

**A THESIS
ON THE APPLICATION OF NEURAL NETWORK COMPUTING TO
THE CONSTRAINED FLIGHT CONTROL ALLOCATION PROBLEM**

by

Robert L. Grogan

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

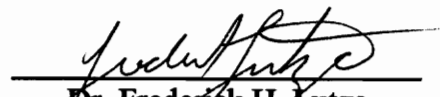
in

Aerospace Engineering

APPROVED:


Dr. Wayne C. Durham, Chairman


Dr. Krishnan Ramu


Dr. Frederick H. Lutze

May 1994
Blacksburg, Virginia

LD
5655
V855
1994
G764
C.2

**A THESIS
ON THE APPLICATION OF NEURAL NETWORK COMPUTING TO
THE CONSTRAINED FLIGHT CONTROL ALLOCATION PROBLEM**

by

Robert L. Grogan

Committee Chairman: Dr. Wayne C. Durham
Aerospace Engineering

Abstract

The feasibility of utilizing a neural network to solve the constrained flight control allocation problem is investigated for the purposes of developing guidelines for the selection of a neural network structure as a function of the control allocation problem parameters. The control allocation problem of finding the combination of several flight controls that generate a desired body axis moment without violating any control constraint is considered. Since the number of controls, which are assumed to be individually linear and constrained to specified ranges, is in general greater than the number of moments being controlled, the problem is nontrivial. Parallel investigations in direct and generalized inverse solutions have yielded a software tool (namely CAT, for Control Allocation Toolbox) to provide neural network training, testing, and comparison data. A modified backpropagation neural network architecture is utilized to train a neural network to emulate the direct allocation scheme implemented in CAT, which is optimal in terms of having the ability to attain all possible moments with respect to a given control surface configuration. Experimentally verified heuristic arguments are employed to develop guidelines for the selection of neural network configuration and parameters with respect to a general control allocation problem. The control allocation problem is shown to be well

suited for a neural network solution. Specifically, a six hidden neuron neural network is shown to have the ability to train efficiently, form an effective neural network representation of the subset of attainable moments, and independently discover the internal relationships between moments and controls. The performance of the neural network control allocator, trained on the basis of the developed guidelines, is examined for the reallocation of a seven control surface configuration representative of the F/A-18 HARV in a test maneuver flown using the original control laws of an existing flight simulator. The trained neural network is found to have good overall generalization performance, although limitations arise from the ability to obtain the resolution of the direct allocation scheme at low moment requirements. Lastly, recommendations offered include: (1) a proposed application to other unwieldy control allocation algorithms, with possible accounting for control actuator rate limitations, so that the computational superiority of the neural network could be fully realized; and (2) the exploitation of the adaptive aspects of neural network computing.

Acknowledgments

I have great appreciation for all of guidance, inspiration, and support my advisor Dr. Wayne "Bull" Durham has granted me during my graduate studies. Without all of his effort on my behalf, this thesis would not be possible. I am also grateful to him for opening my eyes to the world of aircraft and for his friendship.

With appreciation, I acknowledge Dr. Krishnan Ramu for serving on my graduate committee, being my mentor on neural networks, advisor on life, and a friend. Additionally, I extend my thanks to Dr. Frederick Lutze for serving on my graduate committee and for his helpful contributions to this thesis.

To Pam, my fiancée, I am especially thankful for her love, support, understanding, and patience during my absence. Special recognition also goes out to my mother, Charlotte Grogan, for supporting me through my college career.

Without the motivation from all of my friends, family, and professors, I certainly could not have attained my goals.

Table of Contents

Abstract **ii**

Acknowledgments **iv**

Table of Contents **v**

List of Tables **vii**

List of Illustrations **viii**

1.0 Introduction **1**

 1.1 Flight Control Allocation Background 1

 1.2 Neural Network Background 3

 1.3 Goals and Intentions 5

2.0 Problem Statement **6**

3.0 Neural Network Description and Theoretical Development **8**

 3.1 Fundamentals of Neural Network Computing 8

 3.1.1 The Neuron 8

 3.1.2 Multi-Layer Neural Networks 11

 3.1.3 Neural Network Phases of Operation 12

 3.2 The Backpropagation Learning Rule 14

 3.3 Extended Delta-Bar-Delta Backpropagation 16

4.0 Relationships Through Experiment **20**

 4.1 Consistency and Neural Network Parameter Selection 20

 4.1.1 Learning Rule Selection 21

 4.1.2 Training Set Selection 25

 4.1.3 Test Set Selection 25

 4.1.4 Choice of Squashing Function 26

4.1.5 Scaling 28

4.2 Quest for the Magic Number 28

4.2.1 Importance of the Investigation 28

4.2.2 Background 29

4.2.3 Experimental Results 34

4.2.3.1 Efficient Training 34

4.2.3.2 The Neural Network Attainable Moment Subset . . 39

4.2.3.3 Generalization Ability 44

4.3 Questions of Repeatability 46

5.0 Simulation Implementation..... 49

5.1 Overview of Approach 51

5.2 Nominal Aircraft Model 51

5.3 Training Set Generation 53

5.4 Intelligent Neural Network Configuration
and Parameter Selection 57

5.5 Test Maneuver Selection 58

5.6 Training Analysis 60

5.7 Neural Network Deployment Module 64

5.8 Maneuver Performance 65

6.0 Summary and Recommendations 75

References 79

Appendix A: Glossary of Flight Control Terminology 81

Appendix B: Glossary of Neural Network Terminology 83

Appendix C: Sample NeuroAllocator Subroutine 86

Vita 93

List of Tables

Table 4.1.1.1: EDBD Learning Coefficients 22

Table 4.1.1.2: Length of Training Sets 24

Table 4.2.2.1: The XOR Function 30

Table 5.2.1: F/A-18 HARV Control Deflection Limits 52

Table 5.3.1: Moment Coefficient Limits 55

Table 5.6.1: Grid Characteristics 60

List of Illustrations

Figure 3.1.1.1:	Neuron Model	9
Figure 3.1.1.2:	Weight and Input Signal Dot Product	10
Figure 3.1.2.1:	The Multi-Layer Neural Network Architecture	11
Figure 4.1.1.1:	Network Convergence for 3, 5, 7, and 9 Controls	23
Figure 4.1.1.2:	Network Convergence for 11, 13, and 15 Controls	23
Figure 4.1.3.1:	Facet Center Illustration	26
Figure 4.1.4.1:	Hyperbolic Tangent Squashing Function	27
Figure 4.2.2.1:	Plane Representation of XOR Function	30
Figure 4.2.2.2:	Two Input Decision Boundaries	32
Figure 4.2.3.1.1:	Weight Randomization and Undertraining Effects	36
Figure 4.2.3.1.2:	RMS Error vs. Number of Hidden Neurons for Three Controls	37
Figure 4.2.3.1.3:	RMS Error vs. Number of Hidden Neurons for Four Controls	37
Figure 4.2.3.1.4:	RMS Error vs. Number of Hidden Neurons for Five Controls	38
Figure 4.2.3.1.5:	RMS Error vs. Number of Hidden Neurons Three, Four, and Five Controls Comparison	38
Figure 4.2.3.2.1:	Neural Network Generalization for Three Controls	41
Figure 4.2.3.2.2:	The Three Control AMS and NN-AMS Comparison (Viewpoint #1)	42
Figure 4.2.3.2.3:	The Three Control AMS and NN-AMS Comparison (Viewpoint #2)	42
Figure 4.2.3.2.4:	The Three Control AMS and NN-AMS Comparison (Viewpoint #3)	43

Figure 4.2.3.2.5:	The Three Control AMS and NN-AMS Comparison (Viewpoint #4)	43
Figure 4.2.3.3.1:	Effect of Initial Weight Randomization on Neural Network Generalization for Five Controls	44
Figure 4.2.3.3.2:	Effect of Number of Hidden Neurons on Neural Network Generalization for Five Controls	45
Figure 4.2.3.3.3:	Effect of Training Set on Neural Network Generalization for Five Controls	45
Figure 4.3.1:	Five Control AMS for the F/A-18 HARV	47
Figure 4.3.2:	Five Control AMS for Random B Matrix	47
Figure 4.3.3:	RMS Error vs. Number of Hidden Neurons Five Controls Random and Real Control Allocation Parameter Comparison	48
Figure 5.1.1:	General Approach Overview	51
Figure 5.3.1:	Seven Control F/A-18 HARV Attainable Moment Subset	54
Figure 5.3.2:	Initial Input Moment Space Training Grid	55
Figure 5.3.3:	Final Input Moment Space Training Grid	56
Figure 5.4.1:	Seven Control NeuroAllocator Structure	58
Figure 5.5.1:	Desired Moments for Test Maneuver in the F/A-18 HARV AMS	59
Figure 5.6.1:	Neural Network Generalization Performance for Training Grid A	61
Figure 5.6.2:	Neural Network Generalization Performance for Training Grid B	62
Figure 5.8.1:	Rolling Moment Coefficient Time History for Test Maneuver	66
Figure 5.8.2:	Pitching Moment Coefficient Time History for Test Maneuver	67
Figure 5.8.3:	Yawing Moment Coefficient Time History for Test Maneuver	67
Figure 5.8.4:	Right Horizontal Tail Deflection	70
Figure 5.8.5:	Left Horizontal Tail Deflection	70
Figure 5.8.6:	Right Aileron Deflection	71

Figure 5.8.7: Left Aileron Deflection 71

Figure 5.8.8: Combined Rudder Deflection 72

Figure 5.8.9: Right Trailing Edge Flap Deflection 72

Figure 5.8.10: Left Trailing Edge Flap Deflection 73

Figure 5.8.11: Grid Density Resolution Limits 74

1.0 Introduction

1.1 Flight Control Allocation Background

Ailerons, elevators, and rudders first come to mind when one thinks of controlling the rotational motion of an aircraft. More specifically, we usually think of pitching moments as generated by elevators, yawing moments by a rudder, and rolling moments by ailerons. The problem of finding three control deflections to provide the three required moments is solved by a simple matrix inversion. For the next generation of super maneuverable tactical aircraft, that are projected to have as many as twenty primary flight control effectors, the problem becomes nontrivial. An example of such an aircraft is the F/A-18 HARV (High Angle of Attack Research Vehicle) which employs such nontraditional moment controllers as: independent left and right horizontal tails and rudders, leading and trailing edge flaps, spoilers, and three thrust vectoring moment generators.

The controls of the problem to be described are assumed to be strictly moment generators and are all constrained to given limits based upon physical geometry and aerodynamic considerations. The allocation, or blending, of these controls to achieve a specific aerodynamic moment requirement is the control allocation problem.

Traditionally, the generalized inverse (of which the popular pseudo-inverse is a special case) solution has been employed. It can be shown that no single generalized inverse can, for arbitrary moment demands, yield solutions that attain the maximum available moment without violating some control constraint [1]. Another method is a technique called daisy chaining. It can be shown that daisy chaining also cannot everywhere yield maximum

attainable moments, and additionally tends to demand higher control actuator rates than other methods [1].

An algorithm for a direct nonlinear solution has been developed in a series of papers (see references [1], [2], and [3]) and implemented in software named CAT, for Control Allocation Toolbox [4]. The "direct method" of solution consists of the following steps: (1) determination of the subset of attainable moments, which yields a description of the boundary containing the necessary information for control allocation, (2) identification of the point of intersection of a line in the direction of the desired moment with the bounding surface of the attainable moments, and (3) calculation of the controls that generate that intersection, with possible scaling of the controls if the desired moment is interior to the attainable moment subset (AMS).

A glossary of specific flight control terms that may be uncommon to some readers is supplied in Appendix A.

1.2 Neural Network Background

The architecture of the neural network has been derived from research on mammalian brains, specifically the biological functions of the cerebral cortex [5]. Researchers in the field of neural computing have been inspired to model human learning with computers in a manner that is analogous to the elementary functions of the biological neuron. In this artificial adaptation of the human brain three basic characteristics have been preserved by the artificial neural network. Neural networks: learn from experience, generalize from learned responses, and abstract essential patterns from inputs [6].

In the beginning neural networks were of the single layer type composed of linear neurons called perceptrons. These perceptron networks showed great promise until it was discovered the basic linearly non-separable XOR function could not be represented. The linear non-separability issue and the XOR function will be described in detail in Chapter 4 Section 4.2.2 of this thesis.

In the mid 1950's the father of perceptrons, Frank Rosenblatt, proved that if and only if a set of weighted interconnections between perceptrons exist for a problem solution there exists a perceptron training method guaranteed to converge [7]. In other words, if a particular function could be represented by perceptrons, a perceptron could be trained to learn that representation.

The landmark book, *Perceptrons*, by Minsky and Papert, published in 1969, proved several geometric theorems showing that the set of weighted interconnections between perceptrons exists only if the input set is linearly separable and labeled the perceptron as limited and uninteresting [7]. The team of Minsky and Papert additionally mentioned multiple layers might be used to overcome the limitations of perceptrons, but since no

techniques for training multi-layered perceptron networks existed, neural network research declined [6].

Neural networks emerged from obscurity in the mid 1980's when the three independent research efforts of Werbos; Parker; and Rummelhart, Hinton, and Williams developed techniques for training multi-layered neural networks [6]. One such technique is called backpropagation, which continues to be one of the most successful and widely understood neural network training algorithms. Since the rebirth of neural networks many neural network algorithms and structures have been developed. Each neural network type has advantages and limitations associated with specific applications.

Neural networks have been successfully used in a wide variety of fields. Some example applications are: power conversion, medical ultrasound imaging, noise filtering, terrain mapping, character recognition, the effect of solar activity on orbit prediction, image and data compression, stock market prediction, and the determination of satellite orientation by star identification. Neurobiologists and neuropsychologists were the first to attempt to develop a model of human brain learning [6], but it was only a matter of time for the engineers to realize their potential. Only recently have neural networks gained attention in the area of aerospace guidance, navigation, and control. Of particular interest, neural network theory has been used for selecting optimal gain schedules for flight control [8], feedback linearization in aircraft control [9], and identification and control of aircraft dynamics [10].

A glossary of neural network terminology is supplied in Appendix B.

1.3 Goals and Intentions

The goal of this research is to apply neural network computing to the solution of the constrained flight control allocation problem. Considering that much of the available literature on neural network development is very implementation oriented, the selection of the neural network structure, parameters, and learning rule is usually based entirely on trial and error experience. The first objective of this investigation is to replace the trial and error approach and provide guidelines for the intelligent selection of the defining neural network descriptors. The scope of this selection is limited to a modified backpropagation learning rule, based on its popularity and accessibility. Secondly, based on the guidelines, a neural network will be created and trained to emulate the direct method of control allocation. Lastly, the trained neural network will be installed into a module, that could be installed in an existing flight simulator, and performance will be evaluated on the reallocation of seven control surfaces of the F/A-18 HARV in a simulation of a series of two-roll-reversals test maneuver.

2.0 Problem Statement

An m -dimensional control space is considered, $u \in \mathbb{R}^m$. The controls are constrained to minimum and maximum values, defined by the subset Ω :

$$\Omega = \{u \in \mathbb{R}^m \mid u_{i_{\text{Min}}} \leq u_i \leq u_{i_{\text{Max}}}\} \subset \mathbb{R}^m \quad (2.1)$$

The subset of controls which lie on the boundary of Ω , $\partial(\Omega)$, are denoted by u^* .

$$u^* \in \partial(\Omega) \quad (2.2)$$

These controls generate moments through a mapping B onto n -dimensional moment space through a linear matrix multiplication of u ,

$$B : \mathbb{R}^m \rightarrow \mathbb{R}^n \quad (2.3)$$

$$Bu = M \quad (2.4)$$

where $m > n$. B is the control effectiveness matrix with respect to the moments. A requirement that the controls be independent is met if every $n \times n$ partition of B is non-singular.

Denote by Φ the image of Ω in \mathbb{R}^n , $\Phi \subset \mathbb{R}^n$. The subset Φ therefore represents all the moments that are attainable within the constraints of the controls.

Moments which lie on the boundary of Φ , $\partial(\Phi)$, are denoted by an asterisk:

$$M^* \in \partial(\Phi) \quad (2.5)$$

A unit vector in the direction of \mathbf{M} will be denoted by $\hat{\mathbf{M}}$,

$$\hat{\mathbf{M}} = \frac{\mathbf{M}}{|\mathbf{M}|} \quad (2.6)$$

The control allocation problem is defined as follows: given \mathbf{B} , Ω , and some desired moment \mathbf{M}_d , determine the controls $\mathbf{u} \in \Omega$ that generate that moment for the largest possible magnitude of \mathbf{m} in the direction $\hat{\mathbf{M}}_d$. That is, we desire a rule for allocating the controls that generates the maximum moment in a given ratio (direction) without exceeding the constraints on the controls.

3.0 Neural Network Description and Theoretical Development

The biological influence is evident in the terminology used for the various elements of neural networks. Since neural network computing is a relatively new and emerging field, much of the terminology and notation is non-standard. For the purposes of this work, the most common and correct terms are used. The NeuralWorks [7] commercial software package is utilized for neural network development, training, and testing.

3.1 Fundamentals of Neural Network Computing

3.1.1 The Neuron

Neural networks are composed of highly interconnected processing elements or neurons. The structure of every neuron in a neural network is exactly alike and has three essential characteristics. Each neuron in a neural network has:

- (1) Multiple inputs and one output signal with a central processing unit.
- (2) Decision making capability due to its logic with threshold firing.
- (3) Self stabilization capability due to inhibitory response.

The basic structure of a neuron is quite simple as illustrated in Figure 3.1.1.1.

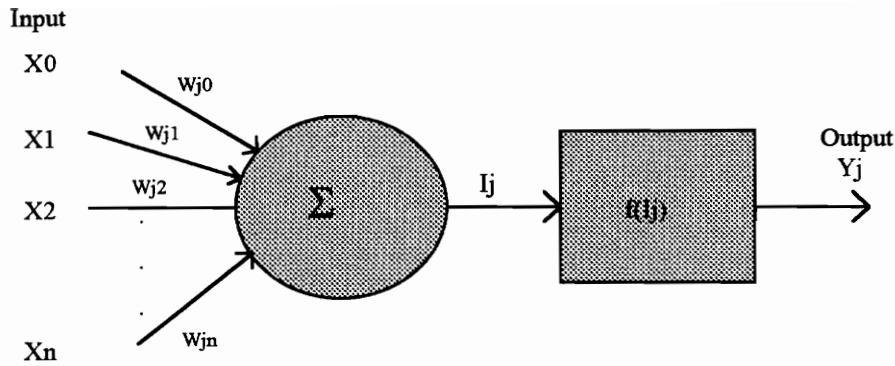


Figure 3.1.1.1: Neuron Model

The model for a neuron is given by the equation

$$I_j = \left[\sum_{i=0}^n w_j x_i \right]$$

$$y = f(I_j)$$
(3.1.1.1)

where each input signal, x_i , to the neuron has a weight, w_j , so that the total input is the weighted sum of all input signals from connecting neurons. In vector notation, the total weighted input signal is the dot product of the weight and input vectors or $I = \mathbf{w} \cdot \mathbf{x}$

where

$$\mathbf{w} = [w_0 \quad w_1 \quad w_2 \quad \dots \quad w_n]$$

$$\mathbf{x} = [x_0 \quad x_1 \quad x_2 \quad \dots \quad x_n]^T$$
(3.1.1.2)

The dot product is equivalent to the projection of the weight vector onto the input vector given by $\mathbf{w} \cdot \mathbf{x} = |\mathbf{w}| |\mathbf{x}| \cos \alpha$, where α is the angle between the input and weight vectors as shown in Figure 3.1.1.2.

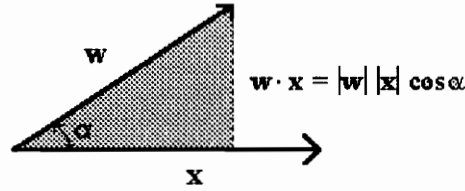


Figure 3.1.1.2: Weight and Input Signal Dot Product

This projection will be greatest in magnitude when the two vectors are most closely aligned [5]. Therefore, neurons having weight vectors nearest in direction to a given input signal will fire and produce a larger output than neurons whose weight vectors are not in the direction of the input vector. The squashing function is then applied resulting in the neuron output.

The squashing function (f) operates on the weighted summation of inputs of each neuron (I_j) and provides the necessary nonlinearity in the system. It can be shown that without this nonlinear squashing function a two layer neural network is exactly equivalent to a single layer network having a weight matrix equal to the product of the two weight matrices [6]. Thus, the squashing function is necessary to extend the capability of the neural network to map nonlinear functions. The squashing function can be a threshold function or a continuous function of the input. Some common choices for the squashing function include the: sigmoid, sine, linear, ramp, Gaussian, step, and hyperbolic tangent functions. The output from the nonlinear squashing function is the output (y) from the neuron. In a multi-layer neural network this output becomes the weighted input to the next layer of neurons.

3.1.2 Multi-layer Neural Networks

The basic neuron itself is not exciting, but the power of neural networks stems from the manner in which neurons are interconnected into networks. Neurons are grouped into layers, and each layer is interconnected to the adjacent layer by connection weights.

There is some confusion over the convention of how many layers a neural network possesses. For example, some refer to a neural network with an input, one hidden, and an output layer as a three layer network, others refer to the same neural network as a two layer network since it has two layers of weights. Adopting the later convention, the neural network used in this study has two layers (see Figure 3.1.2.1), that is, one layer of weights connecting the input and the one hidden layer and one layer of weights connecting the one hidden layer and the output layer. It has been shown that only one hidden layer is necessary to map any arbitrary nonlinear relationship [7].

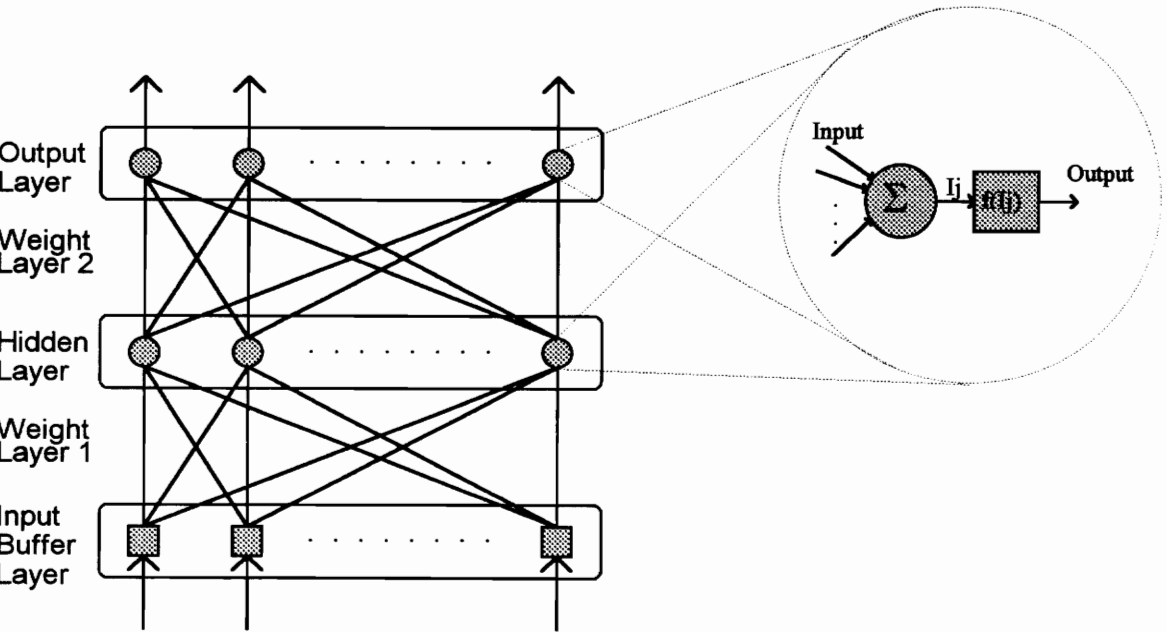


Figure 3.1.2.1: The Multi-Layer Neural Network Architecture

The input buffer layer is where data is presented to the network. The neurons in the input layer act as distribution nodes and have no summing or squashing function. The next layer of neurons is referred to as the hidden layer. The output of the input buffer layer becomes the weighted input to each neuron in the hidden layer. The squashing function is applied to the weighted sum of inputs at each hidden neuron producing an output that becomes the weighted input to each neuron in the output layer. The weighted sum of inputs to each output neuron is passed through a squashing function producing the network output.

Each neuron in the multi-layered neural network has the equation:

$$x_j^{[\ell]} = f\left\{\sum_i (w_{ji}^{[\ell]} \cdot x_i^{[\ell-1]})\right\} \quad (3.1.2.1)$$

$$x_j^{[\ell]} = f\{I_j^{[\ell]}\}$$

where

$\ell = 1, 2, \dots, L$

$i = 1, 2, \dots, N^{[\ell]}$

$N^{[\ell]} = \text{number of neurons in layer } \ell$

$L = \text{number of layers in the neural network}$

$x_j^{[\ell]} = \text{current output of the } j\text{th neuron in layer } \ell$

$w_{ji}^{[\ell]} = \text{weight joining the } i\text{th neuron in layer } [\ell - 1] \text{ to the } j\text{th neuron in layer } \ell$

$I_j^{[\ell]} = \text{weighted sum of inputs to the } j\text{th neuron in layer } \ell$

3.1.3 Neural Network Phases of Operation

A neural network is trained to discover relationships by learning from examples. If no desired output is specified, training is called unsupervised. During unsupervised learning a neural network organizes to respond to certain inputs. Supervised training requires a

group of input and desired output pairs. Each input-output pair is a training pattern and the collection of all training patterns is a training set. If the desired output presented to the neural network is different from the input, the neural network is called hetero-associative. An auto-associative neural network has equal input-output pairs. In the training phase, the neural network learns by adjusting the weights in the layers to minimize an error function to achieve the desired input to output mapping. Training is an iterative process requiring the presentation of many training patterns. The learning rule determines how the weights adapt in response to a training pattern. Learning rules typically have parameters that govern how learning progresses. These parameters may or may not change during training depending on the specific learning rule.

The neural network used in this text is of the multi-layer feed-forward hetero-associative type. Training is supervised and controlled by the backpropagation learning rule. In addition, an extension to backpropagation, called extended delta-bar-delta, that features time varying learning parameters is used.

During the test phase, previously unseen inputs, along with the corresponding desired outputs, are presented to the neural network. Unlike the training phase, weights are not adjusted, but the network output is compared with the desired output for the evaluation of the performance of the trained neural network. Testing patterns in the test set are typically different from the training patterns, in order to evaluate the generalization performance of the neural network and not the ability to memorize the training data.

The recall phase of operation is similar to the testing phase except the desired output is unknown. This is the phase a neural network operates in during implementation. A trained neural network is presented with an input and determines the output based upon the learned relationship.

3.2 The Backpropagation Learning Rule

The backpropagation learning rule is an iterative training procedure that involves the minimization of the total squared error between the actual and desired outputs of a training set, with respect to the neural network weights. This minimization is accomplished by means of the method of steepest descent.

E_T is the error function defined as the sum of squared error over all training patterns and is given by

$$E_T = \frac{1}{2} \sum_{p=1}^{NP} \sum_{k=1}^{NO} \left\{ (d_{kp} - o_{kp})^2 \right\} \quad (3.2.1)$$

where d_{kp} is the desired output and o_{kp} is the actual output for the k^{th} output neuron and the p^{th} training pattern, NO is the number of output neurons, and NP is the number of training patterns in the training set. Note that E_T is the total error over all training patterns. In this investigation, weights are updated once every training epoch, where one epoch is one full pass through all training patterns in the training set. This method of accumulating weight changes over an entire epoch is called cumulative backpropagation. In cumulative backpropagation a composite error function, defined as the sum of all of the individual error functions, is minimized.

According to the method of steepest descent, the update of the weights is proportional to the negative gradient of the error function with respect to the weights and is defined as:

$$\Delta w_{ji}^{[\ell]} = -\eta \left(\frac{\partial E_T}{\partial w_{ji}^{[\ell]}} \right) \quad \forall j, i, \ell \quad (3.2.2)$$

where $\Delta w_{ji}^{[\ell]}$ is the change in the weight connecting neuron i to neuron j in layer ℓ , η is the learning rate, and $\ell = 1, 2, \dots, L$, where L is the number of layers and in this investigation $L=2$.

Weights are adjusted to reduce the error by the equation:

$$\Delta w_{ji}^{[\ell]}(t+1) = \eta \sum_{p=1}^{NP} o_{ip}^{[\ell-1]} \delta_{jp}^{[\ell]} + \alpha \Delta w_{ji}^{[\ell]}(t) \quad \forall j, i, \ell \quad (3.2.3)$$

where

$$\delta_{jp}^{[\ell]} = \begin{cases} f'(I_{jp})(d_{jp} - o_{jp}) & \text{if } j \text{ is an output neuron } (\ell = L = 2) \\ f'(I_{jp}) \sum_{k=1}^{NO} w_{kj} \delta_{kp} & \text{if } j \text{ is a hidden neuron } (\ell = 1) \end{cases} \quad (3.2.4)$$

where $f'(I_{jp})$ is the derivative of the squashing function associated with neuron j .

For the hyperbolic tangent squashing function, the selection of which will be detailed later, the above equation becomes:

$$\delta_{jp}^{[\ell]} = \begin{cases} (d_{jp} - o_{jp}) o_{jp} (1 - o_{jp}) & \text{if } j \text{ is an output neuron } (\ell = L = 2) \\ o_{jp} (1 - o_{jp}) \sum_{k=1}^{NO} w_{kj} \delta_{kp} & \text{if } j \text{ is a hidden neuron } (\ell = 1) \end{cases} \quad (3.2.5)$$

Additionally, o_i is the output of neuron i , δ_j is the local error associated with neuron j , d_j is the desired output of neuron j , and t is the training epoch. $w_{ji}^{[\ell]}(0)$, the initial network weights, are usually chosen to be small random numbers. α is the momentum learning parameter. This term, which is proportional to the amount of the previous weight change, is included to add stability by limiting large oscillations in weight updates and to avoid local minima in the error surface [11].

The backpropagation neural network is trained according to the following steps:

- i. A training vector is selected from the training set and is applied as the network input.
- ii. The output of each neuron in the output and hidden layers is calculated using equation (3.1.2.1).
- iii. The local errors given by equation (3.2.4) are calculated and stored for each neuron in the output and hidden layers and accumulated over the entire training set.
- iv. Delta weights are found according to equation (3.2.3) to minimize the error according to the method of steepest descent.
- v. Weights are updated by adding delta weights to the corresponding weights of the previous update.
- vi. RMS (root-mean-square) error between the network output and desired output is calculated.
- vii. The above process is repeated for a specified number of training epochs or until convergence to a specified RMS error for the entire training set.

3.3 Extended Delta-Bar-Delta Backpropagation

The extended delta-bar-delta (EDBD) was proposed to compensate for some of the shortcomings of the backpropagation algorithm. Specifically, this technique applies a heuristic approach to improve the rate of convergence of the steepest descent procedure. This heuristic technique accounts for variations in the error surface by assigning individual time varying learning and momentum rates to each weighted connection in the neural network.

The heuristic approach is taken to automatically adjust the learning and momentum rates for each weight connection and are updated along with the weights for each training epoch.

Instead of performing a steepest descent search, weights are updated on the basis of the gradient of the error with respect to the weight itself, and on an estimate of the curvature of the local error surface [7].

A fully detailed discussion of the EDBD algorithm and supporting equation derivations can be located in Reference [7]. The following notation is used in the amending equations to the standard backpropagation algorithm:

$E(t)$ local error of a particular neuron

$w(t)$ value of a connecting weight

$\Delta w(t)$ weight update

$\eta(t)$ connection learning rate

$\Delta \eta(t)$ connection learning rate update

$\alpha(t)$ connection momentum rate

$\Delta \alpha(t)$ connection momentum rate update

$\beta(t)$ gradient component of weight change

$\beta^*(t)$ weighted, exponential average of previous gradient components

and are all evaluated at training epoch, t .

The constant learning parameters used in the EDBD algorithm are:

θ	convex weighting factor
K_η	constant learning rate scale factor
K_α	constant momentum rate scale factor
G_η	constant learning rate exponential factor
G_α	constant momentum rate exponential factor
J_η	constant learning rate decrement factor
J_α	constant momentum rate decrement factor
η_{\max}	learning rate upper bound
α_{\max}	momentum rate upper bound
λ	recovery tolerance parameter

The contribution of the error associated with a particular weight is given by:

$$\beta(t) = \frac{\partial E(t)}{\partial w(t)} \quad (3.3.1)$$

The standard backpropagation learning rule for the update of weights is of the form:

$$\Delta w(t+1) = \eta \beta(t) + \alpha \Delta w(t) \quad (3.3.2)$$

where η and α are the specified fixed constant learning and momentum rates, respectively.

This rule is modified in the EDBD approach and becomes:

$$\Delta w(t+1) = \eta(t) \beta(t) + \alpha(t) \Delta w(t) \quad (3.3.3)$$

The heuristics for updating the learning and momentum rates employ a weighted average of the error gradient components governed by

$$\beta^*(t) = (1 - \theta)[\beta(t) + \theta\beta(t-1)] \quad (3.3.4)$$

The update rules for the learning and momentum rates are defined by the following equations:

$$\Delta\eta(t) = \begin{cases} K_\eta e^{-G_\eta|\beta^*(t)|} & \text{if } \beta^*(t-1)\beta(t) > 0 \\ -J_\eta\eta(t) & \text{if } \beta^*(t-1)\beta(t) < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\Delta\alpha(t) = \begin{cases} K_\alpha e^{-G_\alpha|\beta^*(t)|} & \text{if } \beta^*(t-1)\beta(t) > 0 \\ -J_\alpha\alpha(t) & \text{if } \beta^*(t-1)\beta(t) < 0 \\ 0 & \text{otherwise} \end{cases} \quad (3.3.5)$$

Further measures are taken to prevent large oscillations in weight updates. These include limits placed on the learning parameters according to:

$$\begin{aligned} \eta(t) &\leq \eta_{\max} \\ \alpha(t) &\leq \alpha_{\max} \end{aligned} \quad (3.3.6)$$

and the use of a best weight recovery feature, such that, if the current error is less than the minimum error, weights are saved as the current best. The tolerance parameter (λ) is used to control recovery such that, if the current error is greater than the recovery tolerance times the minimum error, learning parameters are decremented and weights revert to the saved best according to:

$$E[t] > E_{\min} \lambda \quad (3.3.7)$$

4.0 Relationships Through Experiment

Various neural network configurations were explored in order to gain understanding of the relationship between the control allocation problem and the neural network structure. The goal of the following experiments is to provide an implementor with reasonable guidelines for the development of a neural network control allocator.

It is tempting to say that the so-called optimal neural network configuration was sought with regard to the control allocation problem at hand but, this would be inaccurate. The current theoretical foundation on which neural networks were built does not provide any rationale behind the selection of many of the neural network parameters. Experimental techniques were used to attempt to determine the cause and effect relationships that exist.

By no means will this investigation exhaust the many choices in selecting neural network configurations, but will address the parameters with the largest impact on performance with regard to control allocation. Additionally, most of the experimental procedures may be used in the determination of neural network configurations for generic neural network applications.

4.1 Consistency and Neural Network Parameter Selection

Consistency and repeatability must be addressed for an experiment to be representative of any arbitrary combination of controls, the effectiveness of those controls, and their limits. Thus, allocation problems, training data, and neural network parameters were chosen carefully. Much care was taken to ensure that while the parameter in question was varied in a controlled manner all other independent variables remained fixed.

4.1.1 Learning Rule Selection

There was great difficulty in determining a set of learning coefficients to use, so that, training remained consistent with the addition of controls. The set of learning coefficients that provided a smoothly converging RMS error for one allocation problem were generally not the same for all allocation problems. To alleviate this dilemma, this investigation uses the extended delta-bar-delta (EDBD) learning rule. As described in Chapter 3, this learning rule features individual time varying learning coefficients for each connection in the network. The learning coefficients are updated based on the curvature of the local error surface [7]. Since this method relied on an algorithm that was common to all experiments there was consistency for all allocation problems. Additionally, EDBD increases the convergence rate of the backpropagation steepest descent error minimization. This was particularly desirable when experiments involved the training of hundreds of different neural network configurations.

The EDBD learning rule used various constant coefficients, listed in Table 4.1.1.1, to specify how the learning coefficients were adjusted during training. An upper bound was specified to be $\eta_{\max}=2$ on the learning rate and $\alpha_{\max}=1$ on the momentum rate. A tolerance parameter of $\lambda = 1.5$ was used to control recovery such that, if the current error was greater than the recovery tolerance times the minimum error, weights reverted to the saved best and the learning parameters were decremented. These coefficients were used throughout the entire investigation. There was little to guide the selection of learning coefficients but, unlike the selection of constant learning and momentum rates in standard backpropagation, convergence was fairly insensitive to changes, on the order of 50%, in the EDBD parameters. This choice of constant coefficients provided efficient and smoothly converging training for up to a fifteen control allocation problem. This training

was characterized by a fairly rapid and smooth decrease in the RMS error, the avoidance of wild jumps in weight space, and even distributions of relatively small network weights during training. The exact function of each coefficient is described in detail in Chapter 3.

Table 4.1.1.1: EDBD Learning Coefficients

EDBD Coefficient		Constant Value
Learning Rate Scale Factor	K_{η}	0.02
Momentum Rate Scale Factor	K_{α}	0.01
Learning Rate Exponential Factor	G_{η}	1.0
Momentum Rate Exponential Factor	G_{α}	1.0
Learning Rate Decrement Factor	J_{η}	0.02
Momentum Rate Decrement Factor	J_{α}	0.01
Convex Weighting Factor	θ	0.7

The smooth convergence provided by these constant coefficients is demonstrated in Figure 4.1.1.1 and Figure 4.1.1.2. Small bumps of the RMS error occurred as thresholds were met which determined whether the learning parameters were incremented or decremented. The figures depict the reduction in the RMS error of the output layer for allocation problems from three to fifteen controls.

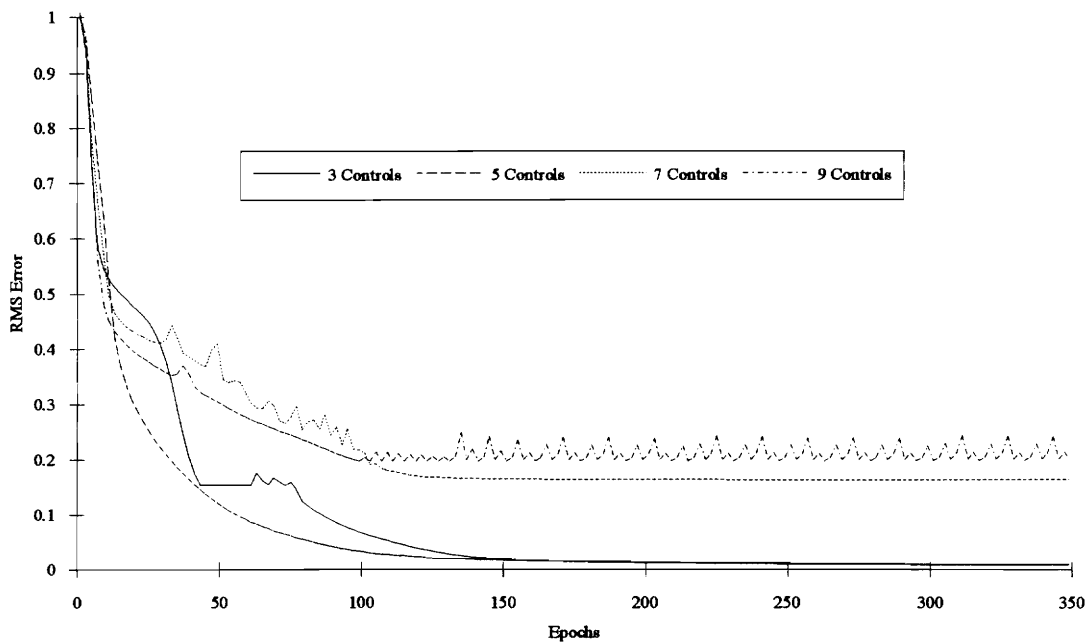


Figure 4.1.1.1: Network Convergence for 3, 5, 7, and 9 Controls

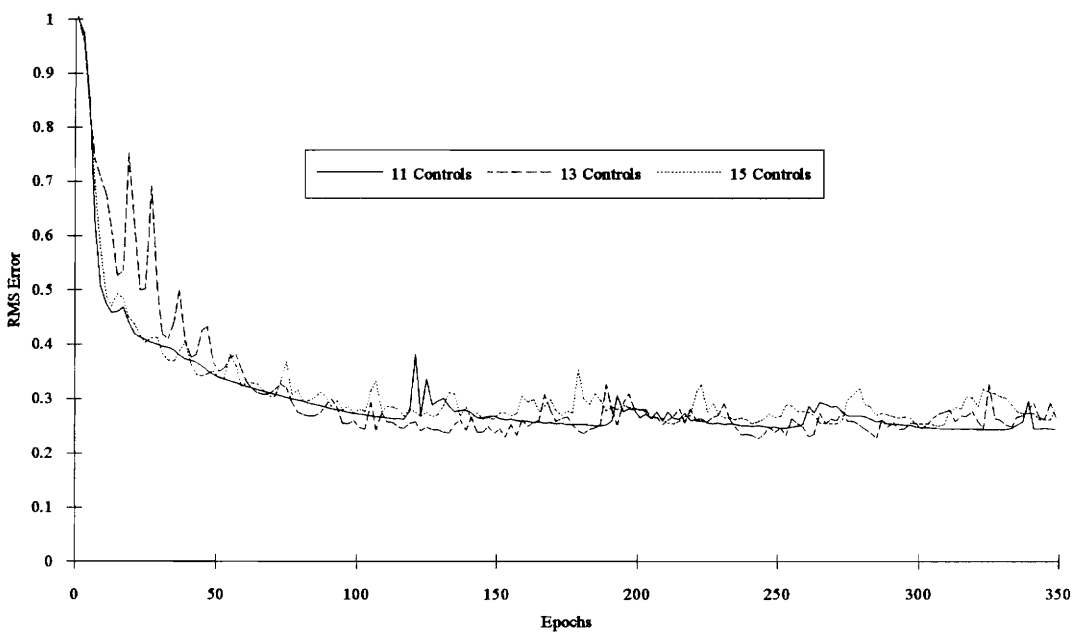


Figure 4.1.1.2: Network Convergence for 11, 13, and 15 Controls

Six hidden neurons were used for each neural network, for reasons to be detailed later. Before training, all weights in the network were randomized using the same random number seed, providing a normal weight distribution between +0.1 and -0.1. This method of consistent weight randomization was used throughout the entire investigation before each training phase and upon the addition of a new output or hidden neuron. The control deflection limits were set to -1 to +1 for each allocation problem. The control effectiveness B matrices were filled with random numbers. The training sets consisted of the vertices of the AMS. Table 4.1.1.2 contains the length of each training set or number of vertices for each allocation problem.

Table 4.1.1.2: Length of Training Sets

Number of Controls	Length of Training Set	Iterations for 350 Epochs
3	8	2800
5	22	7700
7	44	15400
9	74	25900
11	112	39200
13	158	55300
15	212	74200

4.1.2 Training Set Selection

A representative feature of any control allocation problem is the number of nodes of its attainable moment subset. Nodes are the points generated by placing all m controls at one or the other of their constraining values, and are viewed as the vertices on the bounding surface of the AMS. The number of vertices is a feature that is independent of control effectiveness, relies solely on the number of controls, and is given by the equation $m(m-1)+2$. The vertices were chosen to be the points where training data was generated. Therefore, the number of vertices was equivalent to the number of training patterns included in a training set for each allocation problem. The control deflection limits were set to -1 to +1 for each random allocation problem.

The training set was presented to the EDBD neural network sequentially. Weights and learning parameters were updated every epoch, which was one pass through all patterns in the training set. In terms of the control allocation problem this means that weights and learning parameters were updated only after the network had seen the entire training set or AMS. This selection of sequential presentation and epoch was particularly useful to maintain consistency for all allocation problems and for analysis.

4.1.3 Test Set Selection

A test set was selected which would challenge the ability of a neural network to generalize. For this reason, the test set contained the centers of each facet of the AMS. The test phase consisted of presenting the moment vectors to each facet center along with the desired controls, calculated by CAT, and evaluating the performance of the trained neural network. A facet is any face that lies on the boundary of the AMS, where a face is generated in control space by placing all but two of the m controls at either of the

constraints and allowing the two to vary [2]. Faces in the AMS are the images of the facets in control space and the facets are the faces that lie on the boundary. As illustrated in Figure 4.1.3.1, facets are uniquely described by four vertices. An AMS has $m(m-1)$ facets where each facet center is calculated from two of the describing vertex moment vectors on opposite corners as:

$$\bar{m}_c = \frac{\bar{m}_1 + \bar{m}_2}{2}$$

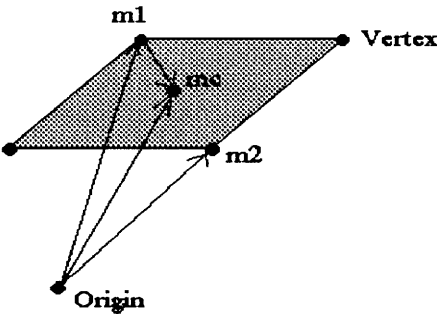


Figure 4.1.3.1: Facet Center Illustration

4.1.4 Choice of Squashing Function

The squashing function chosen for this investigation was the hyperbolic tangent. The hyperbolic tangent function is defined as

$$f(x) = \tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

and is shown in Figure 4.1.4.1.

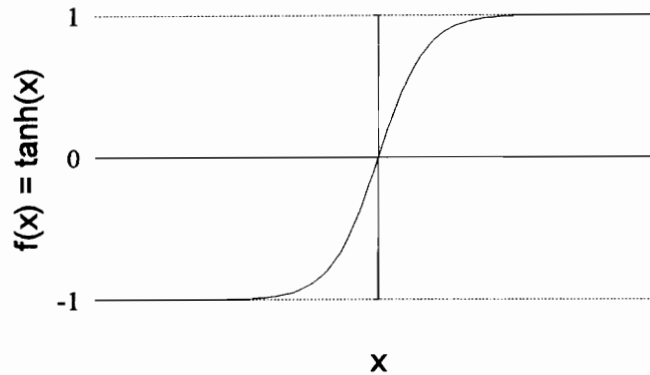


Figure 4.1.4.1: Hyperbolic Tangent Squashing Function

The hyperbolic tangent function was chosen from both intuitive and theoretical viewpoints. Intuitively, since the controls for almost every allocation problem encountered will have bipolar deflection limits, which may or may not be symmetric about zero, lends itself to the hyperbolic tangent function having an output range of -1 to +1. Giving equivalent weighting to high and low end outputs eliminates the bias, in training, experienced by squashing functions with output ranges of 0 to 1 [7]. This is particularly important in control allocation since it is desirable to use the least amount of control possible without sacrificing performance. A bias toward higher outputs would result in larger control deflections which, in turn, results in more control effort and higher actuator rates.

A more theoretical justification for the use of a hyperbolic tangent squashing function can be found in Reference [12]. Since it is believed the choice of squashing function does not significantly effect the test performance of a trained neural network, the impact on training performance is of more concern. The hyperbolic tangent squashing function is shown to have properties which result in more efficient neural network training.

4.1.5 Scaling

The neural networks used both in the experimental and implementation phases of this investigation used the MinMax Table feature of the NeuralWorks software to scale both the input and output to the neural network. This software facility scans the data sets and records the highs and lows of each data field enabling the input and output to be scaled for presentation to the neural network and the results to be de-scaled for output. For every training and testing procedure all data was scaled to a range of -1 to +1. This scaling was appropriate to prevent the saturation of the hyperbolic tangent activation function.

4.2 Quest for the Magic Number

4.2.1 Importance of the Investigation

There is no other neural network feature that effects the aspects of multi-layered neural network development and implementation as the choice of hidden neurons. The number of hidden neurons determines the number of weights and equations that ultimately will represent the nonlinear functional relationship desired. Thus, the focus shifts on how many hidden neurons were necessary to provide an adequate nonlinear map for control allocation.

Not only do the number of hidden neurons effect the nonlinear mapping performance of a neural network, but the additional weights and equations of each hidden neuron impact training time. More important is the increase of computations required for

implementation. This is of great concern in a system such as an aircraft where controls must be allocated in real time.

The best choice of hidden neurons should provide acceptable performance for any number of controls with the minimum number of hidden neurons. Therefore, the number of hidden neurons selected for the hidden layer will from this point on be referred to as *the Magic Number*.

4.2.2 Background

The notion of including a hidden layer of neurons developed out of a need for an additional feature space to solve problems that could not be separated linearly. This linear separability issue arose when the pioneers of neural networks attempted to solve the exclusive OR or XOR problem. In fact, neural networks were all but abandoned for two decades after it was proven that the single layer perceptron neural network could solve only those problems that were linearly separable. Discovering exactly why a single layer neural network has this limitation will allow insight into how neural networks and hidden layers function.

A problem can be linearly separated into categories if all given outputs in one category lie on one side of a hyperplane in input space and all those outputs in another category lie on the other side [7]. For the two inputs of the XOR function, shown in Table 4.2.2.1, the hyperplane is a line.

Table 4.2.2.1: The XOR Function

Point	Input X_1	Input X_2	Output
A	0	0	0
B	1	0	1
C	0	1	1
D	1	1	0

The equation of this line for two inputs is given by the dot product of the weight vector with the input vector.

$$\mathbf{w} \cdot \mathbf{x} = w_0 + w_1x_1 + w_2x_2 = 0$$

This line defines the decision boundary in the two dimensional input space, as shown in Figure 4.2.2.3, for the XOR function.

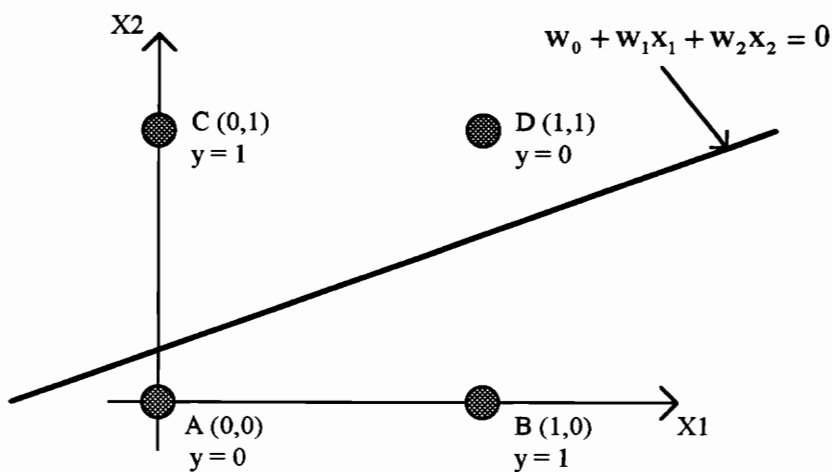


Figure 4.2.2.1: Plane Representation of XOR Function

On one side of the decision boundary the output is one and on the other side it is zero. Therefore, to represent the XOR function given in Table 4.2.2.1, points A and D must be separated from points B and C by this decision boundary. The XOR function is not linearly separable since no straight line can be drawn to separate the points in this manner. Therefore, one layer perceptron neural networks cannot represent linearly non-separable functions such as the XOR.

The linear separability limitation can be overcome by the addition of an additional layer of neurons, the so-called hidden layer. A two layer neural network can represent nonlinear mappings in convex open or closed regions [6]. A region is convex if and only if any two points in that region can be connected by a straight line not leaving that region. A boundary encloses all points in a closed region. This is important since it is well known that the AMS for any aircraft will always be both closed and convex, if all of the controls are independent.

In Reference [6], it is shown that a convex polygon of any desired shape can be created by adding hidden neurons to a neural network with two inputs. Thus, a neural network with a layer of hidden neurons can be used to solve the XOR problem. Figure 4.2.2.2 illustrates the use of four hidden neurons to form a rectangular decision boundary that separates points B and C from points A and D, separating the one outputs from the zero outputs, respectively. Each hidden neuron provides a straight line decision boundary given by the equation:

$$\bar{L} = w\bar{x}$$

$$\begin{Bmatrix} L_1 \\ L_2 \\ L_3 \\ L_4 \end{Bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \end{Bmatrix}$$

where w is the weight matrix between the input and hidden layers

\bar{L} is the vector of decision boundary lines

\bar{x} is the input vector

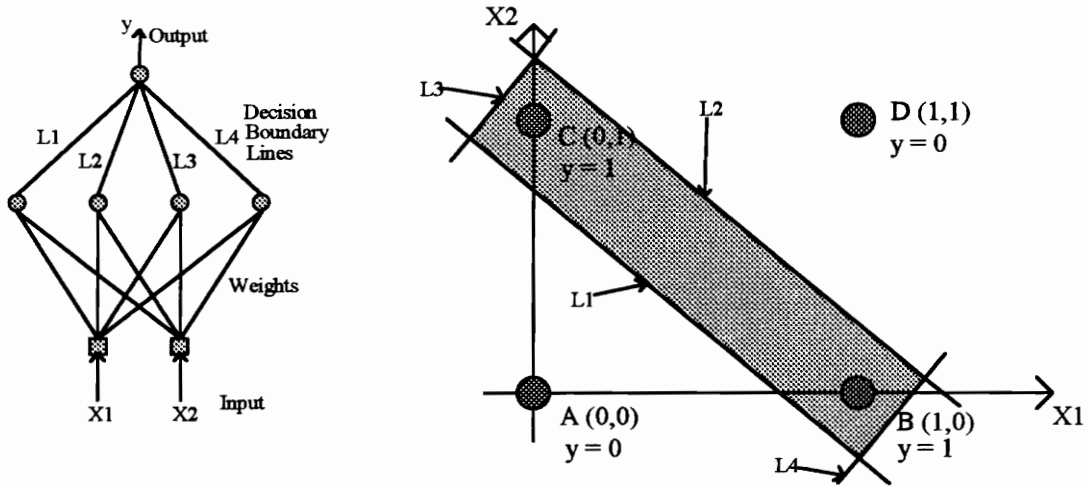


Figure 4.2.2.2: Two Input Decision Boundaries

Extending this notion into three dimensional input moment space each hidden neuron can be thought of creating a plane decision boundary. Thus, the ultimate goal is to determine how many hidden neurons are needed to enclose the three dimensional convex and closed region called the attainable moment subset.

A theorem proved by Russian mathematician Andrei Kolmogorow and applied in the field of neural networks by Robert Hecht-Nielsen guarantees the two layer neural network can solve all linearly non-separable problems [5]. The theorem states that any n dimensional

vector may be exactly mapped to another m dimensional vector by a network with n neurons in the input layer, m neurons in the output layer, and $2n+1$ neurons in the hidden layer [5]. This means the number of neurons to include in the hidden layer is independent of the number of controls. Moreover, since the number of inputs will always be the three components of the desired moment vector an upper limit of $2(3)+1 = 7$ may be set on the number of neurons in the hidden layer. This two layer, seven hidden neuron neural network can theoretically map any three dimensional moment vector to m dimensional control space.

A minimum number of hidden neurons is not addressed in this theorem. There may exist a smaller more efficient network that also may perform the desired mapping. Furthermore, the actual training and structure of the neural network are not provided.

It is conjectured [13] that the number of hidden neurons may be reduced to six. As previously shown, each hidden neuron provides a plane decision boundary in moment space. Therefore any AMS may be encased by a six sided prism, each neuron in the hidden layer creating one side. One may also purport that a prism of four or five sides may enclose an AMS, but it is thought that this may leave large peaks in the error surface, thus increasing the likelihood of becoming trapped in a local minimum [13].

The next section endeavors to experimentally demonstrate that six hidden neurons are indeed sufficient to solve any control allocation problem and is *the magic number* of hidden neurons.

4.2.3 *Experimental Results*

In order to verify that the magic number of hidden neurons for any general control allocation problem is six, three distinct experiments were performed. The three experiments were designed to:

1. measure the ability of the neural network to train efficiently using six hidden neurons.
2. test the theory that decision boundaries form an effective neural network representation of the AMS.
3. test the ability of the neural network to generalize the internal relationship between moments and controls.

Upon completion of the experiments, controlled tests were performed to verify that the results would hold for a general control allocation problem. This verification procedure was followed since the experiments used randomly generated data for the control effectiveness data and control limits of ± 1 . It was particularly important that the results would hold for the real F/A-18 HARV data that was used in the implementation phase of the investigation.

4.2.3.1 *Efficient Training*

Neural networks were trained and tested on the vertices of the AMS for three, four, and five control allocation problems. For each allocation problem the number of training iterations was fixed and the RMS error achieved for that number of iterations and neurons in the hidden layer was noted. For a given number of controls a neural network that can

train efficiently for a chosen number of hidden neurons was denoted by a range of minimum RMS error, provided that the neural network was adequately trained.

It was not entirely clear when a neural network graduated from the undertrained stage to the stage when a stable range of minimum RMS error developed and conclusions could be drawn. The effects of weight randomization before training can account for this dilemma. The effects of weight randomization were even more gross as neurons and therefore more weights were added to the network. The addition of neurons occurred both when neurons were added to the hidden layer and output neurons were included for additional controls.

An attempt was made to limit weight randomization effects so that results describing the relationship between the neural network structure and control allocation problem would not be corrupted. The effect of initial weight randomization on undertraining is shown in Figure 4.2.3.1.1. This neural network was trained using the aircraft data for the F/A-18 HARV with five controls. The six lines represent the RMS error achieved for three initial weight randomizations trained for 90 epochs and the same three initial weight randomizations trained for 136 epochs for three to nine hidden neurons. As expected, no significant increase in performance was found for more than six hidden neurons when the neural network was adequately trained at 136 epochs and a stable range of minimum RMS error was allowed to develop. Note, that for the neural networks trained for 90 epochs, no conclusions could be drawn since the neural network was undertrained and a stable range of minimum RMS error had not formed. The effect was exaggerated as neurons were added. Note the greater deviations for using eight and nine hidden neurons. Beyond nine hidden neurons the neural network trained for 136 epochs began to lose the stable range of minimum RMS error. This loss was irrelevant since a stable range of minimum RMS error was seen for fewer hidden neurons and specifically for six hidden neurons.

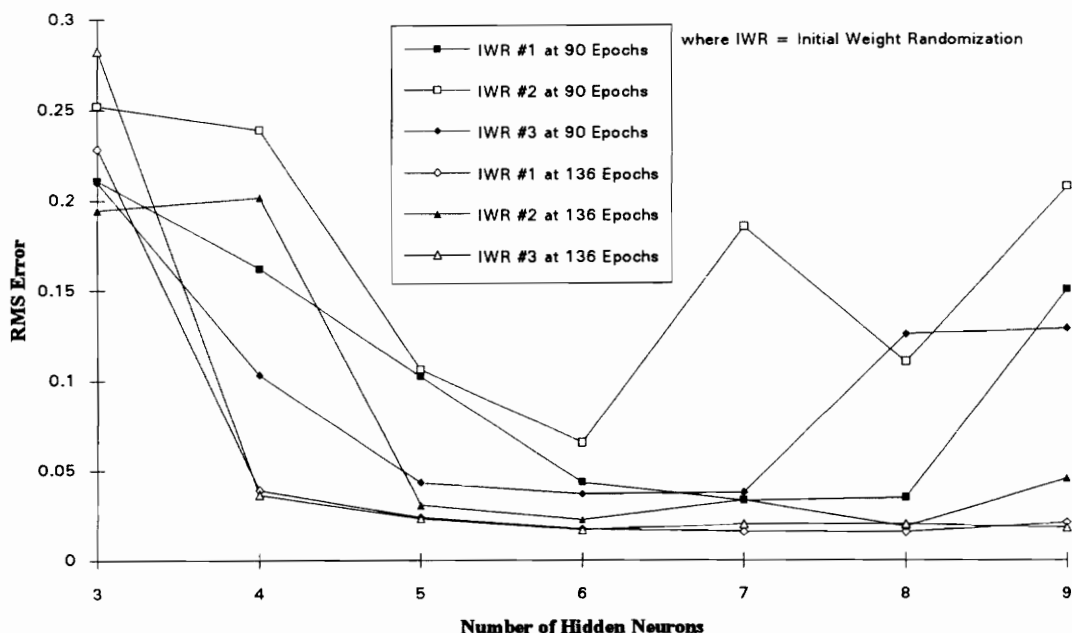


Figure 4.2.3.1.1: Weight Randomization and Undertraining Effects

Figures 4.2.3.1.2, 3, and 4 illustrate the ability of the neural network to train efficiently using six or more hidden neurons for the three, four, and five control allocation problems using various initial weight randomizations. The initial weight randomization of each neural network is located in the legend of each figure. Each neural network for each choice of hidden neurons was trained for 136 epochs. Figure 4.2.3.1.5 provides the above results combined on one chart for comparison. The stable range of minimum RMS error achieved for 136 epochs was both consistent and repeatable using six, seven, and eight hidden neurons for various initial weight randomizations and numbers of controls. Neural networks possessing four or five hidden neurons had, indeed, become trapped in local minimums during training for certain initial weight randomizations, and could not attain the same repeatable performance as neural networks possessing six or more hidden neurons.

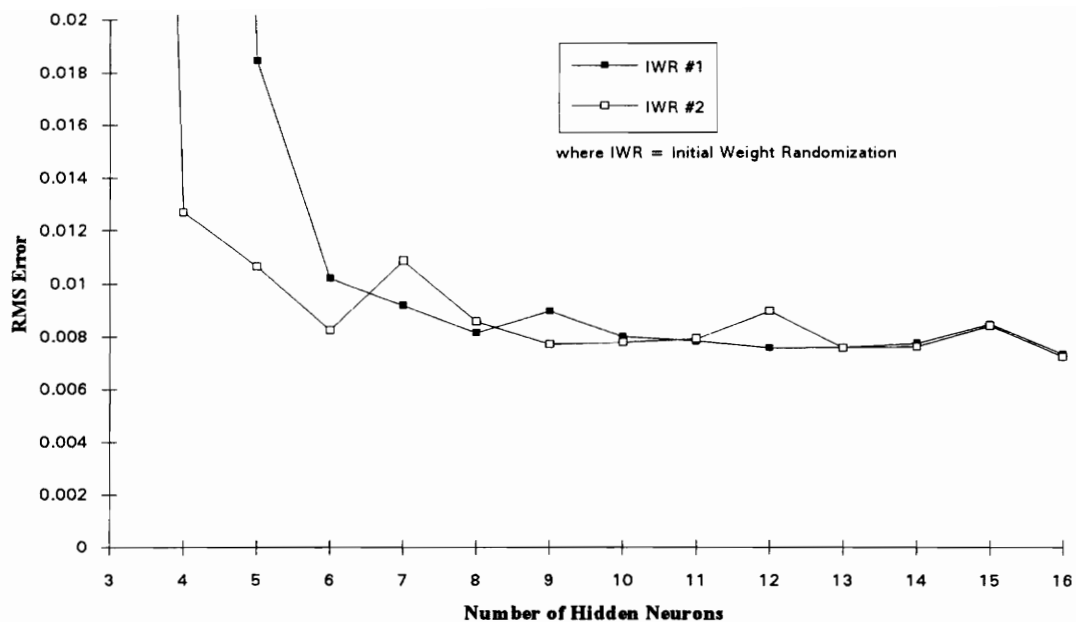


Figure 4.2.3.1.2: RMS Error vs. Number of Hidden Neurons for Three Controls

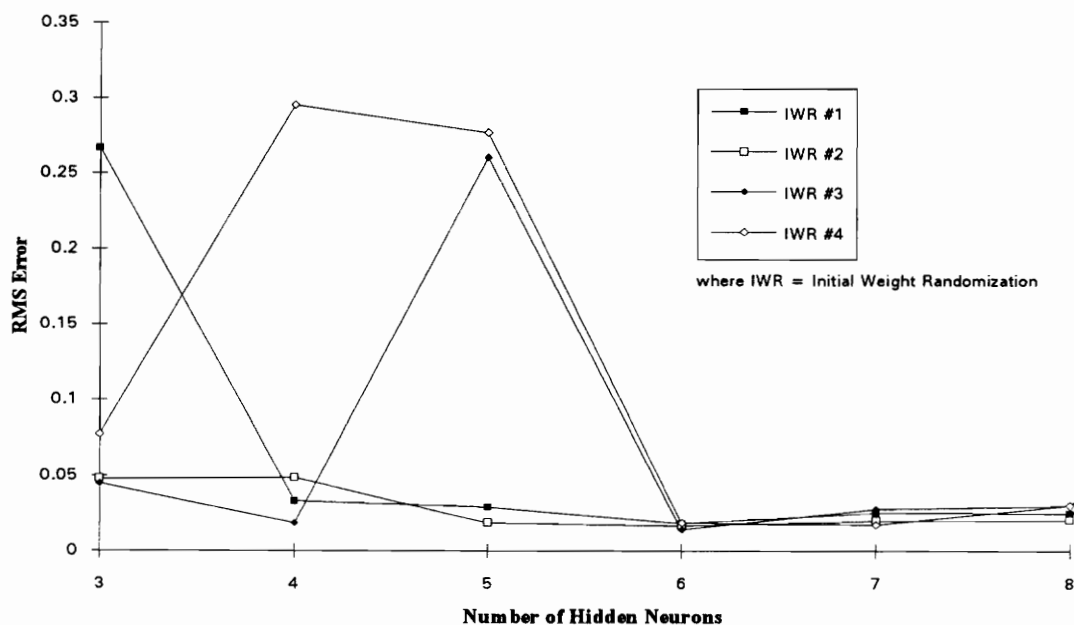


Figure 4.2.3.1.3: RMS Error vs. Number of Hidden Neurons for Four Controls

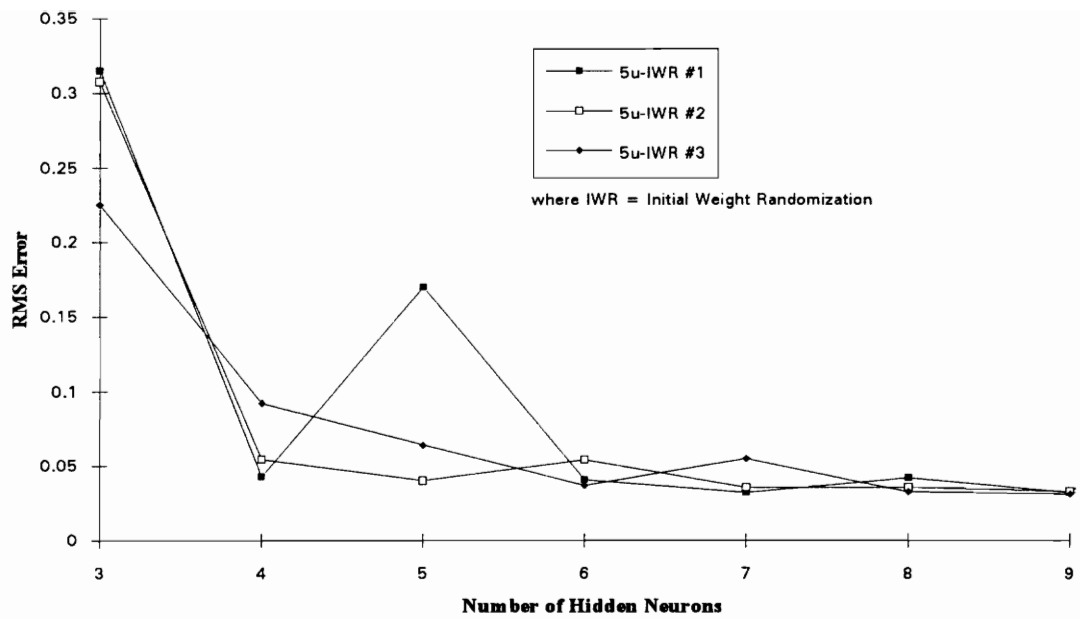


Figure 4.2.3.1.4: RMS Error vs. Number of Hidden Neurons for Five Controls

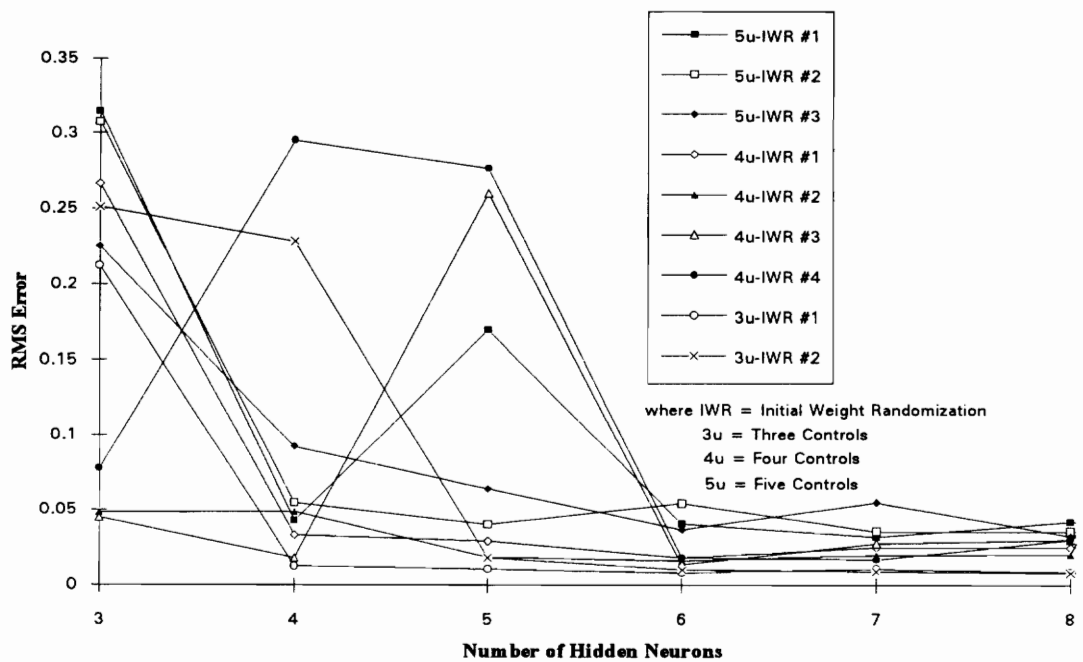


Figure 4.2.3.1.5: RMS Error vs. Number of Hidden Neurons
Three, Four, and Five Controls Comparison

4.2.3.2 *The Neural Network Attainable Moment Subset*

This experiment was designed to test the generalization ability of a neural network with six hidden neurons and the premise that this generalization occurs in a manner that emulates the facets of a three control AMS or generically forms a six sided prism (one side for each hidden neuron) near the bounds of any AMS.

Mathematically, the three control allocation problem has a closed form solution and is solved by the simple inversion of the control effectiveness matrix. However, with the exception of the number of output neurons, the neural network solution to the three control problem is not fundamentally different from problems with many more controls. Thus, because of the simplicity and easy visualization, while retaining the basic concept behind neural network control allocation, the three control problem was chosen.

A Mathematica [14] routine was developed which accepted the vertices of the three control AMS as input and created a wireframe illustration. The eight vertices of the AMS were found by placing the three controls at their eight combinations of minimum and maximum deflections in control space. Because the three control problem can be solved directly, the vertices in control space remain vertices in moment space. Therefore, the vertices of the AMS were found by pre-multiplying the vertices of control space with the inverted control effectiveness matrix. The vertices were then connected by straight lines to form the wireframe AMS.

Additional inputs to the Mathematica routine included the points that generated the neural network representation of the AMS or the NN-AMS. The points forming the NN-AMS were output from a neural network trained on the eight vertices of the AMS and tested on

the six facet centers, twelve points along the edges, and again on the eight vertices of the AMS.

A neural network with six hidden neurons was trained on only the eight vertices of the three control allocation problem AMS. To assess the performance of the neural network it was desirable to train to the point of maximum generalization based on the test set. This point was located by training the neural network for a number of epochs and comparing the RMS error of the output layer for the test and training sets and then repeating this procedure until test performance began to deteriorate. Although the RMS error for the training set continually decreased with training, the RMS error of the test sets increased when the neural network lost its generalization capability and began to memorize the training set data. The neural network is said to be overtrained if memorization occurs. Figure 4.2.3.2.1 depicts the point of maximum generalization for the three control allocation problem. The point of maximum generalization was selected to be the point where the RMS error of the training set intersected the RMS error of the test set at 16 epochs.

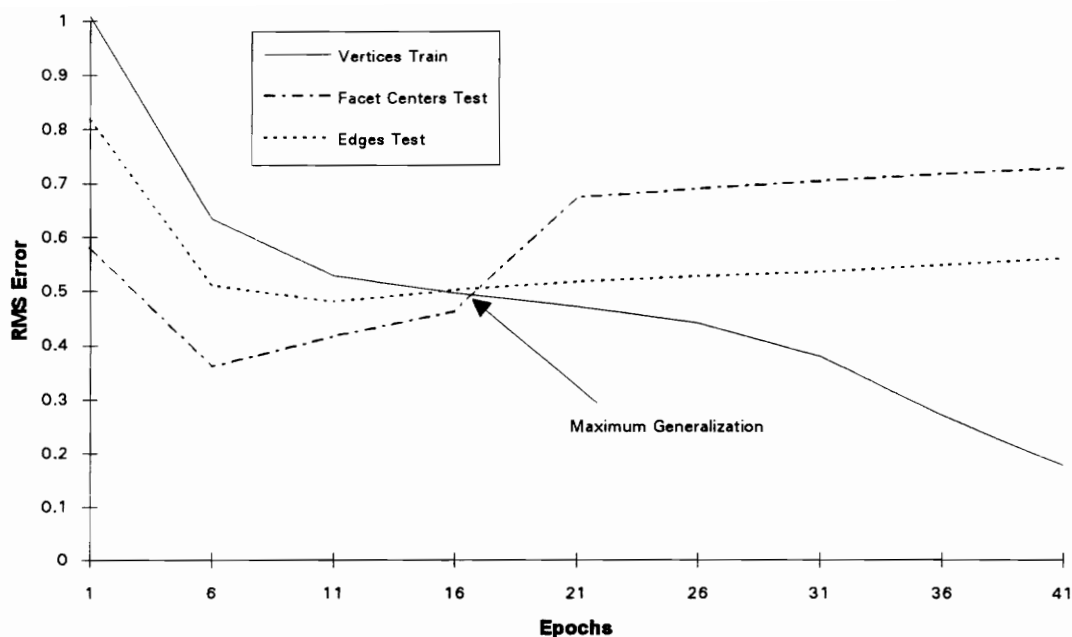


Figure 4.2.3.2.1: Neural Network Generalization for Three Controls

Figure 4.2.3.2.2, 3, 4, and 5 were generated using the Mathematica routine. The routine reads four data files containing the eight vertices of the AMS, the eight vertices of the NN-AMS, the six facet centers of the NN-AMS, and a point along each of the twelve edges of the NN-AMS. The figures depict the NN-AMS inside the AMS for various points of view and the points to which the edge points were mapped. Polygons were formed by connecting the eight vertices of the NN-AMS and the six facet centers to create the facets of the NN-AMS. As demonstrated from the locations of points along the edges, all of the points from a vertex along an edge to another vertex did not necessarily fall in a straight line but, did lie in the vicinity of the facets of the NN-AMS.

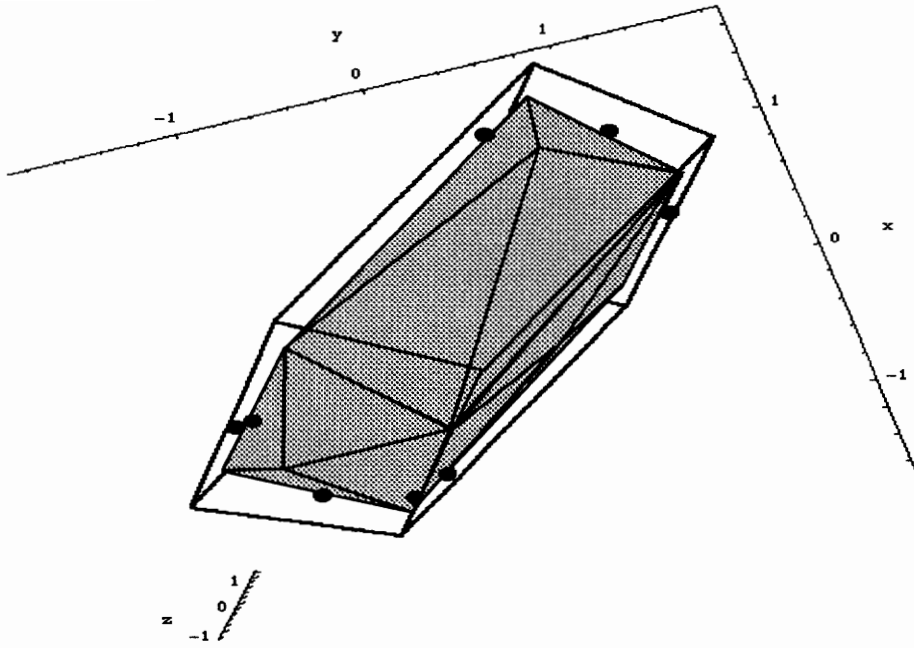


Figure 4.2.3.2.2: The Three Control AMS and NN-AMS Comparison (Viewpoint #1)

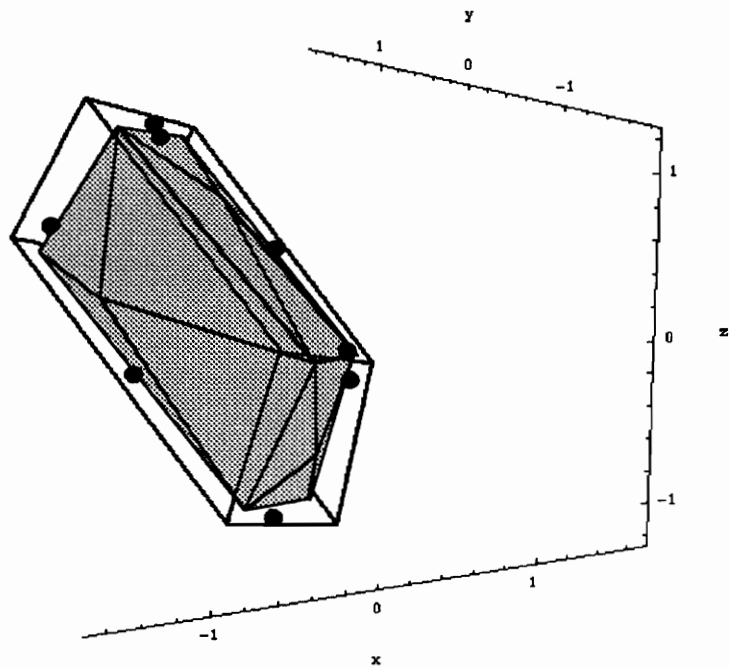


Figure 4.2.3.2.3: The Three Control AMS and NN-AMS Comparison (Viewpoint #2)

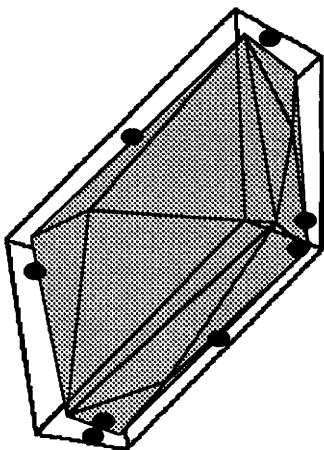


Figure 4.2.3.2.4: The Three Control AMS and NN-AMS Comparison (Viewpoint #3)

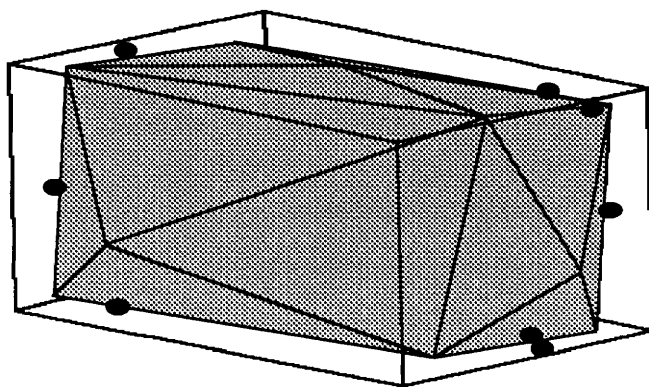


Figure 4.2.3.2.5: The Three Control AMS and NN-AMS Comparison (Viewpoint #4)

4.2.3.3 *Generalization Ability*

As previously discussed, the number of training epochs had a serious consequence on the test performance of a neural network. Thus, it was imperative to discover what affects the generalization ability of the neural network. The effects of initial weight randomization, number of hidden neurons, and training set selection for a five control allocation problem are found in Figures 4.2.3.3.1, 2, and 3, respectively.

The choice of initial weight randomization and number of hidden neurons (if greater than or equal to six) had a negligible effect on the generalization performance of the neural network. However, Figure 4.2.3.3.3 shows that the test performance was affected by the choice of training data. Although both neural networks were tested on the facet center test set, a neural network trained on a grid rather than the vertices of the AMS did not overtrain so abruptly.

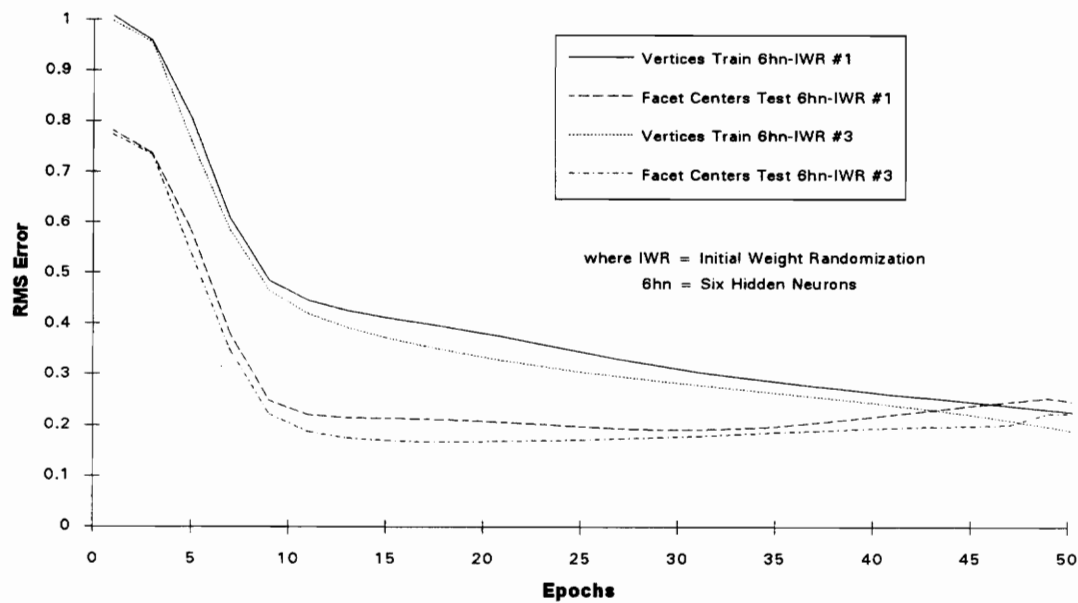


Figure 4.2.3.3.1: Effect of Initial Weight Randomization on
Neural Network Generalization for Five Controls

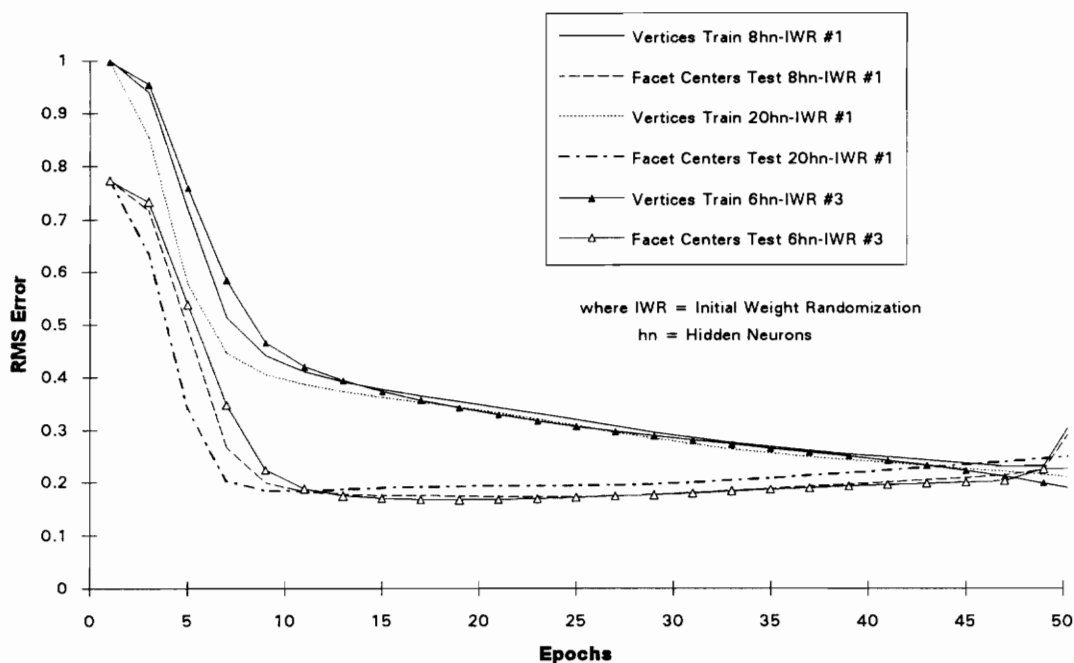


Figure 4.2.3.3.2: Effect of Number of Hidden Neurons on Neural Network Generalization for Five Controls

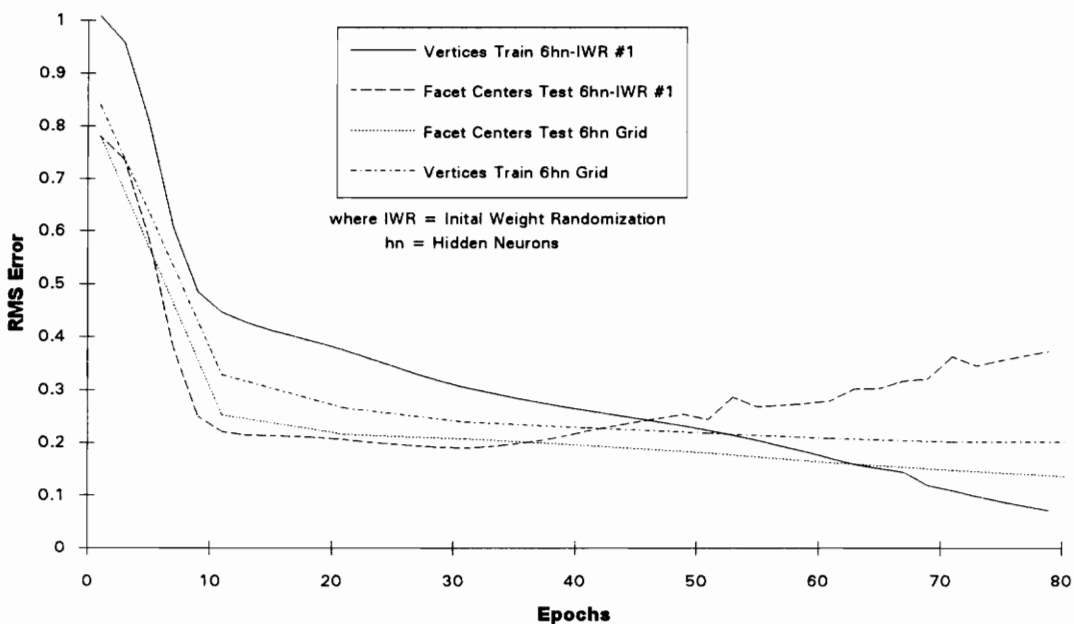


Figure 4.2.3.3.3: Effect of Training Set on Neural Network Generalization for Five Controls

4.3 Questions of Repeatability

As a consequence of the above results, no common point of overtraining could be found for all control allocation problems. Similar experiments must be run to determine the point of maximum generalization. Although, once one experiment is complete, it does not have to be repeated for different initial weight randomizations or additional hidden neurons.

The results relating to the magic number of hidden neurons were certainly repeatable and hold for any control allocation problem. For example, Figures 4.3.1 and 2 contain the attainable moment subsets for two different five control allocation problems. Figure 4.3.3 illustrates that six remains the best number of hidden neurons, although the attainable moment subsets have very different appearances. Additionally, the choice of initial weight randomization had little effect on network training efficiency or test performance.

Thus, it is concluded that a neural network trained using backpropagation, for a control allocation problem, can be efficiently trained using six neurons in the hidden layer. Furthermore, the initial neural network weights have little effect on the results obtained. The only requirement upon the outset of a new control allocation problem is to find the point of maximum generalization.

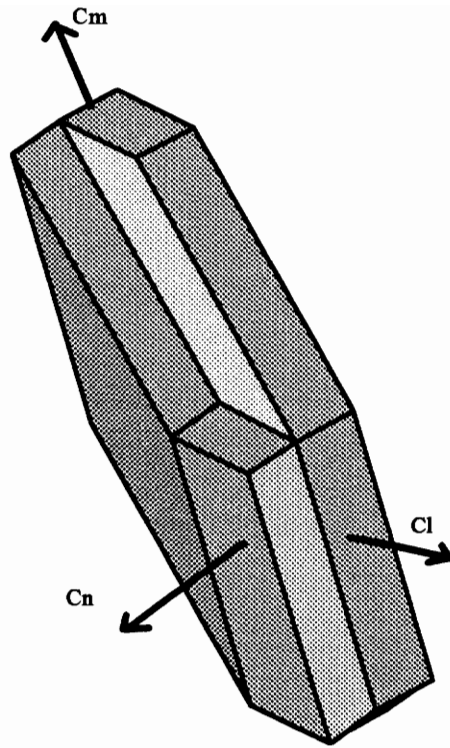


Figure 4.3.1: Five Control AMS for the F/A-18 HARV

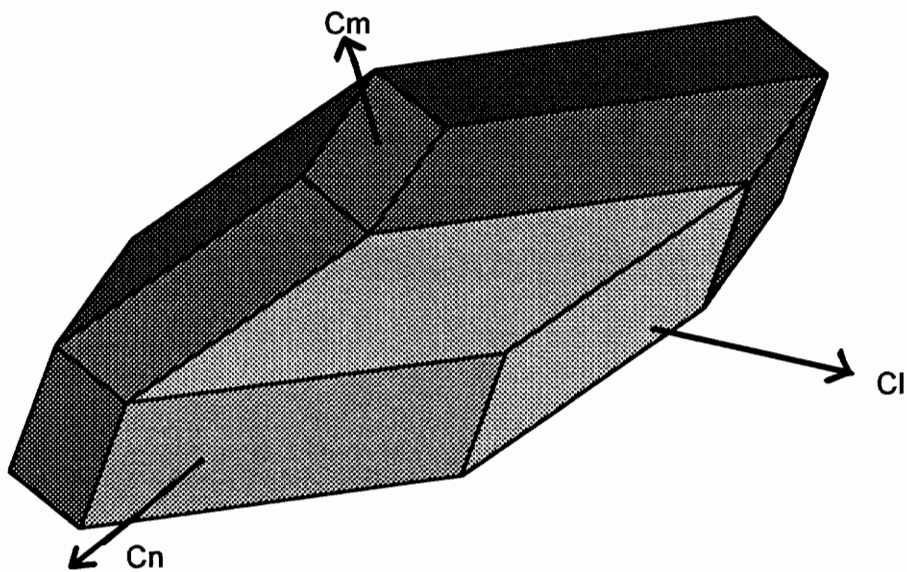


Figure 4.3.2: Five Control AMS for Random B Matrix

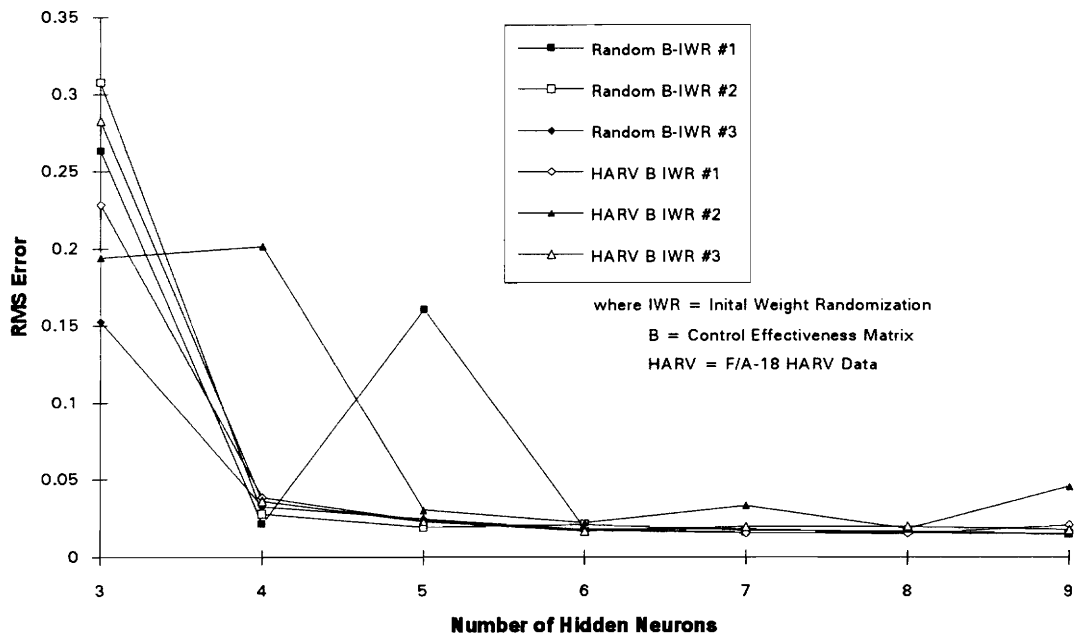


Figure 4.3.3: RMS Error vs. Number of Hidden Neurons
Five Controls Random and Real Control
Allocation Parameter Comparison

5.0 Simulation Implementation

For the implementation phase of this investigation an allocation problem with seven controls has been selected. The perspective of a flight control engineer who has been presented with the challenge of designing a real time control allocator for an aircraft digital flight control system or a real time aircraft simulation is taken. The engineer is expected to have the following tools available:

- A basic knowledge of control allocation and an allocation scheme available to generate neural network training sets.
- A familiarity with neural networks and software for neural network development.
- Software to generate training grids.

In this implementation the following tools corresponding to the above were utilized:

- The direct allocation method implemented in CAT (Control Allocation Toolbox) developed by Dr. W. C. Durham at the Virginia Polytechnic Institute and State University [4] and modified to extract neural network training and testing data in the preferred format.
- NeuralWorks [7] neural network development software distributed by NeuralWare, Inc., Pittsburgh, PA.
- A FORTRAN subroutine developed by the author for the generation of three dimensional symmetric rectangular grids of given size and density.

The ultimate goal is to create a subroutine that accepts the control surface deflections of an existing flight control system as input and reallocates the control surface deflections using a trained neural network that emulates the direct allocation scheme implemented in CAT. This stand-alone subroutine (termed a NeuroAllocator) may then be installed as a module in an existing flight simulator.

The methods contained in this chapter may be extended for the development of a flight control NeuroAllocator for any generic aircraft using any type of control allocation scheme.

5.1 Overview of Approach

The general procedure taken to realize the neural network control allocator in preparation for real time flight simulation implementation was as follows:

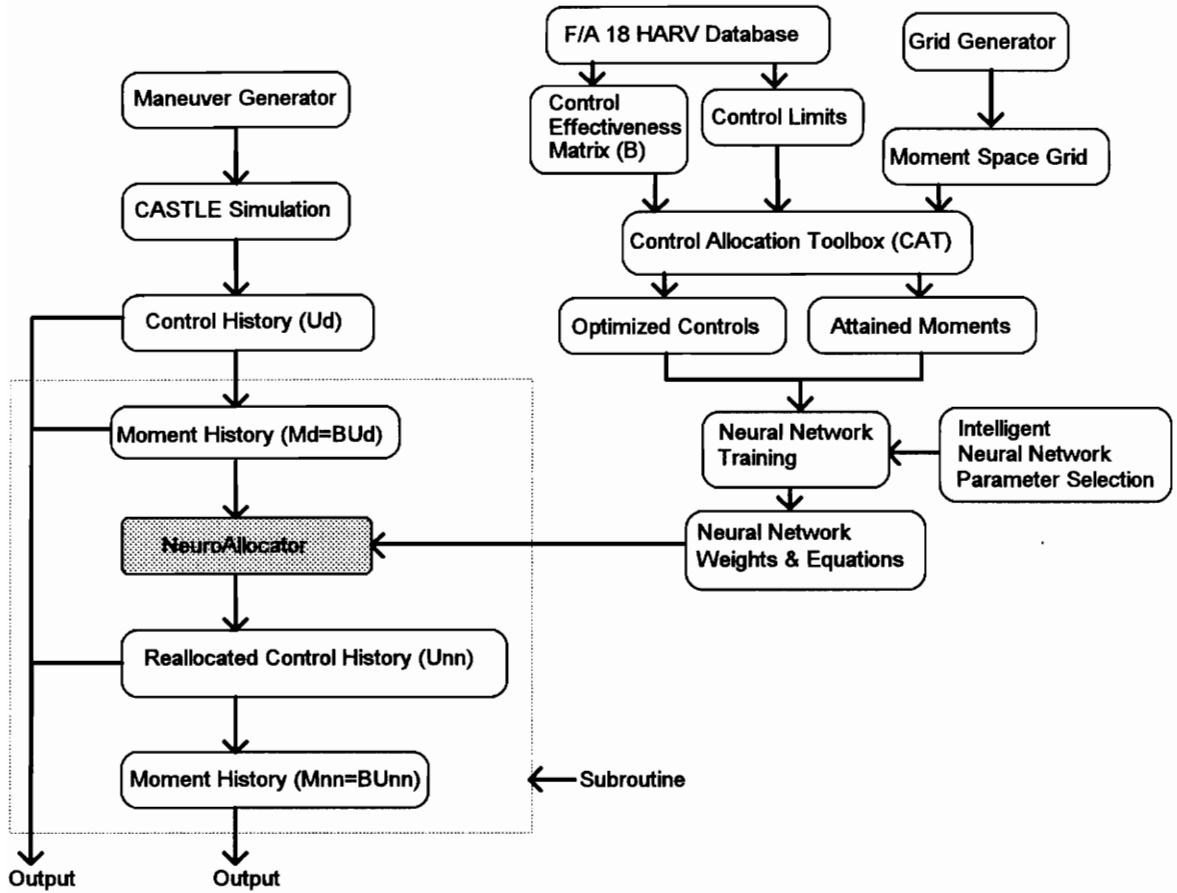


Figure 5.1.1: General Approach Overview

5.2 Nominal Aircraft Model

The nominal aircraft model is that of the F/A-18 HARV at an angle of attack of 28.5° . This model was chosen in anticipation of future implementation in the Controls and Simulation Test Loop Environment (CASTLE) modular flight simulator developed by the

Naval Air Warfare Center (NAWC). Version 1.4 of CASTLE supporting the simulation of the F/A-18A was used to generate the test maneuver in a prior investigation (see Reference [15]). The control effectiveness or B matrix containing the partial derivatives of the three moments with respect to the seven control surfaces, provided in Table 5.2.1, at an angle of attack of 28.5° is:

$$\mathbf{B}_{\text{F18 HARV w/ Flaps}} = \begin{bmatrix} 6.47e-4 & -6.47e-4 & 6.00e-4 & -6.00e-4 & 5.83e-5 & 1.06e-3 & -1.06e-3 \\ -7.35e-3 & -7.35e-3 & -6.00e-4 & -6.00e-4 & 0.00e+0 & 4.80e-4 & 4.80e-4 \\ -1.00e-7 & 1.00e-7 & -1.50e-4 & 1.50e-4 & -6.67e-4 & 0.00e+0 & 0.00e+0 \end{bmatrix}$$

Table 5.2.1: F/A-18 HARV Control Deflection Limits

	Control Surface	Minimum Limit	Maximum Limit
U ₁	Right Horizontal Tail	24° trailing edge up	10.5° trailing edge down
U ₂	Left Horizontal Tail	24° trailing edge up	10.5° trailing edge down
U ₃	Right Aileron	25° trailing edge up	25° trailing edge down
U ₄	Left Aileron	25° trailing edge up	25° trailing edge down
U ₅	Combined Rudders	30° trailing edge left	30° trailing edge right
U ₆	Right Trailing Edge Flap	10° trailing edge up	40° trailing edge down
U ₇	Left Trailing Edge Flap	10° leading edge up	40° leading edge down

A futile attempt was made to extract the control effectiveness parameters from the CASTLE simulation for the F/A-18A in the prior investigation. The data from the HARV database has been shown to produce good results [15]. Since the F/A-18 HARV data for the leading edge flaps were not available, these surfaces will be left as scheduled by the original CASTLE control laws. Additionally, the left and right rudder of the CASTLE simulation were treated as a single combined rudder control by the neural network control allocator.

5.3 Training Set Generation

A neural network trained on points of a grid on and within the AMS has been shown, in Section 4.2.3.3, to provide superior generalization performance to a neural network trained on the vertices of the same AMS. This superior performance was characterized by allowing a lower RMS error to be achieved at an extended number of training epochs, as shown in Figure 4.2.3.3.3. Additionally, grid training is necessary for sufficient performance of general test maneuvers having desired moment vectors that may occur within the bounds of the AMS.

The choice of a training grid for the neural network control allocator had the following characteristics. The training grid:

- (1) spanned the three dimensional attainable moment subset.
- (2) had sufficient density to provide adequate test performance.
- (3) was limited to those points on the bounds and within the AMS.
- (4) contained the defining feature of the AMS, namely the vertices.

A grid generation program was developed to create a three dimensional rectangular grid of given size and density. Once generated this grid was input to CAT which provided the control deflections corresponding to each point in the input moment space grid. Each input moment space grid vector and corresponding output control vector is referred to as a training pattern and the collection of all training patterns of the grid is the training set. To determine the dimensions of the grid, the AMS for the F/A-18 HARV using seven controls was visualized using CAT, as illustrated in Figure 5.3.1. The minimum and maximum moment coefficients in each direction are given in Table 5.3.1.

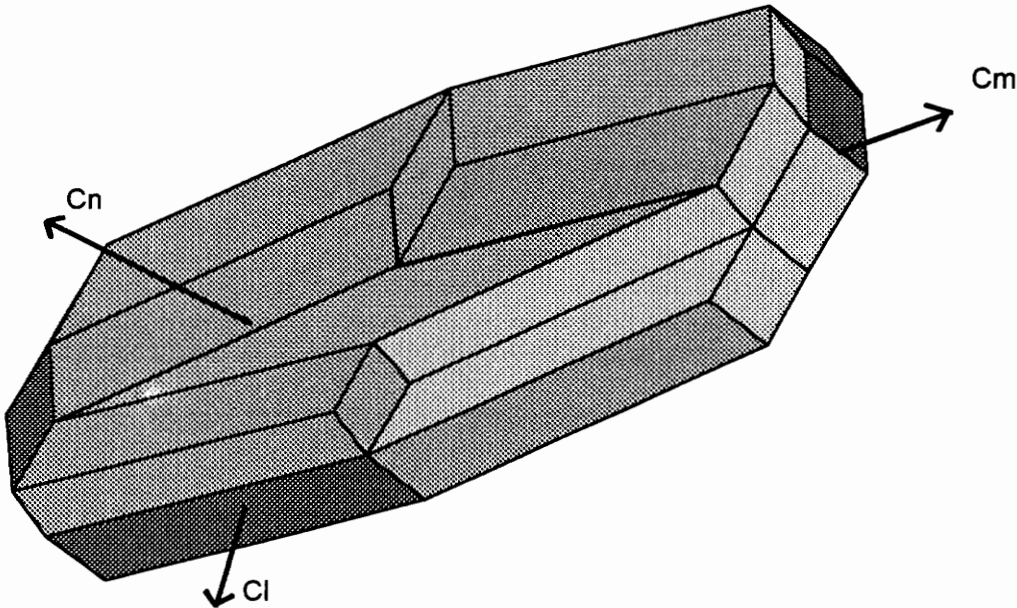


Figure 5.3.1: Seven Control F/A-18 HARV Attainable Moment Subset

Table 5.3.1: Moment Coefficient Limits

Moment Coefficient	Minimum	Maximum
C_l	-0.107	0.107
C_m	-0.194	0.421
C_n	-0.028	0.028

Note that the AMS was unsymmetric in the pitching moment coefficient direction. Since the grid generator was only capable of creating rectangular symmetric grids, the initial grid of the AMS, depicted in Figure 5.3.2, was not satisfactory.

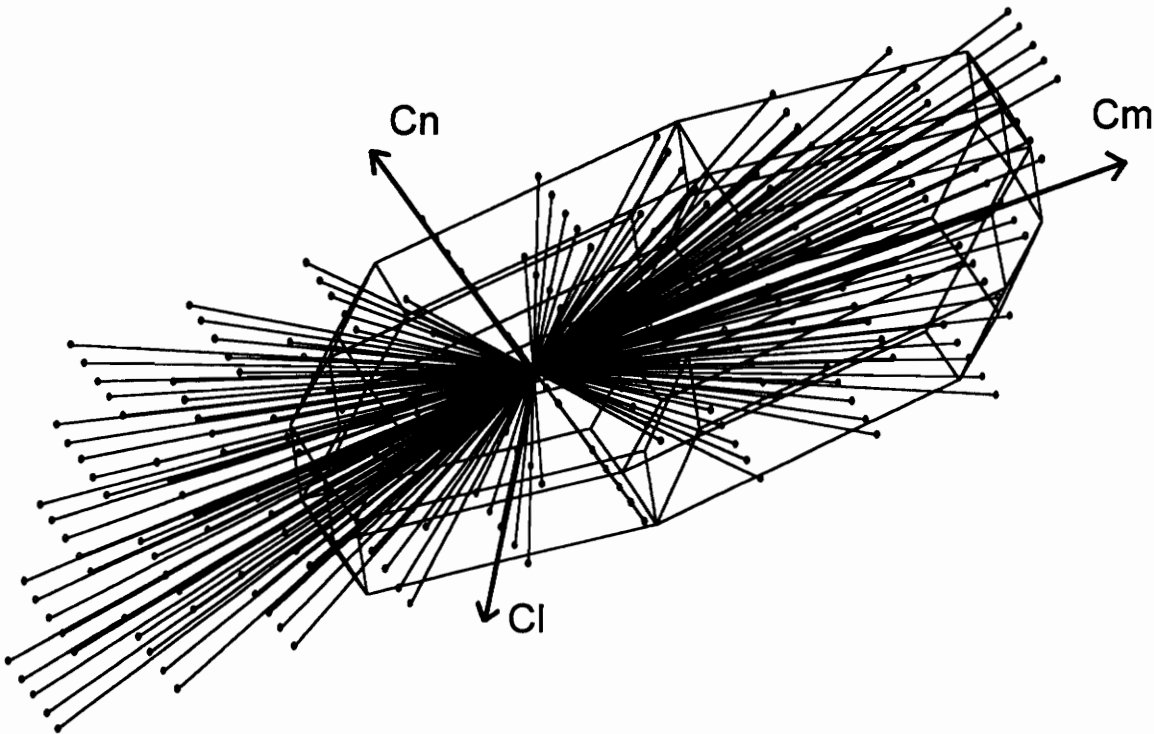


Figure 5.3.2: Initial Input Moment Space Training Grid

Fortunately, as CAT allocated the controls corresponding to the grid points it also calculated the moments actually attained, thereby scaling back unattainable moments outside the boundary of the AMS. A consequence of having many points outside the AMS being scaled back to its boundary was a clustering of training points on the boundary of the AMS. Finally, the vertices and corresponding controls were appended to each training grid to ensure their existence. An example of a complete training grid having 270 training points is shown in Figure 5.3.3.

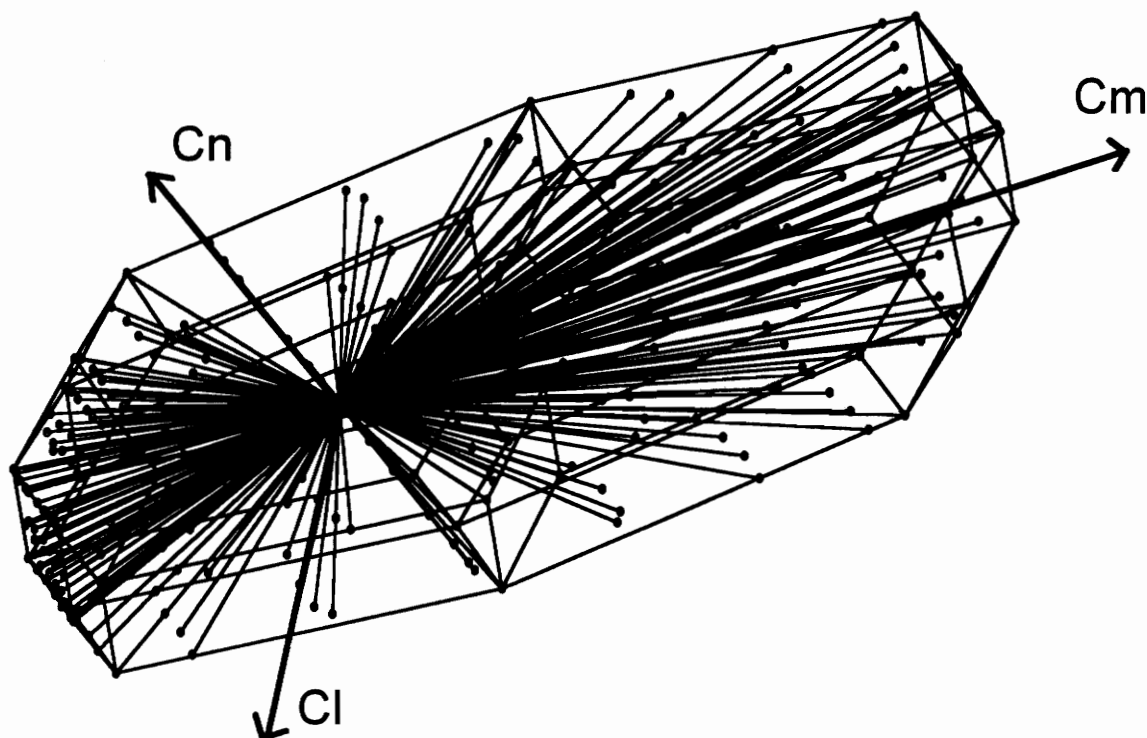


Figure 5.3.3: Final Input Moment Space Training Grid

The only desired characteristic of the training set that is not certified is its density. It is advisable to start with a grid of nominal density based on reasonable training times, such

as the grid shown, and adjust for the desired test precision. The nominal grid used in this case contained 270 points, using a spacing between grid points of 0.05 in the C_I direction, 0.1 in the C_m direction, and 0.015 in the C_n direction. By starting with a grid of nominal density and finding its point of maximum generalization during training, the continued reproduction of the experiment for each subsequent grid density may be avoided since the number of training epochs required to attain maximum generalization was found to be fairly insensitive to adjustments in grid density, as shown in Section 4.2.3.3.

5.4 Intelligent Neural Network Configuration and Parameter Selection

As opposed to the trial and error approach commonly taken to develop a neural network, the neural network structure and parameters were selected intelligently based upon the experiments of Chapter 4. The EDBD backpropagation learning rule used in the experiments was used for training the neural network to be implemented. The same learning coefficients used in the experiments and listed in Table 4.1.1.1 were used. In addition, the hyperbolic tangent squashing function and the scaling method described in Sections 4.1.4 and 4.1.5, respectively, were selected.

In the quest for the magic number, it was demonstrated that a six hidden neuron neural network can train efficiently, form an effective neural network representation of the AMS, and retain good generalization performance. The point of maximum generalization will be found by again testing on the facet centers. Finally, any initial weight randomization may be employed since it was shown not to affect generalization ability.

Using the above rationale the neural network structure used to solve the seven control allocation problem is illustrated in Figure 5.4.1.

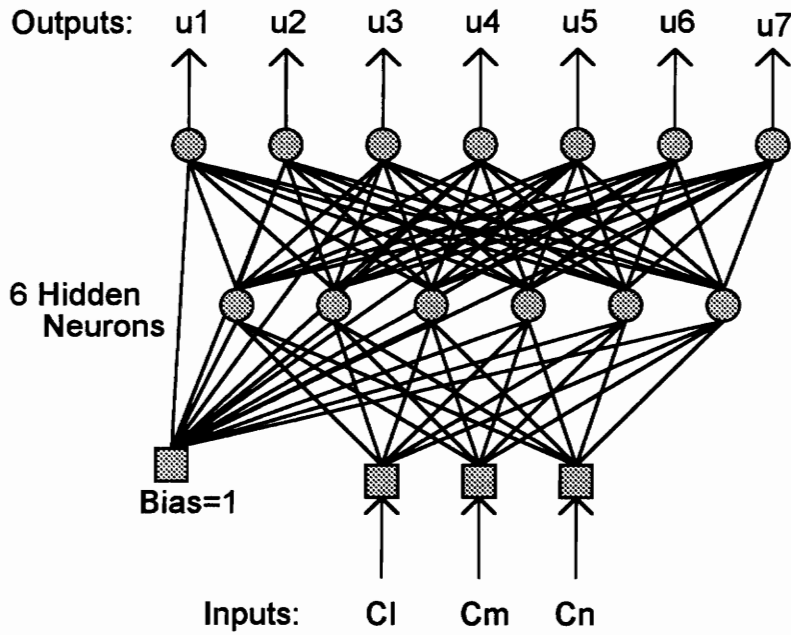


Figure 5.4.1: Seven Control NeuroAllocator Structure

5.5 Test Maneuver Selection

Neural network performance will be evaluated on a high angle of attack roll from wings level to 60° of bank, to -60° of bank, and back to wings level. Figure 5.5.1 shows the time history of moment vectors required to produce this maneuver inside of the wireframe F/A-18 HARV AMS.

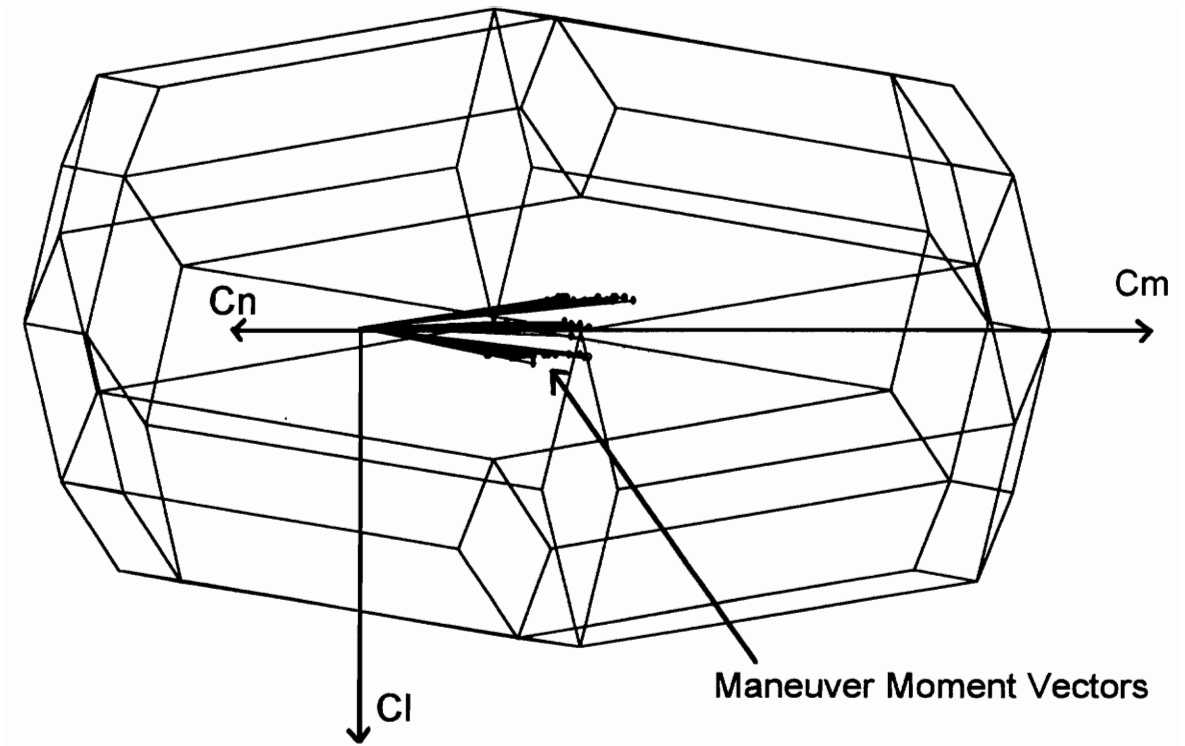


Figure 5.5.1: Desired Moments for Test Maneuver in the F/A-18 HARV AMS

The rationale behind the selection of this maneuver, given in Reference [15], was to generate a maneuver that would:

- (1) saturate at least one control using the original control law, since the optimal control allocator could be shown to eliminate this saturation.
- (2) fall into the low dynamic pressure (high α) flight regime, to again saturate the controls and also maximize control actuator rates.
- (3) have a relatively small departure from the initial angle of attack since the control effectiveness matrix was derived at the initial condition and changes considerably with angle of attack.

The maneuver was performed using the maneuver generator in the CASTLE simulation. The control augmentation system (CAS) of the F/A-18A is a fly-by-wire redundant, full authority digital flight control system. The Auto Flap Up configuration for up and away flight was used during the maneuver simulation [15]. The CAS of the F/A-18A uses gain scheduling, cross axis interconnection, and closed loop control for the original allocating of the ten control surface deflections for the maneuver.

5.6 Training Analysis

As stated in the previous chapter, the only requirement upon the outset of a new control allocation problem is to find the point of maximum generalization. The neural network described in the previous section was trained using grids ranging from 270 points to 2800 points. The characteristics of the two grids used in the performance analysis are provided in Table 5.6.1.

Table 5.6.1: Grid Characteristics

Characteristic		Grid A	Grid B
Number of Points:		270	2800
Maximum Limit in Direction:	Cl	0.1	0.1
	Cm	0.4	0.4
	Cn	0.03	0.03
Interval Between Points in Direction:	Cl	0.05	0.015
	Cm	0.1	0.015
	Cn	0.015	0.015

After every epoch, which in this case was one pass through a training grid, a reading of the RMS error achieved by testing the facet centers of the AMS was recorded. Figure 5.6.1

demonstrates that the point of maximum generalization for this particular network and grid A was achieved at 61 epochs.

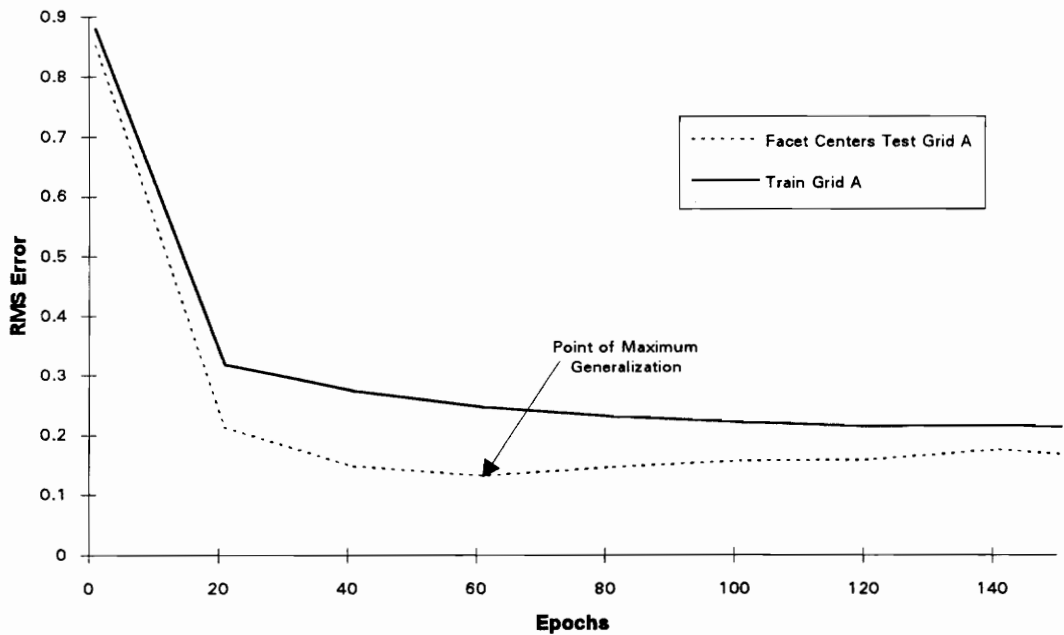


Figure 5.6.1: Neural Network Generalization Performance for Training Grid A

The neural network trained for 61 epochs on grid A was tested on the selected maneuver which had desired moments well within the bounds of the AMS and unsatisfactory results, discussed in detail in the following section, were found. Large deviations from the desired performance demanded a reevaluation of grid density, training epochs, and the testing set.

Grid B achieved a lower RMS error for the training and facet center test sets using fewer training epochs than grid A. The neural network trained on grid B was then subjected to a test using points interior to the bounding surface of the AMS, in addition to the facet centers. The RMS error achieved by the training set, the facet center test set, and the interior points test set was recorded every training epoch, as illustrated in Figure 5.6.2.

Although the facet center test RMS error continued to decrease after 20 training epochs and did not predict overtraining for many training epochs, the interior points test demonstrated that the neural network had become overtrained at only 13 epochs.

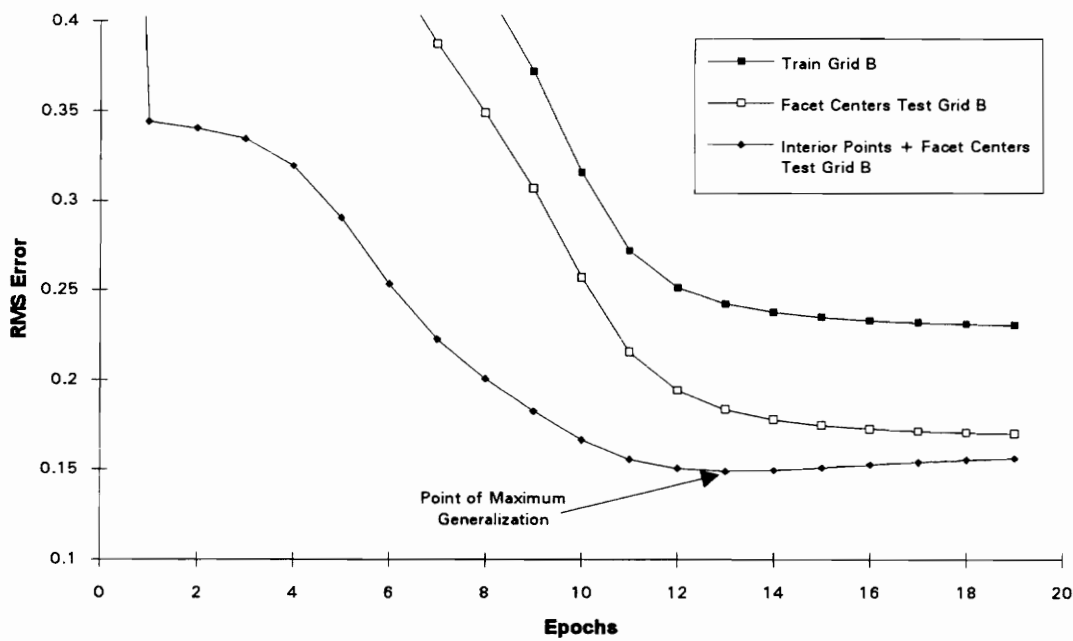


Figure 5.6.2: Neural Network Generalization Performance for Training Grid B

Therefore, for general test maneuvers it is very important to choose testing data that is representative of the entire input moment space. If not, the point of maximum generalization may be greatly overestimated resulting in inferior performance. Furthermore, a substantial number of training grid points is necessary to provide an acceptable RMS error. The need for a dense training grid is discussed in detail in Section 5.8.

Inspection of a weight histogram during training revealed an even distribution of relatively small weights throughout the network, as expected. This signifies the neural network had an adequate number of weights to represent the nonlinear functional relationship between the moments and controls [7].

This neural network was trained in total isolation of the knowledge of the maneuver to be performed. Thus, no bias was given to any octant of the AMS or spacing of grid points in any direction.

5.7 Neural Network Deployment Module

During the training analysis the point of maximum generalization for training grid B was found to occur at 13 epochs. Once this point was found the neural network was reinitialized and trained the specified number of epochs. At this point the weights of the fully trained neural network were extracted and used as the weight matrices in the following set of equations.

$$\begin{Bmatrix} X_{hn1} \\ X_{hn2} \\ X_{hn3} \\ X_{hn4} \\ X_{hn5} \\ X_{hn6} \end{Bmatrix} = \text{Tanh} \left[W_1 \begin{Bmatrix} 1 \\ C_1 \\ C_m \\ C_n \end{Bmatrix} \right]$$

$$\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{Bmatrix} = \text{Tanh} \left[W_2 \begin{Bmatrix} 1 \\ X_{hn1} \\ X_{hn2} \\ X_{hn3} \\ X_{hn4} \\ X_{hn5} \\ X_{hn6} \end{Bmatrix} \right]$$

where W_1 is 6x4 (#of HN) x (# of Inputs + Bias)

where W_2 is 7x7 (#of Outputs) x (# HN + Bias)

These were the equations that were translated into FORTRAN and deployed in the subroutine NeuroAllocator block. The equations are quite simple and consist of a matrix-vector multiplication, a pass through the squashing function, another matrix-vector

multiplication, and another pass through the squashing function. An example of the subroutine module is located in Appendix C.

The subroutine accepts a vector of control deflections from an existing flight simulator control law. The desired moment the pilot wishes to achieve associated with the original control deflections was found and input to the NeuroAllocator. By employing the above equations the controls were reallocated according to the direct control allocation scheme the neural network was emulating. The original desired moment, reallocated controls, and moment achieved by the reallocated controls were output from the subroutine for comparison.

5.8 *Maneuver Performance*

A driver program was created to call the NeuroAllocator in order to simulate the real time operation of the CASTLE flight simulator. The program called the NeuroAllocator module at each time step (0.02 sec) for the duration of the series of the two-roll-reversals test maneuver. The user determined if the trailing edge flaps were to be scheduled by the NeuroAllocator during the first pass of the subroutine. Additionally, the user may choose to use neural networks trained on grids of various density for the remainder of the simulation.

Each time the driver program called the NeuroAllocator module, the subroutine read the next sequence of ten control surface deflections from a user defined input file containing the time history of control deflections for a desired test maneuver. Leading edge flaps were not reallocated and were left as scheduled by the CASTLE CAS. The left and right rudder control deflections were treated as a single combined rudder. This leaves either five (or seven, if the trailing edge flaps are to be rescheduled) of the ten original controls

to be reallocated by the NeuroAllocator. The results presented in this chapter are for reallocated trailing edge flaps.

Figures 5.8.1, 2, and 3 illustrate the rolling, pitching, and yawing moment coefficients, respectively, for the 20 seconds of the test maneuver. Results are presented for the desired moment coefficients obtained using the CASTLE CAS, and the moments obtained using the reallocated controls from the direct allocation scheme (from CAT), and from the NeuroAllocator trained on grids A and B. Note, that since the direct allocation scheme could perfectly reproduce the desired moments, the time history from CAT is indistinguishable from the desired moment coefficients using the CASTLE CAS.

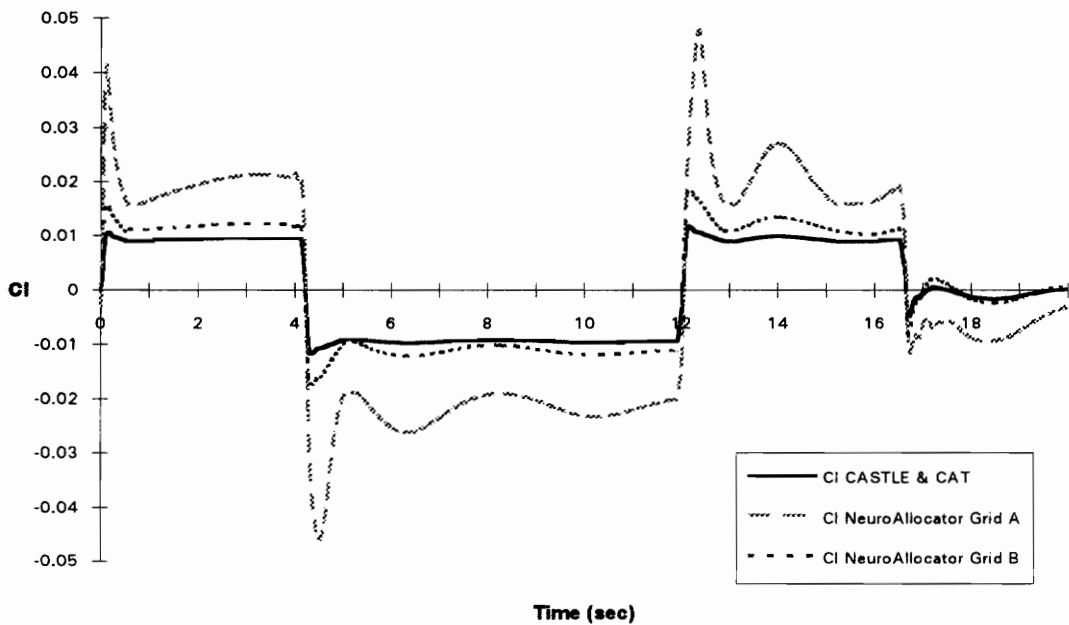


Figure 5.8.1: Rolling Moment Coefficient Time History for Test Maneuver

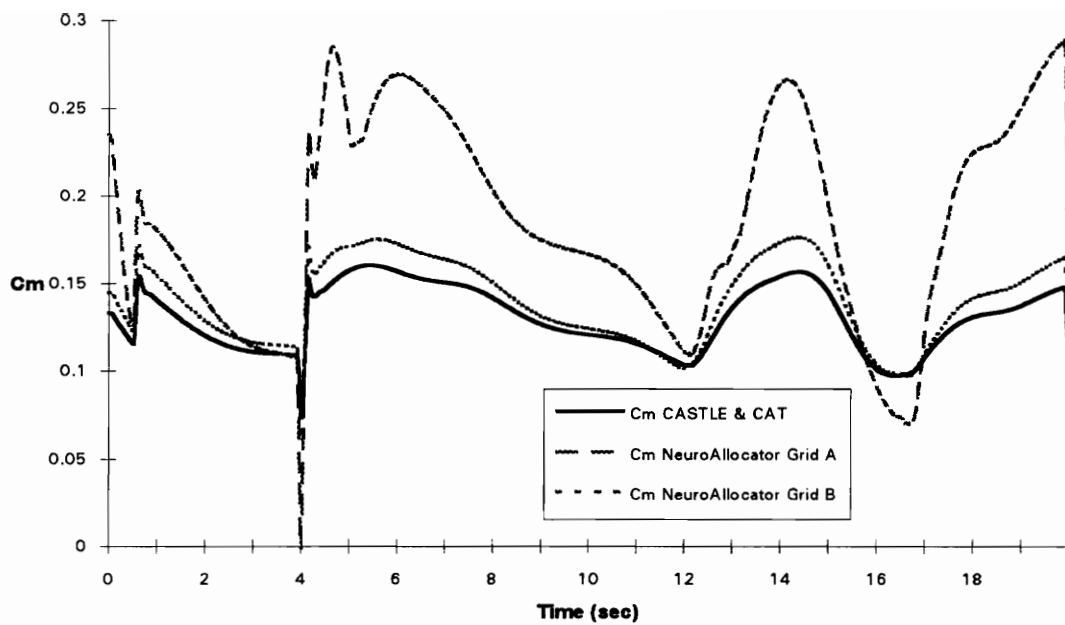


Figure 5.8.2: Pitching Moment Coefficient Time History for Test Maneuver

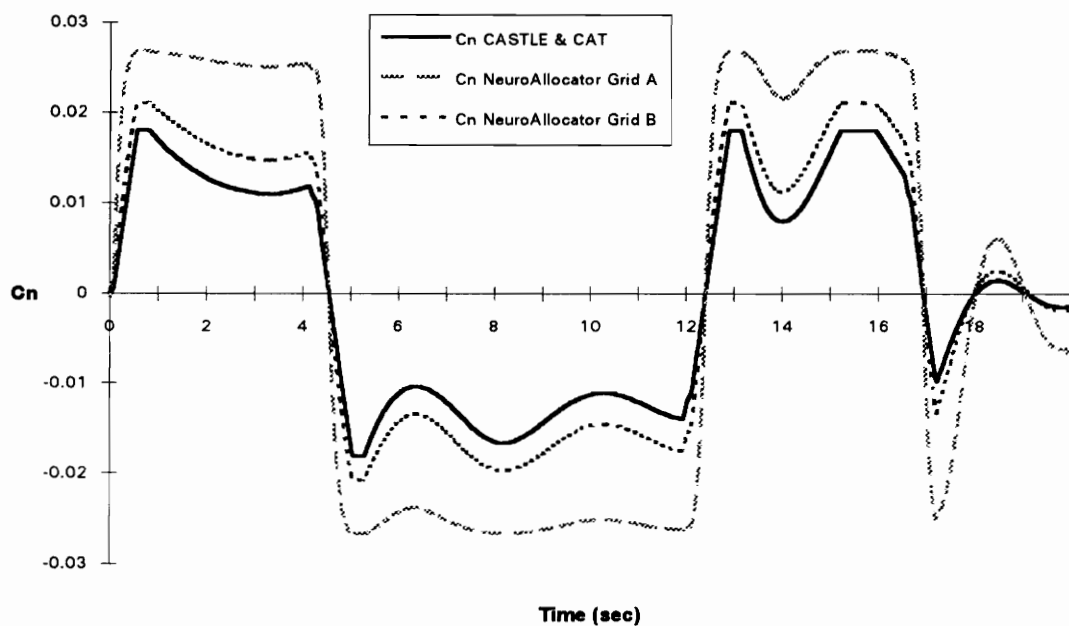


Figure 5.8.3: Yawing Moment Coefficient Time History for Test Maneuver

The performance of the NeuroAllocator to reproduce the desired moments was substantially improved with the density of grid A (270 points) to grid B (2800 points). For grids of density between grid A and B the same trends were found and gradually approached the results of grid B with increasing density. Grid B nearly obtained the desired moments but fell short of perfectly matching the desired time history as was achieved by CAT, the control allocation scheme the neural network was attempting to emulate. Since 2800 training patterns is a sizable number of grid points for a training set, on the basis of training times of up to 30 minutes on a 486DX33 machine, density was not increased beyond that amount.

As shown in Figures 5.8.1, 2, and 3, although the resulting moments produced by the NeuroAllocator control deflections match phase with the desired moments, there was an error in magnitude. This error, specifically a magnitude error greater than that desired, can be attributed to the fact that the training grids had a greater density of points on the bounding surface of the AMS and were sparse in the interior. It is thought that this created a bias to extend the moment vector that ultimately commanded larger control deflections resulting in larger moments.

Although it is thought that continued increases in grid density will gradually drive the resulting moments of the NeuroAllocator to those desired, it is advantageous to discover why such great numbers of training grid points are necessary rather than using brute force by creating increasing large unwieldy training grids to achieve better performance.

Observing the time histories of control deflections that create the moments to produce the test maneuver lends insight into the limitations of the neural network ability to emulate CAT. Figures 5.8.4, 5, 6, 7, 8, 9, and 10 illustrate a comparison of the original CASTLE

CAS, the reallocated CAT, and the reallocated NeuroAllocator control deflection time histories for the test maneuver.

The NeuroAllocator should ideally follow the reallocated control deflections of CAT, but it is clearly demonstrated by the horizontal tail and aileron deflections that the NeuroAllocator cannot achieve the resolution of CAT. The NeuroAllocator had, in effect, smoothed out the swiftly varying control deflection demands of CAT by generalizing the relationship between moments and controls between grid points. A consequence of this type of nonlinear curve fit to the CAT control deflections was the reduction of actuator rate demands.

Each time history of control deflections shows that the reallocated controls were quite different from the original CASTLE CAS control deflections with the exception of the rudder. The direct allocation scheme that the NeuroAllocator was emulating, by design, takes full advantage of all available controls without saturation unless the desired moment lies on the bounding surface of the AMS. This effect was most evident in the reallocation of the trailing edge flaps which were barely utilized by the original control law.

Since the test maneuver occurred well within the bounds of the AMS, the resulting reallocation did not saturate any control surface at any time. In fact, the saturation of the rudder during the maneuver by the original control law was eliminated in the reallocation of the controls.

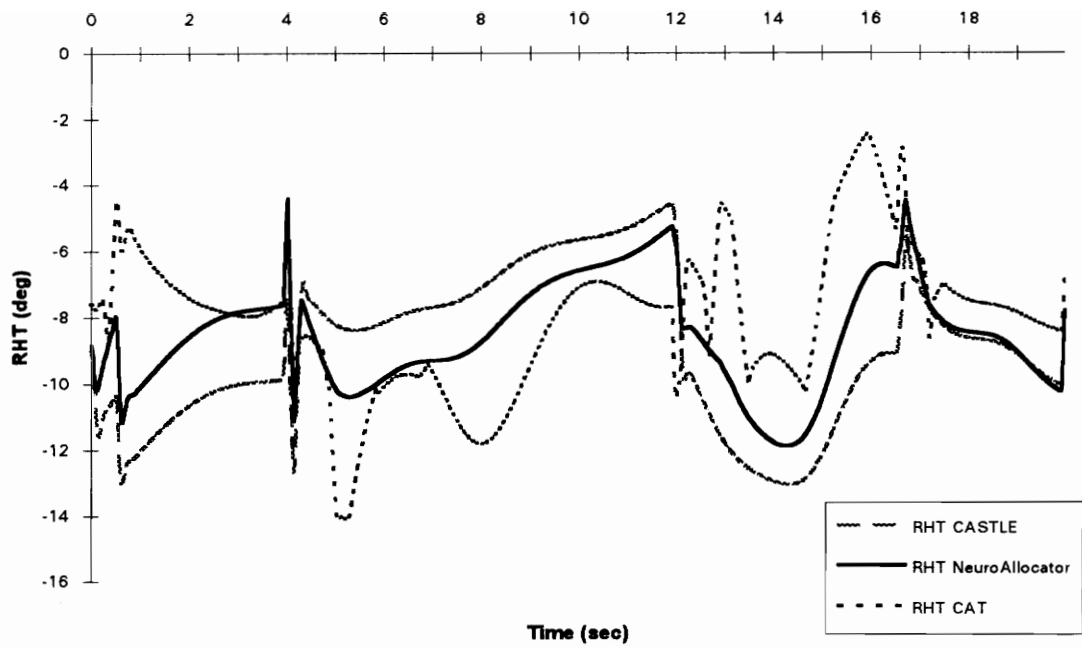


Figure 5.8.4: Right Horizontal Tail Deflection

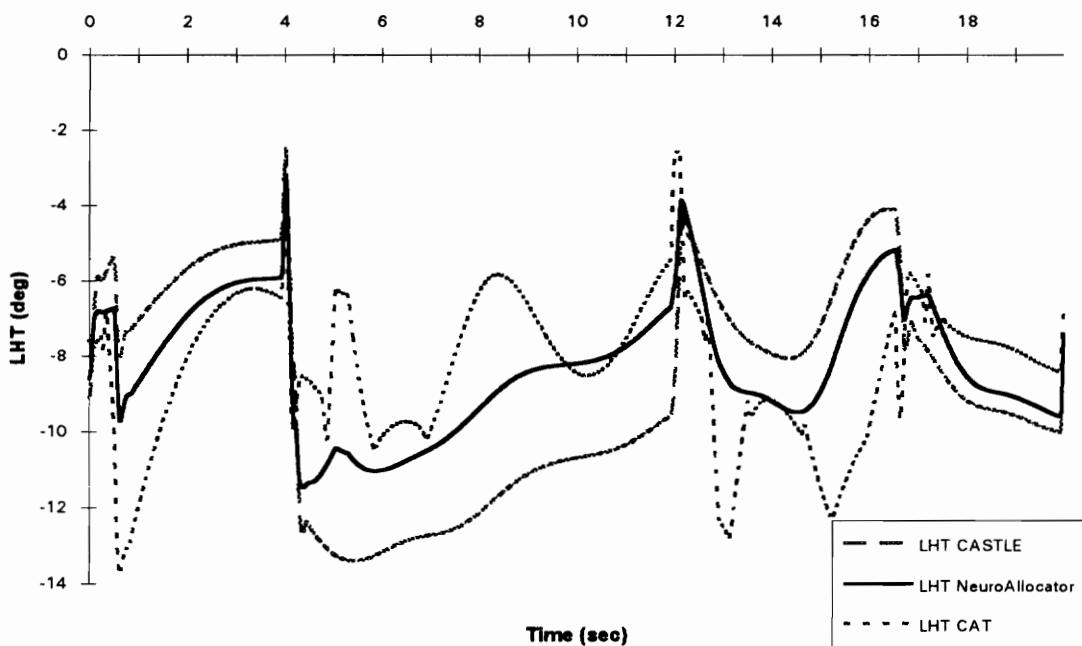


Figure 5.8.5: Left Horizontal Tail Deflection

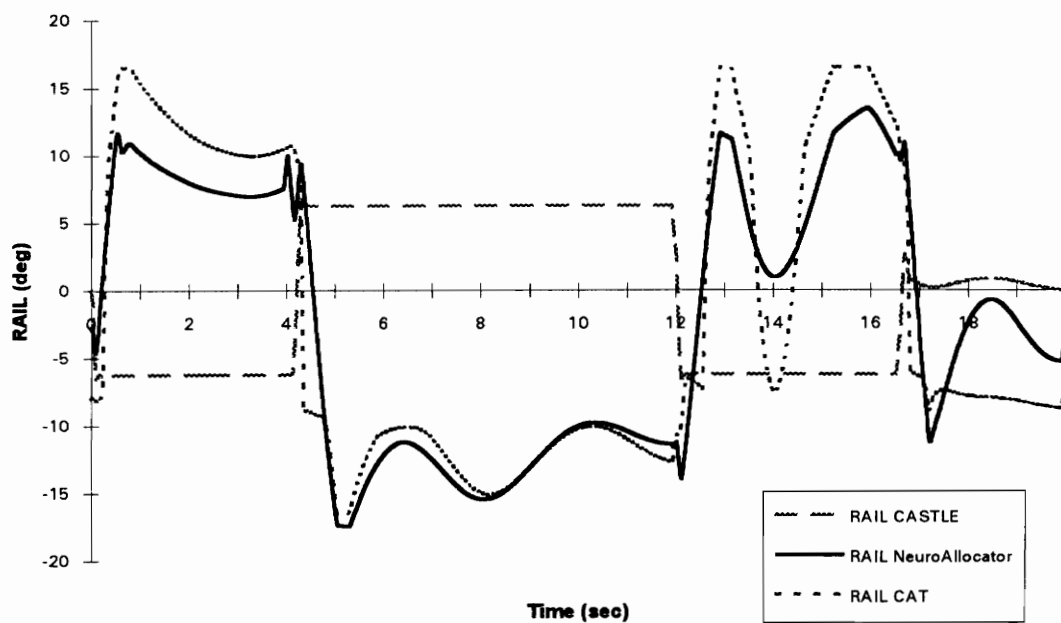


Figure 5.8.6: Right Aileron Deflection

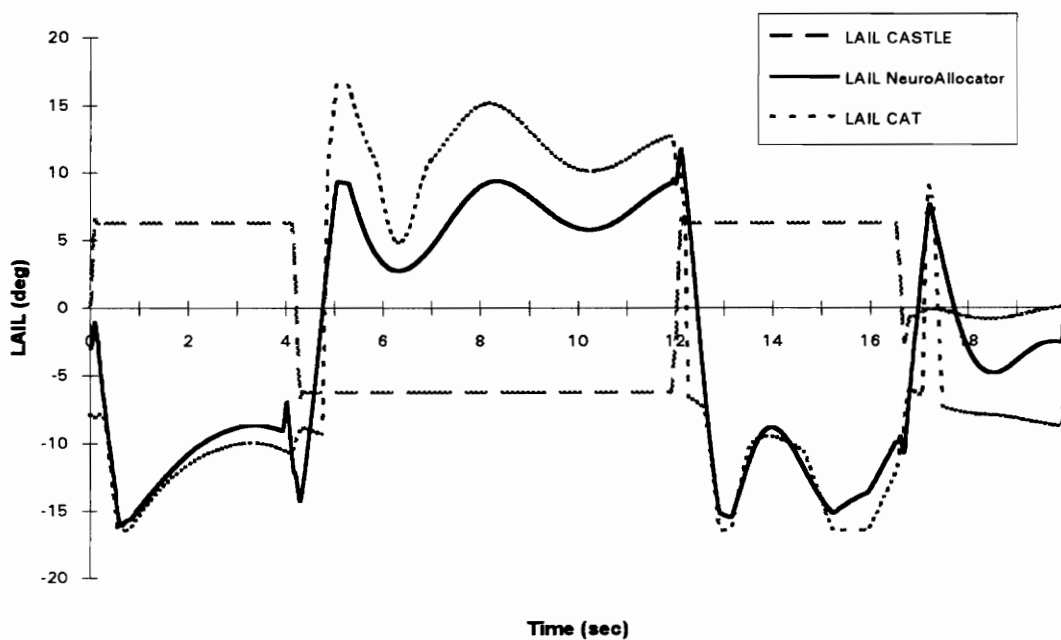


Figure 5.8.7: Left Aileron Deflection

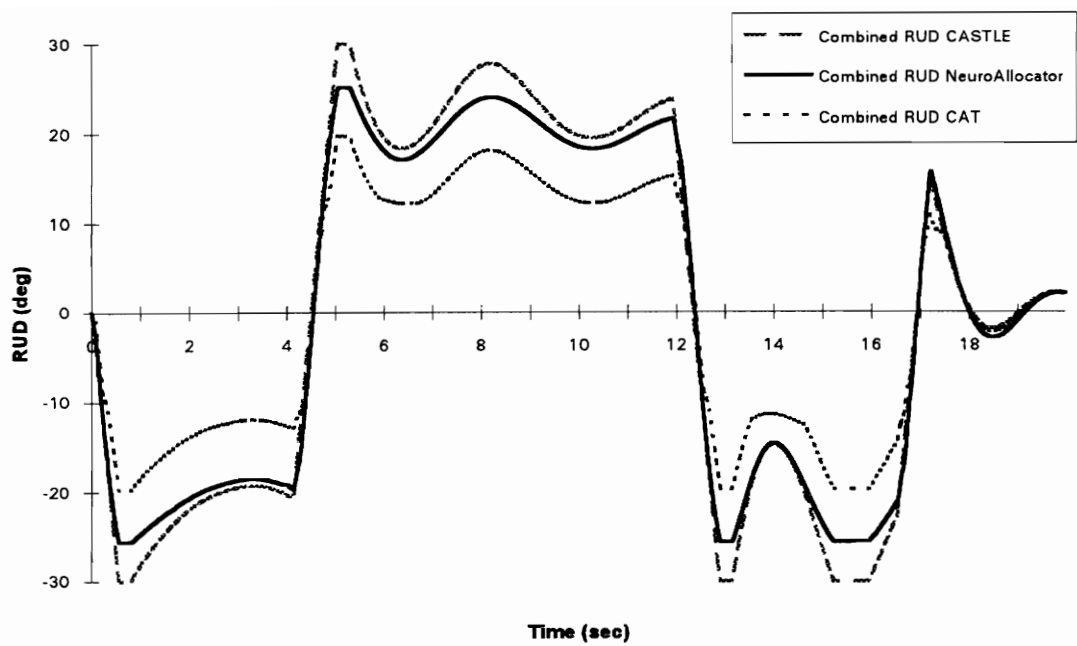


Figure 5.8.8: Combined Rudder Deflection

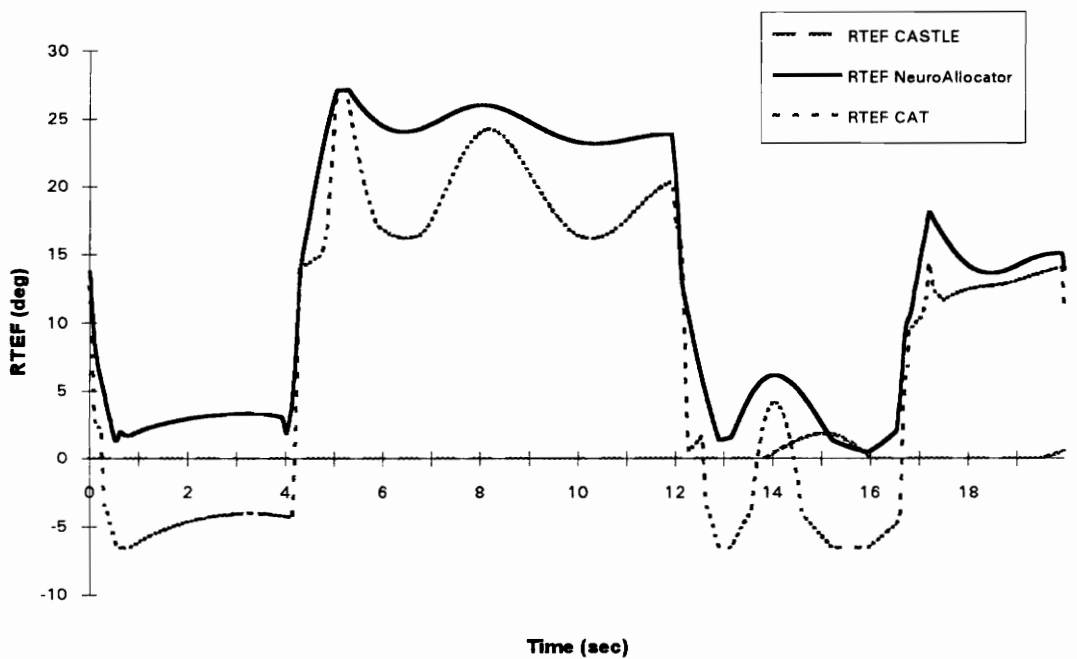


Figure 5.8.9: Right Trailing Edge Flap Deflection

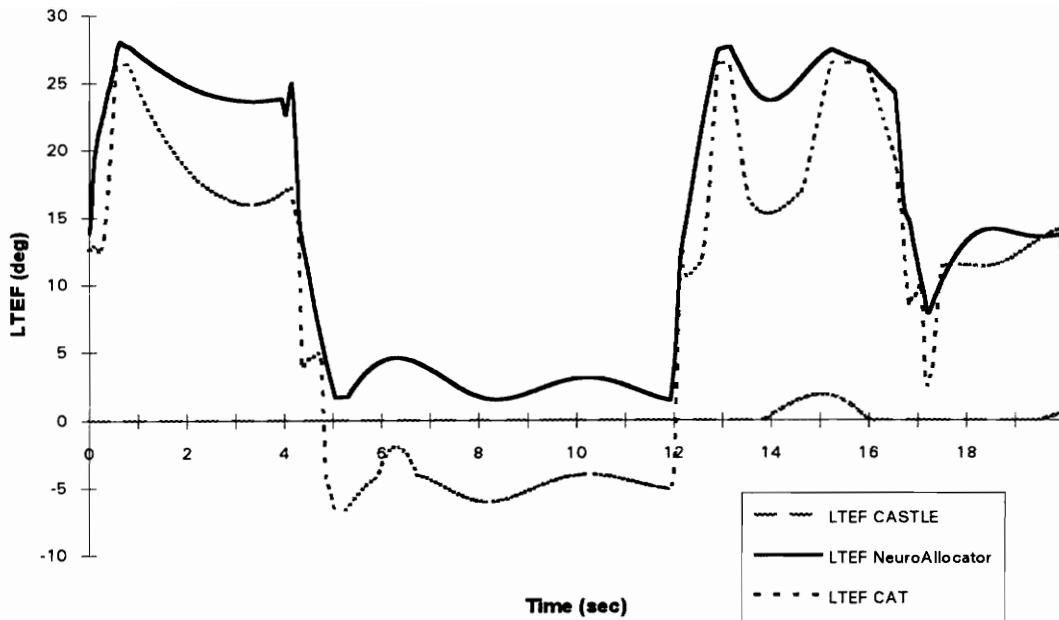


Figure 5.8.10: Left Trailing Edge Flap Deflection

The ability of the NeuroAllocator to obtain the same resolution of CAT was limited by the grid density interior to the surface of the AMS. Figure 5.8.11 illustrates a cross section of an AMS spanned by a two dimensional training grid. A desired moment is shown at three different magnitudes in the same direction. CAT determines the direction of any desired moment and finds the corresponding control associated with the intersection of the desired moment with the bounding surface of the AMS. In other words, interior to the surface of the AMS are miniature scaled representations of the bounding surface. The facets on the bounding surface, which are defined by at least 4 training points at the vertices, become smaller and smaller with decreasing distance to the origin. Thus, facets on a smaller surface on a miniature AMS near the origin are not well defined by this type of grid and more than one facet will lie between two grid points. The resolution limits were

particularly apparent for the test maneuver since the desired moments were well interior to the bounding surface of the AMS.

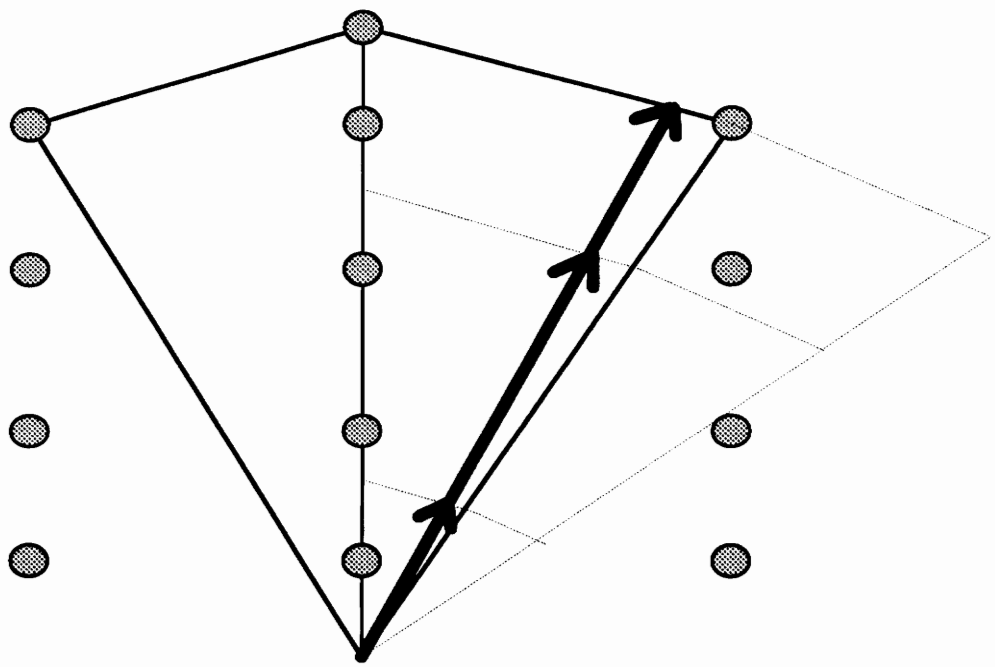


Figure 5.8.11: Grid Density Resolution Limits

6.0 Summary and Recommendations

The described extended delta-bar-delta backpropagation neural network has been successfully applied to the stated flight control allocation problem. The control allocation problem has been shown to be well suited for a neural network solution, since the input space is described by a closed and convex attainable moment subset, if the controls are independent.

Guidelines have been developed by heuristic arguments, which were experimentally verified, for the selection of neural network configurations and parameters. Specifically, a six hidden neuron neural network has been shown to have the ability to train efficiently, form an effective neural network representation of the AMS, and independently discover the internal relationships between moments and controls. These general guidelines may be extended to emulate any control allocation scheme for the solution of any flight control allocation problem having any number of flight control surfaces.

On the basis of the guidelines, a subroutine module has been created to implement the developed flight control NeuroAllocator in an existing flight simulator. The NeuroAllocator subroutine, that was used to demonstrate the advantages and disadvantages of the neural network, reallocated seven of the control surfaces in a real time test maneuver of the F/A-18 HARV.

The neural network, trained to emulate the direct allocation scheme implemented in CAT, has been found to have limitations arising from the ability of the training grid to achieve the resolution of CAT at low moment requirements. Computationally speaking, it is feasible to implement the direct method in real time. Since the neural network cannot attain the same performance, the direct method is not in danger of being replaced.

However, the direct method has been shown to have flaws associated with high actuator rate demands.

The most obvious recommendation to increase the performance of the neural network, pertaining to the emulation of the direct allocation scheme, is the development of a more sophisticated training grid generator. More resolution has been shown to be needed in the region of low moment demands near the origin of the AMS. Points could be distributed along vectors to the vertices and facet centers. This would guarantee all facets on the surface and miniature facets within the AMS are clearly defined. The spacing of the grid points along each vector would have to be addressed. Additionally, this would eliminate the clustering of training points on the bounding surface of the AMS.

A less obvious modification would be an increase in the slope of the hyperbolic tangent squashing function to increase the performance of the NeuroAllocator. It has been suggested [13] that this increase in slope would partially correct the magnitude error seen in the moment coefficient time histories since the neural network has matched the correct phase.

The increased resolution needed by the direct allocation scheme was accompanied by a high demand on actuator rates. Although, the neural network could not exactly attain the desired moments for the test maneuver, the nature of the neural network to generalize tended to limit rate demands. A parallel investigation has led to a blended solution using the pseudo-inverse for low moment demands and the direct method for high moment demands that the pseudo-inverse cannot attain [16]. In the investigation, low actuator rate requirements have been found to occur at low moment demands for the pseudo-inverse and at high moment demands for the direct method.

Large dense training grids would not be necessary to learn the pseudo-inverse relationship between moments and controls, and a neural network has been shown to attain the necessary resolution for high moment demands near the bounding surface of the AMS. The elegance of the neural network to condense complex algorithms into simple equations would be useful for the emulation of variations on the direct method that are computationally cumbersome and difficult to realize in real time, such as the above.

Future investigations should explore the adaptive nature of neural networks. In practice, the effectiveness of the controls (the B matrix) is not constant, but varies with flight condition. This problem could be addressed by additional inputs corresponding to the major contributors in this variation, such as airspeed and angle of attack. The direct method would have to use a look-up table to find the moment coefficient derivatives and recalculate the AMS for each flight condition. To recalculate the AMS on the fly would be difficult in real time. Thus, a neural network could be trained off-line on attainable moment subsets at strategic locations throughout the flight envelope. In essence, this would require more dimensions to be incorporated into the training grid to account for the entire flight envelope. Perhaps, configurations that feature breaking the neural network into a set of parallel networks, one for the moments, which could utilize the previous results and one for the flight condition, should be examined.

Another feature of the adaptability of the neural network that may be exploited may account for arising uncertainties in control effectiveness data. A neural network could be trained off-line on existing control effectiveness data. A second neural network could be linked in parallel to the first and trained on-line to compensate for modeling errors and parameter drift in real time, similar to a method that has been used for feedback linearization in aircraft control [9]. Conversely, the neural network trained on-line could

be used to extract more accurate estimates of control effectiveness data for implementation in other allocation schemes.

The performance of alternative neural network types should be addressed, such as the Radial Basis Function (RBF) and Self-Organizing Map (SOM) architecture. The RBF neural network features an internal representation of hidden processing elements that are radially symmetric and has recently gained popularity in applications in which backpropagation is typically employed. RBF neural networks have been shown to have properties that offer many advantages over backpropagation [7]. Among these advantages include: faster training, superior learning of decision boundaries, especially in bounded regions, and improved extrapolation in regions of sparse training data. SOM can be used as a front end to a backpropagation neural network [7]. The SOM neural network offers advantages beyond the scope of this thesis.

Other issues of note include: stability and robustness, control failure accommodation, the number of computations required as compared to other allocation schemes, and adaptability to the requirements of various flight phases and aircraft classifications.

In closing, it is the author's opinion that neural networks have enormous potential in the field of aircraft control. Research in this area is in its infancy. Future advanced aircraft may feature intelligent flight control systems entirely composed of neural networks. It is hoped that this work has accomplished the goal of providing future researchers with a guide that may serve as the foundation for the development of a more advanced and realistic NeuroAllocator that may become a part of large neural network flight control system.

REFERENCES

1. Durham, Wayne C., "Constrained Control Allocation," *Journal of Guidance, Control, and Dynamics*, Volume 16, Number 4, July-August 1993, Pages 717-725.
2. Durham, Wayne C., "Constrained Control Allocation: Three Moment Problem," *Journal of Guidance, Control, and Dynamics*, Volume 17, Number 2, January-February 1994, Pages 330-336.
3. Durham, Wayne C., "Attainable Moments for the Constrained Control Allocation Problem," accepted for publication, *Journal of Guidance, Control, and Dynamics*, April 1994.
4. Durham, Wayne C., "CAT: Control Allocation Toolbox," Application Software for the Solution and Visualization of Constrained Flight Control Allocation Problems, Department of Aerospace Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, Copyright © 1993, by Wayne C. Durham.
5. Caudill, Maureen, "Neural Networks Primer," *AI Expert*, Miller Freeman Publications, San Francisco, CA, 1990.
6. Wasserman, Philip D., *Neural Computing*, Van Nostrand Reinhold, New York, 1989.
7. NeuralWare, Inc., "NeuralWorks Users Manual", Pittsburgh, PA, 1993.
8. DiGirolamo, R., "Flight Control Law Synthesis Using Neural Network Theory," AIAA-92-4390-CP, *Proceeding of the 1992 AIAA Guidance, Navigation, and Control Conference*, Hilton Head Island, South Carolina, August 1992.
9. Calise, Anthony J., Kim, Byoung Soo, Kam, Moshe, Azam, Misbahul, "Neural Networks for Feedback Linearization in Aircraft Control," AIAA-92-4391-CP, *Proceeding of the 1992 AIAA Guidance, Navigation, and Control Conference*, Hilton Head Island, South Carolina, August 1992.
10. Ahmed-Zaid, F., Ioannou, P. A., Polycarpou, M. M., Youssef, H. M., "Identification and Control of Aircraft Dynamics Using Radial Basis Function Neural Networks," AIAA-92-4393-CP, *Proceeding of the 1992 AIAA Guidance, Navigation, and Control Conference*, Hilton Head Island, South Carolina, August 1992.

11. Williams, Kenneth E., "Prediction of Solar Activity With A Neural Network And Its Effect On Orbit Prediction," *Johns Hopkins APL Technical Digest*, Volume 12, Number 4, 1991.
12. Kalman, Barry L., Kwasny, Stan C., "Why Tanh: Choosing a Sigmoidal Function," *Proceedings of the 1992 International Joint Conference on Neural Networks*, Volume 4, Baltimore, Maryland, June 1992.
13. Ramu, Krishnan, Private Consultations, September 1993 - May 1994.
14. Wolfram, Stephen, *Mathematica*, Second Edition, Addison-Wesley Publishing Company, Inc., 1991.
15. Keiser, John, "Constrained Control Optimization Study and Solution Implementation," Project Report, Department of Aerospace Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, May 1993.
16. Bordignon, Kenneth A., Durham, Wayne C., "Closed-Form Solutions to the Constrained Control Allocation Problem," to be presented as paper AIAA-94-3552 at the *1994 AIAA Guidance, Navigation, and Control Conference*, Scottsdale, Arizona, August 1994.

Appendix A. Glossary of Flight Control Terminology

AMS	Attainable Moment Subset, Φ
B	Matrix of control effectiveness
CAS	Control Augmentation System
CASTLE	Controls and Simulation Test Loop Environment
CAT	Control Allocation Toolbox: Application Software for the Solution and Visualization of Constrained Flight Control Allocation Problems
C_l	Rolling Moment Coefficient
C_m	Pitching Moment Coefficient
C_n	Yawing Moment Coefficient
Face	A geometric feature of the constrained control and attainable moment subsets. Faces are generated in m-space by placing all but two of the m controls at either of their two constraint and allowing two to vary. Faces in the attainable moment subset are the images of the faces in the constrained control subset.
Facet	Any face that lies on the boundary of the constrained control or attainable moment subsets. All faces in the constrained control subset are facets, but a face in the attainable moment subset may or may not be a facet.
HARV	F/A-18 High Angle of Attack Research Vehicle
LAİL	Left Aileron Control Surface Deflection

LHT	Left Horizontal Tail Control Surface Deflection
LTEF	Left Trailing Edge Flap Control Surface Deflection
m	Number of controls
M	Vector of moments
n	Dimension of moment space, 3
Node	A geometric feature of the constrained control and attainable moment subsets. Nodes are generated in m-space by placing each of the m controls at either of its two constraints, yielding 2^m nodes in the subset of constrained controls. Nodes in the attainable moment subset are the images of the nodes in the constrained control subset.
Ω	Constrained Control Subset
RAIL	Right Aileron Control Surface Deflection
RHT	Right Horizontal Tail Control Surface Deflection
RTEF	Right Trailing Edge Flap Control Surface Deflection
RUD	Rudder Control Surface Deflection
u	Vector of controls
Vertex	Any node that lies on the boundary of the constrained control or attainable moment subsets. All nodes in the constrained control subset are vertices, but a node in the attainable moment subset may or may not be a vertex.

Appendix B. Glossary of Neural Network Terminology

Adaptability The ability to self-adjust in response to external stimuli.

Backpropagation An iterative training procedure that involves the minimization of the total squared error between the actual and desired outputs of a training set, with respect to the neural network weights.

Cumulative Backpropagation Backpropagation training procedure where weight updates are accumulated over an entire training epoch.

Decision Boundary Boundaries by which output decisions are made by separating regions of an input space.

Delta Weight Weight update.

EDBD Extended Delta-Bar-Delta: a modified backpropagation learning rule using individual time varying learning coefficients for each neural network weight.

Epoch One full pass through all training patterns in a training set.

Generalization The ability of a neural network produce an output from a previously unseen input on the basis of learned input-output examples.

Hidden Layer Any layer in a neural network that is not an input or output layer.

Hidden Neuron A neuron in the hidden layer.

Learning Rule Algorithm determining how weights adapt in response to a learning example.

NeuroAllocator	Flight control allocator module using a neural network, trained to emulate a desired control allocation scheme.
Neuron	The basic building block of a neural network, analogous to the biological neuron nerve cell. Neural networks are composed of highly interconnected neurons.
NN-AMS	Neural network representation of the attainable moment subset
Perceptrons	The original artificial neural network, similar to a neuron with the exception of a predetermined threshold replacing the squashing function, usually organized into a single layer neural network.
RBF	Radial Basis Function neural network architecture
RMS	Root-Mean-Square
SOM	Self-Organizing Map neural network architecture
Squashing Function	A threshold function or a continuous function operating on the weighted sum of the input to a neuron.
Supervised Learning	Training such that a neural network adapts to create a specified input-output mapping.
Threshold	Constant used as a comparison level by a variable.
Training Pattern	An input-output example for training a neural network.
Training Set	The collection of all training patterns used to train a neural network.

Unsupervised Learning Training such that a neural network organizes to respond to certain inputs.

Weight Interconnection between two neurons.

Weight Layer Matrix of weights connecting two neural network layers.

Appendix C. Sample NeuroAllocator Subroutine

Enclosed in Appendix C is the FORTRAN code for the NeuroAllocator subroutine module described in detail in Chapter 5. The subroutine accepts a vector of control deflections from the original CASTLE CAS control law as input and reallocates the control surface deflections using a trained neural network that emulates the direct allocation scheme implemented in CAT. Each time the driver program calls the NeuroAllocator module, the subroutine reads the next sequence of ten control surface deflections from a user defined input file containing the time history of control deflections for a desired test maneuver. The desired moment associated with the original control deflections is calculated and input to the NeuroAllocator. Because of space limitations the equations included here are only for the neural network trained on Grid B for 13 epochs that was used to obtain the results presented in Section 5.8. The original desired moment, reallocated controls, and moment achieved by the reallocated controls are output from the subroutine for comparison.

```

subroutine neural(dlhtd,drhtd,dlad,drad,drudd,dtefl,dtefr,dlefl,dlefr,drudl,drudr,
$              nnlhtd,nnrhtd,nnlad,nnrad,nnrudd,nntefl,nntefr,
$              yin,nnmom,ilshot,flaps,time)
implicit none
real*4 dlhtd,drhtd,dlad,drad,drudd,dtefl,dtefr
real*4 dlefl,dlefr,drudl,drudr
real*4 nnlhtd,nnrhtd,nnlad,nnrad,nnrudd,nntefl,nntefr
real*4 yin(3),yout(7),nnmom(3),uin(7),xout(38),time
integer*4 flaps,ilshot,grid

```

```

c
c The following are the control deflections from the original CASTLE CAS control laws
c dlhtd = left horizontal tail deflection
c drhtd = right horizontal tail deflection
c dlad = left aileron deflection
c drad = right aileron deflection
c drudl = left rudder deflection
c drudr = right rudder deflection
c drudd = combined rudder deflection
c dtefl = left trailing edge flap deflection
c dtefr = right trailing edge flap deflection
c
c The following are the allocated control deflections output from the neural network
c nnlhtd = left horizontal tail deflection
c nnrhtd = right horizontal tail deflection
c nnlad = left aileron deflection
c nnrad = right aileron deflection
c nnrudd = combined rudder deflection
c nntefl = left trailing edge flap deflection
c nntefr = right trailing edge flap deflection
c
c subroutine input:
c dlhtd,drhtd,etc. are the controls for a maneuver read from file bull.dat
c subroutine asks user if trailing edge flaps are being used
c flap information is input by user on first pass
c
c subroutine output:
c nnlhtd,nnrhtd,etc. are the neuro-allocated controls, also in yout array
c yin are the moments gotten from multiplying the appropriate B matrix
c with the input control - the CASTLE moments
c nnmom are the moments gotten from multiplying the appropriate B matrix
c with the neural-allocated controls - the NN moments
c

```

```

if (ilshot.eq.0) then
  open (99,file='control.dat',status='unknown')
  write(*,*) 'use trailing edge flaps? (1=yes,0=no)'
  read(5,*) flaps
  write(*,*) 'pick a grid'
  read(5,*) grid
endif
read (99,*) time,dlefl,dlefr,dtefl,dtefr,dlad,
$drad,dlhtd,drhtd,drudl,drudr
c
  drudd=(drudl+drudr)*0.5
  uin(1)=dlhtd
  uin(2)=drhtd
  uin(3)=dlad
  uin(4)=drad
  uin(5)=drudd
  uin(6)=dtefl
  uin(7)=dtefr
c=====
c pre-multiply input control deflections with B to get the
c moments required for desired maneuver for input to neural network
  call bwack(uin,yin,flaps)
c=====
  if (flaps.eq.0) then
c    5u NN code goes here
c=====
    else
c=====
c    7u NN code goes here
    iff(grid.eq.1)then
c These equations are for the neural network trained on Grid B for 13 epochs
c and used to obtain the results presented in Chapter 5
c Read and scale input into network
      Xout(2) = Yin(1) * (10.25641) + (0.025641039)
      Xout(3) = Yin(2) * (3.4188034) + (-0.35042737)
      Xout(4) = Yin(3) * (36.346455)

c Code for NEURON 1 in hidden layer
      Xout(5) = (-0.031317133) + (-0.32951489) * Xout(2) +
/      (-0.93620998) * Xout(3) + (0.12593168) * Xout(4)
      Xout(5) = tanh( Xout(5) )

```

c Code for NEURON 2 in hidden layer

$$\begin{aligned} \text{Xout}(6) &= (0.0023961561) + (0.79035228) * \text{Xout}(2) + \\ &/ \quad (-0.44962245) * \text{Xout}(3) + (0.24002163) * \text{Xout}(4) \\ \text{Xout}(6) &= \tanh(\text{Xout}(6)) \end{aligned}$$

c Code for NEURON 3 in hidden layer

$$\begin{aligned} \text{Xout}(7) &= (0.067462869) + (-0.12598433) * \text{Xout}(2) + \\ &/ \quad (-0.32188138) * \text{Xout}(3) + (0.3785848) * \text{Xout}(4) \\ \text{Xout}(7) &= \tanh(\text{Xout}(7)) \end{aligned}$$

c Code for NEURON 4 in hidden layer

$$\begin{aligned} \text{Xout}(9) &= (0.031298164) + (-0.87158149) * \text{Xout}(2) + \\ &/ \quad (0.468503) * \text{Xout}(3) + (0.79977751) * \text{Xout}(4) \\ \text{Xout}(9) &= \tanh(\text{Xout}(9)) \end{aligned}$$

c Code for NEURON 5 in hidden layer

$$\begin{aligned} \text{Xout}(10) &= (-0.021355228) + (-0.86691993) * \text{Xout}(2) + \\ &/ \quad (-0.59443933) * \text{Xout}(3) + (-0.26715013) * \text{Xout}(4) \\ \text{Xout}(10) &= \tanh(\text{Xout}(10)) \end{aligned}$$

c Code for NEURON 6 in hidden layer

$$\begin{aligned} \text{Xout}(11) &= (-0.0051007429) + (0.54082954) * \text{Xout}(2) + \\ &/ \quad (-0.26232728) * \text{Xout}(3) + (0.79318631) * \text{Xout}(4) \\ \text{Xout}(11) &= \tanh(\text{Xout}(11)) \end{aligned}$$

c Code for NEURON 1 in output layer

$$\begin{aligned} \text{Xout}(35) &= (0.043324705) + (0.46323586) * \text{Xout}(5) + \\ &/ \quad (0.46617943) * \text{Xout}(6) + (0.07843034) * \text{Xout}(7) + \\ &/ \quad (-0.56374753) * \text{Xout}(9) + (0.16793905) * \text{Xout}(10) + (0.26864043) * \text{Xout}(11) \\ \text{Xout}(35) &= \tanh(\text{Xout}(35)) \end{aligned}$$

c Code for NEURON 2 in output layer

$$\begin{aligned} \text{Xout}(36) &= (0.069513872) + (0.74444646) * \text{Xout}(5) + \\ &/ \quad (0.058761761) * \text{Xout}(6) + (0.3167201) * \text{Xout}(7) + \\ &/ \quad (-0.05283942) * \text{Xout}(9) + (0.65420818) * \text{Xout}(10) + (0.05478288) * \text{Xout}(11) \\ \text{Xout}(36) &= \tanh(\text{Xout}(36)) \end{aligned}$$

c Code for NEURON 3 in output layer

$$\begin{aligned} \text{Xout}(37) &= (-0.0057014348) + (0.10217991) * \text{Xout}(5) + \\ &/ \quad (0.26666474) * \text{Xout}(6) + (-0.12476267) * \text{Xout}(7) + \\ &/ \quad (-0.99169153) * \text{Xout}(9) + (0.0897654) * \text{Xout}(10) + (-0.0867637) * \text{Xout}(11) \\ \text{Xout}(37) &= \tanh(\text{Xout}(37)) \end{aligned}$$

c Code for NEURON 4 in output layer

```
Xout(8) = (-0.010617553) + (0.53805619) * Xout(5) +  
/ (0.024169981) * Xout(6) + (0.36718798) * Xout(7) +  
/ (0.69565201) * Xout(9) + (0.45057669) * Xout(10) + (0.35412234) * Xout(11)  
Xout(8) = tanh( Xout(8) )
```

c Code for NEURON 5 in output layer

```
Xout(14) = (0.034770161) + (-0.033673961) * Xout(5) +  
/ (-0.24017468) * Xout(6) + (-0.19916892) * Xout(7) +  
/ (-0.60106784) * Xout(9) + (0.27526799) * Xout(10) + (-0.7800905) * Xout(11)  
Xout(14) = tanh( Xout(14) )
```

c Code for NEURON 6 in output layer

```
Xout(17) = (-0.12954383) + (-0.31490016) * Xout(5) +  
/ (0.36128873) * Xout(6) + (0.036798634) * Xout(7) +  
/ (-0.18624724) * Xout(9) + (-0.7580961) * Xout(10) + (0.52218318) * Xout(11)  
Xout(17) = tanh( Xout(17) )
```

c Code for NEURON 7 in output layer

```
Xout(16) = (-0.039411657) + (-0.039239563) * Xout(5) +  
/ (-0.65613067) * Xout(6) + (-0.1111136) * Xout(7) +  
/ (0.29073375) * Xout(9) + (0.4273434) * Xout(10) + (-0.51065058) * Xout(11)  
Xout(16) = tanh( Xout(16) )
```

c De-scale and write output from network

```
Yout(1) = Xout(35) * (21.5625) + (-6.75)  
Yout(2) = Xout(36) * (21.5625) + (-6.75)  
Yout(3) = Xout(37) * (31.25)  
Yout(4) = Xout(8) * (31.25)  
Yout(5) = Xout(14) * (37.499999)  
Yout(6) = Xout(17) * (31.25) + (15)  
Yout(7) = Xout(16) * (31.25) + (15)  
endif
```

c

endif

c save neural allocated controls output from neural network

```
nnlhtd = yout(1)  
nnrhtd = yout(2)  
nnlad = yout(3)  
nnrad = yout(4)  
nnrudd = yout(5)
```

c

```

    if (flaps.eq.1) then
        nntefl = yout(6)
        nntefr = yout(7)
    endif

c=====
c pre-multiply neural network output control deflections with B to get the
c moments attained for desired maneuver for output
    call bwack(yout,nnmom,flaps)
c=====

    return
end

subroutine bwack(u,m,flaps)
    implicit none
    real*4 u(7),m(3),b(3,7)
    integer*4 flaps

c=====
c subroutine to multiply matrix B with vector u to get vector m
c m=Bu
c=====

c the B matrix for f/a-18 HARV with trailing edge flaps
    b(1,1)=6.47e-4
    b(1,2)=-6.47e-4
    b(1,3)=6.00e-4
    b(1,4)=-6.00e-4
    b(1,5)=5.83e-5
    b(1,6)=1.06e-3
    b(1,7)=-1.06e-3
    b(2,1)=-7.35e-3
    b(2,2)=-7.35e-3
    b(2,3)=-6.00e-4
    b(2,4)=-6.00e-4
    b(2,5)=0.00e+0
    b(2,6)=4.80e-4
    b(2,7)=4.80e-4
    b(3,1)=-1.00e-7
    b(3,2)=1.00e-7
    b(3,3)=-1.50e-4
    b(3,4)=1.50e-4
    b(3,5)=-6.67e-4
    b(3,6)=0.00e+0
    b(3,7)=0.00e+0

```

```
m(1)=b(1,1)*u(1)+b(1,2)*u(2)+b(1,3)*u(3)+b(1,4)*u(4)+b(1,5)*u(5)
m(2)=b(2,1)*u(1)+b(2,2)*u(2)+b(2,3)*u(3)+b(2,4)*u(4)+b(2,5)*u(5)
m(3)=b(3,1)*u(1)+b(3,2)*u(2)+b(3,3)*u(3)+b(3,4)*u(4)+b(3,5)*u(5)
```

c

```
if (flaps.eq.1) then
m(1)=m(1)+b(1,6)*u(6)+b(1,7)*u(7)
m(2)=m(2)+b(2,6)*u(6)+b(2,7)*u(7)
m(3)=m(3)+b(3,6)*u(6)+b(3,7)*u(7)
endif
```

```
return
end
```

Vita

The author was born on January 13, 1970, and raised in Scranton, Pennsylvania. He graduated with honors from Scranton Central High School in 1988 and went on to receive the degree of Bachelor of Science in Aerospace Engineering in May of 1992 from The Pennsylvania State University.

The author was granted membership in Tau Beta Pi, the National Engineering Honor Society; Sigma Gamma Tau, the National Honor Society in Aerospace Engineering; and the Golden Key National Honor Society during his undergraduate studies. Additionally, he is an active member in the American Institute of Aeronautics and Astronautics having: presented work related to his undergraduate thesis on the Thermally Induced Vibration of Flexible Spacecraft Solar Array Booms at the 1992 Mid-Atlantic Student Conference, and a paper related to this thesis accepted for presentation and publication in the 1994 AIAA Guidance, Navigation, and Control Conference and Proceedings.

This thesis partially fulfills the requirements towards the degree of Master of Science in Aerospace Engineering from the Virginia Polytechnic Institute and State University. Upon completion of his graduate studies in May 1994, the author will secure the position of Senior Engineer with AlliedSignal Aerospace Guidance and Control Systems in New Jersey.

A handwritten signature in black ink that reads "Robert L. Grogan". The signature is written in a cursive, flowing style with a large, prominent 'R' and 'G'.