THE APPLICATION OF ARTIFICIAL NEURAL SYSTEMS TO THE TRAINING OF AIR COMBAT DECISION-MAKING SKILLS

Michael X. Crowe
Advanced Concepts Department
Ball Systems Engineering Division
5580 Morehouse Drive
San Diego, CA 92121-1709

ABSTRACT

The Air Force Human Resources Laboratory (AFHRL) has established a program to design, develop, and validate an expert model of pilot decision-making in Air Combat Maneuvering (ACM) to be used in the training of fighter aircraft pilots. The intent of this program is to create a computer-based simulation which can encapsulate the expertise of combat pilots in ACM strategy, tactics, and offensive and defensive decision-making. The resulting expert system is to be incorporated into a flight simulation package to support the training of these ACM skills to student combat pilots. The development of the ACM Expert System is based on the latest advancements in the technology of **Artificial Neural Systems (ANS)**.

To effectively train student combat pilots in ACM skills, it is desirable to move beyond the textbook, allowing the student to interact with a simulated adversary aircraft via computer. Unfortunately, it is difficult to capture ACM expertise in computer software which will provide the student with realistic and reasonable adversary behavior. Most existing systems use "pre-canned" profiles or simple trajectory generators. The more advanced adversary simulators use rule-based expert systems to represent and recall pilot expertise and create a more reactive system. However, traditional expert systems suffer from major inadequacies which have limited their success. By using an ANS approach to this problem, actual human ACM performance data is being used to "train" an expert system in ACM decision-making skills. This system is capable of simulating human ACM performance by learning to associate the recognition of a tactical situation with the selection of the proper course of action. The objective of this paper is to describe current efforts to apply ANS technology to the training of ACM decision-making skills.

INTRODUCTION

Program Objectives

The objective of the Air Combat Maneuvering Expert System (ACMES) program is to create an expert model of pilot decision-making in Air Combat Maneuvering (ACM) using Artificial Neural Systems (ANS) technology. This expert system trainer is to be capable of providing ACM decision training to F-15, F-16, and T-38 pilots by planning and selecting a series of optimal flight maneuvers under realistic adversary situations. The expertise of ACM tactics, offensive and defensive decision-making to determine the best course of action under specific tactical conditions, and certain training objectives are to be encapsulated in a computer-based expert system. The expert model is then to be integrated with a man-in-the-loop flight simulation software package utilizing a three-dimensional, color graphics display and interface system. This system will allow student pilots to fly against simulated adversaries whose maneuver responses are both reactive and realistic. Furthermore, the expert system will be capable of guiding the student through optimal maneuver sequences for training purposes.

The current focus of this program is how best to apply ANS techniques to augment and extend traditional expert system technology. Many techniques have been explored over the last two decades for capturing, representing, and recalling knowledge in a fast and reliable fashion, but all have suffered from major inadequacies in terms of knowledge acquisition techniques, knowledge base size, speed of recall, generalization of knowledge, and the ability to handle incomplete or inaccurate data. Though some improvements have been made using traditional Artificial Intelligence (AI) methods, current systems continue to become bogged down by size and speed constraints, and many of the aforementioned technical hurdles remain. The field of Artificial Neural Systems, also known as **Connectionism** and **Artificial Neural Networks**, has shown much promise in overcoming some of the more intractable elements of simulating intelligent behavior [2], and therefore, is being explored as a possible approach to accomplishing the objectives of the ACM Expert System program.

Top-Level Description of ACM Decision-Making

The intent of the ACM Expert System is to simulate the behavior of fighter pilots who have acquired a certain proficiency in air-to-air combat. By directly modeling human experts, the details and nuances of successful ACM performance can be captured and used to train others to achieve that level of performance. In an air-to-air engagement, once an enemy aircraft has been detected, the pilot begins a decision process based both on his own goals and expectations and on the actions of the enemy. The basic goal of an aircraft pilot during ACM is to destroy the enemy aircraft while simultaneously avoiding destruction of his own aircraft. The pilot accomplishes this goal by maneuvering his aircraft through three-dimensional space in an effort to obtain a position (relative to the enemy aircraft) which allows the enemy aircraft to be attacked. At the same time, the pilot must avoid the enemy's attempts to maneuver into an attack position against the pilot's own aircraft.

Though the above description provides an adequate top-level outline of the fundamentals of air combat, there are a number of other factors which influence the performance of ACM, such as the limitations of the aircraft and weapon systems, the use of countermeasures, weather conditions, and the prevailing rules of engagement. A fielded ACM training system would require that these variables be taken into account to provide the proper "success" criteria profile which would emphasize and build upon all the relevant elements of effective maneuvering and weapons employment. However, the basic requirement of air combat, and the focus of the ACM Expert System, is the generation a successful sequence of maneuvers based on the relative geometry of the engaged aircraft. A "successful" engagement is one in which the elimination of the threat is coupled with the survival of ownship. This is the framework upon which the ACM Expert System has been established.

In general terms, a **maneuver** is defined simply as a change in the current flight path of the aircraft, for example, alterations in position, orientation, or velocity over time. Maneuvers can, however, be described at different levels. A simple maneuver might consist of a slight change in the aircraft's pitch and thrust to attain a new altitude, or it might be a complicated sequence of flight path commands which result in the aircraft flying an inside loop with a barrel roll. A coordinated sequence of smaller flight path changes connected in a specific way is referred to as a basic fighter maneuver (BFM). Variations and combinations of these BFMs are often used during certain ACM situations as strategies to gain or maintain a tactical advantage. The question for the pilot (and for the ACM Expert System) is how to select the right set of changes to his flight path, or the proper BFM, which brings him safely into position to destroy the enemy aircraft.

There are three basic issues at the design level which have been used to guide the development of an Expert System to simulate the decision-making of pilots during ACM: 1) What does the pilot need to know when he makes ACM decisions, 2) How is that information combined to arrive at maneuvering decisions, and 3) What is the form of the pilot's decision output that leads to changes in the aircraft's flight path? At the implementation level, these issues form the basis of how to apply neural network technology to bring about new solutions in the simulation of ACM decision-making. The goal of this effort is to train an expert system, using examples of human performance, to take tactical situation data as input and produce the proper maneuver response as output.

THE NEURAL NETWORK APPROACH

Introduction to Artificial Neural Systems

Artificial Neural Systems (ANS) technology provides a methodology for combining input knowledge to produce a corresponding, appropriate output response through a "self-organization" of the representational system. In other words, a neural network produces a mapping which relates the input space to the output space. This methodology may be applied to any problem where the underlying function of association between input and output is complex or unknown. As applied to the ACM Expert System problem, this mapping capability is utilized to model human performance by learning the association between input and output conditions. Rather than capturing expertise as a set of logical "if-then" rules, as is done in traditional AI expert systems, a neural network develops expertise by adapting its internal arrangement in response to examples of expert behavior. While some ACM simulations rely on the diagnostic, pattern recognition capabilities of AI expert systems [4], others use a value-driven, trajectory prediction technique to determine a course of action [1]. An ANS-based expert model can combine these approaches in that the system can learn to associate the recognition of an input pattern with the selection of the proper course of action.

A prototype, neural-network-based ACM Expert System has been developed, called the Artificial Neural System for the Representation and Collection of ACM Decision-Making Expertise, or ARCADE. The ARCADE neural network creates a mapping or association from the tactical situation (input space) to the appropriate maneuver response (output space), as is shown in Figure 1. The source of ACM expertise for the ARCADE networks is data selected from the Simulator for Air-to-Air Combat (SAAC), which is a man-in-the-loop, multi-dome air combat simulator located at Luke Air Force Base. The underlying assumption of this expert system development process is that, given a sufficient representation of the input and output parameters, there exists a general relationship between tactical conditions and maneuver responses that can be resolved and duplicated computationally. The internal arrangement and behavior of a neural network which allows such a mapping to be accomplished is reminiscent of the massively parallel and highly distributed processing arrangement found in biological nervous systems. Specifically, neural networks are biologically motivated models of information processing in that they use the interactions of many simulated neurons to store, recognize, and recall knowledge.

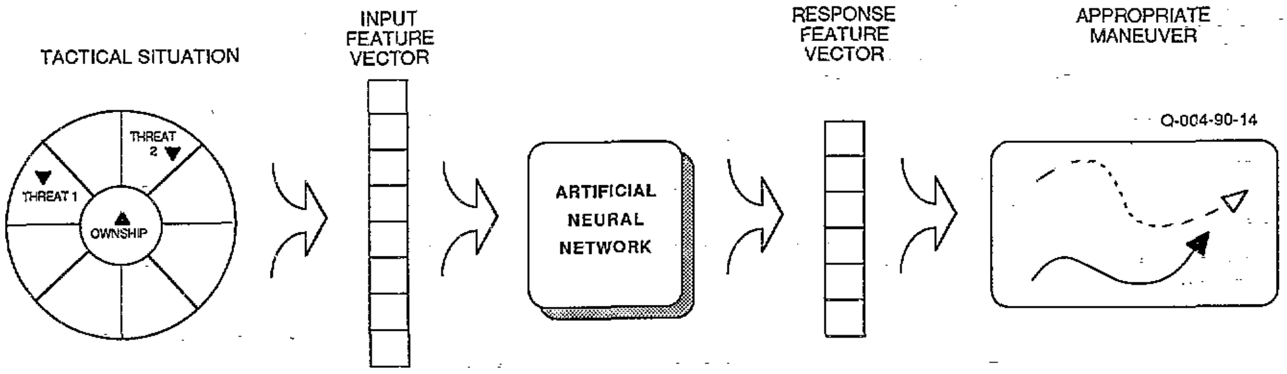

Figure 1. Approach to the ACM Expert System Using a Neural Network to Associate Tactical Situations to Appropriate Maneuver Responses.

Neural networks are often constructed as a hierarchy of **layers**. Each layer contains some number of simulated neurons, technically known as **processing elements (PEs)**, which are interconnected throughout the network. The strengths and structure of these interconnections are what determine the system's ultimate operation. The strength of a connection, that is, the extent to which one PE influences the **state** of another, is also known as the **weight** of the connection. A pictorial representation of a typical processing element in a neural network is shown in Figure 2.

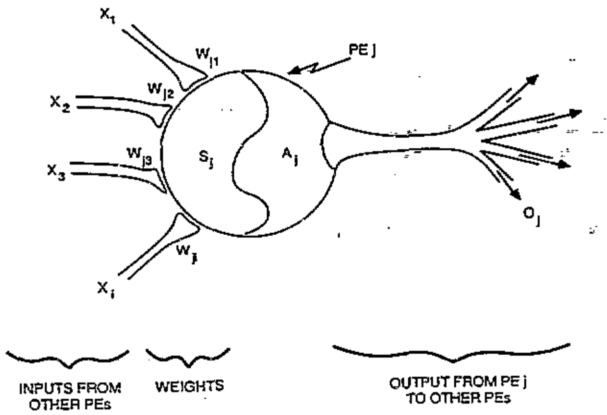

Figure 2. Schematic Representation of a Typical Neural Network Processing Element.

In the figure, processing element **j** receives some number of inputs, i, from other PEs in the network. The levels of each of these input signals, x_i, are multiplied by their connection weights, w_{ji}, and summed together to produce S_j, the total signal into j. In general, this summation represents the external influence of other PEs on PE **j**, and is used to calculate an update value to $A_j$new, the current level of activation, or state, of PE **j**. The new activation level is then applied to a threshold function to determine O_j, the output signal to be produced by j. The output is then received by other PEs in the network which each perform a similar process to the one just described. The activity of an individual PE as described above can be represented symbolically by the following equations:

$$S_j = \sum_i x_i w_{ji}, \qquad (1)$$

$$A_j\text{new} = f(A_j\text{old}, S_j), \qquad (2)$$

$$O_j = g(A_j\text{new}) \qquad (3)$$

where S_j is the summation value of PE **j**, $A_j$new is the new activation level of PE **j**, $A_j$old is the old activation level of PE **j**, f() is some activation function which produces the new activation based on the current activation and summed input, O_j is the output of PE **j**, and g() is some thresholding function for determining the level of output. Thresholding equations generally have a sigmoidal form as in equation (4).

$$O_j = \frac{1}{1 + e^{-A_j new}} \qquad (4)$$

Since the weights and the PE activation levels can have both positive and negative values, the processing elements can have both excitatory and inhibitory influences on other PEs in the network. Processing element output levels are usually continuous values scaled between 0 and 1 or between +1 and -1. Weight values may vary over a wider range, and a larger absolute weight value represents more potential influence among PEs. A weight value of zero indicates that no connection exists between those two PEs.

Many ANS paradigms have been developed to accomplish the determination of a mapping between input and output data. Each paradigm has a different approach for associating the various layers of processing elements and updating the weights between PEs in response to training. The specific neural network paradigm used in the ARCADE system is known as the **Multilayer Back-Propagation (MBPN)** algorithm and is the most common and most successful neural network algorithm in current usage. The form and function of the MBPN paradigm is particularly suited to the development of an expert system which learns to simulate intelligent behavior through exposure to actual examples of human performance.

The internal rules and procedures for how the neural network arrives at a set of associations for ACM performance need not be specified by the system designer. Rather, ACM expertise is represented in the patterns of activations and the weighted connections of the network's processing elements. The neural network requires only that the problem be represented in terms of an input vector which represents the current tactical situation, and a corresponding output vector which determines the correct maneuver response. During the network's learning process, expert performance is used to generate the values of these input and output vectors which are then assigned to certain processing elements of the neural network as their activation levels. Through a series of associated input and output examples, these assigned activation levels methodically influence and adjust the rest of the network until a general solution to the mapping between input and output is found.

In the ARCADE network, there are three or more layers of processing elements. The first layer is the input layer to which the tactical situation is presented, and the last layer is the output layer which produces the response vector. The component values of the input and output vectors are represented by the activation levels of the PEs in those layers. For example, the value for the current altitude of the blue aircraft may be represented by the activation level of a certain PE in the input layer, and a commanded velocity value might be mapped to a specific output layer PE. Between the input and output layers are one or more **hidden** layers of PEs which are responsible for building up explicit and implicit associations between the input and output feature vectors. This multi-layer arrangement allows associations of a more abstract nature to be formed than would be possible with just two connected layers. A general diagram of a typical three-layer back-propagation network structure can be found in Figure 3. The weights in such a structure would occur where connections are made from the input layer to the hidden layer and from the hidden layer to the output layer.

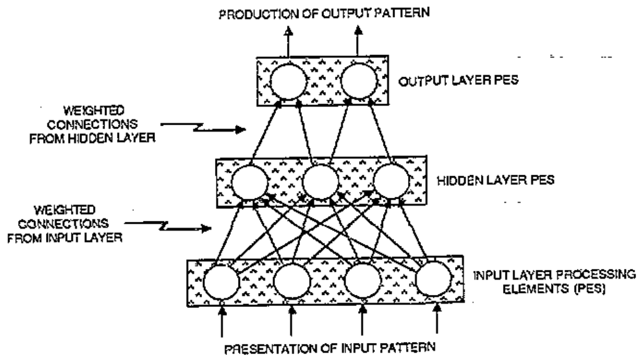
PRODUCTION OF OUTPUT PATTERN
OUTPUT LAYER PES
WEIGHTED CONNECTIONS FROM HIDDEN LAYER
HIDDEN LAYER PES
WEIGHTED CONNECTIONS FROM INPUT LAYER
INPUT LAYER PROCESSING ELEMENTS (PES)
PRESENTATION OF INPUT PATTERN

Figure 3. **Structure of a Typical Three-Layer Back-Propagation Network.**

The back-propagation algorithm works by employing a concept known as the **generalized delta learning rule.** This means that the neural network is **trained** to associate specific sets of input and output data by being taught the difference, or **delta**, between the results it produces and the actual desired result. Neural network training simply means presenting the network with repeated examples of associated input and output data and allowing the system to adjust itself when mapping errors occur. Naturally, the errors will be quite large when training begins but will gradually decrease as training continues. When fully trained, the network will be capable of reproducing the response performance of the data upon which it was trained. In many cases, a properly trained network is capable of providing the correct output response to "noisy" input data or even to input upon which it has never been specifically trained. Neural networks are capable of arriving at **generalized** solutions based on a limited set of training data.

Advantages of ANS Technology

Artificial Neural Systems provide some very specific advantages over other knowledge engineering technologies. Primarily, neural networks allow the knowledge engineer to overcome the knowledge acquisition bottleneck that plagues the current generation of expert systems. Since the neural network is trained on examples of existing performance data, there is no requirement to generate a precise protocol and translation of the expert's knowledge structure. Expertise need not be explained in detail; rather, the network achieves its goals by simply copying the expert behavior and correcting its own mistakes. Another advantage of ANS architectures is their fault tolerance capabilities. This includes the ability to respond with reasonable output to noisy or incomplete data, and the graceful degradation of system responses in novel situations. Most expert systems will simply fail to function at all when presented with situations for which they were not explicitly programmed. Rule-based expert systems also have a reputation for being very slow, especially when the set of rules begins to achieve any appreciable size. The neural network approach provides the possibility of real-time, simulated expert response to a wide range of input/output conditions. This is because, unlike the training process, which may require many iterations of example data, a response to input requires only a single feed-forward pass through the network with a constant time requirement. Finally, the inherent structure of neural networks allows a natural mapping to parallel hardware systems that are now becoming available.

Hardware and Software Description

Neural network processing can place an extreme burden on conventional computer hardware, particularly during training of the network, so specialized equipment is now being utilized to provide greater size and speed capabilities. The development and implementation of the ARCADE system makes use of a customized neural network processing board called the ANZA-Plus

Neurocomputer from Hecht-Neilsen Neurocomputers (HNC). The ANZA-Plus coprocessor is part of an 80386-based computer system which is optimized for training and executing neural network software. A software development package for building and interfacing to various neural networks is included with the ANZA-Plus system. The ANZA-Plus coprocessor has a maximum combined capacity to represent 2.5 million processing elements (PEs) and interconnections. The combined number of PEs and interconnections for the initial ARCADE networks are currently in the range of two to four thousand. The memory capacity for representation of the network and storage of the training data on the ANZA-Plus board is 10 megabytes. The advertised processing speed for back-propagation training on the ANZA-Plus is 1.5 million interconnect updates per second. During run mode, the processing speed jumps to 6 million sustained interconnect updates per second. The host computer for this program is a Compaq 386/20 Portable system running under the DOS 3.31 operating system.

In addition to the software routines supplied by HNC for interfacing with the ANZA-Plus Neurocomputer, a wide variety of ANS software support tools have been created under this effort for the design, development, and validation of various networks. One of the primary tools is a Neural Network Training System which allows an MBPN network designer to create a network for training or to continue the training of an existing network. The user is prompted with various network definition parameters, and the network is instantiated accordingly on the coprocessor. The training program then loads the specified training data and begins processing of the network. During training, the user is provided with a full color display of the network structure, some of the training parameters, and a dynamic picture of how training is progressing. Another tool used during the development of neural networks is the Weights Analysis Utility. When a network has been trained and saved using the Neural Network Training System, the training parameters and the current weight values are stored in two disk files. These files can then be used to produce a readable profile of the network's structure, training conditions, and weights. The Weights Analysis Utility allows the user to quickly examine or compare the details of any saved network. It provides a complete breakdown of each weighted connection in each layer of the network.

NEURAL NETWORK REPRESENTATION OF THE ACM ENVIRONMENT

Representation Factors

A critical element in the design and development of a neural network is the selection of specific components of the problem domain to be represented in the form of input and output feature vectors. It is important that the problem be represented by input and output components which are relevant to the real-world association between those elements and for which an underlying mapping exists. The initial representation for the ARCADE system assumes a minimal set of tactical input data that might be required by a single combat pilot when making ACM decisions in a one versus one engagement. Similarly, the components of the output response were chosen to meet a minimal set of aircraft flight control commands to direct aircraft maneuvering. The time dependency of the tactical situation data is also represented by the input vector.

Initial training data for the ARCADE networks was taken from a series of engagements flown in the Simulator for Air-to-Air Combat (SAAC) during AFHRL's Performance Measurement Validation Study. All engagements were flown by active-duty Air Force pilots against the same adversary in F-16 versus F-16 situations, and the same weapons loadout and rules of engagement were used each time. This provided maximal consistency of the training data for use with the neural network.

Before obtaining specific SAAC data for network training, certain preselection criteria had to be met. In order to facilitate the learning of the network, the training data must be as consistent as possible, and ACM scenario conditions present within the data should be restricted to only those conditions which are represented by the input vector. For example, the network should not be trained to maneuver with and without weapons if the input representation provides no indication to the system as to weapons availability. The SAAC data format allows quick identification of aircraft types, weapon loadouts, mission odds, and rules of engagement to help provide this overall consistency. It is also useful to select ACM data at a consistent level of proficiency, presumably from consistently "good" performances, to the extent that this is possible.

Internal Network Structure

The structure of the neural network's input vector is based primarily on what the pilot (and presumably the expert system) needs to know about the prevailing tactical situation in order to arrive at a reasonable maneuver response. More specifically, the input vector represents the top-level, dynamic variables which define the tactical situation between the competing goals of the pilot and enemy aircraft. The term **top-level** refers to situational data that is directly perceived by the pilot from the environment which relates to the present situation. Relative positions, orientations, and altitudes of the two aircraft are examples of top-level data. An example of data that is not top-level would be whether the current geometry is defensive, offensive, or neutral for the pilot. This implicit **meta-level** knowledge will eventually be derived by the network as training progresses. **Dynamic** data are those parameters which change in value over the course of the ACM profile. Non-dynamic data may be excluded from the input vector because the system will simply learn to operate within the constant constraints dictated by these factors where they are present in the training data. For example, the aerodynamic limitations of the aircraft need not be spelled out specifically to the neural network via the input vector, yet will become an implicit part of the simulation's maneuver responses by virtue of their existence in the responses found in the training data. Of course, like the human pilot, such a neural network would likely have to be retrained to effectively fly a different aircraft with different aerodynamic limitations.

In current versions of the ARCADE neural network, the input vector contains parameters which define the current tactical geometry in terms of slant range, antenna train angle (ATA), target aspect angle (TAA), closing velocity (V_c), blue aircraft angle of attack (AOA), blue altitude, blue heading, altitude difference, blue and red airspeed, blue and red pitch orientation, blue and red roll and turn orientation, and blue G information. This means that the activation level of each processing element in the input layer will be specific for representing the current value of each of these components. Certainly, there are other dynamic parameters which may be included, but this set forms a baseline representation and seems to capture enough information about ownship status and the relative location of the enemy aircraft to provide a basis for reasonable and realistic ACM performance. The above input parameters represent what the pilot needs to know about the current situation in order to make his next maneuvering decision. The relationship of some of these parameters can be seen in Figure 4, which depicts the geometry of a typical ACM engagement.

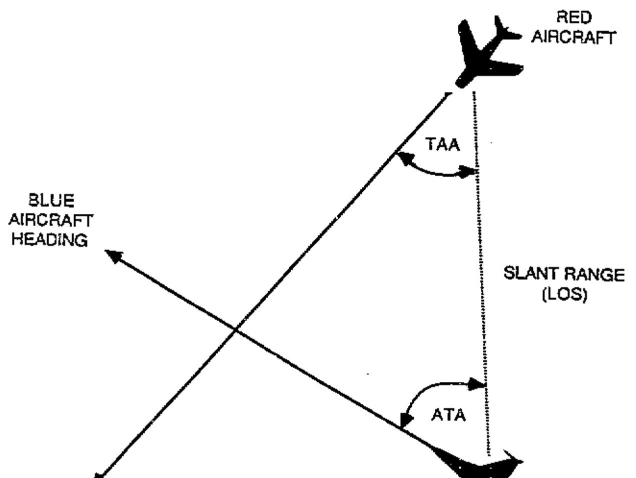

Figure 4. Geometry of a Typical Air Combat Intercept.

Another aspect of the input feature vector is the element of time. A pilot makes his maneuvering decisions based not only on a "snapshot" picture of how things look now, but also on how the situation has been evolving prior to this point. In the ARCADE model, the input feature vector contains multiple copies of the parameter set described above, each copy representing a different point in time over the last few seconds. The resulting input layer may, for example, consist of one set of input parameters for the present situation, another for the situation one second ago, and additional sets for two, five, and ten seconds in the past. Such an arrangement would result in a total of eighty-five input elements (five sets of seventeen) as depicted in Figure 5. The current implementation of the ARCADE network operates at two cycles per second, that is, each half-second, the system is provided with new tactical input data and produces an associated maneuver response.

On the other side of the ARCADE network is the output response layer which represents how the simulated pilot should maneuver his aircraft under the prevailing conditions. Not only must the output vector provide a sufficient representation of the maneuver response, it must also be capable of being translated into a set of parameters that can drive the operation of an aerodynamic model. The ARCADE system produces output parameters compatible with a flight command input set similar to that used by the Blue Max II aerodynamic model. Since certain assumptions are made by the flight path model, such as roll-coupling in turns, there is no requirement for the output vector to specify a command for each of the aircraft's degrees of freedom. The output processing elements of the network therefore represent only three flight path commands: a desired pitch angle, a desired heading, and a commanded velocity.

Unlike the input vector, the output PEs need not contain multiple representations over time. Each time the neural network is provided with a set of tactical parameters for the last few seconds, it will provide the aircraft flight model with the set of flight path commands to control the current piece of the desired maneuver. It is necessary, however, to use a time window on the ACM output data stream to determine the desired flight path commands for network training relative to the time now. Figure 6 shows how such a time window is used to read the input and output data parameters from a digital tape of actual ACM flight path data. Where the input values are based, in part, on the previous conditions, the output values are determined by looking ahead from the current maneuver state to the state as it will appear, say three seconds, later. The pitch, heading, and velocity on the tape at that future time are used as the current desired output values for the network.

In addition to the three flight command values, the size of the output time window is also sent to the aerodynamic model to define how quickly the flight condition variables should be changed. For example, on a given cycle, the aerodynamic model will receive a set of desired pitch, heading, and velocity values to be attained within a three second time slot. Commands beyond the aerodynamic limits of the aircraft would therefore be executed at the limits until the desired conditions are met or until the flight commands are changed. The overall architecture of the ARCADE system is shown in Figure 7.

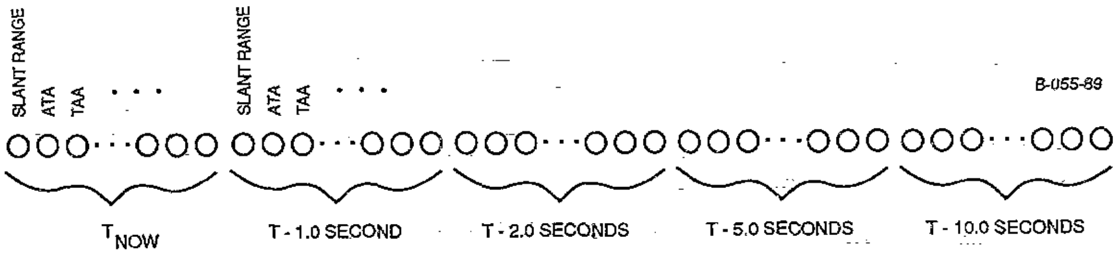

Figure 5. Neural Network Input Layer for the ARCADE System.

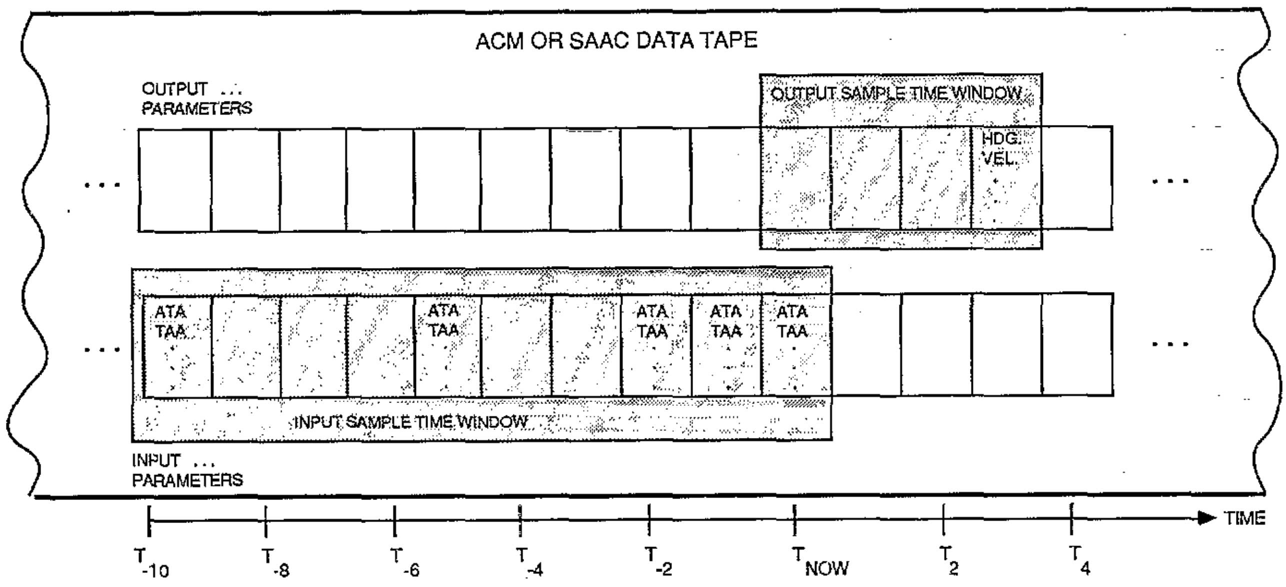

Figure 6. Data Windows for Reading Information from the ACM Data Tapes.

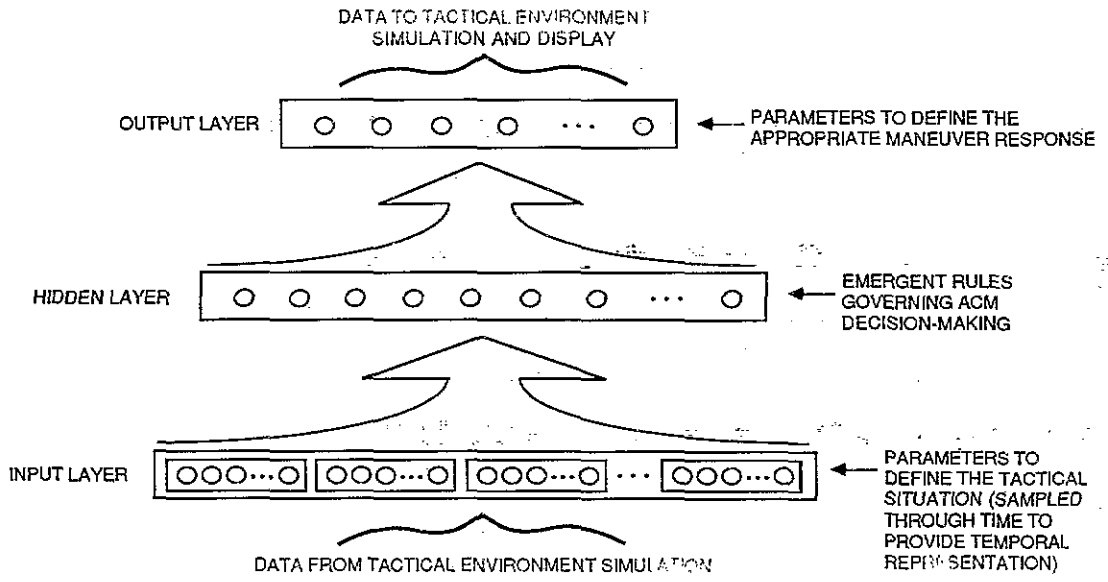
DATA TO TACTICAL ENVIRONMENT SIMULATION AND DISPLAY
OUTPUT LAYER
PARAMETERS TO DEFINE THE APPROPRIATE MANEUVER RESPONSE
HIDDEN LAYER
EMERGENT RULES GOVERNING ACM DECISION-MAKING
INPUT LAYER
PARAMETERS TO DEFINE THE TACTICAL SITUATION (SAMPLED THROUGH TIME TO PROVIDE TEMPORAL REPRESENTATION)
DATA FROM TACTICAL ENVIRONMENT SIMULATION

Figure 7. Internal Architecture of the ACM Expert System Neural Network.

The Neural Network Training Process

The ability of the neural network to successfully learn and recall the proper mapping which associates tactical conditions to maneuver responses is based entirely on the structure and connection weights between the various layers of the system. The input and output layers provide the interface to the outside world, and through them, the network is trained to accomplish the desired mapping. However, it is the internal structure, the connections to, from, and between the hidden layers of the ARCADE neural network where the system learns how to properly maneuver an aircraft in response to certain tactical situations. These internal connections and weights embody the expertise of the simulated pilot, and do so by learning associations between what is seen on the input layer and what is desired on the output layer. While the representations of the input and output vectors are made very explicit prior to training the network, it is difficult to predict with certainty what the processing elements and interconnections of the middle layers will come to represent. After training the network, however, there will be certain **emergent** properties of the network's internal conformation which will represent the abstract nuances of successful ACM performance. Meta-level knowledge structures like maintaining speed, building energy, and gaining lateral separation, become embodied in the activation levels and connection weights of the system without being explicitly programmed by the system designer.

This is one of the fundamental benefits of the neural network approach. Determining how a human pilot combines the information coming into his brain into a useful response is very difficult to spell out as a set of rules. The amount of knowledge and the types of combinations of that knowledge are vast, and often, the pilot is not capable of furnishing a detailed explanation of precisely why a certain action was taken. But, the neural network, with the proper representation and training, will self-organize to arrive at its own understanding of how maneuver responses are formed. The weights and connections of the hidden layers will eventually store the maneuver strategies, which are learned automatically through example, as an implicit set of internal rules.

By examining the states of the PEs in the hidden layers and their associated connection weights, one might hope to gain some insight into how ACM expertise is formed. Unfortunately, it is almost as difficult to determine the logical rule structure by examining the hidden layers of a neural network as it is to elicit expertise from a human expert. In the hidden layers, some tendencies and generalizations may be evident from the overall activation levels in response to certain input data, but there is certainly no explicit "if-then" structure to be found. This poses some problems in terms of having the system provide a rationale for each of its decisions. One possible approach to this problem is to create a pattern recognition system which is capable of classifying the sequences of **micro-maneuvers** produced by the ARCADE network and determining which one of a specific set of basic fighter maneuvers is being developed by the model. If a BFM pattern can be determined from the maneuver responses, some clue as to which tactics or strategies are being employed may be gained.

Supervised neural network training, as is used in the back-propagation algorithm, consists of repeatedly providing the system with input data, observing the response at the output layer, and correcting that output to the desired response if it is in error. The results of this correction step are propagated back through the layers of PEs, making the appropriate weight changes where necessary. Each weight adjustment moves the overall state of the network in the direction of a global solution to the input/output mapping. Such a procedure obviously requires a known set of input/output correlation vectors which can be used to train the network. For the ARCADE system, selected data of pilot ACM performance in the SAAC is used as the training platform. The initial ARCADE training set consists of something on the order of 5000 input/output vector pairs, or associations.

Using this body of input/output feature vector associations, the neural network is presented with a set of input values which represent the tactical situation over the last few seconds. From this data, the system uses equations (1), (2), and (3) to arrive at an output set of specific flight parameters to control the evolution of the appropriate maneuver. This **calculated** response is then compared to the correct, or **target** response from the SAAC data, and the difference between the two, the error, is fed back into the network to correct the association between input and output by adjusting the weights in the appropriate direction. In effect, this teaches the network which output to produce when it sees that input data again. As the process continues in this fashion, the neural network eventually provides the correct response for each input situation upon which it was trained. At this point, training is considered complete for this network, and it may be used in a **run** mode where no back-propagation of errors is required.

Back-propagation training is a multi-step process, each step providing adjustments to a single layer of interconnections. Assuming a three-layer system as shown in Figure 3, the first back-propagation of errors, or deltas, produced at the output layer of PEs, adjusts the weights of the connections between the hidden layer and the output layer. In the second step of the three-layer back-propagation training process, weight adjustments are made to the connections between the hidden layer of PEs and the elements of the input layer.

The back-propagation algorithm is a **gradient-descent** heuristic, which means that the weight changes will minimize the squares of the differences (error) between the actual and the target output values. This error function is known as **Mean Squared Error (MSE)**. The back-propagation process attempts to move across the multi-dimensional weight space so as to continually reduce this error function. The weight space may be visualized as a landscape with various wells, valleys, hills, and ridges. Somewhere in this landscape is a lowest point or **global minimum**, which represents the optimal performance of the network. The BPN process adjusts the weights so that the weight surface is traversed in the steepest fashion. However, it does not guarantee that the global minimum will be found; the process may get trapped in a valley which represents only a **local minimum**. Finding the global minimum is the major goal of network processing, and the techniques for doings so (and avoiding local minima) are a primary focus of ANS research. One cycle of the network's feed-forward operation, including weight updates if necessary, is referred to as a single **iteration**. The entire training process may take thousands of such iterations before the system settles into a minimum well, and the proper mapping is learned.

Since the gradient descent process attempts to minimize Mean Squared Error, the most common method for measuring the performance of a back-propagation network is to calculate the Mean Squared Error over the entire set of training data. MSE is only one of the measures of effectiveness used to evaluate the performance of an ARCADE network. Since the eventual performance of the network is a generalization of the training data, the only way to really validate the network is to observe its performance in a simulation of a dynamic ACM engagement, and evaluate that performance relative to actual human behavior under similar conditions.

To produce an effective generalization in a neural network, the training data must be selected so as to adequately span the entire **solution space**. Exactly what the solution space is can be difficult to specify in complex, multi-dimensional problems like the ACM representation, but, in general, the solution space consists of all the possible associations of input with output. At a practical level, only a small subset of all the possible associations will ever occur in the real world, so the solution space is substantially smaller than it might be, though it is no less difficult to specify. The important point regarding the solution space is that the training data be selected such that it represents a relatively uniform distribution over the entire space. Unless the solution space can be specified in detail, it is easiest to choose data in a random fashion from real-world examples and assume that this selection will adequately span the possible conditions to which the network will be required to respond. The network's performance will be improved, however, if certain guiding principles are used to produce a more intelligently constructed database. When the training data is selected appropriately, the network will form a mapping which allows it to generalize solutions from the relatively small set of training examples. This means that the neural network will be capable of responding correctly to novel situations that were not seemingly reflected in the training data. The ability of the neural network to generalize successfully from the training data is crucial because the amount of training data, relative to the entire solution space, is very sparse.

In addition to the creation of an optimal network training file, it is also necessary to determine the most effective structure for the hidden layer(s) and to fine-tune the learning equations. There are certain useful heuristics in the development of the neural network's internal structure, but discovering and applying those heuristics is an acquired skill in these early days of ANS research. The number of PEs used in the hidden layer(s) is partially a speed versus accuracy question in that more PEs means more connections and therefore, longer training cycles. At the same time, a sufficient number of PEs must be present to successfully accomplish the desired mapping. Depending on the specific association that is being learned, the number of PEs and connections directly determine the network's capacity. The exact capacity of the network, or number of associations learned, is difficult to measure since the network is expected to generalize over many more associations than it is actually trained on. The question of capacity is more correctly stated as the performance of a network on a general set of test data as measured by the minimization of Mean Squared Error (MSE). In general, the number of PEs is gradually increased from one until the optimal minimization of MSE is found. Another factor, however, must be taken into consideration. If too many internal PEs are present, the system may become locked in on the training data associations, but be completely unable to generalize to the broader scope of the problem. When this happens, the network will exhibit very good performance on the training set, but a broader test set will show poor correlation between input and output.

After the training file and the network structure have been defined, the next step is to find the optimal values for each parameter in the network initialization and training process. For example, when the network is created, the weights must be assigned some initial value before training begins. Without any previous data to guide this initialization, the best approach is to simply assign random values to the weights. However, both the seed value for the random number generator and the allowable range for the random values themselves may affect the eventual performance of the trained network. In effect, they determine the starting point for the gradient descent process on the weight surface. The only way to determine the best random seed value is by trial and error, but the optimal range for the initial weight values can be determined in a more systematic fashion. Similarly, experimental procedures are used to determine optimal values for the network's learning rate and other factors of the weight update equations. Each value is chosen so as to optimize (in a local fashion) the reduction of MSE for a given training set.

The optimal length of training, that is, the total number of training iterations, is best determined by observing MSE as training progresses and by noting the absolute differences in the desired and computed output values. In general, it is desirable to continue training until the output deltas in the training set become very small, with absolute errors below one percent; however, the actual performance is best tested in a run mode with a broader set of data. Every problem will require a different amount of training to achieve a level of optimal performance. Care must also be taken not to over-train the network and therefore degrade generalization capability. The gradient descent algorithm proceeds asymptotically, so the vast majority of learning occurs at the beginning of training, and the final stages comprise a slower, fine-tuning process. This phenomenon is shown pictorially in Figure 8. If a given network is going to converge to a solution, performance should be tested when the MSE plot begins to flatten out.

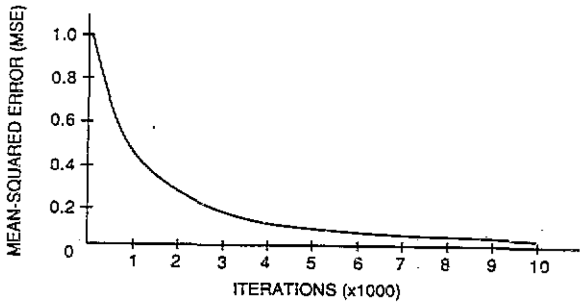

Figure 8. The Asymptotic Reduction of MSE as a Function of Network Training Iterations.

CURRENT STATUS OF THE ARCADE SYSTEM

Prior to the development of the ARCADE system, a scaled-down, prototype neural network environment was created for proof-of-principle testing purposes. The goal of this system was to provide a framework upon which to develop the ARCADE system, while allowing system concepts to be investigated in a workable environment with a controlled database. Instead of using actual ACM performance data, the prototype system was trained to simulate the behavior of a simple Lead Pursuit and Intercept algorithm. To accomplish this, an algorithm was designed which would successfully fly a lead pursuit profile followed by an intercept maneuver for any initial relative geometry of two aircraft. Intercept is achieved by the pursuit aircraft when the target's position and velocity are matched within a certain tolerance for a certain time period. The algorithm works by generating lateral and axial acceleration commands based on the current relative geometry of the two aircraft. For each set of commands, the flight path is updated and new geometry calculated at an update rate of one per second. A body of neural network training data was then collected by sampling the performance of the Lead Pursuit/Intercept (LPI) algorithm over a wide range of starting geometries. The intent was to use this body of knowledge to train a neural network to arrive at a generalized solution to the LPI problem.

The structure of the LPI neural network is shown in Figure 9. The tactical situation is represented on the input layer in terms of angles, ranges, and velocities. The corresponding output is represented on the input layer as the lateral and axial acceleration commands. A single layer of fifteen hidden PEs was used to complete the most successful network structure. The training file was constructed of approximately 1500 associations of tactical situations and maneuver responses generated by the algorithm. Training was carried out for approximately 250,000 iterations, or about 160 passes through the training set.

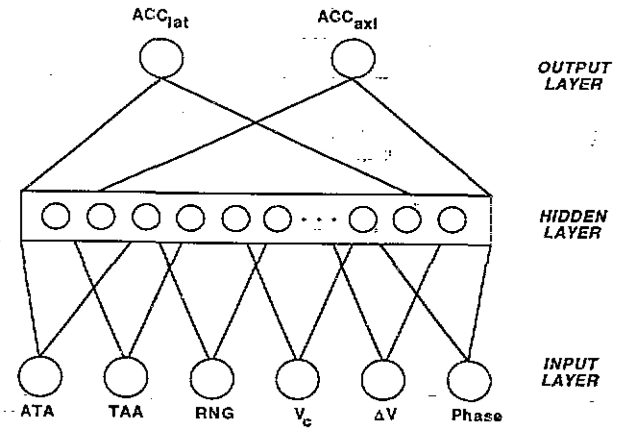

Figure 9. Network Structure of the Lead Pursuit/Intercept Demonstration System.

To validate the performance of the LPI network, it was incorporated into a software package which includes a flight path update routine and a flight profile plotting feature. This system allows the network to provide maneuver responses in a dynamic flight situation, and the results are plotted simultaneously with the responses of the algorithm under the same conditions. Results of this demonstration program using an optimally trained LPI network exhibit a behavior remarkably similar to that of the algorithm. In most cases, the network's pursuit profile develops very near to or directly on top of the algorithm's profile. Furthermore, the network is capable of attaining the intercept conditions in much the same fashion as the algorithm. By maneuvering the target aircraft, via input from a joystick control, a variety of unusual geometries and novel situations can be created. Because the LPI neural network was trained to learn a generalized solution, it is capable of generating successful responses, sometimes uniquely so, to even these novel input conditions.

Building on the successes of the LPI network, work is in progress to convert that software into a form that will support the ARCADE research. Prototype neural networks have been developed using actual SAAC ACM data, and these networks indicate that the appropriate mapping can be successfully learned. To fully validate the ARCADE network's performance, however, it must be incorporated into a dynamic flight path generator and display system and exercised as were the LPI networks. It is anticipated that the results from this system will be available in the near future.

CONCLUSION

The research goals of the ARCADE project were to ascertain the applicability of ANS technology to expert systems tasks in general and to support the simulation of ACM decision-making in the training environment. In the experiments conducted thus far under this program, neural networks have aptly displayed their unique capabilities to overcome some of the more difficult aspects of knowledge engineering. ANS approaches have been shown to be capable of producing robust, generalized solutions even under novel circumstances. By capturing and simulating the expertise of human pilots in a neural network, students may be provided with expert training devices which come very close to the look and feel of real air-to-air combat. Other research confirms the successes of this program [3, 5, 8], and it is expected that ANS technology will continue to provide new solutions to the simulation of human performance for training purposes.

REFERENCES

[1] Burgin, G. H., Fogel, L. J., and Phelps, J. P., An Adaptive Maneuvering Logic Computer Program for the Simulation of One-on-One Air-to-Air Combat, Vol. 1, Decision Science Inc. Report No. NASA CR-2582, 1975.

[2] Gallant, S. I., "Connectionist Expert Systems", Communications of the ACM, Vol. 31, 152-169, (1988).

[3] Mozer, M. C., "RAMBOT: A Connectionist Expert System That Learns By Example", Proceedings from the First IEEE International Conference on Neural Networks, Vol. 2, (1987).

[4] Partridge, M., "Advanced Tactical Simulation: Using an Expert System to Simulate an Air-to-Air Combat Environment", Proceedings from the 11th Interservice/Industry Training Systems Conference, (1989).

[5] Rodin, E. Y. and Amin, S. M., "Prediction and Identification of Tactical Air Combat Maneuvers: Neural Network Implementation of a Qualitative Approach", Proceedings from the 5th Aerospace Applications of Artificial Intelligence Conference, (1989).

[6] Rumelhart, D. E. and McClelland, J. J. (Eds), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vols 1 and 2, MIT Press, Cambridge, MA, 1986.

[7] Shaw, R. L., Fighter Combat: Tactics and Maneuvering, U.S. Naval Institute Press, Annapolis, MD, 1985.

[8] Tesauro, G. and Sejnowski, T. J., "A Parallel Network That Learns to Play Backgammon", Artificial Intelligence, Vol. 39, 357-390, (1989).

ABOUT THE AUTHOR

Mr. Crowe is a Program Manager at Ball Systems Engineering Division's Advanced Concepts Department and is currently serving as principal investigator for the Air Combat Maneuvering Expert System program. He received a Bachelor's degree in Cognitive Science from the University of California at San Diego and has over ten years of experience in systems simulation, human factors analysis, and intelligent systems modeling.