AL-TR-1992-0037

AD-A254 653

ARMSTRONG LABORATORY

(2)

NEURAL NETWORK MODELS OF AIR COMBAT MANEUVERING

Roger W. Schvaneveldt
Alan E. Benson

New Mexico State University
Box 30001
Las Cruces, NM 88003

Timothy E. Goldsmith

University of New Mexico
Albuquerque, NM 87131

Wayne L. Waag

HUMAN RESOURCES DIRECTORATE
AIRCREW TRAINING RESEARCH DIVISION
Williams Air Force Base, AZ 85240-6457

July 1992

Final Technical Report for Period September 1988 - December 1991

AIR FORCE MATERIEL COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5000

NOTICES

Wayne L. Waag

WAYNE L. WAAG
Project Scientist

Dee H. Andrews

DEE H. ANDREWS, Technical Director
Aircrew Training Research Division

[illegible]

W. M. KORNOVICH, JR., LTC, USAF
Deputy Chief, Aircrew Training Research Division

REPORT DOCUMENTATION PAGE	Form Approved OMB No 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE July 1992	3. REPORT TYPE AND DATES COVERED Final – September 1988 – December 1991

4. TITLE AND SUBTITLE Neural Network Models of Air Combat Maneuvering	5. FUNDING NUMBERS C - F33615-88-C-0010 PE - 62205F PR - 1123 TA - 35 WU - 09
6. AUTHOR(S) Roger W. Schvaneveldt Alan E. Benson Timothy E. Goldsmith Wayne L. Waag	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) New Mexico State University Box 30001 Las Cruces, NM 88003	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Armstrong Laboratory Human Resources Directorate Aircrew Training Research Division Williams Air Force Base, AZ 85240-6457	10. SPONSORING/MONITORING AGENCY REPORT NUMBER AL-TR-1992-0037
11. SUPPLEMENTARY NOTES Armstrong Laboratory Contract Monitor: Wayne L. Waag, (602) 988-6561	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	12b DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The primary goal of this project was to explore the applicability of artificial neural network (NN) models in the domain of air combat maneuvering (ACM). The work investigated several models. (a) NN models that select ACM on the basis of training with the production rules of a model, Air Combat Expert Simulation (ACES), (b) NN models that mimic the action selections of the Automated Maneuvering Logic (AML) System, (c) NN models that predict the outcome of engagements flown in the Simulator for Air-to-Air Combat (SAAC) given summary measures of various parameters measured during the engagements, and (d) NN models that predict future aircraft control inputs in SAAC engagements given the values of flight parameters at particular points in time.

These various models incorporate knowledge about air combat maneuvers and components of maneuvers as well as rudimentary knowledge about maneuver planning and situational awareness. For most of the models, validation tests were conducted using data different from that used in training the models. The authors provide details on each of these efforts as well as a review of the ACES model, a presentation of the basics of NNs, and an overview of a software system developed for the implementation and testing of the NN models

14. SUBJECT TERMS Air combat Air combat maneuvering Flight training	Flight simulation Flight simulators Neural networks	Performance measurement	15. NUMBER OF PAGES 64 16 PRICE CODE
17 SECURITY CLASSIFICATION OF REPORT Unclassified	18 SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19 SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20 LIMITATION OF ABSTRACT UL

Contents

Tables

PREFACE

This final technical report for the Air Combat Maneuvering Expert System (ACMES) Program Research and Development Announcement (PRDA) was prepared for the Armstrong Laboratory, Human Resources Directorate, Aircrew Training Research Division (AL/HRA) under Contract Number F33615-88-C-0010 with New Mexico State University (NMSU). The primary goal of this project was to explore the applicability of artificial neural network models in the domain of air combat maneuvering. The research was conducted under Work Unit 1123-35-09, Expert Pilot Decision Making Models. Contract monitor was Dr. Wayne L. Waag (AL/HRA).

NEURAL NETWORK MODELS OF AIR COMBAT MANEUVERING

Summary

The primary goal of this project was to explore the applicability of artificial neural network (NN) models in the domain of air combat maneuvering (ACM). The work investigated several models: (a) NN models that select ACM on the basis of training with the production rules of a model, Air Combat Expert Simulation (ACES); (b) NN models that mimic the action selections of the Automated Maneuvering Logic (AML) System; (c) NN models that predict the outcome of engagements flown in the Simulator for Air-to-Air Combat (SAAC) given summary measures of various parameters measured during the engagements; and (d) NN models that predict future aircraft control inputs in SAAC engagements given the values of flight parameters at particular points in time.

These various models incorporate knowledge about air combat maneuvers and components of maneuvers as well as rudimentary knowledge about maneuver planning and situational awareness. For most of the models, validation tests were conducted using data different from that used in training the models. The authors provide details on each of these efforts as well as a review of the ACES model, a presentation of the basics of NNs, and an overview of a software system developed for the implementation and testing of the NN models.

Introduction

Problem

Although a great deal of research has been conducted on the perceptual-motor skills of fighter plane pilots, the cognitive and decision-making skills have received relatively little attention. A better understanding of cognitive skills can make important contributions to pilot training and to various aspects of performance assessment. In previous work, we have collected data and developed models relating to pilots' cognitive structures and to the selection of maneuvers in air-to-air combat (Goldsmith, Schvaneveldt, & Brunderman, 1985; Schvaneveldt, Anderson, Breen, Cooke, Goldsmith, Durso, DeMaio, & Tucker, 1983; Schvaneveldt, Durso, Goldsmith, Breen, Cooke, Tucker, & DeMaio, 1985; Schvaneveldt, Goldsmith, Durso, Maxwell, Acosta, & Tucker, 1982). While this research has yielded interesting results, there is still much work to be done to develop better and more complete models. In this project, we explore the applicability of artificial neural net (NN) models to the domain of air-combat maneuvering.

Research Goals

Several goals were pursued in this research. First, we needed software tools for the implementation and testing of NN models. The special tools were required to accommodate the particular forms of data used to train and test the networks. Standard, off-the-shelf, software packages were available, but the programming required to get the data into the required form rivaled the effort required to produce an NN tool itself. To achieve maximum control over the nature of our models and the format of data, we chose to develop the software tools as part of the project.

A second goal was to investigate the representation of a production system model in an NN architecture. Classical production systems are often described as "brittle." Such systems only work under conditions that are explicitly represented in the model. If events lead to conditions that were not explicitly anticipated, the model simply fails. NN models, in contrast, exhibit "graceful degradation." Such systems will generally produce reasonable responses even when conditions change, particularly if there is considerable similarity between a previously learned situation and a new situation. NN models are also capable of learning to improve performance based on encounters with new situations, whereas production systems must be improved by adding new rules or by modifying old ones. In sum, there are several reasons to explore the training of an NN model using the rules of a production system. This work is reported in Study 1.

As a further investigation of the applicability of NN models to the representation of the principles embodied in other models, Study 2 trained an NN model using the performance of the Automated Maneuvering Logic (AML) as a teacher. The goal was to determine the extent to which the actions selected by AML could be incorporated into an NN model.

Finally, we also pursued the goal of applying NN models to data obtained from the Simulator for Air-to-Air Combat (SAAC) at Luke AFB, AZ. The SAAC system supplies detailed information about air-to-air engagements flown against a human or a programmed adversary. An application of NN models to performance measurement and a comparison of NN models with standard statistical data analysis is reported in Study 3. Study 4 attempts to develop models to predict the future values of control variables (i.e., g loading and roll angle) using current values of various state variables.

Background

Summary of Air Combat Expert Simulation

Air Combat Expert Simulation (ACES) is a computer simulation of expert fighter pilots' cognitive skill in air-combat maneuvering (ACM). ACES selects single air combat maneuvers for particular airspace situations. This work has been fully reported in Goldsmith, et al. (1985). The ACES project had its roots in earlier research by Schvaneveldt and his colleagues into the nature of cognitive skills of fighter pilots (Schvaneveldt, et al., 1982, 1983, 1985). This research examined the underlying conceptual framework used by pilots in performing ACM. Both experts and novices were studied, and an attempt was made to uncover the cognitive dimensions on which these two groups differed. Conceptual structures were represented by multidimensional spaces and Pathfinder networks (see Schvaneveldt, Durso, & Dearholt, 1989; Dearholt & Schvaneveldt, 1990 for the definition of Pathfinder networks and a discussion of their properties). ACES built upon this earlier research by representing the decision-making processes involved in selecting maneuvers in air combat in a computer model. Techniques of knowledge engineering in artificial intelligence (AI) and expert-novice research in cognitive psychology were employed to derive the model.

ACES uses a production system architecture to select basic fighter maneuvers (BFM) in a one-versus-one situation. The database of the production system represents an airspace situation at a particular point in time. Particular maneuvers are related to situations by way of a set of condition-action pairs called production rules. The condition side of the rules specifies a set of flight characteristics for each aircraft; the action side selects a particular maneuver. Table 1 shows the airspace state information referenced by the condition sides of ACES production rules, and Table 2 shows the maneuvers selected by the rules. A sample production rule is shown in Table 3.

Table 1. Conditions Referenced in ACES Production Rules.

My:	Bogey's:	Relative:
aspect angle	aspect angle	angle off
3-9 position	3-9 position	slant range
clock position	clock position	closure rate
altitude		altitude difference
airspeed		

Table 2. Actions (Maneuvers) Selected by ACES Production Rules.

AC	Accelerate	LT	Lead Turn
BR	Barrel Roll	LY	Low Yo Yo
BT	Break Turn	NT	Nose Low Slice Turn
DT	Defensive Turn	OT	Optimum Turn Vertical
FG	Fire Guns	QP	Quarter Plane
FM	Fire Missile	RV	Reversal
HR	High AOA Roll	SI	Scissors
HY	High Yo Yo	SN	Separation
LR	Lag Roll		

Table 3. An Example of an ACES Production Rule.

If (conditions)

my aspect angle is between 120 and 180 degrees
bogey's aspect angle is between 0 and 60 degrees
we are within 1,000 feet of altitude
slant range is between 1,000 and 9,000 feet
angle off is between 0 and 45 degrees

then (action)

select break turn

A validation study was conducted comparing the selections of ACES rules to the selections made by National Guard pilots at Kirtland Air Force Base, NM. Forty scenarios were shown to the pilots who were asked to select their top three choices (provided that there were as many as three reasonable choices) from the list of maneuvers (Table 2). These same scenarios were run through the ACES production rules, and the top three selections of ACES were identified. Table 4 summarizes these results. Individual pilots agree with the group consensus about the best maneuver more than ACES agrees with the group consensus. Interestingly, however, ACES shows more overlap with the set of maneuvers found in the top three choices than do the selections of individual pilots. These results suggest that ACES has some difficulty in properly ordering the alternatives, but it does quite well at identifying the best alternatives.

Table 4. Results of the ACES validation study.

Comparisons of group - individual maneuver selections	Individual pilots	ACES
Top Choice of Pilot Group	46.1%	25.0%
Top 3 Choices of Pilot Group	52.8%	73.3%

A transfer-of-training study was performed in a preliminary test of ACES as a desktop training system. Student pilots in fighter lead-in training at Holloman Air Force Base, NM were given an opportunity to interact with ACES by selecting maneuvers and observing their outcome in a simulated air-to-air engagement. Instructor pilot (IP) ratings on later training sorties for these students were compared to similar ratings of students not using ACES during training. The ratings for the two groups were not significantly different. The evaluation was problematic for several reasons including little control over when and how the students interacted with ACES and inherently low variability in the IP rating scores. However, both student and IP responses to questionnaires indicated that the system was valuable as a training tool.

Limitations of Production System Models

There are several reasons to be concerned about the ultimate appropriateness of production systems for modeling pilot decision making. Classical production systems are often described as "brittle." Such systems only work under conditions that are explicitly represented in the rules found in the model. When conditions are encountered that were not explicitly anticipated, the model simply fails. NN models, in contrast, exhibit "graceful degradation." Such systems will generally produce reasonable responses even when conditions change, particularly if there is considerable similarity between a previously learned situation and a new situation. NN models are also capable of learning to improve their performance based on encounters with new situations, whereas production systems must be improved by adding new rules or by modifying old ones. Another advantage of NN models is their ability to accept and produce continuously valued variables would make them particularly appropriate for representing the parameters that describe airspace states and for ordering the priority of alternative actions. ACES handled continuous variables by specifying ranges of values (e.g., aspect angle between 0 and 30 degrees). A rule then reacts to all values in the range as equivalently appropriate and all values outside the range as equivalently inappropriate. Clearly, it would be better to make more continuous distinctions among the various values both inside and outside the range. For example, it seems more appropriate to treat 30 and 31 degrees as only slightly different rather than completely different (as the rule would do).

Thus, there are several reasons to explore the applicability of NN models in ACM. The next section discusses some of the basic properties of NNs and their implementation.

Artificial Neural Networks

In contrast to rule-based models, NNs select maneuvers using densely interconnected networks of simple processing units. These simple units, called nodes, receive information from external sources (i.e., from input to the network or from other nodes), sum this information, and then propagate an activation level to all connected nodes.

The advocates of NNs frequently mention the ability of such systems to learn the appropriate mapping of inputs to outputs from examples and to successfully generalize that learning to new examples. Perhaps the crux of NN modeling is the application of appropriate learning algorithms to appropriate processing network topologies such that a set of connection weights is found that leads to desired performance. One of the most basic learning algorithms found in NN models is the Hebbian contiguity, or associative rule (Hebb, 1949). This simple learning rule states that if two simple processors are simultaneously active and are connected, then the relationship between them should be strengthened.

Some types of learning rules rely on external teachers for feedback. In such networks, learning occurs in an iterative feedback loop composed of four parts (Lippmann, 1987). First, a pattern is presented and activation is propagated through the layers of the network. Second, the output activation is compared against the correct output (i.e., the true output information associated with a given input pattern), and an error term is computed. Third, interconnections (i.e., weights) are modified using some scheme that reduces the error measure computed in part two. Finally, go back to step one and repeat this process. This iterative learning scheme continues until all training patterns produce the correct output.

NNs are able to generalize from previously learned responses to incomplete or novel instances of stimuli. To the extent that a new stimulus is similar to a stimulus pattern that has already been trained into the network, a similar pattern of processing will occur across the network resulting in a similar response (Arbib, 1986).

For our purposes, we thought that the ability of NNs to accept and produce continuously valued variables would make them particularly appropriate for representing the parameters that describe airspace states and for ordering the priority of alternative actions.

In a production system, examples of airspace scenarios that elicit maneuvers are defined by the condition half of a production rule. The representation of when to select a particular maneuver is local to that rule. NNs, on the other hand, store information in the entire set of connection weights that define a network. In this case, the representation of when and how a maneuver is selected is not found in an explicit set of conditions that have to be matched but, rather, it exists everywhere within the network (i.e., in every connection weight).

Nonlinear Multilayered Perceptron Model

The multilayer perceptron (MLP) model has one or more layers of processing nodes between input information and the output layer. A layer is a set of processing nodes connected to successive layer nodes via a matrix of weights. That is, a weight matrix represents the connectivity in a layer where the row dimension of the matrix corresponds to inputs for a given layer and the column dimension represents outputs for that layer. The computational power of the MLP stems from the application of nonlinear activation functions, as well as the associated family of nonlinear learning algorithms such as the back propagation gradient descent (Rumelhart & Zipser, 1986).

A feed-forward network operates by passing activation from the inputs for each layer to the outputs in the layer via the weights on the connections between the inputs and the outputs. The net input to a given node is passed through a nonlinear quashing function which keeps the activation of a unit between zero and one. This passing of activation through the layers is repeated until the final outputs are computed in this way. Learning in such networks is a matter of finding a set of weights which will compute a desired input-output mapping. If the mapping is linear, a single layer is sufficient. With nonlinear mappings, "hidden nodes" are required resulting in at least two layers in the system[1].

The learning rule, also called the modified delta or back-propagation rule, most commonly associated with this model is very similar to the perceptron convergence rule developed earlier by Rosenblatt (Nilsson, 1990). The delta rule

$$\Delta W_{jk} = \varepsilon \delta_k X_j \qquad (1)$$

[1]Two layers are sufficient if there are a sufficient number of hidden nodes, but more layers may be used.

states that the weight connecting layer j to layer k will be changed by an amount proportional to the error found at the layer k node, δ_k, and the activation received from the layer j node, X_j, where ε is a learning rate parameter. The major difference between the non-linear multilayer and the single-layer perceptron models is found in the method for generating the δ_k term.

The error signal, δ_k, for an output node is generated via

$$\delta_{pk} = (t_{pk} - X_{pk}) X_{pk} (1 - X_{pk}) \tag{2}$$

where $(t_{pk} - X_{pk})$ corresponds to the difference between the correct output activation for a given node, k, and an input pattern, p. The $X_{pk}(1 - X_{pk})$ term, sometimes labeled the quashing function, is simply the derivative of the sigmoid activation function, F_{act} (Rumelhart, & Zipser, 1986).

$$Fact_j = \frac{1}{1 + e^{-net_j}} \quad \text{where } net_j = bias_j + \sum^{i} a_i w_{ij} \tag{3}$$

The derivative of the sigmoid serves to vary the size of the error ($t_{pk} - X_{pk}$) in terms of amount of activation produced by a node. That is, the derivative of the sigmoid reaches its maximum as X_{pk} activation approaches 0.50. In other words, the sigmoid increases as output activation approaches 0.50 and decreases as output activation approaches 0.00 or 1.00. As the output activation approaches 0.00 or 1.00 the node is obtaining criterion performance such that the amount of change per weight should be getting smaller. The error signal generation procedure, unfortunately, is not so straightforward in the case of hidden layers.

What is the appropriate or true activation, t_{pk}, for a given hidden layer node? At the output layer the answer is given directly in terms of criterion performance as provided by an external teacher. For hidden layers the solution was provided by Widrow and others (Widrow & Hoff, 1960; Rumelhart, Hinton, & Williams, 1986). The solution was to simply take the weighted sum of all successive layer error terms connected to a specific hidden-layer node as the error term:

$$\delta_{pj} = X_{pj} (1 - X_{pj}) \sum_{i=1}^{j} \delta_{pk} W_{kj} \tag{4}$$

That is, for the k^{th} hidden layer node the error terms for all successive layer nodes, δ_{pk}, are multiplied by the weights, W_{kj}, that form the connections between the j and k layers and summed. This aggregate error term is then multiplied by the sigmoid function to produce the error term for that node. Again, the error term is fed to the delta rule and weights are adjusted.

The entire procedure continues until the output activation pattern for all training stimuli yield difference scores (i.e., t_{pk} - X_{pk}) that are very small (Lippmann, 1987). That is, the process continues until all input patterns are correctly classified.

Back propagation is termed a gradient-descent method in that a measure of error, δ, for every weight in the network is being reduced by the learning algorithm. One way of thinking of this is that there is some n-dimensional space where the weights that define a network at a given time reside. The weights can be thought of as representing a surface in this space. To the extent that input patterns are incorrectly classified (i.e., produce inappropriate output activation) then there is error associated with the weights that define the surface in the n-dimensional space. The back propagation algorithm attempts to minimize this error by modifying the weight space (Plaut, Nowlan, & Hinton, 1986). When a network has been trained to classify a set of stimulus patterns, the weight space that provides the solution is said to be at a minimum in the sense that the error associated with the weight surface is minimized. That is, the error found in all the weights has been minimized such that all training set input patterns are transformed by the weight layers resulting in correct classifications.

Just as with any gradient-descent method, the back propagation procedure is subject to becoming trapped in a "local minimum" and, consequently, failing to find the best solution for a problem. With some problems, the local minima may be numerous causing great difficulty with the learning of the input-output mapping.

Artificial Neural Network Models of ACM

In the research performed under this contract, four classes of NN models of air combat maneuvering have been investigated. The first class performs basic fighter maneuver selection and is a direct descendant of ACES. The second class descends from the AML. The third and fourth classes were developed from data collected from the SAAC at Luke AFB, AZ. The third class uses summary data to predict outcomes (win, lose, or draw) and is compared to results from discriminant analysis. The fourth class uses parameter values at particular points in time to predict future control parameters (i.e., bank and g loading).

Software System for Training and Testing Neural Networks

Although there are several commercially available software packages for training and testing NN models, several considerations led us to develop a system that was specifically adapted to our needs. As we discuss more fully in Study 1, the representation of production rules from the ACES model as training patterns for an NN model presented several problems. Overcoming these problems required a system that provided appropriate inputs based on the specification of a range of acceptable values. We also needed a method for handling the values of inputs that were not specified by the ACES rules. Finally, we needed an easy method for saving sets of connection weights after varying amounts of training to permit an evaluation of the generalization of the trained models at various levels of training. Existing software would be difficult to use with the special problems we faced.

The system we developed was written in Turbo Pascal for the IBM PC and compatibles. The systems consists of a set of three programs: NNTRAIN, NNTEST, and NNPLOT. NNTRAIN embodies the back-propagation learning algorithm to adjust connection weights as various training patterns are presented. The program allows the user to specify the number of layers and the number of nodes in each level. NNTEST uses the weights developed through training on a set of patterns to test performance on a new set of patterns to determine the extent to which the model generalizes to new patterns. NNPLOT plots the scores obtained across training epochs to provide information about the course of learning. Several variations on scoring methods and presentation of training patterns are possible as described in the following sections from the documentation.

The NNTRAIN and NNTEST programs allow for several options that can be realized in one of two ways. Various flags may be included on the command line following NNTRAIN or NNTEST, or flags may be specified in files in the current directory named "train.flg" or "test.flg," respectively. These files may contain the same flags as the command line. Flags allow specification of various parameters, files, and options. Some flags require one or more following values. Others simply cause the described effect. Most flags have defaults. The help screens for each command include a list of the flags which can be used with the command, the values which must accompany the flags (if a value is required), the default for the flag, and a short description of the effect of the flag.

An example command line: NNTRAIN -d test.def -l 3 -n 4 3 2 1 -q w 95 -r .25

In the example, the -d flag specifies the definition file is to be "test.def." This file identifies the file containing training patterns, and specifies within that file what data is to be used and which columns define input and output parameters. The -l flag specifies that there will be three layers to the the network, layers here being defined as the number of layers of weights in the net. The -n flag defines the number of nodes in each level of the network, in this case the network is to have four input nodes, three nodes in the first hidden level, two nodes in the second hidden level and one output node. The -q flag indicates when the program is to quit. In this example, w is specified after the -q flag, meaning that the program will quit when the winner criterion exceeds 95% (see sections on scoring criteria and quitting methods for details). Finally, the -r flag indicates that the learning rate is to be set at .25.

NNTRAIN: Training a neural network.

flag	value	default	description
-a	file	none	name of summary file for appending summary data
-d	file	prompt	name of definition file
-j	integer	no join	get ranges for inputs from pairs of patterns and average
-l	integer	prompt	number of layers of weights
-m	e, t or w		save weights if better (error, tolerance, or winner)
-n	integers	prompt	number of nodes within each level
-p		false	permute the order of training patterns each epoch
-q	mtd crit	manual	method and criterion for stopping
-r	real	0.45	network learning rate
-s	integer		multiple of epochs for saving weights
-t	real	0.1	value for tolerance error scoring
-u	real r, p or real		"unknown" input values and method for setting values for the "unknowns"
-v			verbose, display more information during training
-w	file	random	file with weights to initialize net

The **-a flag** followed by a filename specifies a file to receive summary statistics after the completion of the NNTRAIN program. If the file specified does not exist it will be created; if it does, the current summary will be appended to it. The statistics for each run of program are written on a single line in the file. The summary statistics recorded are: output error, percent correct (winner scoring), percent correct (tolerance scoring), learning rate, definition filename, number of layers, network architecture, and last epoch.

The **-d flag** followed by a filename specifies a file containing information about the training pattern file, including the pattern file name, and which patterns to use in the training.

The **-j flag** allows ranges for inputs to be specified by using pairs of patterns. The number of patterns skipped and taken (in the definition file) should be determined by the number of pattern pairs. The integer following the -j flag specifies the number of random values in the range which are averaged to determine a value for each presentation of the pattern. The larger the value, the more the averages will tend toward the center of the range.

The **-l flag** followed by an integer specifies the number of layers in the network to be trained.

The **-m flag** causes weights to be stored whenever a new maximum percent correct or minimum output error state is reached during training. The weights are stored in files specifying the epochs (e.g., 110.wts are the weights after 110 training epochs). The -m flag must be followed by e, t, or w indicating whether to use output error, percent within tolerance, or percent winners, respectively.

The **-n flag** followed by a series of integers specifies the number of nodes in each level of the network. The network has one more level than it has layers.

The **-p flag** signals the program to permute the training pattern order on each epoch. This will eliminate any incidental effects of the order the training patterns. This flag also will slow the program to some extent.

The **-q flag** is used to control when the NNTRAIN program quits training the network. There are four different methods that may be used to stop the program and each method requires a criterion. Thus, the flag requires two following parameters, a method and a criterion. An example of the flag might be: -q t 0.05

The methods (mtd) and criteria are as follows:

mtd	criterion	the program will stop when
c	integer	the criterion number of epochs is completed
e	real	the overall error is less than the criterion
t	real	all outputs are within the criterion of correct values
w	real	the percent correct (winners) meets the criterion

If the -q flag is not given the program will quit when the q key is depressed. The q key will also stop the program if any of the other methods are being used as well.

Note that the winner criterion only functions when there are binary parameters in the output units and there is more than one output unit. See the scoring methods help section for details.

The **-r flag** can set the learning rate in the NNTRAIN program by specifying a value following the -r flag. The learning rate can be thought of as the step size taken during gradient descent. Learning rate values of over .45 are not recommended as they can lead to chaotic network behavior.

The **-s flag** will save the weights generated after given multiples of epochs. The multiple is specified by the -s flag (e.g., -s 100) will save the weights after every 100 epochs. The weights are stored in a file specifying the number of epochs and a ".wts" extension (e.g., 100.wts).

The **-t flag** followed by a real number specifies the tolerance level to be used by the tolerance scoring strategy. For a pattern to be scored correct, every output unit must be with the tolerance of the correct value.

The **-u flag** controls how the program deals with "unknown" values in input patterns. After the -u two entries are required, the first specifies the value of unknown values in the training patterns. Because training patterns are read in as real numbers this value must be a real number. The second item following the -u flag controls what the program does with the unknown values. If a constant is specified, that constant is used every time the program encounters an unknown value. If an r is specified, random values between zero and one are generated each time the unknown value is found. If a p is specified, pairs of random values (random and 1-random) are assigned to pairs of unknown inputs.

The **-w flag** will use weights stored in the filename specified. These weights were presumably saved after a previous run of the NNTRAIN program. When the weights option is used, the NNTRAIN program will read the network architecture from the file, and thus no layer or number of nodes information need be specified.

NNTEST: Testing new patterns.

flag	value	default	description
-a	file	none	name of summary file to append stats
-d	file	prompt	name of definition file
-i	integer		increment for weights files
-o	file	none	store outputs for each test pattern
-t	real	0.1	tolerance value for tolerance scoring
-v		false	verbose, display more information during training
-w	file	prompt	file(s) with weights to define net

The NNTEST program evaluates the weights produced by the NNTRAIN program against a new set of test patterns. This analysis provides an index of the ability of the model created by the NNTRAIN program to generalize to new patterns.

The **-a flag** causes a summary of the test statistics for each of the weights files to be appended to the file specified with the -a flag.

The **-d flag** followed by a filename specifies a file containing information about the test pattern file, including the pattern file name, and which patterns to use in the testing.

The **-i flag** is used to define a set of weights files to be tested. The files have names such as "100.wts" where the 100 indicates the number of epochs of training preceding the saving of the weights. Several such files in increments of 100 epochs (e.g., 200.wts, 300.wts) can be tested by using the flag, -i 100 so the program will continue analyzing such files until it can find no more.

The **-o flag** causes the outputs produced for each test pattern to be stored in a file specified with the -o flag.

The **-t flag** followed by a real number specifies the tolerance level to be used by the tolerance scoring strategy. For a pattern to be scored correct, every output unit must be with the tolerance of the correct value.

The **-w flag** followed by a file specification indicates which weight files should be tested. The file specification can include wildcards so if -w *.wts were given, all files in the current directory with the extension .wts would be used to evaluate the test patterns.

NNPLOT: Plotting the scores over epochs.

flag	value	default	description
-a	integer	1	averages the given number of epochs
-f	file	prompt	name of error file

The NNPLOT program uses the error information from the error output file generated by the NNTRAIN program. NNPLOT will prompt for the name of this error file. The program will plot the gradient of the three scoring methods across epochs. The initial plot will show the first 1,800 epochs in three panels of 600 epochs each. When there are more than 1,800 epochs the user may scroll down one panel at a time using the n key.

The **-a flag** allows the user to average epochs together. If this flag is used each panel will display 600 times the value given following the -a flag.

In the plot display, error information is shown in green, percent incorrect using the winner strategy is displayed in magenta, and percent incorrect using the tolerance strategy is displayed in yellow. In each panel the top bar indicates the maximum error for all epochs displayed, while the bottom indicates the minimum error for all epochs displayed. For percent incorrect scores, the top and bottom of the panels indicate 100 and zero percent incorrect.

The Patterns File: Pattern file organization.

The NNTRAIN and NNTEST programs require two files to run. One of these is a data file containing a rectangular matrix of numbers. Data from a single observation must be recorded on each line while information concerning individual parameters must be organized in columns. Columns are defined by at least one space between parameters, and thus do not need to be strictly aligned. A sample data file follows:

```
0 0 0
0 1 1
1 0 1
1 1 0
```

The definition file is used to specify what columns are the inputs as well as the outputs. The definition file also specifies which training patterns (lines) are to be used. This example represents the XOR problem. The first two columns represent the input training patterns, while the last column specifies the output pattern.

The Definition File: Definition file organization.

The NNTRAIN and the NNTEST programs require a definition file which specifies a data file and the parts of the data file to be used as inputs and outputs. The definition file must consist of eight lines. A sample definition file follows:

```
xor.pat  Pattern file name
xor.scr  Score file name
0        Number of lines to be skipped in the pattern file
4        Number of lines to be used as training patterns
2        Number of input columns
1 2      The column numbers of the input parameters
1        Number of output columns
3        The column number(s) of the output parameters
```

In cases where only one item is required on a line, only the first item is read. When reading column numbers (line 6 and 8) the program reads the number of items specified on the previous line. Aside from these values no other items are read from the file; thus comments can be inserted on any line after the definition parameters have been entered.

One limitation of this coding scheme is that only contiguous lines can be read. In other words, you may not specify more than one value of lines to be skipped.

The Flags Files: Flag file organization.

In addition to setting options from the command line, the NNTRAIN and the NNTEST programs can read options from files named "train.flg" and "test.flg," respectively. These files must be in the current directory.

Each option in these files must be on separate lines. Once the option is completely specified comments may be added. If the same options are specified on the command line, the options from the flags file are disregarded. A sample flags file follows:

```
-d test.def    Use test.def as the definition file.
-l 3           Network will have 3 layers.
-n 4 3 2 1     Use 4 3 2 1 architecture.
-q w 95        Quit when the percent correct (winner strategy) exceeds 95%.
-r .25         Use a learning rate of .25.
```

Scoring Procedures: The error, winner, and tolerance criteria.

The NNTRAIN and NNTEST programs use three different measures of performance. The first of these is the error criterion. The error measure is the total squared difference between the activation levels of nodes in the network and the correct activation levels.

The winner criterion assumes that outputs are encoded as binary values (one or zero) and that there is more than one output unit, and only one of these output units is set to one. The winner criterion calculates the percentage of training patterns correctly identified. This calculation uses an algorithm that identifies correct trials on the basis of whether the most active output node corresponds to the single output parameter set to one.

The tolerance criterion also calculates the percentage of training patterns correctly identified. In this case, correct trials are based on whether the output nodes are within a specified criterion value of the corresponding correct outputs. Note that for a pattern of activation in the output nodes to be judged correct, all the output units must be within the tolerance level. If not all output activations are within the tolerance, a single trial is given a score based on the percentage within the tolerance. The tolerance value may be set using the -t flag. Additionally in the NNTRAIN program, the tolerance value can be set through the -q flag when it is followed by t and the new tolerance value.

All three scoring criterion measures are sent to a score file specified in the definition file. If the score file exists already, the scoring from the current run are appended to the end of the file.

These programs were used to train and test the NN models described in the following four studies.

Study 1: Application of Neural Networks to ACES

Objective

Recall that ACES selects maneuvers using condition-action or if-then rules. In all, ACES has 38 production rules that select 12 offensive and 5 defensive basic fighter maneuvers. In many cases there are multiple rules defining a particular maneuver selection. For example, there are 6 rules for selecting the acceleration maneuver. During operation, ACES samples an ACM scenario and attempts to find matches between rules and ACM conditions. When a match is found, a rule is selected.

Frequently, the conditions for more than one rule are satisfied. When more than one rule has been selected, a tie-breaking or conflict resolution procedure is applied in an attempt to determine the best maneuver. ACES' conflict resolution scheme selects the most general rule. That is, the maneuver rule that has the fewest airspace descriptors defined (i.e., fewest number of conditions defined) is selected.

A potentially serious problem found in many rule-based systems that was also identified within ACES is that conflict resolution schemes operate at a syntactic level. In other words, the information initially used to select a production (i.e., the condition half of rules) is not used in the conflict resolution process. Rather, conflict resolution is based upon meta-information such as how many or how few conditions are specified by the set of rules. It may be the case that subgroups of airspace conditions within rules carry important discriminative information beyond that found in the selection measures employed by conflict resolution strategies. Many conflict resolution procedures ignore the configural aspects of the selection information contained within a rule.

One approach to capturing very specific configurations of conditions in airspace scenarios is to write large numbers of very specific rules. Unfortunately, in dynamic environments such as ACM this can lead to vast numbers of rules. In an attempt to overcome this inherent limitation of rule based expert systems, an NN model of ACM was implemented.

Method

Among the problems that must be solved in the development of an NN model for selecting BFM is the selection of appropriate patterns on which to train the network. Since the production rules in ACES were based on interviews with expert pilots and were validated against experts' decisions, it was thought that there was an important knowledge base contained within those rules. The problem is how to represent the rules as training patterns for training an NN model. There are actually three problems here: (a) How should the inputs (e.g., angle-off, closure rate) be coded for presentation to the network? (b) Since many of the inputs specify ranges rather than particular values, how should ranges be represented in the training patterns? and (c) Since each production rule only references some, but not all, of the possible input variables, how should unreferenced variables ("unknowns") be represented in the training patterns? Table 5 illustrates one set of answers to these questions by showing one method of transforming production rules into training patterns for an NN.

With a range of possible values for a specific airspace condition, the midpoint of the range was used in the network training pattern. This approach assumes that the midpoint is the ideal value for that particular rule. Training on that point should maximize the response of the network to that value and decreasing responsiveness should occur with values at increasing distance from the midpoint. Thus, the midpoint was taken to be

Table 5. Production Rules Represented as NN Training Patterns

Production Rule	Network Training Pattern
select **acceleration** if	
my altitude = __	NN inputs: (midpoints of ranges)
my aspect angle = 0° to 45°	
my airspeed = __	[_ 22.5 _ 0 _ _ _ 1 6 7 _ -100 _]
my 3-9 position to him = behind	
my clock position = __	
bogey's aspect angle = __	feedback pattern (target outputs):
bogey's airspeed = __	
bogey's 3-9 position to me = front	(the first position codes acceleration)
bogey's clock position = 5 to 7	[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
range = __	
altitude difference = -450' to 250'	
angle off = __	
select **break turn** if	
my altitude = __	NN inputs: (midpoints of ranges)
my aspect angle = 120° to 180°	
my airspeed = __	[_ 150 _ _ _ 30 _ _ _ 5000 500 22.5]
my 3-9 position to him = __	
my clock position = __	
bogey's aspect angle = 0° to 60°	feedback pattern (target outputs):
bogey's airspeed = __	
bogey's 3-9 position to me = __	(the 15th position codes break turn)
bogey's clock position = __	[0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
range = 1000' to 9000'	
altitude difference = 0' to 1,000'	
angle off = 0° to 45°	

representative of the range, and generalization to new test cases was expected to occur according to a generalization gradient established around the midpoint[2]. The underlined blank spaces in both the production rule and the prototype pattern correspond to conditions that are unspecified.

[2]It should be noted that the midpoint may not always be the best choice. For example, for angles with a range from 0 to 30 degrees, the 0 degree point may represent the ideal point rather than 15 degrees, the midpoint. We tried using these values where they could be determined, but it made little difference in the outcome. We also tried sampling from the various points within the range during training, again with little effect on the outcome.

That is, any airspace descriptor value can bind to this condition. We employed various strategies for setting the values on these "unknown" inputs: (a) The unknowns were set to a random value over the total range of the variable; (b) The unknowns were set to the midpoint of the range of the variable; or (c) The unknowns were set to some other constant value such as zero which meant that no input came from that unit[3]. The maneuver categories are simply represented as a binary vector where each vector position corresponds to a particular maneuver. In this example acceleration is the first value in the vector and is represented as 1 while all other maneuvers are represented by 0s. The 38 rules in ACES (which could be represented as either 37 or 38 patterns) were configured as prototype training patterns with their associated maneuvers.

Several different architectures were used to explore the problem space represented by the training patterns. In each case, the network was trained for a varying number of passes through the training patterns, and the trained networks were tested for generalization. The generalization test involved presenting the same 40 scenarios originally used to test the ACES model, and the selections of the neural net were compared to ACES selections and the selections of the Air National Guard pilots.

During the testing, it appeared that the original set of training patterns led to biases in favor of the maneuvers that were represented by more rules. Consequently, the patterns for the maneuvers that had only a single rule were duplicated, increasing the number of patterns in the training set. Including three copies of the singletons produced 57 patterns, and including four copies of each singleton produced 67 training patterns.

In an attempt to simplify the learning for the neural nets, another set of training patterns was created by eliminating some of the 12 input variables. Altitude and airspeed were eliminated because they were infrequently referenced by the production rules. The clock positions were also not referenced by many rules, and besides, they are redundant with the information provided by the aspect angles and angle off. This reduction of inputs led to patterns with 8 inputs. Still another set of training patterns

[3]Unfortunately this strategy conflicts with the coding of the values of inputs as levels of input variables when they may take on the value of zero. The "zero strategy" did not work very well in most cases.

Table 6. Summary of Several NN Models Trained on ACES Rules

	Model		Percent matches for model and pilots selections					
	Trng. Set -		First Model	First Model	Top 3 Model	First Pilots	First Model	Top 3 Model
ID	Unkwn. Method	Trng. Epochs	First Pilots	Top 3 Pilots	Top 3 Pilots	Top 3 Model	First ACES	Top 3 ACES
ACES	----	----	25.0	77.5	73.3	85.0	100.0	100.0
8-17	37-rand	110	20.0	45.0	35.0	30.0	**47.5**	49.3
8-32-17	37-rand	32,000	20.0	42.5	31.7	37.5	25.0	53.7
8-32-17-17	37-rand	9,500	22.5	45.0	33.3	40.0	25.0	51.5
8-17	57-rand	360	32.5	52.5	**43.3**	57.5	7.5	55.9
8-8-17	57-rand	1,000	20.0	40.0	40.0	52.5	7.5	57.4
8-17-17	57-rand	470	25.0	50.0	40.0	52.5	7.5	60.3
8-17-17	57-0.5	207	32.5	55.0	38.3	**60.0**	2,5	53.7
8-17	67-rand	10	30.0	40.0	30.0	37.5	7.5	38.2
8-8-17	67-rand	290	32.5	50.0	42.5	55.0	2.5	**75.7**
8-17-17	67-rand	50	32.5	50.0	33.3	**60.0**	20.0	44.1
8-8-17-17	67-rand	440	**35.0**	52.5	40.0	55.0	5.0	56.6
12-17	38-rand	1,153	12.5	30.0	30.8	22.5	17.5	44.1
12-17-17	38-rand	12,522	22.5	40.0	33.3	27.5	25.0	48.5
12-12-20-17	38-rand	13,200	20.0	45.0	36.7	37.5	25.0	48.5
16-17	37-zero	55	20.0	40.0	31.7	35.0	15.0	42.6
16-16-17	37-zero	300	15.0	30.0	28.3	37.5	5.0	43.4
16-17	57-zero	50	25.0	50.0	39.3	55.0	5.0	50.0
16-17	57-rand	190	27.5	52.5	37.5	47.5	2.5	52.9
16-16-17	57-rand	90	**35.0**	**55.0**	40.0	57.5	2.5	50.0
40-17	67-0.2	30	17.5	22.5	22.5	32.5	0.0	31.6
40-40-17	67-0.2	22	20.0	25.0	13.3	25.0	7.5	19.9

Table 6. Summary of Several NN Models Trained on ACES Rules

Model			Percent matches for model and pilots selections					
	Trng. Set -		First Model	First Model	Top 3 Model	First Pilots	First Model	Top 3 Model
ID	Unkwn. Method	Trng. Epochs	First Pilots	Top 3 Pilots	Top 3 Pilots	Top 3 Model	First ACES	Top 3 ACES
ACES	----	----	25.0	77.5	73.3	85.0	100.0	100.0
8-17	37-rand	110	20.0	45.0	35.0	30.0	**47.5**	49.3
8-32-17	37-rand	32,000	20.0	42.5	31.7	37.5	25.0	53.7
8-32-17-17	37-rand	9,500	22.5	45.0	33.3	40.0	25.0	51.5
8-17	57-rand	360	32.5	52.5	**43.3**	57.5	7.5	55.9
8-8-17	57-rand	1,000	20.0	40.0	40.0	52.5	7.5	57.4
8-17-17	57-rand	470	25.0	50.0	40.0	52.5	7.5	60.3
8-17-17	57-0.5	207	32.5	55.0	38.3	**60.0**	2,5	53.7
8-17	67-rand	10	30.0	40.0	30.0	37.5	7.5	38.2
8-8-17	67-rand	290	32.5	50.0	42.5	55.0	2.5	**75.7**
8-17-17	67-rand	50	32.5	50.0	33.3	**60.0**	20.0	44.1
8-8-17-17	67-rand	440	**35.0**	52.5	40.0	55.0	5.0	56.6
12-17	38-rand	1,153	12.5	30.0	30.8	22.5	17.5	44.1
12-17-17	38-rand	12,522	22.5	40.0	33.3	27.5	25.0	48.5
12-12-20-17	38-rand	13,200	20.0	45.0	36.7	37.5	25.0	48.5
16-17	37-zero	55	20.0	40.0	31.7	35.0	15.0	42.6
16-16-17	37-zero	300	15.0	30.0	28.3	37.5	5.0	43.4
16-17	57-zero	50	25.0	50.0	39.3	55.0	5.0	50.0
16-17	57-rand	190	27.5	52.5	37.5	47.5	2.5	52.9
16-16-17	57-rand	90	**35.0**	**55.0**	40.0	57.5	2.5	50.0
40-17	67-0.2	30	17.5	22.5	22.5	32.5	0.0	31.6
40-40-17	67-0.2	22	20.0	25.0	13.3	25.0	7.5	19.9

One model is somewhat more similar to ACES. The 8-8-17 model trained on 67 patterns with random values for the unknowns produced a 75.7% overlap with ACES' selections. Most of the models, however, failed to reproduce ACES' selections which suggests that ACES' rules had not been captured very well by the learning procedure (backward error propagation). The failure of the NN models to achieve the same performance level as ACES (let alone surpass ACES' performance) is, at first, somewhat surprising. The claims made for the capabilities of NN models would lead one to expect that given a large enough network, any input-output relationship could be captured by the model. In one sense, this was true of most of these nets because during the learning phase most of the nets reached very high levels of accuracy on the learning patterns themselves. Although as Table 6 shows, performance fell below that of ACES on the generalization test.

There are several possible reasons that the neural net failed to precisely capture ACES' rules. The adaptation of the rules to training patterns for the NN models required some decisions about the appropriate treatment of continuous inputs, inputs which specify a range, and inputs for the "unknown" parameters. As a result, the training patterns do not directly correspond to the rules. In fact, the training was accomplished by a (theoretically) infinite number of training patterns whenever a random selection was used to set values for unknown parameters and/or for the values in the range of specified parameters. Despite our best efforts and several different attempts, we were unable to find a representation of the training patterns that produced an NN model capable of generalization performance as good as the performance of ACES original rules. This problem is clearly in need of further investigation.

Study 2: Application of Neural Networks to AML

Objective

In Study 2, we examine the ability of NN models to incorporate the logic of another model, the AML model used in simulators to provide a maneuvering adversary for air-to-air combat training. The training patterns used to develop an NN model of AML do not suffer from the problems we encountered in developing an NN model of ACES. Thus, this study should give an idea of the extent to which our earlier failures with NN models for ACES were due to the problems associated with deriving training patterns frcm production rules.

There are two additional motivations for the work. First, given AML's status in the ACM simulated arena, it would be of general interest to be able to simulate its logic with a neural network. Second, one criticism of AML is that its logic is too rigid. Being sensitive to this, human pilots are often able to detect its artificial nature. Once detected they can usually determine its weakness and then consistently beat it. Perhaps the generalization of an NN model would help soften the decision boundaries of AML. If so, maneuvering might appear more realistic to an opponent.

Background

AML is a computer simulation directed at selecting and executing maneuvers for aircraft engaged in air combat. First, we describe the basic components of AML. For a complete description of the program see the two volume report by Burgin, Fogel, and Phelps (1975).

An important factor that distinguishes AML is the level of action employed by the model. Whereas ACES and the classical maneuver networks function at the basic fighter maneuver level, AML operates on small action units called elemental maneuvers. Fundamental to the notion of elemental maneuvers is the concept of a maneuver plane. A maneuver plane is defined as the geometric plane that contains an aircraft's velocity vector and is perpendicular to the earth's horizon. From this maneuver plane, then, consider the set of planes produced by rotating the maneuver plane about the velocity vector in integer multiples of some angle. For AML this angle is typically between 5 and 15 degrees. During each decision point, those elemental maneuvers considered for execution next (called trial maneuvers) are members of the set of rotated maneuver planes.

Trial maneuvers are grouped into sets according to different flight regimes. Four basic flight regimes are identified: (a) normal flight, (b) low speed recovery, (c) dive recovery, and (d) over the top. The exact set of trial maneuvers considered under each flight regime may vary with each implementation of AML. Table 7 shows the trial maneuvers used for each regime in an implementation of AML at Luke AFB, Arizona.

Table 7. Specific Trial Maneuvers Considered under Each Flight Regime in an Implementation of AML at Luke AFB, AZ.

normal flight
- same maneuver as executing
- roll to level flight
- normal level g roll to intercept opponent
- max g roll to intercept opponent
- max g defense (+90°) from opponent
- max g defense (-90°) from opponent

low speed recovery
- negative g roll to wings level
- zero g roll to wings level
- low speed zero g with current roll
- recovery: one g roll to wings level
- one g roll to intercept opponent

dive recovery
- max g pull up
- dive angle max g pull up off + side
- recovery: max g pull up off - side

over the top
- max g roll to inverted wings level
- over the top
- max g roll to inverted wings off + side
- max g roll to inverted wings off - side
- max g intercept opponent

Each trial maneuver is associated with values for the three basic control variables: load factor, bank angle, and thrust. As seen in Table 7, the load factor is identified with each trial maneuver. Bank angle is also inherent in the definition of the maneuvers. Thrust is assumed to be set to full afterburner unless the situation requires rapid deceleration. This occurs when an overshoot is imminent or when one wants to force the opponent to overshoot. In this case, the throttle is set to idle and effects for speed brakes are introduced into the flight equations.

The selection of a particular trial maneuver is based on the future positions of both the attacker, the AML controlled aircraft, and its target. The target's future position is determined from its past coordinates recorded at 1-s intervals. Polynomial equations are used to extrapolate along its flight path for some fixed prediction time (typically 2 to 4 s). The extrapolation assumes a constant acceleration vector and flight within a fixed plane. The target's attitude is assumed to be perpendicular to the derived plane. Any errors resulting from this extrapolation technique are sufficiently small to make the overall decision process feasible.

Next, the attacker's future position is predicted for each trial maneuver within the flight regime identified as appropriate. Although exact future positions for the interval between decision points could be made via flight equations, an extrapolation technique is employed instead. This technique helps conserve computing time. At Luke AFB, AML predicts the attacker's future position with the same extrapolation technique used for the target. The prediction time for the attacker is always the same as for the target.

Once the predicted situations are computed, AML selects that trial maneuver resulting in the most favorable conditions for the attacker. A value is derived for each situation by answering a series of yes-no questions about the two aircraft's relative characteristics. Yes answers are given a value of one and no answers a value of zero. A positive weight is associated with each question that reflects the question's importance. That trial maneuver resulting in the highest weighted sum is chosen for execution. The specific set of questions used may vary across implementations of AML. The questions asked in the version at Luke AFB are given in Table 8.

To summarize, AML selects low level or elemental maneuver actions by evaluating several predicted future situations every 0.5 s. AML's action unit is the elemental maneuver where elemental maneuvers can be thought of as the subcomponents that might compose fighter maneuvers. AML's decision-making processes were motivated by performance rather than psychological processes. Finally, AML's training function is to drive aircraft in real-time flight simulators.

Table 8. Questions used to Evaluate Predicted Situations in an Implementation of AML at Luke AFB, AZ.

1. Is opponent in front of me?
2. Am I in front of opponent?
3. Can I see opponent?
4. Can opponent see me?
5. Am I within a certain cone behind opponent?
6. Is opponent with a certain cone behind me?
7. Am I in attitude and position so I can fire at opponent?
8. Is opponent in attitude and position so he can fire at me?
9. Are we closing between 0 and 300 feet/second?
10. Is line-of-sight angle from me to opponent less than 60 degrees?
11. Is rate of change of line-of-sight angle from me to opponent negative?
12. Is my specific energy rate (Ps) greater than given constant?

Method

That part of the ACES code for defining airspace scenarios was used to define a set of 35 training patterns here. Each pattern was defined by the same 12 airspace descriptors used by ACES to select basic fighter maneuvers. The AML algorithm was then applied to each pattern to determine which of the 17 trial maneuvers would be selected. Not all of the information in the set of 12 inputs was needed by the algorithm to select a maneuver. Additionally, AML required some information not contained in the 12 inputs, such as prior locations to extrapolate future position of opponent. Nonetheless, we used the same set of inputs used by ACES in order to maintain consistency across the models.

Results and Discussion

A 12-8-17 NN model was trained under back propagation learning to select the 17 trial maneuvers. At the end of 7,130 epochs of training all 35 patterns were correctly classified. The results suggest that an NN model is indeed capable of implementing the AML selection logic. The question of whether maneuvers selected by neural networks would appear less artificial than AML was not addressed. Rather than pursue this issue further, we decided to concentrate on building NN maneuvering models directly from SAAC pilot data. We discuss the results of this effort in the next two studies.

Study 3: Prediction of Air Combat Outcomes

Objective

This section reviews several empirically derived NN models of air combat outcomes. The models were based on engagements flown in the SAAC at Luke AFB, AZ. The engagement data were collected as part of another project aimed at identifying and validating performance measures of air combat maneuvering. More specific information on the nature of the engagements, characteristics of the pilots, and the method of data collection can be found in Waag, Raspotnik, and Leeds (1992). This research was based on 792 air combat engagements flown in the SAAC simulator at Luke Air Force Base for which 28 airspace state parameters were collected and averaged. The results of the engagements also were recorded as either win, draw, or lose. From a subset of these data, NN models were created using the back propagation-feed forward learning algorithm. These NN models were tested against new cases and were also compared to models based on discriminant analysis.

Method

The engagement data were first separated into a training and a testing data set. Of the 792 engagements available, the first 400 were used to train the NN model, and the last 392 were used to test the model's ability to generalize to new patterns. Of the 28 air state parameters, 11 were chosen to be used in both the NN and the discriminant analysis models. These parameters are shown in Table 9. (The same parameters were selected as those used in the earlier Multiship Air Combat R&D report from the Armstrong Laboratory at Williams Air Force Base.)

Two different NN architectures were used to create models of air combat. The first architecture was a perceptron (11-3 architecture), where 11 input units connected directly to the 3 possible outputs (win, draw, or lose). The second architecture was a multilayer NN (11-11-3 architecture), with 11 inputs, 11 nodes in the hidden layer, and 3 outputs.

Table 9. Parameters Included in the Models

AAMI defensive	Throttle position
AAMI offensive	Percent time throttle half to mil
AAMI percent offensive	Missiles fired in parameters
AAMI average offensive	Missiles fired out of parameters
Closing rate	Gun Misses
Ps (energy management)	

Note. AAMI = All Aspects Maneuvering Index

The NN models were trained with varying numbers of training epochs. The perceptron architecture was trained for 50 and 1,200 epochs. Because the multilayer architectures learn much slower, they were trained with both 300 and 3,000 epochs.

Results

The generalization performance of the various NN models is shown in Tables 10 through 13. All the models classify correctly approximately 80% of the generalization test patterns. The figure in the lower right corner of the tables represents the overall percent correct classifications with each engagement weighted equally. Averages of the row and column percentages are also shown.

The perceptron models (Tables 10 and 11) performed slightly better than the multilayer models (Tables 12 and 13) using the overall percent correct. The differences between the models appear to be due to differential performance on classifying patterns into the win, draw, or lose conditions. Specifically, the perceptron models correctly classified an average of less than 50% of engagements resulting in draws. Because draw outcomes accounted for only 13% of the testing data, this poorer performance did not have a great effect on the overall performance measure. The mean percent correct measure equally weights each outcome. With the mean percent correct measure, the multilayer models classified engagements slightly better than the perceptron models.

Table 10. Performance of Perceptron Network (1,200 Epochs)

Engagement outcome	Model classification				
	Win	Draw	Lose	Overall	Percent correct
Win	152	10	17	179	84.91
Draw	8	29	14	51	56.86
Lose	5	8	149	162	91.98
Overall	165	47	180	392	77.91
Percent correct	92.12	61.70	82.78	78.87	84.18

Table 11. Performance of Perceptron Network (50 Epochs)

Engagement outcome	Model classification				
	Win	Draw	Lose	Overall	Percent correct
Win	149	6	24	179	83.24
Draw	10	21	20	51	41.17
Lose	5	4	153	162	94.44
Overall	164	31	197	392	72.95
Percent correct	90.85	67.74	77.66	78.75	82.40

Table 12. Performance of 11-11-3 Network (3,000 Epochs)

Engagement outcome	Model classification				
	Win	Draw	Lose	Overall	Percent correct
Win	147	15	17	179	84.48
Draw	10	31	10	51	60.74
Lose	3	14	145	162	89.50
Overall	160	60	172	392	78.24
Percent correct	91.88	51.67	84.30	75.95	82.39

Table 13. Performance of 11-11-3 Network (300 Epochs)

Engagement outcome	Model classification				
	Win	Draw	Lose	Overall	Percent correct
Win	143	23	13	179	79.88
Draw	8	33	10	51	64.71
Lose	3	21	138	162	85.18
Overall	154	77	161	392	76.59
Percent correct	92.86	42.86	85.71	73.81	80.10

The similarity in the performance of the linear (11-3 perceptron) models and the more complex nonlinear (11-11-3 multilayer) models suggests that this [illegible]oblem space is primarily linear. Not much is gained in generalization performance by using the more complex, nonlinear, model. This conclusion can be examined further by comparing the performance of these NN models to the more traditional discriminant analysis model.

Discriminant Analysis Model

A model of air combat outcomes was also created using discriminant analysis. Discriminant analysis uses a least squares technique to classify patterns of data into categories. The performance of the discriminant analysis model on the engagement data is shown in Table 14. This model yields a level of performance very similar to that of the NN models.

Table 14. Matches between Actual Outcomes and Discriminant Analysis Classification

	Model classification				
Engagement outcome	Win	Draw	Lose	Overall	Percent correct
Win	141	13	25	179	78.77
Draw	8	29	14	51	56.65
Lose	5	15	142	162	87.65
Overall	154	57	181	[illegible]	74.35
Percent correct	91.56	50.88	78.45	73.63	79.59

Table 15 shows the correlations among the actual outcomes and the classifications made by the models (win, draw, and lose coded as 1, 2, and 3, respectively).

Both NN models and discriminant analysis models assign some priority to the possible outcomes. Discriminant analysis does this by calculating the probabilities of each outcome, while the NN models assign differing activations to the output units. By dividing the output activations by the sum of the total output activations, a measure similar to the discriminant analysis probabilities can be created, and the two measures can be compared.

Table 15. Correlations among Outcomes (Win, Lose, or Draw): ACM Data

	Actual	Disc.	Perc. 50	Perc. 1,200	11 11 3 300	11 11 3 3,000
Actual Outcome	1.00	.75	.79	.82	.81	.81
Discriminant Analysis	.75	1.00	.93	.89	.81	.88
Perceptron 50 epochs	.79	.93	1.00	.94	.86	.92
Perceptron 1,200 epochs	.82	.89	.94	1.00	.90	.96
Net 11 11 3 300 epochs	.81	.81	.86	.90	1.00	.92
Net 11 11 3 3,000 epochs	.81	.88	.92	.96	.92	1.00

The outcome probabilities of the discriminant analysis model and the relative activation of the outcomes in the 4 NN models were correlated across the 392 test engagements. Table 16 shows the correlations among the magnitude of the output units for the various models.

By a small margin, the largest correlation with the actual data is the 1,200 epoch perceptron model. Again this result might be tainted by the smaller proportion of draws in this data set. The correlations also show a strong relationship between the various models. The correlation between the discriminant analysis model and the 300 epoch multilayer model is the weakest with an *r* of .80, while strongest correlation is between the 50 epoch perceptron model and the 1,200 epoch perceptron model with an *r* of .97. Given the similar performances in classifying patterns of data into categories, the high correlations between models should be expected.

Table 16. Correlations among Output Units: ACM Data

	Actual	Disc.	Perc. 50	Perc. 1,200	11-11-3 300	11-11-3 3,000
Actual outcome	1.00	.75	.79	.82	.81	.81
Discriminant analysis	.75	1.00	.96	.93	.80	.92
Perceptron 50 epochs	.79	.96	1.00	.97	.83	.95
Perceptron 1,200 epochs	.82	.93	.97	1.00	.86	.96
11-11-3 Net 300 epochs	.81	.80	.83	.86	1.00	.89
11-11-3 Net 3,000 epochs	.81	.92	.95	.96	.89	1.00

One unexpected aspect of the correlational analysis is the relatively low *r* between the discriminant analysis model and the actual outcomes compared to its performance classifying patterns of data. While the discriminant analysis model performed similarly to the NN models in classification, its correlation with the actual data was much lower than any of the other models. The discrepancy between classification and correlation scores suggests that while the number of correct classifications between both types of models is similar, the discriminant analysis models tend to make bigger mistakes. In other words, the discriminant analysis models tend to classify incorrectly more losses as wins, compared to the NN models that tend to incorrectly classify losses as draws (assuming, of course that a draw is closer to a loss than is a win).

Discussion

Although the performance of the discriminant analysis model is slightly below that of NN models, discriminant analysis can be used to identify which variables have the most predictive utility for categorization. Thus, it may be useful to employ discriminant analysis in deciding what parameters should be used as inputs to an NN model.

The results discussed thus far have centered on the relationships of the various models to the actual results. However, these data also describe the nature of the prediction of air combat outcomes. The principal finding demonstrated by the current research is that this problem space is essentially linear because of the approximate equivalence of classification performance between the nonlinear multilayer models and either the linear perceptron or the discriminant analysis models.

To summarize Study 3, four points can be made. First, NN models can achieve a reasonable degree of accuracy in predicting the outcome of air combat engagements based on only summary air state data. Second, among NN models there is little difference between perceptron models and more complex multilayer models. Third, discriminant analysis yields results very similar to that of the NN models which is to be expected if linear networks perform as well as nonlinear systems. Consequently, the results of these empirical models suggest that this problem space for predicting air combat engagement outcomes is primarily linear.

Study 4: Prediction of g Loading and Roll

Objective

In Study 4, we describe an effort to develop NN models to predict aircraft flight control variables. As in Study 3, and in contrast to Studies 1 and 2, our interest here is in deriving network models directly from empirical data. In addition, the models are aimed at actually controlling the inputs required to maneuver an aircraft. Hence, the models are similar to AML as maneuvering models, but rather than being based on an artificial maneuvering logic, they are derived from actual flight data.

Some of the specific questions in this part of the work include: (a) How accurate are neural network models in predicting flight control inputs from past airspace information? (b) How does the accuracy of the predictions vary as a function of increasing prediction time? (c) To what extent do the NN models need to use nonlinear information? (d) How well do models derived from one source of engagement data generalize to other engagement data? and (e) How does performance degrade as the set of predictor variables is reduced to be more psychologically meaningful?

Method

As with the models in Study 3, the models described in Study 4 were based on engagements flown in the SAAC at Luke AFB, AZ. The database for Study 4 consisted of four 1-v-1 engagements flown between a single subject matter expert (SME) and F-15 and F-16 pilots recruited specifically for the project. Each engagement resulted in a win, loss, or draw. Although the SME typically won the majority of the engagements flown against his opponents, an occasional pilot would outmaneuver the SME a majority of times. For Study 4, we selected SAAC data from a pilot who beat the SME three out of the four engagements. Because of the high skill level of this pilot, we assumed that these engagements were representative of particularly good maneuvering. We refer to the four engagements as Engagement 1, Engagement 2, Engagement 3, and Engagement 4. The four engagements lasted 51.5, 79.5, 148, and 59.5 s, respectively.

The SAAC data record consists of 32 variables that describe airspace situational information for each of the two aircraft, flight and weapons characteristics, and performance measures (see Table 17). Values for each variable were recorded once every 0.5 s. We refer to each 0.5 s data record as a pattern. Hence there were 103, 159, 296, and 119 patterns for the four engagements, respectively, for a total of 677 patterns.

Table 17. The 32 Variables in the SAAC Engagement Database.

antenna train angle	indicated air speed
angle off tail	earth axis x position
angle off tail azimuth	earth axis y position
angle off tail elevation	earth axis x velocity
range	earth axis y velocity
closing velocity	earth axis z velocity (vertical velocity)
line of sight azimuth	earth axis z acceleration
line of sight elevation	sideslip - absolute value of beta
aami for heat missile	pitch
aami for guns	roll
max aami for all weapons	yaw
romp	pitch rate
gs	roll rate
angle of attack	yaw rate
altitude	weapon code 1=radar, 2=heat, 3=guns
turn rate	specific excess energy

Selection of Input and Output Variables

The NN models are intended to predict future aircraft states given current situational information. Specifically, we chose to predict future g and roll angles. Knowledge of these two variables, along with thrust setting, offers a complete specification of flight control inputs. Our models attempted to predict the state of these variables for 1, 2, and 4 s into the future. Further, all of the models attempted to predict the future state of the SME's aircraft. We chose to model the SME because he is an acknowledged expert and also because there exists a large number of SAAC engagements from which the models may be both derived and later tested.

After eliminating g and roll angle from the set of data variables, we selected a subset of the remaining 30 variables in the SAAC database to serve as inputs to the models. To select this subset, we first correlated each of the 30 variables for both aircraft with the future values (at 1, 2, and 4 seconds) of g and roll. The primary purpose of this correlational analysis was to insure that we did not eliminate variables that were particularly good predictors of the outputs by themselves. However, our interest was not simply in maximizing prediction. Rather, we wished to construct models that were constrained by both the number of variables represented and by the type of information contained. In particular, we wished to exclude variables that were descriptions of low-level information (e.g., specific excess energy, side-slip) especially if they were not particularly predictive. In addition, we hoped to select inputs that resembled the set of inputs we used in our previous models of ACES and AML. This correspondence of the inputs should facilitate comparisons across models.

Based on these criteria we selected a candidate set of inputs consisting of 7 variables specific to each aircraft plus 2 variables giving relative information. These 16 variables along with their correlations (all of the correlations reported in this section are Pearson correlations) to the output variables across the 3 prediction times are shown in Table 18.

Several points need to be made regarding these correlations. First, the correlations were based on Engagement 3 data only. Engagement 3 was selected to serve as the source of training data for the models because it was the longest engagement (almost twice as long as the next longest engagement). Because of its length (approximately 2.5 min), the pilots had an opportunity to engage in rather lengthy maneuvering employing both strategies and counterstrategies. In contrast, short engagements are likely to culminate in quick kills and are more likely to reflect lucky decisions on the part of the winner or uninteresting errors on the part of the loser.

Second, the correlations in Table 18 were obtained by first offsetting the values for g and roll by the required number of patterns to reflect the prediction time. For example, to predict SME g 1 s in the future, the first two (two because patterns reflect 0.5 s) values of g were dropped and the remainder of the g's moved back two patterns. Because there were 296 patterns for Engagement 3, the correlations were based on 294, 292, and 288 patterns for predictions of 1, 2, and 4 seconds in the future, respectively.

Notice first in Table 18 that there is considerable variability in the predictiveness of the input variables. The correlations range from around 0 to .96. Also, the predictions of g are generally higher than of roll angle. The best predictors for g are the SME aircraft variables (particularly indicated airspeed, vertical velocity and altitude), whereas the best predictors of SME roll angle are his opponent's variables. In the case of g, the correlations tend to decrease with prediction time, as might be expected. However, the correlations actually increase for predicting roll angle as future time increases. We were curious about this phenomenon, wondering just how far out this trend might hold. So we correlated the input variables with roll angle for times beyond 4 s. The results showed that the correlations peak at 4 s and decline thereafter. We have not yet determined whether this phenomenon is generally true for predicting roll angle or is specific to this engagement. In any case, we can already see some interesting differences between predicting future g and roll angle simply based on the correlational analysis.

Table 18. Correlations Between 14 Variables and Values of SME g and Roll for 1, 2, and 4 Seconds in the Future.

	SME g			SME Roll		
	1 s	2 s	4 s	1 s	2 s	4 s
Opponent aircraft variables						
antenna train angle	.10	.11	.04	-.06	-.11	-.10
angle of attack	-.33	-.32	-.33	-.10	-.01	.11
altitude	.58	.56	.50	-.18	-.21	-.27
turn rate	.19	.15	.09	-.21	-.21	-.20
indicated air speed	.69	.70	.71	-.13	-.16	-.22
vertical velocity	.00	-.04	-.05	-.30	-.31	-.34
pitch	-.02	.02	.03	.23	.24	.28
SME aircraft variables						
antenna train angle	.09	.09	.01	-.01	-.10	-.20
angle of attack	-.34	-.49	-.61	-.05	-.13	-.08
altitude	.55	.55	.54	-.21	-.23	-.27
turn rate	.44	.44	.42	.09	.05	-.10
indicated air speed	.96	.94	.83	.10	.08	.01
vertical velocity	.64	.66	.59	-.12	-.08	.00
pitch	-.41	-.50	-.58	.13	.10	.03
Relative variables						
range	.60	.52	.39	-.03	-.02	-.03
closing velocity	-.06	-.06	.05	.11	.13	.05

Selection of Training and Test Patterns

Having selected the input and output variables for the NN models we are now ready to identify a set of training and test patterns. As mentioned earlier, we selected Engagement 3 to serve as the primary source of training data. Because we wished to use Engagement 3 for both training and testing the models, we split its 296 patterns in half. The first half would serve as training patterns and the second half as test patterns. We partitioned Engagement 3 patterns in two different ways. In the first method, which we call the Interleaved Training Set, we selected the odd-numbered patterns (which would now occur in 1-s intervals) to serve as training patterns and used the even-numbered patterns to test generalization. One might argue that this is not a very stringent test of generalization because of the close connection in time between the training and test patterns. However, because of the very temporal nature of the task, we felt it was an important case to consider. Further, the second method of partitioning the patterns would allow us to examine this time-dependency relation between training and test patterns. In this second method, which we call the Split-Half Training Set, the training patterns come from the first half of the engagement and the test patterns from the second half.

The Interleaved Training Set consisted of 144 patterns, the minimum number of distinct patterns available for training across the three prediction times (148 one-second patterns minus 4 patterns). The Split-Half Training Set consisted of 140 patterns (148 half-second patterns minus 8 patterns). In both cases, the number of test patterns was the same as the number of training patterns.

Each variable was first transformed to the range [0,1] in order to standardize the network weights. A linear transformation converted the values by subtracting the minimum from each score and then dividing this difference by the range. The minimum and maximum for each variable was based on all 677 patterns across the four engagements.

Network Topology

We investigated two basic topologies for the NN models, one with a layer of hidden nodes (i.e., multilayer network) and one without (i.e., single-layer perceptron model). The comparison of models with and without a hidden layer addresses the question of the nonlinearity of the task. For models with a hidden layer, the number of hidden nodes was chosen to be approximately one-fourth the number of input nodes. This value was selected after trying several different numbers of hidden nodes, usually in the range of one-tenth to one-half of the number of inputs. Based on both the degree of learning of the training patterns and the extent of generalization for the test patterns, we found that the one-fourth value gave the best overall performance. The output of all the networks consisted of one node for either g or roll.

Network Training

All of the networks were trained using a standard back-propagation learning method. The learning rate was set at .45. Recall that a training epoch is the presentation of each pattern in the training set once to a model. For each epoch, the training patterns were randomly ordered. Each model was submitted to 5,000 epochs of training. This upper bound of 5,000 epochs was chosen to ensure that the models had adequate time to train. In fact, the set of weights that resulted in the maximum percent classification, and hence the set used to test generalization, often occurred much earlier in the training.

The tolerance criterion was set at .04. That is, a pattern was considered successfully classified if the actual output value was within .04 of the desired output value. Given a tolerance value, the performance of a model is measured by the percent of patterns (either in training or testing) that resulted in outputs within the specified tolerance. The .04 criterion mapped into actual units of approximately one-half g and 15 degrees of roll angle. Hence, a pattern was considered correctly classified if the output value was within these limits.

Results

We first performed a simple regression analysis to determine how much of the variance in future outputs could be accounted for by a simple linear combination of the input variables. The analysis was based on 294, 292, and 288 patterns for predicting the SME's output values 1, 2, and 4 s in the future, respectively. The regression analysis was performed on data from Engagement 3.

Table 19 shows the multiple R values for each of the six regression models along with the t-scores for each of the predictor variables. Several observations made from examining the simple correlations in Table 18 are corroborated here. First, the prediction of future g is higher than future roll angle. Approximately 80% of the variance in g 4 s in the future can be accounted for by a simple linear combination of the variables. In contrast, at best only about 14% of the variance of future roll angle is explained. Based on this analysis we should expect a perceptron model to do well in modeling future g but poorer for roll angle. We turn to these models shortly. Second, although many of the variables that were individually highly correlated with the outputs are good predictors in the regression model, there are some exceptions. For example, although antenna train angle is not highly correlated with the outputs individually, it does play a significant role in several of the regression models.

Table 19. Multiple R Values and t-Scores of Individual Predictor Variables for the Six Linear Regression Models Predicting SME g and Roll at Future Times of 1, 2, and 4 Seconds.

	SME g			SME roll		
	1 s	2 s	4 s	1 s	2 s	4 s
Multiple R	.96	.94	.90	.31	.30	.37
Opponent aircraft variables						
antenna train angle	-2.55	-2.99	-2.42	-.11	-.86	-3.32
angle of attack	-.54	.39	1.15	-2.33	-2.05	-2.97
altitude	-.64	.10	-1.04	2.22	.90	-2.10
turn rate	1.41	.32	-1.28	-1.38	-.40	.77
indicated air speed	-1.01	1.61	6.02	-3.08	-3.27	-3.13
vertical velocity	-1.46	-3.41	-6.45	-1.45	-1.59	-.08
pitch	-1.01	-1.99	-3.77	-1.30	-.81	2.18
SME aircraft variables						
antenna train angle	-1.79	-.81	2.22	-.15	-1.26	-6.33
angle of attack	2.23	-5.49	-5.92	.28	-2.39	-2.95
altitude	.68	-.04	1.00	-2.51	-1.15	1.83
turn rate	6.13	.76	-1.80	2.96	2.57	1.17
indicated air speed	18.62	13.23	5.90	3.09	2.13	1.72
vertical velocity	4.09	2.90	1.14	-2.95	-1.38	-.42
pitch	1.69	-1.17	-4.41	-2.31	-.60	1.04
Relative variables						
range	1.87	-1.84	-5.41	-.22	1.42	3.78
closing velocity	-3.72	-3.34	1.33	1.08	.04	-5.64

NN Models: Interleaved Training and Test Patterns

The first set of NN models attempt to predict future g and roll angle using the 16 input variables shown in Tables 18 and 19. These models were trained and tested on the Interleaved Training Set. As mentioned earlier, the training patterns always came from SME data. Two different topologies were used, a 16-1 single-layer network and a 16-4-1 multilayer network. The results of both training and a generalization test are shown in Table 2.

The most striking result is the difference in predictiveness between g and roll angle. Considering the 16-1 network first, the percent of patterns correctly trained for g was roughly three times the number for roll angle. In agreement with the correlation and regression analyses just reported, we find that roll angle is a more complex and likely nonlinear classification problem.

Table 20. Percent Predictions in Tolerance for Training and Test Patterns for a Perceptron (16-1 topology) and a Multilayer Network (16-4-1 topology) based on the Interleaved Training Set.

	Training		Generalization	
	16-1	16-4-1	16-1	16-4-1
g				
1 s	91	93	92	95
2 s	89	89	88	91
4 s	80	85	77	86
Roll				
1 s	26	43	26	40
2 s	26	50	21	41
4 s	26	49	24	40

By considering the performance of the multilayer network we can determine how much a nonlinear model improves classification. We see from Table 20 that the 16-4-1 networks resulted in only slightly better classification performance of future g, but considerable improvement occurred in predicting roll angle. Even here, however, at best 50% of the patterns in the training set were correctly classified. These results suggest that the set of input variables may be insufficient for predicting roll angle.

Note also in Table 20 that, for both g and roll angle, the improvement that occurs with the 16-4-1 models increases with higher prediction times. Perhaps it is not too surprising that longer prediction times require more complex and hence more configural models.

Finally, we examine how well the trained models generalize to a new set of test patterns. The results, also shown in Table 20, are that, overall, the models generalize quite well. The percentage of correct classifications for test patterns is about as high as for training patterns. Generalization of the models trained to predict g was particularly impressive; in several cases the percentage of correct classification for the novel test patterns was higher than for the training patterns. Generalization was somewhat poorer for roll angle, indicating again the greater difficulty in predicting this variable.

Recall that the training and test patterns for the Interleaved Training Set were selected from every other half-second of data across the engagement. Such a close correspondence in time between the two sets of patterns would likely enhance the ability of the models to generalize. A more stringent and perhaps more realistic test is whether a model trained during one segment of an engagement could predict the output variables during another segment. We turn to this issue in the next set of models.

NN Models: Split-Half Training and Test Patterns

The same types of models were trained here as for the last analysis but now with the Split-Half Training Set. Table 21 shows the results. Examining first the models trained to predict g, we find that the training performance for both the single-layer and multilayer networks is comparable to the earlier models, obtaining around 85 to 95% correct. We also see that the models' predictions decrease as future time increases, and there is a slight advantage of the multilayer network over the single layer. However, the generalization data tell a somewhat different story than before. Generalization still looks strong for predictions of 1 s in the future, but as prediction time increases, the models' ability to generalize falls off rather sharply. This pattern of results suggests that the high generalization performance found for the Interleaved Training Set was in fact due to the close temporal association of the training and test patterns. Given the more realistic task of predicting g for a segment of the engagement different from the training segment, and for times beyond 1 s, the models' generalization performance begins to decrease. Notice also that under these more difficult conditions, the multilayer network generalizes best. With the 16-4-1 model, it is possible to predict g 4 s into the future on 63% of the test patterns.

Consider next the models trained to predict roll angle. First notice that the training performance for both topologies was higher than for the Interleaved Training Set. Although this difference could be due simply to a different sample of training patterns, we suspect that it is a result of something else. Recall that for the Split-Half Training Set the patterns come from half-second data whereas the Interleaved Training Set contains 1-s data. The closer the patterns are in time the more similar they are. Therefore, it would seem to be an easier task to train a model on half-second than full second data, particularly if the classification task is difficult to begin with. Also, notice again the marked improvement of the multilayer over the single-layer network. In fact the performance of the multilayer model for predicting roll angle is beginning to approach the performance of the multilayer model for g (77 vs. 88). However, the picture changes drastically when we consider the generalization data.

Put simply, the roll angle models do not generalize with the Split-Half Training Set. Two points need to be made. First, the generalization found for the Interleaved Training Set was very likely due to the close time correspondence between the training and test patterns. When tested on a completely different segment of the engagement, the models predict poorly. Second, despite the advantage of the multilayer network in learning to classify the training patterns, there is little if any generalization advantage. The likely conclusion seems to be that the set of input patterns is insufficient for predicting future roll angle. We will return to this point later during the summary. For the moment, however, we continue to focus on g by investigating a new set of models with reduced information in the input variables.

Table 21. Percent Predictions in Tolerance for Training and Test Patterns for a Perceptron (16-1 topology) and a Multilayer Network (16-4-1 topology) based on the Split-Half Training Set.

	Training		Generalization	
	16-1	16-4-1	16-1	16-4-1
g				
1 s	92	97	97	82
2 s	88	94	70	70
4 s	85	88	31	63
Roll				
1 s	42	65	6	6
2 s	41	70	5	4
4 s	43	77	8	10

NN Models: More Psychologically Realistic Inputs

The question we address here is whether our success so far in modeling g might be due to having included input variables that relate to g through simple flight-related associations. For example, g, turn rate, and airspeed are simple derivatives of one another. Perhaps simply knowing the current values of these variables allows good prediction of future g. If so, a more interesting question would seem to be to assess how well future g could be predicted from knowledge of more psychologically meaningful variables such as the opponent's relative position and orientation, and relative information such as range and closing velocity. To pursue this issue we eliminated five variables from the set of inputs used in the previous models: the SME's airspeed, turn rate, angle of attack, and the opponent's airspeed and angle of attack. We used the remaining 11 variables to train single-layer (11-1 topology) and multilayer (11-3-1 topology) models to predict future g. The models were trained from the Split-Half Training Set. The results are given in Table 22.

Table 22. Percent Predictions of g in Tolerance for Training and Test Patterns for a Perceptron (11-1 topology) and a Multilayer Network (11-3-1 topology) based on the Split-Half Training Set.

	Training		Generalization	
	11-1	11-3-1	11-1	11-3-1
g				
1 s	67	88	30	40
2 s	59	87	11	38
4 s	64	84	0	53

As might be expected, the percentage of patterns learned during training dropped for the reduced set of inputs. However, the reduced models for predicting g were still better than were the full models for predicting roll angle. Notice also that the decline in percent classification for the reduced models was less for the multilayer than single-layer network and less for longer than shorter prediction times. In fact, the reduced multilayer model for predicting g 4 s into the future learned as well as the comparable full model. Unfortunately, this trend does not hold for the generalization test.

The generalization performance of the reduced models falls considerably below the full models, suggesting that at least part of the earlier success was due to the low-level flight relations among the variables. Despite the overall poorer showing, the multilayer network predicted around one-half of the SME's g values 4 s into the future.

Discussion

In summary, the results of predicting future flight control variables by NN models were somewhat mixed. First, clear differences were found in predicting g and roll angle. Although NN models were fairly successful at predicting future g, the results were disappointing with roll angle. Second, there was an overall advantage of multilayer over single-layer models. The nonlinear dimensions of the task, and hence the multilayer advantage, were more pronounced for predicting roll angle than g, for predicting longer than shorter future times, and for predictions based on a reduced rather than a full (i.e., 11 vs. 16) set of inputs. Third, the relatively high performance found with the Interleaved Training Set was likely a function of the close temporal connection between the training and test set.

It is important to keep in mind some of the limitations of the current work. Although we attempted to select an initial set of input variables that were good predictors, it is possible that some particularly good combinations of variables were excluded. Perhaps a more thorough analysis of sets of input variables would turn up more predictive inputs. Also, the models were based on data from a single engagement and a single pilot. Although we believe our rationale for choosing the particular engagement and pilot was justified, additional work is clearly needed to extend the modeling efforts beyond these limitations. Finally, a finer-grained analysis of prediction time seems warranted. Several intriguing results deserve further exploration, including the better training performance for roll angle under longer prediction times and the occasional better generalization under longer times for predicting g.

General Conclusions and Future Directions

As is usually the case in research, our work under this contract has raised more questions than it has answered. Our attempts to train NN models using ACES' rules revealed several problems in defining training patterns from production rules. Our limited success in capturing the ACES logic suggests either that we have not yet hit upon the correct method for representing production rules in NN training patterns or that the types of neural nets we have investigated do not lend themselves to representing the logic contained in ACES. ACES is a classical symbolic model representing decisions in discrete rules while the NN models used numerical values of variables directly. Perhaps the difference between the discrete symbolic and analog representations is responsible for the differences in the two approaches. As another possibility, the back propagation learning algorithm is a hill climbing procedure, and as such is subject to the problem of getting stuck on local maxima (or minima, depending on the structure of the problem). There are other methods, such as simulated annealing, that are less sensitive to the problem of local maxima. Some additional work employing such methods to learn ACES' rules would be of value.

Our work in modeling AML logic was more limited, but the results we obtained suggested that NNs are capable of incorporating AML logic. Unfortunately, AML has serious limitations as an automated adversary so we concentrated more on developing models from SAAC data.

The models based on SAAC summary data indicated that linear NN models were quite comparable to models based on discriminant analysis as we expected. Further, nonlinear NN models did not perform any better than the linear ones. This finding shows that linear models are sufficient for this problem which is not easily determined using regression procedures. However, once the linearity of the problem is established, the well-established statistical models are probably preferable to the NN models because of the well-developed analytical procedures available with the statistical models.

The models trained to predict aircraft control variables in the SAAC engagement database produced some interesting results. We found that we could more accurately predict future g's than future roll angle. Predicting g's was largely linear whereas predicting roll angle was superior with nonlinear models. When the generalization test was conducted on entirely different time segments of an engagement, prediction of g's was still relatively good, the models failed to predict roll angle at all. We should emphasize that these models were trained on segments of a single engagement, and more extensively trained models may behave differently.

Finally, we suggest some additional research issues for future work. First, it is interesting to consider the performance of AML in light of our attempts to model g and roll angle with NN models. One might ask why AML does as well as it does given the complexity of the task. The answer is that AML's decision logic is based on extrapolating the future position of its opponent from past situational information. We wonder how much improvement in the current NN models would result from incorporating similar information. Instead of formally extrapolating the opponent's future state and representing it explicitly as inputs, one possibility would be to give the model input nodes that corresponded to historical situational information of the opponent. The question then is whether an NN model could learn to perform the necessary extrapolation of future information and then use this information to predict future states.

A second line of future work involves building models specific to particular airspace situations (e.g., offensive, defensive, neutral). Given the dynamic nature of ACM, it is perhaps unreasonable to expect single models to predict future flight control variables under all conditions. However, if engagement data could first be parsed into engagement types, appropriate models could then be invoked.

References

Arbib, M. I. (1986). *Brains, Machines, and Mathematics.* New York: Springer-Verlag.

Burgin, G. H., Fogel, L. J., & Phelps, J. P. (1975). *An Adaptive Maneuvering Logic Computer Program for the Simulation of One-on-One Air-to-Air Combat.* (Vols. I and II). Contractor Report No. NASA CR-2582, National Aeronautics and Space Administration, Wasington DC 20546.

Dearholt, D. W., & Schvaneveldt, R. W. (1990). Properties of pathfinder networks. In R. Schvaneveldt (Ed.), *Pathfinder associative networks: Studies in knowledge organization.* (pp. 1-30). Norwood, NJ: Ablex.

Goldsmith, T. E., Schvaneveldt, R. W., & Brunderman, J. (1985). *Simulating Air Combat Expertise.* Technical Paper No. AFHRL-TP-85-37, Williams, AFB, AZ: Operations Training Division, Air Force Human Resources Laboratory.

Hebb, D. O. (1949). *The Organization of Behavior.* New York: Wiley.

Lippmann, R. (1987, April). An introduction to computing with neural nets. *IEEE ASSP Magazine.* pp. 4-22.

McClelland, J. L., Rumelhart, D. E., & the PDP Research Group, (Eds). (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol 2. Cambridge: MIT Press.

Nilsson, N. (1990). *The Mathematical Foundations of Learning Machines.* San Mateo: Morgan Kaufmann Publishers.

Plaut, D. C., Nowlan, S. J., & Hinton, G. E. (1986). Experiments on learning by back propagation. Department of Computer Science Technical Report, Carnegie-Mellon University (CMU-CS-86-126).

Rosenblatt, F. (1962). *Principles of Neurodynamics.* New York: Spartan.

Rumelhart, D. E, McClelland, J. L., & the PDP Research Group (1986). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (Vol 1). Cambridge, MA: MIT Press.

Rumelhart, D. E. & Zipser, D. (1986). Feature discovery by competitive learning. In D. Rumelhart & J.McClelland (Eds.), *Parallel Distributed*

Processing, Explorations in the Microstructure of Cognition, Volume 1: Foundations. Cambridge: MIT Press.

Rumelhart, D. E., Hinton, G. E., & Williams, (1986). Learning internal representations by error propagation. In D. Rumelhart & J.McClelland (Eds.), *Parallel Distributed Processing, Explorations in the Microstructure of Cognition*, Vol. 1: *Foundations*. Cambridge: MIT Press.

Schvaneveldt, R. W., Anderson, M., Breen, T. J., Cooke, N. M., Goldsmith, T. E., Durso, F. T., DeMaio, J. C., & Tucker, R. G. (1983). *Conceptual Structures in Fighter Pilots*. AFHRL-TP-83-2, Flying Training Division, Air Force Human Resources Laboratory.

Schvaneveldt, R. W., Durso, F. T., & Dearholt, D. W. (1989). Network structures in proximity data. In G. Bower (Ed.), *The psychology of learning and motivation: Advances in research and theory*, Vol. 24 (pp. 249-284). New York: Academic Press.

Schvaneveldt, R. W., Durso, F. T., Goldsmith, T. E., Breen, T. J., Cooke, N. M., Tucker, R. G., & DeMaio, J. C. (1985). Measuring the structure of expertise. *International Journal of Man-Machine Studies*, 23, 699-728.

Schvaneveldt, R. W., Goldsmith, T. E., Durso, F. T., Maxwell, K., Acosta, H. M., & Tucker, R. G. (1982). *Structures of Memory for Critical Flight Information*. AFHRL-TP-81-46, Flying Training Division, Air Force Human Resources Laboratory.

Waag, W. L., Raspotnik, W. B., & Leeds, J. L. (1992). Development of a composite measure for predicting engagement outcome during air combat maneuvering (ACM). AL-TR-1992-0002. Williams AFB, AZ: Armstrong Laboratory.

Widrow, B. & Hoff, M. C. (1960). Adaptive switching circuits. In 1960 IRE WESCON Convention Record. New York: IRE, pp. 96-104.