_u) \tag{7–36}$$

$$\dot{\lambda_u} = 0 \tag{7–37}$$

where we set $\alpha_{cmd} = 0$. The system has an equilibrium point about the origin. We investigate the domain $D = [-0.5, 0.5] \times [-0.25, 0.25] \times [-0.5, 0.5] \times [0.5, 2.0]$ of the closed-loop system which is shown in Figure 7-7. The gray boxes in the figure reveal the regions of the state-space where initial conditions have guaranteed convergence to the origin. The region of attraction was verified using extensive simulations.

7.4 Summary

We developed a sparsely activated robust deep learning controller for hypersonic flight vehicle control. This controller was trained systematically using a large set of sample trajectories. We included control effectiveness and time delay terms in sample trajectories during optimization in order to establish stability and time delay margins. These margins were verified for a particular equilibrium point by using a region of attraction method via forward reachable sets. We found the sparse deep learning controller to be superior in terms of both transient and robustness compared to a standard fully-connected deep learning controller and a more traditional gain-scheduled controller.

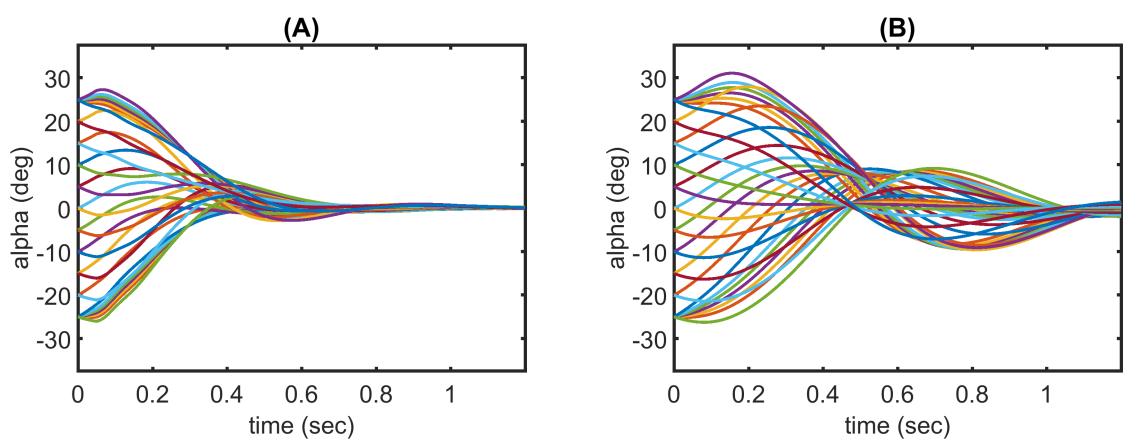

Figure 7-4. Traditional step responses for A) DLC and B) GS with uncertainty values $\lambda_u = 0.6$, $\rho_u = -5$, and $\tau_d = 3$.

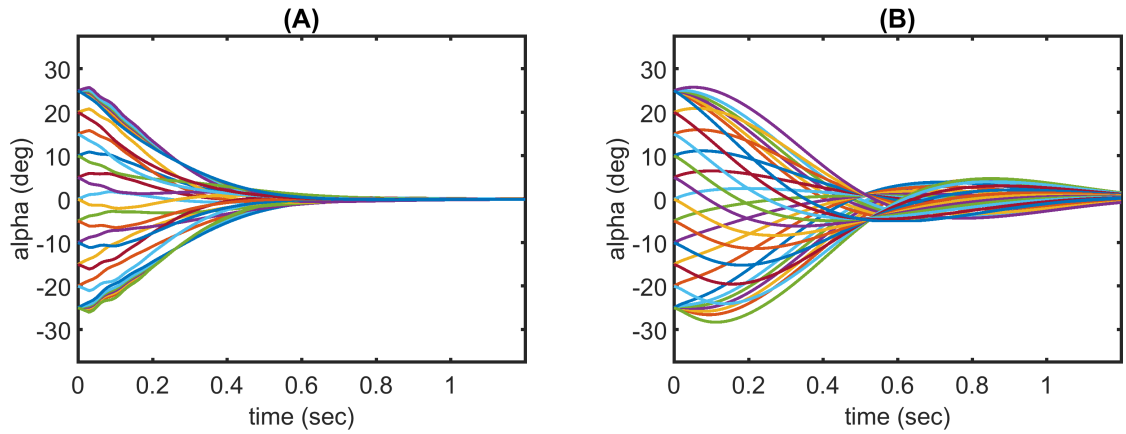

Figure 7-5. Traditional step responses for A) DLC and B) GS with uncertainty values $\lambda_u = 1.5$, $\rho_u = 3$, and $\tau_d = 1$.

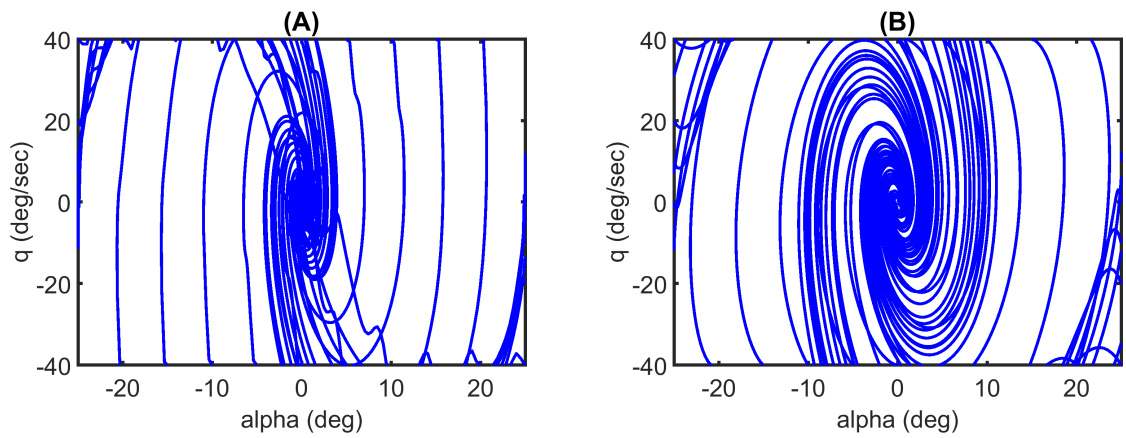

Figure 7-6. Phase portrait plots for A) DLC and B) GS with uncertainty values $\lambda_u = 0.5$, $\rho_u = -0.5$, and $\tau_d = 0$.

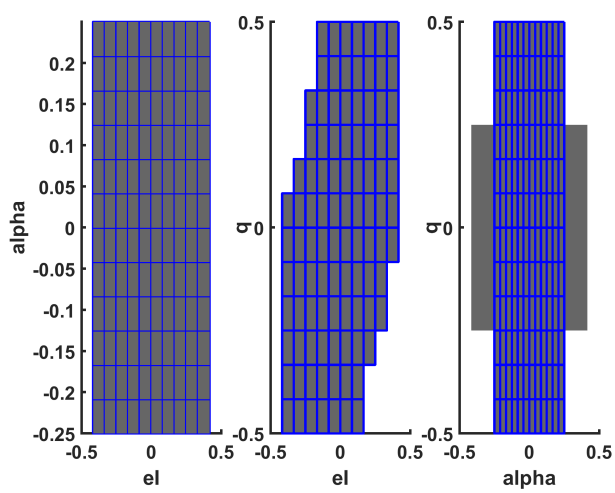

Figure 7-7. Region of attraction estimate via forward reachable sets for $\lambda_u = [0.5,\ 2.0]$.

CHAPTER 8
CONCLUSIONS

The dissertation attacks various problems related to hypersonic control using deep and sparse neural network controllers. The main problems that were addressed include the computational limitations, vast dynamical changes, and flexible body effects of hypersonic vehicles (HSVs). These issues were addressed through two related methodologies. First, we developed a sparsely connected adaptive controller which improves transient performance on control tasks with persistent region-based uncertainties. The controller operates by partitioning the flight envelope into regions and utilizing only a small number of neurons while operating in each region. We improved the controller by including additional adaptive terms to counter-act control degradation. We showed that by enforcing a specified dwell time condition and utilizing a robust adaptive term, we could guarantee a uniformly ultimate bounded (UUB) Lyapunov stability result. Simulation studies showed the significant benefits of employing the SNN architecture over traditional radial basis function (RBF) and single hidden layer (SHL) schemes. The second approach that was used to solve HSV control challenges utilized a deep recurrent neural network architecture for the controller. We developed a novel training procedure that simultaneously addressed performance and robustness based goals. We analyzed the stability margins of the closed-loop system using recently developed region of attraction (ROA) estimation methods. In addition, we developed a sparse deep learning controller (S-DLC) which led to significantly improved parameter convergence. This convergence resulted in a high-performance robust controller for hypersonic flight. Simulation studies show the effectiveness of the DLC against a traditional gain-scheduled approach with uncertainties in various forms.

Similar to much research in control literature, we make use of control-oriented models (reduced order) that make many assumptions about the dynamics of the flight vehicle. For instance, although we include the first several bending modes in our model,

we assume sufficient decoupling between the longitudinal and lateral dynamics of the vehicle and do not consider propulsion or propulsion-based effects. In addition, the architecture developed in Chapters 4 and 5 relies on the adaptive controller to compensate for effects not captured in the linearized (gain-scheduled) dynamics of the flight vehicle. Hence, it would be interesting to employ the sparse adaptive controller in various control architectures which allow for more realistic models. Similarly, future work could include using the sparse deep network concept and training methodology established in Chapters 6 and 7 to obtain a more accurate flight vehicle model for control.

As mentioned throughout this document, the persistence of excitation (PE) condition is often relied upon for many system identification and adaptive control tasks. This condition requires the input to have sufficient energy in order for the estimated parameters to converge properly. Another interesting avenue of research could be to pursue the benefits of the sparse neural network architecture in system identification tasks.

Future work could also include investigating the effectiveness of the higher-order terms in the neural network adaptive laws. It would be interesting to analyze various control platforms, activation functions, and learning rates in the sparse neural network architecture. Sparse multi-layer neural network adaptive control could also be another direction to pursue.

Finally, the expressive power of deep neural networks is well-known. However, due to its highly nonlinear and elaborate structure, deep neural networks are often difficult to analyze. Another area of interest is to establish more accurate and suitable analysis tools for deep neural network-based controllers.

APPENDIX A
DEVELOPMENT OF BOUNDS FOR THE NEURAL NETWORK ADAPTIVE ERROR (CH 4)

Using the second order Taylor series expansion terms generated in Chapter 5 along with the form of the neural network adaptive controller in (5–28) results in the following bound on the adaptive error:

$$
\begin{aligned}
-(u_{NN} + f_\Lambda(x)) &= \hat{W}_i^T \sigma(\hat{V}_i^T \mu) - W_i^T \sigma(V_i^T \mu) - \epsilon_i \\
&= \hat{W}_i^T \sigma(\hat{V}_i^T \mu) - W_i^T (\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\tilde{V}_i^T \mu) \\
&\quad -\frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)((V_i^T \mu - \hat{V}_i^T \mu) \odot \tilde{V}_i^T \mu) + O(\tilde{V}_i^T \mu)^3) - \epsilon_i \\
&= \tilde{W}_i^T \sigma(\hat{V}_i^T \mu) + W_i^T \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu) - W_i^T \dot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu) \\
&\quad +\frac{1}{2} W_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu \odot \tilde{V}_i^T \mu) \qquad \text{(A–1)} \\
&\quad -\frac{1}{2} W_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \tilde{V}_i^T \mu) - W_i^T O(\tilde{V_i^T} \mu)^3 - \epsilon_i \\
&= \tilde{W}_i^T (\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu) + \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \hat{V}_i^T \mu)) \\
&\quad +(\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu)\tilde{V}_i^T \mu - \frac{1}{2}\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \tilde{V}_i^T \mu)) \\
&\quad +\tilde{W}_i^T (\dot{\sigma}(\hat{V}_i^T \mu) V_i^T \mu - \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot V_i^T \mu)) \\
&\quad +\frac{1}{2} W_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu \odot \tilde{V}_i^T \mu) - W_i^T O(\tilde{V}_i^T \mu)^3 - \epsilon_i.
\end{aligned}
$$

The following upper bound can be established using the definition of h_i from (5–37):

$$
\begin{aligned}
||h_i - \epsilon_i|| &= ||\tilde{W}_i^T (\dot{\sigma}(\hat{V}_i^T \mu) V_i^T \mu - \frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot V_i^T \mu)) \\
&\quad +\frac{1}{2} W_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu \odot \tilde{V}_i^T \mu) - W_i^T O(\tilde{V}_i^T \mu)^3 - \epsilon_i|| \\
&= ||\tilde{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu) V_i^T \mu - W_i^T \dot{\sigma}(\hat{V}_i^T \mu)\tilde{V}_i^T \mu + W_i^T (\sigma(\hat{V}_i^T \mu) - \sigma(V_i^T \mu)) \qquad \text{(A–2)} \\
&\quad -\frac{1}{2}\tilde{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot V_i^T \mu) + \frac{1}{2} W_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(V_i^T \mu \odot \tilde{V}_i^T \mu) \\
&\quad -\frac{1}{2}\ddot{\sigma}(\hat{V}_i^T \mu)((V_i^T \mu - \hat{V}_i^T \mu) \odot \tilde{V}_i^T \mu) - \epsilon_i||.
\end{aligned}
$$

After grouping terms the bound becomes

$$
\begin{aligned}
||h_i - \epsilon_i|| &= ||[\hat{W}_i^T \dot{\sigma}(\hat{V_i^T}\mu) V_i^T \mu - W_i^T \dot{\sigma}(\hat{V_i^T}\mu) V_i^T \mu] \\
&\quad -[W_i^T \dot{\sigma}(\hat{V_i}^T \mu)\hat{V}_i \mu - W_i^T \dot{\sigma}(\hat{V_i^T}\mu) V_i^T \mu] \\
&\quad -\frac{1}{2}\tilde{W_i^T}\ddot{\sigma}(\hat{V_i^T}\mu)(\hat{V_i}^T \mu \odot \hat{V_i^T}\mu) + \frac{1}{2}\hat{W}_i^T \ddot{\sigma}(\hat{V_i^T}\mu)(\hat{V_i}^T \mu \odot \tilde{V_i^T}\mu) \\
&\quad +W_i^T \sigma(\tilde{V_i^T}\mu) - \epsilon_i|| \qquad \text{(A–3)} \\
&= ||\hat{W}_i^T \dot{\sigma}(\hat{V_i}^T \mu) V_i^T \mu - W_i^T \dot{\sigma}(\hat{V_i}^T \mu)\hat{V}_i \mu + \frac{1}{2} W_i^T \ddot{\sigma}(\hat{V_i^T}\mu)(\hat{V_i}^T \mu \odot \hat{V_i^T}\mu) \\
&\quad -\frac{1}{2}\hat{W}_i^T \ddot{\sigma}(\hat{V_i^T}\mu)(\hat{V_i}^T \mu \odot V_i^T \mu) + W_i^T \sigma(\tilde{V_i}^T \mu) - \epsilon_i|| \\
&\leq \zeta_i \psi_i.
\end{aligned}
$$

APPENDIX B
PROJECTION OPERATOR DEFINITIONS (CH 3/4)

In order to introduce the projection operator, we first define a chosen smooth convex function, $f : \mathbb{R}^N \rightarrow \mathbb{R}$, which takes the form [16]:

$$f = f(\hat{\Theta}) = \frac{(1+\epsilon)||\hat{\Theta}||^2 - \bar{\Theta}^2}{\epsilon\bar{\Theta}^2} \tag{B–1}$$

where $\epsilon \in [0, 1]$ is often referred to as the projection tolerance and $\hat{\Theta} \in \mathbb{R}^N$ is a vector of adaptive weights with an upper bound denoted by $\bar{\Theta} \in \mathbb{R}$. The projection tolerance, ϵ, and the upper bound on the adaptive weights, $\bar{\Theta}$, are predefined parameters set by the user and used by the adaptive law during run-time. We define the gradient of f by

$$\nabla f = \frac{2(1+\epsilon)}{\epsilon\bar{\Theta}^2}\hat{\Theta} \tag{B–2}$$

and two convex sets (Ω_0 and Ω_1):

$$\begin{aligned}
\Omega_0 &= \{f(\hat{\Theta}) \leq 0\} && \text{(B–3)}\\
&= \{\hat{\Theta} \in \mathbb{R}^N : \ ||\hat{\Theta}|| \leq \frac{\bar{\Theta}}{\sqrt{1+\epsilon}}\} && \text{(B–4)}\\
\Omega_1 &= \{f(\hat{\Theta}) \leq 1\} && \text{(B–5)}\\
&= \{\hat{\Theta} \in \mathbb{R}^N : \ ||\hat{\Theta}|| \leq \bar{\Theta}\}. && \text{(B–6)}
\end{aligned}$$

For use in the adaptive laws, the projection operator operates by the following equation [16]:

$$\begin{aligned}
\dot{\hat{\Theta}} &= \mathrm{Proj}(\hat{\Theta}, \Gamma y) && \text{(B–7)}\\
&= \Gamma \begin{cases} y - \frac{\Gamma\nabla f(\nabla f)^T}{(\nabla f)^T\Gamma\nabla f}\Gamma y f & if\ f(\hat{\Theta}) > 0\ and\ y^T\Gamma\nabla f > 0 \\ y & otherwise \end{cases}
\end{aligned}$$

where $y \in \mathbb{R}^N$ is a known piecewise continuous vector. Then, the following useful property will be utilized in Lyapunov stability analysis:

$$(\tilde{\Theta})(\Gamma^{-1}\,\mathrm{Proj}(\hat{\Theta}, \Gamma y) - y) \leq 0 \tag{B–8}$$

where we define the adaptive weight error as $\tilde{\Theta} = \hat{\Theta} - \Theta$.

APPENDIX C
DEVELOPMENT OF UPPER BOUNDS BASED ON SPARSE NEURAL NETWORK ADAPTIVE UPDATE LAWS (CH 4)

In the following equations, we will show how an upper-bound of zero is developed for a number of terms in (5–46) by using the chosen form of the adaptive control laws stated in (5–24) to (5–27). We refer to the group of terms that will be upper-bounded as V_{impact}, $W_{impact,}$ and K_{impact}.

For instance, the effect of the adaptive law stated in (5–25) on $\dot{V}$ results in:

$$\begin{aligned}
V_{impact} &= -2e^T PB\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu)\tilde{V}_i^T \mu + e^T PB\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \tilde{V}_i^T \mu) + \mathrm{trace}(\tilde{V}_i^T \Gamma_V^{-1} \dot{\hat{V}}_i) \\
&= -2e^T PB\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu)\tilde{V}_i^T \mu + 2\ \mathrm{trace}(\tilde{V}_i^T \Gamma_V^{-1} \mathrm{Proj}(\hat{V}_i, \Gamma_V \mu e^T PB\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu))) \\
&\quad + e^T PB\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \tilde{V}_i^T \mu) \\
&\quad + \mathrm{trace}(\tilde{V}_i^T \Gamma_V^{-1} \mathrm{Proj}(\hat{V}_i, -\Gamma_V \mu e^T PB\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)\, \mathrm{diag}(\hat{V}_i^T \mu)) \qquad \text{(C–1)} \\
&= 2\ \mathrm{trace}(\tilde{V}_i^T (\Gamma_V^{-1} \mathrm{Proj}(\hat{V}_i, \Gamma_V \mu e^T PB\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu)) - \mu e^T PB\hat{W}_i^T \dot{\sigma}(\hat{V}_i^T \mu))) \\
&\quad + \mathrm{trace}(\tilde{V}_i^T (\Gamma_V^{-1} \mathrm{Proj}(\hat{V}_i, -\Gamma_V \mu e^T PB\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)\, \mathrm{diag}(\hat{V}_i^T \mu)) \\
&\quad + \mu e^T PB\hat{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)\, \mathrm{diag}(\hat{V}_i^T \mu))) \\
&\leq 0. \qquad \text{(C–2)}
\end{aligned}$$

Similarly, the impact of $\dot{\hat{W}}_i$ yields:

$$\begin{aligned}
W_{impact} &= -2e^T PB\tilde{W}_i^T (\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu)) \\
&\quad - e^T PB\tilde{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \hat{V}_i^T \mu) + \mathrm{trace}(\tilde{W}_i^T \Gamma_W^{-1} \dot{\hat{W}}_i) \\
&= -2e^T PB(\tilde{W}_i^T (\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu)) \\
&\quad + 2\ \mathrm{trace}(\tilde{W}_i^T \Gamma_W^{-1} \mathrm{Proj}(\hat{W}_i, \Gamma_W (\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu)) e^T PB)) \\
&\quad - e^T PB\tilde{W}_i^T \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \hat{V}_i^T \mu) \qquad \text{(C–3)} \\
&\quad + \mathrm{trace}(\tilde{W}_i^T \Gamma_W^{-1} \mathrm{Proj}(\hat{W}_i, \Gamma_W \ddot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu \odot \hat{V}_i^T \mu) e^T PB)) \\
&= 2\ \mathrm{trace}(\tilde{W}_i^T \Gamma_W^{-1} \mathrm{Proj}(\hat{W}_i, \Gamma_W (\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu)) e^T PB) \\
&\quad - (\sigma(\hat{V}_i^T \mu) - \dot{\sigma}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu)) e^T PB)
\end{aligned}$$

$$
\begin{aligned}
& + \operatorname{trace}(\tilde{W}_i^T \Gamma_W^{-1} \operatorname{Proj}(\hat{W}_i, \Gamma_W \ddot{\sigma}(\hat{V}_i^T \mu) \operatorname{diag}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu) e^T P B) \\
& - \ddot{\sigma}(\hat{V}_i^T \mu) \operatorname{diag}(\hat{V}_i^T \mu)(\hat{V}_i^T \mu) e^T P B) \\
\leq \; & 0. \qquad \text{(C–4)}
\end{aligned}
$$

Finally, $\dot{\hat{K}}_\Lambda$ in (5–27) forms the following upper bound:

$$
\begin{aligned}
K_{impact} &= 2 e^T P B \Lambda (K_\Lambda - \hat{K}_\Lambda)(u_{BL} + u_{NN} + u_{RB}) + \operatorname{trace}(\tilde{K}_\Lambda^T \Gamma_K^{-1} \dot{\hat{K}}_\Lambda \Lambda) \\
&= -2 e^T P B \Lambda \tilde{K}_\Lambda (u_{BL} + u_{NN} + u_{RB}) + \operatorname{trace}(\tilde{K}_\Lambda^T \Gamma_K^{-1} \dot{\hat{K}}_\Lambda \Lambda) \qquad \text{(C–5)} \\
&= 2 \operatorname{trace}(\tilde{K}_\Lambda^T \Gamma_K^{-1} Proj(\hat{K}_\Lambda, \Gamma_K (u_{BL} + u_{NN} + u_{RB}) e^T P B \Lambda) \\
&\quad - (u_{BL} + u_{NN} + u_{RB}) e^T P B \Lambda) \\
&\leq 0. \qquad \text{(C–6)}
\end{aligned}
$$

APPENDIX D
ALTERNATIVE DWELL-TIME CONDITION APPROACH (CH 4)

Suppose that for each time segment Δt_S there exists $\mu_S \geq 1$ such that at any switching time (t_S) we have

$$V_{\Delta t_{S+1}}(t_S) \leq \mu_S V_{\Delta t_S}(t_S^-) \tag{D–1}$$

where $V_{\Delta t_{S+1}}(t_S)$ refers to the Lyapunov candidate value during the t_{S+1} segment at time t_S. Now consider a function in the following form [30, 89]:

$$W(t) = e^{c_V t} V_i(t) \tag{D–2}$$

where from (5–66) and (D–2) we can show that

$$\dot{W}(t) \leq e^{c_V t} c_V \bar{k}_T \tag{D–3}$$

for every time segment. Consider a interval of time $[t_{S-1},\, t_S]$ with one switch, where

$$W(t_{S+1}) \leq \mu_S (W(t_S) + \int_{t_{S-1}}^{t_S} e^{c_V t} c_V \bar{k}_T \, dt) \tag{D–4}$$

is derived from (5–66), (D–1), and (D–3). Rearranging terms and simplifying results in

$$V_{\Delta t_{S+1}}(t_S) \leq \mu_S e^{-c_V t_S} (e^{c_V t_{S-1}} V_{\Delta t_S}(t_{S-1}) + \int_{t_{S-1}}^{t_S} e^{c_V t} c_V \bar{k}_T \, dt). \tag{D–5}$$

Using (5–53) and since

$$\int_{t_{S-1}}^{t_S} e^{c_V t} c_V \bar{k}_T \, dt = \bar{k}_T (e^{c_V t_S} - e^{c_V t_{S-1}}) \tag{D–6}$$

then (D–5) becomes

$$V_{\Delta t_{S+1}}(t_S) \leq \mu_S (e^{-c_V T_{dwell}} (V_{\Delta t_S}(t_{S-1}) - \bar{k}_T) + \bar{k}_T) \tag{D–7}$$

which holds for each time segment. If we use the dwell time bound derived in (5–77) and the assumption that $V_{\Delta t_S}(t_{S-1}) > 2\bar{k}_T + \bar{k}_{NN}$, then (D–7) becomes

$$V_{\Delta t_{S+1}}(t_S) \leq \mu_S (V_{\Delta t_S}(t_{S-1}) - \bar{k}_{NN}). \tag{D–8}$$

Since we know (D–1) is always true and

$$V_{\Delta t_{S+1}}(t_S) \leq \bar{\mu}_S(V_{\Delta t_S}(t_S) + \bar{k}_{NN}) \tag{D–9}$$

where $\bar{\mu}_S \in \mathbb{R}^+$ is a user defined constant based on (5–71). Then, we can set

$$\mu_S = \frac{\bar{\mu}_S(\bar{k}_{NN} + V_{\Delta t_S}(t_S))}{V_{\Delta t_S}(t_S)} \tag{D–10}$$

which results in

$$V_{\Delta t_{S+1}}(t_S) \leq \mu_S V_{\Delta t_S}(t_S) \tag{D–11}$$

which is equivalent to the solution derived in Theorem 5.3.

LIST OF REFERENCES

[1] Z. T. Dydek, A. M. Annaswamy, and E. Lavretsky, "Adaptive control and the nasa x-15-3 flight revisited," *IEEE Contr. Syst.*, vol. 30, no. 3, pp. 32–48, 2010.

[2] B. McGarry. (2015) Air force getting closer to testing hypersonic weapon, engineers say. [Online]. Available: http://www.military.com/daily-news/2015/05/19/air-force-getting-closer-to-testing-hypersonic-weapon.html

[3] R. Aditya, "Direct adaptive control for stability and command augmentation system of an air-breathing hypersonic vehicle," Master's thesis, 2015.

[4] K. Mizokami. (2016) Lockheed to build a mach 20 hypersonic weapon system. [Online]. Available: http://www.popularmechanics.com/military/research/a22970/lockheed-hypersonic-weapon/

[5] M. Creagh, M. Kearney, and P. Beasley, "Adaptive control for a hypersonic glider using parameter feedback from system identification," in *AIAA Guidance, Navigation, and Control Conf.*, 2011, p. 6230.

[6] M. A. Bolender, "An overview on dynamics and controls modelling of hypersonic vehicles," in *Proc. Amer. Control Conf.*, 2009, pp. 2507–2512.

[7] T. Williams, M. Bolender, D. Doman, and O. Morataya, "An aerothermal flexible mode analysis of a hypersonic vehicle," in *AIAA Atmos. Flight Mech. Conf.*, 2006, p. 6647.

[8] C. Cesnik, P. Senatore, W. Su, E. Atkins, C. Shearer, and N. Pitchter, "X-hale: a very flexible uav for nonlinear aeroelastic tests," in *Proc. Struct. Dyn. Mater. Conf.*, 2010, p. 2715.

[9] R. Upadhyay, K. Schulz, P. Bauman, R. Stogner, and O. Ezekoye, "Steady-state ablation model coupling with hypersonic flow," in *AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 2010, p. 1176.

[10] N. J. Falkiewicz, C. E. S. Cesnik, A. R. Crowell, and J. J. McNamara, "Reduced-order aerothermoelastic framework for hypersonic vehicle control simulation," *AIAA J.*, vol. 49, no. 8, pp. 1625–1646, 2011.

[11] T. E. Gibson, "Closed-loop reference model adaptive control: with application to very flexible aircraft," Ph.D. dissertation, Massachusetts Inst. of Technology, 2013.

[12] Z. Han and K. S. Narendra, "New concepts in adaptive control using multiple models," *IEEE Trans. Autom. Control*, vol. 57, no. 1, pp. 78–89, 2012.

[13] F. Lewis and S. S. Ge, "Neural networks in feedback control systems," *Mech. Eng. Handbook*, 2005.

[14] N. Hovakimyan, F. Nardi, A. Calise, and N. Kim, “Adaptive output feedback control of uncertain nonlinear systems using single-hidden-layer neural networks,” *IEEE Trans. Neural Netw.*, vol. 13, no. 6, pp. 1420–1431, 2002.

[15] R. T. Anderson, G. Chowdhary, and E. N. Johnson, “Comparison of rbf and shl neural network based adaptive control,” in *Unmanned Aircraft Systems*. Springer, 2008, pp. 183–199.

[16] E. Lavretsky and K. Wise, *Robust and Adaptive Control: With Aerospace Applications*. Springer Science & Business Media, 2012.

[17] F. Lewis, “Nonlinear network structures for feedback control,” *Asian J. Control*, vol. 1, no. 4, pp. 205–228, 1999.

[18] G. Chowdhary and E. Johnson, “Concurrent learning for convergence in adaptive control without persistency of excitation,” in *IEEE Conf. Decision and Control*, 2010, pp. 3674–3679.

[19] P. A. Ioannou and J. Sun, *Robust adaptive control*. Courier Corporation, 2012.

[20] R. Gadient, “Adaptive control with aerospace applications,” Ph.D. dissertation, 2013.

[21] Z. T. Dydek, “Adaptive control of unmanned aerial systems,” Ph.D. dissertation, Massachusetts Institute of Technology, 2010.

[22] N. Hovakimyan and A. J. Calise, “Adaptive output feedback control of uncertain multi-input multi-output systems using single hidden layer neural networks,” in *Proc. Amer. Control Conf.*, vol. 2. IEEE, 2002, pp. 1555–1560.

[23] T. Yucelen and W. M. Haddad, “Low-frequency learning and fast adaptation in model reference adaptive control,” *IEEE Trans. Autom. Control*, vol. 58, no. 4, pp. 1080–1085, 2013.

[24] A. J. Calise, N. Hovakimyan, and M. Idan, “Adaptive output feedback control of nonlinear systems using neural networks,” *Automatica*, vol. 37, no. 8, pp. 1201–1211, 2001.

[25] P. Patre, W. MacKunis, K. Dupree, and W. Dixon, “A new class of modular adaptive controllers, part i: Systems with linear-in-the-parameters uncertainty,” in *Proc. Amer. Control Conf.*, 2008, pp. 1208–1213.

[26] L. Chen and K. S. Narendra, “Nonlinear adaptive control using neural networks and multiple models,” *Automatica*, vol. 37, no. 8, pp. 1245–1255, 2001.

[27] C. Cao and N. Hovakimyan, “Design and analysis of a novel adaptive control architecture with guaranteed transient performance,” *IEEE Trans. Autom. Control*, vol. 53, no. 2, pp. 586–591, 2008.

[28] V. Stepanyan, K. Krishnakumar, N. Nguyen, and L. Van Eykeren, "Stability and performance metrics for adaptive flight control," in *AIAA Guidance, Navigation, and Control Conf.*, 2009.

[29] J. P. Hespanha, "Uniform stability of switched linear systems: Extensions of lasalle's invariance principle," *IEEE Trans. Autom. Control*, vol. 49, no. 4, pp. 470–482, 2004.

[30] D. Liberzon, *Switching in systems and control*. Springer Science & Business Media, 2012.

[31] S. Kamalasadan, "A new generation of adaptive control: An intelligent supervisory loop approach," Ph.D. dissertation, University of Toledo, 2004.

[32] Z. Liu and Y. Wang, "Fuzzy adaptive tracking control within the full envelope for an unmanned aerial vehicle," *Chin. J. Aeronaut.*, vol. 27, no. 5, pp. 1273–1287, 2014.

[33] L. Yu and S. Fei, "Robustly stable switching neural control of robotic manipulators using average dwell-time approach," *Trans. Inst. Measurement Control*, vol. 36, no. 6, pp. 789–796, 2014.

[34] Y. Huang, C. Sun, C. Qian, and L. Wang, "Non-fragile switching tracking control for a flexible air-breathing hypersonic vehicle based on polytopic lpv model," *Chin. J. Aeronaut.*, vol. 26, no. 4, pp. 948–959, 2013.

[35] J. Lian, Y. Lee, S. D. Sudhoff, and S. H. Zak, "Variable structure neural network based direct adaptive robust control of uncertain systems," in *Proc. Amer. Control Conf.*, 2008, pp. 3402–3407.

[36] J. Lian, J. Hu, and S. H. Zak, "Adaptive robust control: A piecewise lyapunov function approach," in *Proc. Amer. Control Conf.*, 2009, pp. 568–573.

[37] Y. LeCun, Y. Bengio and G. E. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 2015.

[38] R. Schneiderman, L. Deng, and V. Sejnoha, "Accuracy, apps advance speech recognition," *IEEE Signal. Proc. Mag.*, vol. 1053, no. 5888/15, 2015.

[39] M. P. Deisenroth, G. Neumann, J. Peters *et al.*, "A survey on policy search for robotics," *Foundations and Trends in Robotics*, vol. 2, no. 1–2, pp. 1–142, 2013.

[40] R. E. Bellman and S. E. Dreyfus, "Applied dynamic programming," 1962.

[41] S. Levine and V. Koltun, "Guided policy search," in *Proc. Int. Conf. Mach. Learning*, 2013, pp. 1–9.

[42] I. Sutskever, "Training recurrent neural networks," Ph.D. dissertation, University of Toronto, 2013.

[43] S. Levine, "Exploring deep and recurrent architectures for optimal control," *arXiv preprint arXiv:1311.1761*, 2013.

[44] R. Tedrake, “Lqr-trees: Feedback motion planning on sparse randomized trees,” 2009.

[45] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, “Lqr-trees: Feedback motion planning via sums-of-squares verification,” *Int. J. Robotics Research*, 2010.

[46] P. A. Parrilo, “Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization,” Ph.D. dissertation, California Institute of Technology, 2000.

[47] A. Dorobantu, L. Crespo, and P. Seiler, *Robustness analysis and optimally robust control design via sum-of-squares*, 2012.

[48] A. Papachristodoulou and S. Prajna, “On the construction of lyapunov functions using the sum of squares decomposition,” in *IEEE Conf. Decision and Control*, vol. 3, 2002, pp. 3482–3487.

[49] A. Majumdar and R. Tedrake, “Robust online motion planning with regions of finite time invariance,” in *Algorithmic Foundations of Robotics*. Springer, 2013, pp. 543–558.

[50] A. Majumdar, A. A. Ahmadi, and R. Tedrake, “Control design along trajectories with sums of squares programming,” in *IEEE Int. Conf. Robotics Automat.* IEEE, 2013, pp. 4054–4061.

[51] J. Theis, “Sum-of-squares applications in nonlinear controller synthesis,” Ph.D. dissertation, PhD Thesis, University of California, Berkeley, 2012.

[52] M.-W. Seo, H.-H. Kwon, and H.-L. Choi, “Nonlinear missile autopilot design using a density function-based sum-of-squares optimization approach,” in *IEEE Conf. Control Applicat.*, 2015, pp. 947–952.

[53] A. El-Guindy, D. Han, and M. Althoff, “Estimating the region of attraction via forward reachable sets,” in *Proc. Amer. Control Conf.*, 2017.

[54] M. Althoff, “Reachability analysis and its application to the safety assessment of autonomous cars,” *Technische Universitt Mnchen*, 2010.

[55] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen. (2014, August) Ufldl tutorial. [Online]. Available: http://ufldl.stanford.edu/tutorial/

[56] Y. Bengio, A. Courville, and P. Vincent, “Representation learning: A review and new perspectives,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 8, pp. 1798–1828, 2013.

[57] R. Schneiderman, “Accuracy, apps advance speech recognition,” *IEEE Signal. Proc. Mag.*, 2015.

[58] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks,” in *Int. Conf. Artificial Intell. Stat.*, 2011, pp. 315–323.

[59] Q. V. Le, N. Jaitly, and G. E. Hinton, “A simple way to initialize recurrent networks of rectified linear units,” *arXiv preprint arXiv:1504.00941*, 2015.

[60] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, “Maxout networks,” *arXiv preprint arXiv:1302.4389*, 2013.

[61] Q. Wang and J. JaJa, “From maxout to channel-out: Encoding information on sparse pathways,” in *Artificial Neural Networks and Machine Learning*. Springer, 2014, pp. 273–280.

[62] A. Graves, A.-r. Mohamed, and G. Hinton, “Speech recognition with deep recurrent neural networks,” in *IEEE Trans. Acoust., Speech, Signal Process.* IEEE, 2013, pp. 6645–6649.

[63] D. Britz. (2015, October) Recurrent neural network tutorial. [Online]. Available: http://www.wildml.com/

[64] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks.” *Int. Conf. Mach. Learning*, vol. 28, pp. 1310–1318, 2013.

[65] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[66] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

[67] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, “How to construct deep recurrent neural networks,” *arXiv preprint arXiv:1312.6026*, 2013.

[68] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, “On optimization methods for deep learning,” in *Int. Conf. Mach. Learning*, 2011, pp. 265–272.

[69] Y. Nesterov, “A method of solving a convex programming problem with convergence rate o (1/k2),” in *Soviet Mathematics Doklady*, vol. 27, no. 2, 1983, pp. 372–376.

[70] A. S. Lewis and M. L. Overton, “Nonsmooth optimization via quasi-newton methods,” *Math. Programming*, vol. 141, no. 1-2, pp. 135–163, 2013.

[71] K. J. Aström and R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2010.

[72] M. Sharma, E. Lavretsky, and K. A. Wise, “Application and flight testing of an adaptive autopilot on precision guided munitions,” in *AIAA Guidance, Navigation and Control Conf.*, 2006.

[73] B. Dickinson, S. Nivison, A. Hart, C. Hung, B. Bialy, and S. Stockbridge, “Robust and adaptive control of a rocket boosted missile,” in *Proc. Amer. Control Conf.*, 2015, pp. 2520–2532.

[74] B. L. Stevens and F. L. Lewis, *Aircraft control and simulation*. John Wiley & Sons, 2003.

[75] K. A. Wise, “Bank-to-turn missile autopilot design using loop transfer recovery,” *J. Guid. Contr. Dynam.*, vol. 13, no. 1, pp. 145–152, 1990.

[76] K. S. Narendra and A. M. Annaswamy, *Stable adaptive systems*. Courier Corporation, 2012.

[77] H. K. Khalil and J. Grizzle, *Nonlinear systems*. Prentice hall New Jersey, 1996, vol. 3.

[78] P. Ioannou and B. Fidan, “Adaptive control tutorial, society for industrial and applied mathematics,” 2006.

[79] K. S. Narendra and A. M. Annaswamy, “A new adaptive law for robust adaptation without persistent excitation,” *IEEE Trans. Autom. Control*, vol. 32, no. 2, pp. 134–145, 1987.

[80] S. A. Nivison and P. Khargonekar, “Improving long-term learning of model reference adaptive controllers for flight applications: A sparse neural network approach,” in *AIAA Guidance, Navigation, and Control Conf.*, 2017, p. 1249.

[81] F. L. Lewis, A. Yesildirek, and K. Liu, “Multilayer neural-net robot controller with guaranteed tracking performance,” *IEEE Trans. Neural Netw.*, vol. 7, no. 2, pp. 388–399, 1996.

[82] Y. Shin, “Neural network based adaptive control for nonlinear dynamic regimes,” Ph.D. dissertation, 2005.

[83] M. B. McFarland and D. T. Stansbery, “Adaptive nonlinear autopilot for anti-air missiles.” DTIC Document, Tech. Rep., 1998.

[84] G. Chowdhary, H. A. Kingravi, J. P. How, and P. A. Vela, “Bayesian nonparametric adaptive control using gaussian processes,” *IEEE Trans. Neural Netw.*, vol. 26, no. 3, pp. 537–550, 2015.

[85] G. Gybenko, “Approximation by superposition of sigmoidal functions,” *Math. Control Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[86] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, 2003, vol. 2.

[87] T. Zhang, S. S. Ge, and C. C. Hang, “Design and performance analysis of a direct adaptive controller for nonlinear systems,” *Automatica*, vol. 35, no. 11, pp. 1809–1817, 1999.

[88] N. Kim, “Improved methods in neural network-based adaptive output feedback control, with applications to flight control,” Ph.D. dissertation, Georgia Institute of Technology, 2003.

[89] L. Vu, D. Chatterjee, and D. Liberzon, “Iss of switched systems and applications to switching adaptive control,” in *IEEE Conf. Decision and Control*, 2005, pp. 120–125.

[90] M. A. Bolender and D. B. Doman, “Nonlinear longitudinal dynamical model of an air-breathing hypersonic vehicle,” *J. Spacecraft Rockets*, vol. 44, no. 2, pp. 374–387, 2007.

[91] A. Chakraborty, P. Seiler, and G. J. Balas, “Nonlinear region of attraction analysis for flight control verification and validation,” *Control Eng. Pract.*, vol. 19, no. 4, pp. 335–345, 2011.

[92] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *arXiv preprint arXiv:1412.3555*, 2014.

[93] J. W. Curtis and R. W. Beard, “A graphical understanding of lyapunov-based nonlinear control,” in *IEEE Conf. Decision and Control*, vol. 2, 2002, pp. 2278–2283.

[94] S. A. Nivison and P. P. Khargonekar, “Development of a robust deep recurrent neural network controller for flight applications,” in *Proc. Amer. Control Conf.*, 2017, pp. 5336–5342.

BIOGRAPHICAL SKETCH

Scott Nivison was born in Fairbanks, Alaska. After four years of study, he received a Bachelor of Science degree in electrical and computer engineering from the University of Florida in 2009. After the completion of his degree, he began working for the Air Force Research Laboratory, Munitions Directorate at Eglin AFB, Florida. During his time at the lab, he earned a Master of Science degree in electrical and computer engineering in 2012. He began his pursuit of his Ph.D in 2013.