NEURAL NETWORK BASED SEMI-AUTOMATED FORCES: EXPERIMENTAL RESULTS

Amy Henninger, Avelino Gonzalez, and Michael Georgiopoulos
University of Central Florida
Orlando, FL.

ABSTRACT

In concert with the I/ITSEC '99 theme, *Synthetic Solutions for the 21st Century*, this paper explores a variety of ways in which neural networks, synthetic models of human cognition, can be used to improve performance in a distributed simulation exercise. Specifically, it examines the use of neural networks in semi-automated forces (SAF) systems as a means of reducing network bandwidth and processing requirements. To address the first performance measure, reduction of network bandwidth requirements, this research investigates the use of neural networks in lieu of the current, Newtonian, DIS dead-reckoning models. While this concept is demonstrated in a SAF system, it is extendible to other types of players (e.g., manned modules or live/embedded systems) in a distributed simulation. To address the second performance measure, reduction of processing requirements, this research considers the use of neural networks in lieu of SAF behavioral models. This concept does not extend beyond SAF systems.

This paper motivates the need for this research by reviewing how SAF systems work and why they are limited by bandwidth and processor constraints. It also introduces the theory behind the neural networks' architecture and training algorithms as well as the specifics of how the networks were developed for this investigation. Lastly, it illustrates how the networks were integrated with SAF software, defines the networks' performance measures, presents the results of the scenarios considered in this investigation, and offers directions for future work.

ABOUT THE AUTHORS

Amy Henninger is a doctoral candidate in computer engineering at the University of Central Florida, a research associate at the UCF Intelligent Systems Laboratory, and a recipient of the 1998 I/ITSEC Scholarship. She has worked in the simulation community as an engineer for McDonnell Douglas Aerospace, a staff scientist for Science Applications International Corporation, a research assistant at the Institute for Simulation and Training, and a Research Fellow for the Army Research Institute at U.S. Army STRICOM. Previously, Amy has earned B.S. degrees in Psychology, Industrial Engineering, and Mathematics from Southern Illinois University, an M.S. in Engineering Management from Florida Institute of Technology, and an M.S. in Computer Engineering from UCF. Her research interests lie in the areas of artificial intelligence, mathematical modeling, and simulation.

Avelino Gonzalez received his bachelor's and master's degrees in Electrical Engineering from the University of Miami, in 1973 and 1974, respectively. He obtained his Ph.D. degree from the University of Pittsburgh in 1979, also in Electrical Engineering. He is currently a professor in the Electrical and Computer Engineering Department at UCF, specializing in human behavior representation.

Michael Georgiopoulos is an Associate Professor at the Department of Electrical and Computer Engineering of the UCF. His research interests lie in the areas of neural networks, fuzzy logic and genetic algorithms and the applications of these technologies in cognitive modeling, signal processing and electromagnetics. He has published over a hundred papers in scientific journals and conferences.

NEURAL NETWORK BASED SEMI-AUTOMATED FORCES: EXPERIMENTAL RESULTS

Amy Henninger, Avelino Gonzalez, and Michael Georgiopoulos
University of Central Florida
Orlando, Florida

INTRODUCTION

The combination of computer simulation and networking technologies has provided military forces with an effective means of training through the use of Distributed Interactive Simulation (DIS). DIS is an architecture for building large-scale simulation models from a set of independent simulator nodes that represent entities in the simulation (Smith and Petty, 1992). These simulator nodes individually simulate the activities of one or more entities in the simulation and report their attributes and actions of interest to other simulator nodes via the network. DIS nodes simulating combat vehicles, such as M1 Abrams tanks, are crewed by soldiers being trained. The trainees operate the controls of the simulators as they would in the actual vehicles, and the simulators implement actions in the simulated battlefield. Since, in a synthetic battlefield, the trainees need opposing forces against which to train, a type of DIS node known as a Computer Generated Force (CGF) system was developed.

CGFs are computer-controlled behavioral models of combatants used to serve as opponents against whom trainees can fight or as friendly forces with whom the trainees can fight. The behavior of the CGF may be generated by a human operator assisted by software, in which case the class of CGF is referred to as a semi-automated force (SAF), or they may be generated completely by software, in which case they are known as autonomous forces (AFs). At a minimum, the behavior generated by CGFs should be feasible and doctrinally correct. For example, CGF behaviors should be able to emulate the use of formations in orders, identify and occupy a variety of tactical positions (e.g., fighting positions, hull down positions, turret down positions, etc), and plan reasonable routes.

A small number of researchers have experimented with using neural networks to model battlefield behavior for Semi-Automated Forces (SAF) systems used in distributed simulations (e.g., Crowe, 1990; Jaszlics, 1993). Typically, the rationale for these applications is related to increasing the behavioral model's fidelity or facilitating the knowledge acquisition process, and the measures of performance are a subjective interpretation of the behavior's "reasonableness". While these studies suggest that the use of neural networks may indeed provide adequate or even improved SAF behavior representation, they do not recognize that the creative use of neural networks can help solve practical DIS problems related to network bandwidth and processor time utilization.

In addition to maintaining the requirement of providing reasonable behavior, this research investigates the notion of using neural networks as a means of reducing network bandwidth and processing requirements in SAF systems. The following two sub-sections, Network Bandwidth and Processing Requirements, expand on these concepts, respectively.

Network Bandwidth

The first performance measure of this investigation is the reduction of network bandwidth requirements. This section reviews why network bandwidth is a limiting factor in a distributed simulation and how current DIS methods address this problem.

DIS entities can exist simultaneously and interact meaningfully in the virtual world by communicating via a common network protocol. This distributed real-time interaction, however, is limited by the communications requirements imposed by the need to convey large amounts of data between the respective players in the simulation. One method to reduce the amount of data transfer is the use of DIS dead-reckoning models. The term "dead-reckoning", borrowed from navigation, describes the process by which the position of a ship can be estimated from an earlier known position and the elapsed time and motion since that known position. DIS dead-reckoning models build on this concept by applying Newtonian equations of

motion to approximate the state information of a simulated entity. For example, one DIS dead-reckoning model used to approximate an entity's position is given by:

$$p = p_0 + (v_0 * \Delta t) + \frac{a_0 * (\Delta t)^2}{2}$$
$$v = v_0 + a_0(\Delta t) \qquad (1)$$

where p = current position
p_0 = initial position
v = current velocity
v_0 = initial velocity
a_0 = initial acceleration
Δt = elapsed time

In DIS dead-reckoning, each simulator maintains simple, low fidelity dead-reckoning models such as equation (1) of its own state and of the state of all other vehicles with which it may interact. Additionally, each simulator is responsible for maintaining a high-fidelity model of its own vehicles' states. At periodic intervals, these models are compared and, as illustrated in Figure 1 (Lin and Ng, 1993), if the difference between the high fidelity model and the dead-reckoning model exceeds some error threshold, the local simulator will send out an update of the vehicle's true state to the other simulators. This update takes the form of an Entity State (ES) Protocol Data Unit (PDU) and contains information such as that found in equation (1) (e.g., location, velocity, time of last update, etc.). Upon the receipt of an ESPDU, the dead-reckoning models will adjust to reflect the updated data and then continue to predict the near-term position and orientation of the entity from these data.

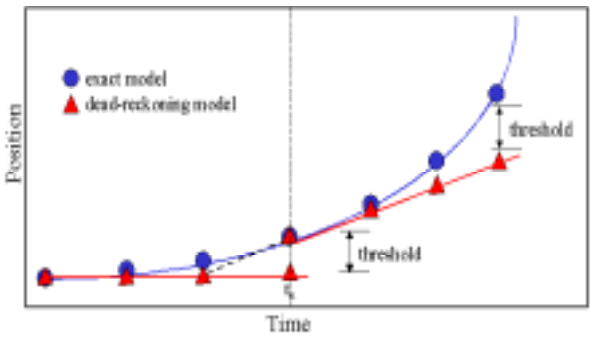


Figure 1. DIS Dead-Reckoning Process

Because the number of ESPDUs required in an exercise depends on the predictive utility of the dead-reckoning model, improving this model's fidelity will result in the reduction of communications updates. This, in turn, will enable more entities to exist on the battlefield for the same amount of network bandwidth. The research presented in this paper attempts to improve the predictive utility of the dead-reckoning model by using neural networks instead of Newtonian equations.

Processing Requirements

The second performance measure addressed in this investigation is the reduction of processing requirements or computing power. This section reviews how current SAF systems are organized and why computing power is and will continue to be a constrained resource in SAF systems.

Traditionally, CGF behaviors have been implemented in procedural languages (e.g., Ada or C) and organized around state transition constructs such as finite state machines (FSMs) or finite automata. For example, a CGF behavior such as "Occupy a Battle Position" might be constructed around states that: "Travel", "Calculate Position", and "Move Into Position". Alternatively, a CGF behavior such as "Conduct a Tactical Road March" might be constructed around states that: "Get to Route", "Follow Route", and "Halt". Any one of these states, in turn, could be 1) an embedded FSM, 2) a simple function call representing some low-level primitive action, or 3) any combination of the two. This type of organization is useful for structuring and communicating the intricacies of the behavior and for providing the CGF some semblance of multi-tasking.

Although representing the behavior of CGF entities is already a complex task, a recent study by the National Research Council (Pew and Mavor, 1998) calls for significant improvements in human behavior representation for military simulations. Various researchers have suggested (e.g., Franceschini et al, 1999 and Reddy, 1995) that these improved behavioral models will be even more complex and require even more computing power. Consequently, improving the algorithmic efficiency of the CGF becomes increasingly important because it will, in turn, reduce the required amount of computing power. As a result, increasingly complex behavioral models may be executed for the same amount of computing power. The research presented in this paper attempts to improve the processing

requirements of the SAF behavioral model by basing it on neural networks instead of procedural code.

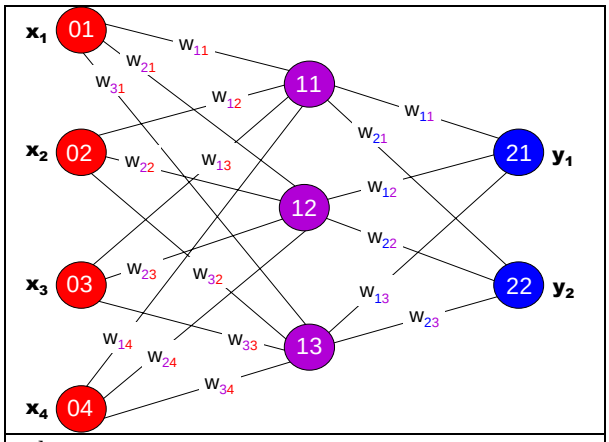
METHODOLOGY

This study used ModSAF, a CGF system for training and research, to focus on the near-term movement behavior of a SAF. The near-term movement behavior was selected because it is computationally challenging and highly observable (Smith, 1994; Schricker et al, 1998; Pew and Mavor, 1999). It also provides a direct correspondence to ESPDUs resulting from errors in entity position. Additionally, since moving in the battlefield is a highly complex behavior depending on many factors, the problem was scoped to specifically consider how a single entity (i.e., a ModSAF M1A2) performs a "Road March". This is accomplished by estimating the changes in an M1A2 entity's speed and orientation given its previous state and the previous state of the simulated world.

Neural Networks - Theory

A variety of researchers have worked in modeling human driving skills such as acceleration, steering, and vehicle following with neural networks (e.g., Pomerlau et al, 1994; Pentland and Liu, 1995; Modjtahedzadeh and Hess, 1991; Fix and Armstrong, 1990; and Nechyba and Xu, 1997). A neural network is a collection of simple processors or nodes interconnected with each other that learn from examples and store the acquired knowledge in their interconnections, referred to as weights. Neural networks can solve a variety of problems related to non-linear regression and discriminant analysis, data reduction, and non-linear dynamic systems. One of the practical characteristics of neural networks is that they lend themselves to parallel distributed processing using simple processing units rather than a complex CPU. This makes their execution very fast. For example, given four processors, the network illustrated in Figure 2, a 4-3-2 feed-forward network, could execute in significantly less time than it would take to execute on a single processor.

The multi-layer feed-forward network is one of the more typical network designs used in neural network applications. The example in Figure 2 is a 3-layer feed-forward network with four nodes in the first layer representing each dimension of the input vector, two nodes in the last layer representing each dimension of the output vector, and a hidden layer consisting of three nodes. This network attempts to develop a matching function between the input and output vectors by using some training algorithm. One of the more popular training algorithms is a method known as back-propagation. This method is based on finding the outputs at the last layer of the network, calculating the errors between the actual and the predicted outputs, and then adjusting the network weights to minimize the error. Weight changes are implemented in a backward fashion starting from the weights converging to the output layer and proceeding backwards to the weights that converge to the hidden layer closest to the output layer. These computations are repeated such that the error is propagated back until the weights converging to the hidden layer closest to the input layer are reached.



where

- x1, x2, x3 and x4 are inputs (k = 1 to 4)
- y1 and y2 are outputs (i = 1 to 2)
- there are three hidden nodes (j = 1 to 3)
- there are three layers (L = 0,1,2)
- node "12" is node 2 in layer 1
- weight "12" connects node 1 from layer L+1 to node 2 of layer L

Figure 2. 4-3-2 Feed-Forward Architecture

In short, back-propagation involves a two step process. The first step, the forward pass, propagates the effects of the inputs forward through the network to reach the output layer. This step is governed by three forms of equations. First, in equation 2.1, the total weighted input to the j^{th} node for pattern p is given by:

$$net_j^{(1)}(p) = \sum_{k=1}^{K=4} x_k^{(0)}(p) w_{jk}^{(1)} \quad \text{for j} = 1 \text{ to } 3 \qquad (2.1)$$

Next, the output of the jth node, $y_j^{(1)}(p)$, is given by:

$$y_j^{(1)}(p) = g(net_j^{(1)}(p)) = \frac{1}{1+e^{-net_j(p)}} \qquad (3.1)$$

where $g()$ is the frequently used sigmoid function.

The net input to the *i*th node for pattern *p* is similar to equation (2.1) and is given by:

$$net_i^{(2)} = \sum_{j=1}^{J=3} y_j^{(1)}(p) w_{ij}^{(2)} \quad \text{for i} = 1 \text{ to } 2 \qquad (2.2)$$

and the output of the *i*th node for pattern *p*, $y_i(p)$, is given by:

$$y_i^{(2)}(p) = g(net_i^{(2)}(p)) = \frac{1}{1+e^{-net_i(p)}} \qquad (3.2)$$

where $g()$ is shown as the sigmoid function.

Lastly, the error function associated with the *p*th input/desired output ($d_i(p)$) pair, is given by:

$$E^p(w) = \frac{1}{2}\sum_{i=1}^{I=2}[d_i^{(2)}(p) - y_i^{(2)}(p)]^2 \qquad (4)$$

Once the error is computed, the weights are adjusted such that E^p is minimized. This occurs in the second pass, the backward pass, by computing the negative gradient of the error function and taking the partial derivatives of this function with respect to the weights (equations 5.1 and 5.2). This allows errors at the output layer to be propagated backward toward the input layer in proportion to the change in activity at the previous layer.

$$\Delta w_{ij}^{(2)} = -\eta \left(\frac{\partial E}{\partial w_{ij}^{(2)}}\right) \qquad (5.1)$$

$$\Delta w_{jk}^{(1)} = -\eta \left(\frac{\partial E}{\partial w_{jk}^{(1)}}\right) \qquad (5.2)$$

where η is a user defined parameter.

By applying the chain rule of derivation (see Rummelhart et al, 1986 for complete derivation), equations 5.1 and 5.2 reduce to:

$$\Delta w_{ij}^{(2)} = \eta \delta_i^{(2)}(p) y_j^{(1)}(p) \quad i = 1,2 \quad j = 1,2,3 \qquad (6.1)$$

$$\text{where } \delta_i^{(2)}(p) = g'(net_i^{(2)}(p))[d_i^{(2)}(p) - y_i^{(2)}(p)]$$

$$\Delta w_{jk}^{(1)} = \eta \delta_j^{(1)}(p) x_k(p) \quad j = 1,2,3 \quad k = 1,2,3,4 \qquad (6.2)$$

$$\text{where } \delta_j^{(1)}(p) = g'(net_j^{(1)}(p)) \sum_{i=1}^{I=2} w_{ij}^{(2)} \delta_i^{(2)}(p)$$

For the example in Figure 2, these equations reduce to:

$$\Delta w_{ij}^{(2)} = \eta\,(y_i(p))(1-y_i(p))(d_i(p) - y_i(p)) \cdot (y_j^{(1)}(p)) \qquad (7.1)$$

$$\Delta w_{jk}^{(1)} = \eta\,(y_j(p))(1-y_j(p)) \sum_{i=1}^{I=2} [(y_i(p)) \cdot (1-y_i(p))(d_i(p) - y_i(p)) w_{ij}^{(2)}] \cdot x_k(p) \qquad (7.2)$$

Each of these weight adjustments directs the network towards a solution to the input/output mapping. That is, these weights are training the network to produce a certain output given a set of inputs. This is one of the fundamental benefits of the neural network approach. With the proper training and representation, the network will self-organize to arrive at a mapping of how the responses are formed and there is no need to acquire and represent an expert's knowledge in terms of rule sets.

Neural Networks – Application

The first step in applying neural networks to a problem involves choosing a training algorithm and network architecture. Next, one must select the variables and their representation scheme in the input/output feature vectors. Lastly, the methods used for data sampling and training must be established. The following three sections review how these issues were addressed in this investigation.

Model Architecture

For this application, a feed-forward architecture with back-propagation training, as introduced in the previous sub-section, Neural Networks – Theory, was used in each of four networks. Of the four neural networks, there were two estimating the M1A2 entity's speed and two estimating the M1A2 entity's orientation. Further, each of these pairs of networks could be distinguished by whether the M1A2 entity was travelling a straight segment of the road or was entering a turn. This combination yielded four networks identified as straight-speed (SS), straight-heading (SH), turn-speed (TS), and turn-heading (TH).

The configuration of each of the four networks may be seen in Table 1. Each network used a sigmoid function at the hidden nodes and a linear transformation at the output nodes.

NN	Arch.	Predictors	Resp.
SS	8-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1}$	ΔS_t
SH	7-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	$\Delta\theta_t$
TS	8-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1}$	ΔS_t
TH	7-20-5-1	$Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}$	$\Delta\theta_t$

Table 1. Architecture by Neural Network Type

Input/Output Representations

The inputs shown in Table 1 were normalized according to equations 8.3 – 8.9 below. Fundamentally, the inputs for each of the networks were a function of the M1A2's state at the last simulation clock and how this state related to the road characteristics and March Order parameters.

$$S_t = S_{t-1} + \Delta S_t$$
$$\text{where } \Delta S_t = f(Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}, Hz_{t-1}) \quad (8.1)$$

$$\theta_t = \theta_{t-1} + \Delta\theta_t$$
$$\text{where } \Delta\theta_t = f(Ra_{t-1}, Rb_{t-1}, Rc_{t-1}, Rp_{t-1}, Rs_{t-1}, HRab_{t-1}, HRbc_{t-1}) \quad (8.2)$$

where

$$Ra_t = S_t / (Da_t + M) \quad (8.3)$$
$$Rb_t = S_t / (Db_t + M) \quad (8.4)$$
$$Rc_t = S_t / (Dc_t + M) \quad (8.5)$$
$$Rp_t = S_t P_t / M \quad (8.6)$$
$$Rs_t = S_t / M \quad (8.7)$$
$$HRab_t = Hab_r \times Hxy_t \quad (8.8)$$
$$HRbc_t = Hbc_r \times Hxy_t \quad (8.9)$$
$$S_t = entity\ speed\ at\ t \quad (8.10)$$
$$Da_t = distance\ to\ previous\ waypoint \quad (8.11)$$
$$Db_t = distance\ to\ current\ waypoint \quad (8.12)$$
$$Dc_t = distance\ to\ next\ waypoint \quad (8.13)$$
$$M = march\ order\ speed \quad (8.14)$$
$$P_t = perpindicular\ distance\ to\ road \quad (8.15)$$
$$Hab_t = direction\ of\ road\ segment\ ab \quad (8.16)$$
$$Hbc_t = direction\ of\ road\ segment\ bc \quad (8.17)$$
$$Hxy_t = entity\ orientation \quad (8.18)$$

Training Procedure

To ensure effective generalization of the neural network, the training examples must be selected such that they provide adequate coverage of the decision space. For this investigation, data for training and validation were gathered over the 45 segment route shown in Figure 3. This route is approximately 7 kilometers long and took the tank about 15 minutes of simulation time to travel. From this period, a total of 13760 examples were generated at a rate of 15 HZ. Of these, 620 examples were used for training the straight-heading network, 319 examples were used for training the turn-heading network, 2104 were used for training the turn speed network, and 4325 were used for training the straight-speed network. An equal number of examples were used for validating the results of each network. The training rate, η, was selected as 0.01 and the initial momentum parameter, α, was .09. The momentum parameter was periodically adjusted to speed the rate of descent along the error surface. Finally, training was performed on a four-processor, 500 Mhz Xeon running RedHat Linux 6.0.

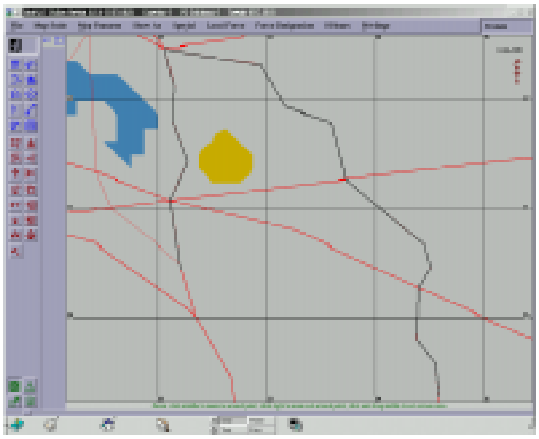


Figure 3. Route Used for Neural Network Training

The training and validation results for each of the four networks may be seen in Table 2.

		Delta Speed ΔS Error(m/s)	Delta Heading $\Delta\theta$ Error(rads)
Straight	**Training**	0.012475±0.03646	0.000150±0.0022
	Validation	0.012508±0.03573	0.000153±0.0024
Turning	**Training**	0.053409±0.08705	0.002010±0.0174
	Validation	0.053997±0.08575	0.002704±0.0175

Table 2. Training and Validation Mean Square Error and Standard Deviation

EXPERIMENTS

Two experiments were performed using data generated by the neural networks. The first experiment used the neural networks in place of the Newtonian dead-reckoning models and the second was in place of the SAF behavioral near-term movement model.

Experiment 1

In the context of dead-reckoning, the estimation from the neural network takes the form of a prediction of where the entity will be positioned next. The comparison of the entity's true position and the position according to the dead-reckoning model takes place in the ModSAF *libentity* library. As such, in Experiment 1, the neural network models replaced the *libentity* library. The implementation of this functionality in ModSAF is illustrated in Figure 4.

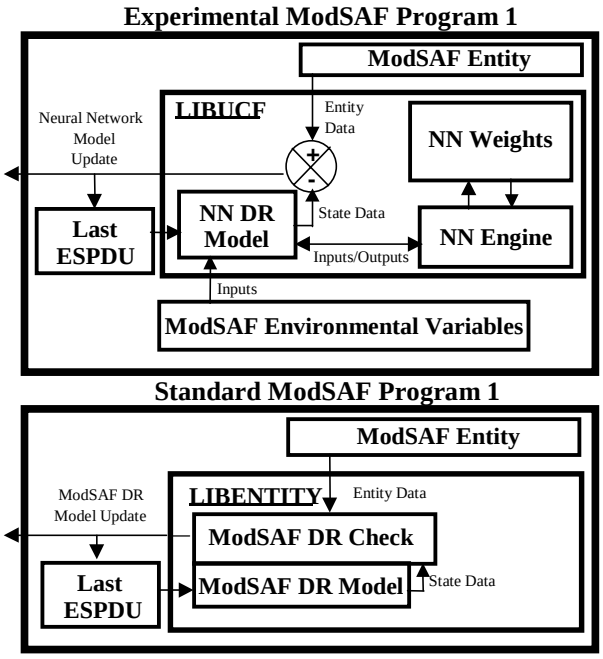


Figure 4. Functional Relationship of Neural Networks to ModSAF Dead-Reckoning Code

Since the neural networks were implemented in the Experimental ModSAF Program 1 as dead-reckoning models, the measure of performance was based on numbers of ESPDUs resulting from both models. That is, errors in location or orientation generated by the experimental neural networks program were compared with the same number generated by the standard ModSAF program. As evidenced in Figure 5, there was a notable reduction in ESPDUs resulting from errors in location but no significant reduction in ESPDUs resulting from errors in orientation. There was, however, a 22% overall combined reduction in ESPDUs caused by errors in location and orientation. Given that ESPDUs account for a majority of the network traffic in a distributed simulation exercise, this would result in a meaningful reduction of bandwidth requirements.

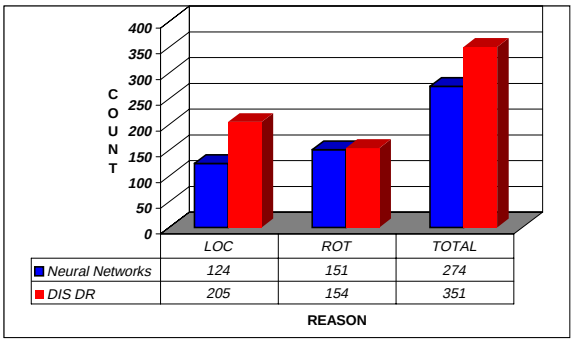


Figure 5: ESPDU Count Using Neural Networks and DIS DR.

Experiment 2

In the behavioral model application, the neural network estimation takes the form of a dynamic control model. Figure 6 illustrates how this functionality was implemented in ModSAF.

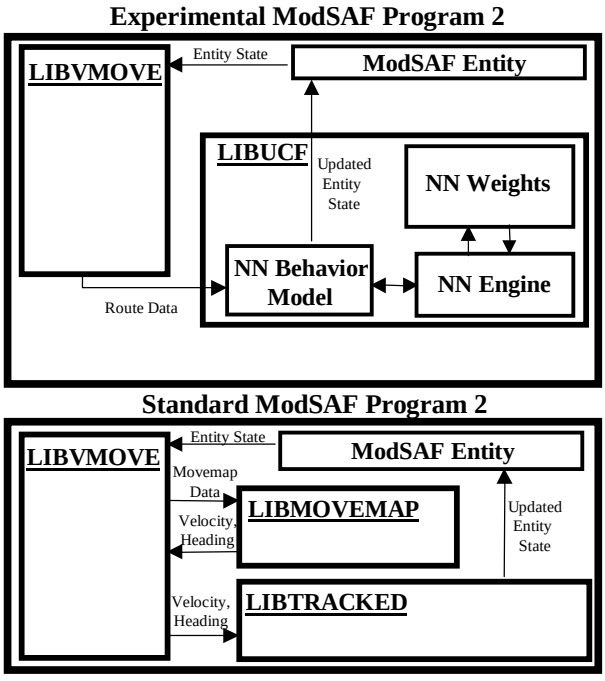


Figure 6. Functional Relationship of Neural Networks to ModSAF Behavior Code

Since the neural networks were implemented in the Experimental ModSAF Program 2 as behavioral/controller models, the measure of performance was based on the execution time resulting from both models. That is, the processing time for the experimental configuration was compared with the processing time for the configuration in Standard ModSAF Program 2. This measure was derived by use of the UNIX "*gettimeofday*" function while executing ModSAF on a Pentium II, 300 Mhz, machine running RedHat Linux 5.2. As reported in Table 3, the processing time for the Experimental ModSAF Program 2 was clocked at an average of 2.925×10^{-4} seconds, and the processing time for Standard ModSAF Program 2 was clocked at an average of 1.7650×10^{-4} seconds. That is, the experimental program required almost two-thirds more processing time than the standard ModSAF near-term movement model required.

	NN	SAF
Mean	2.9250×10^{-4}	1.7650×10^{-4}
Max	1.9870×10^{-3}	4.2075×10^{-4}
Min	2.5594×10^{-4}	1.3600×10^{-4}

Table 3. Execution Time (in seconds) Using Neural Networks and Procedural Code

DISCUSSION

The use of the neural network based SAF model resulted in a 66% increase in processing requirements. This indicates that in terms based solely on processing power, the ModSAF near-term movement model (*libmovemap* and *libtracked* for an M1A2 entity) outperformed the neural networks. It is true, however, that neural networks developed from human skill data do have the potential to yield behavioral models of increased fidelity. Unfortunately, until improved methods for measuring and validating SAF models are in place, this potential can not be adequately quantified. Additionally, as indicated earlier in the paper, neural networks are easily parallelized. If the execution time of a SAF model is an important performance measure given some application, then the investigation of model execution time on a multi-processor machine would be warranted.

The use of the neural network based dead-reckoning models resulted in a 22% reduction of ESPDUs caused by errors in the entity's position and/or orientation. Future work in this aspect of the study includes determining the processing tradeoff for this application. This requires evaluating the processing time of the Newtonian equations as well as the time saved by processing fewer ESPDUs. Also, instrumental in this phase of the project is the improvement of the neural networks' performance. This includes considering alternative types of architectures, inputs, normalization schemes, sampling strategies, etc. Since the ModSAF infrastructure to collect data and evaluate models (i.e., *libucf* in Figures 4 and 5) is now in place, more work can be done to improve these preliminary results.

ACKNOWLEDGEMENTS

This work was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command as part of the Inter-Vehicle Embedded Simulation and Technology (INVEST) Science and Technology Objective (STO) contract N61339-98-K-0001. That support is gratefully acknowledged.

BIBLIOGRAPHY

Crowe, M. (1990). The Appliction of Artificial Neural Systems to the Training of Air Combat Decision-Making Skills. Proceedings of the 12th Interservice/Industry Training Systems Conference, Orlando, FL. Nov 6-8, 1990, pp 302-312.

Fix, E., and Armstrong, H.G., (1990). Modeling Human Performance with Neural Networks. Proceedings of the International Joint Conference on Neural Networks, vol. 1, pages 247-252.

Franceschini, R.W., Petty, M.D., Schricker, S.A., Franceschini, D.J., and McCulley, G. (1999). Measuring and Improving CGF Performance. In Proceedings of the Eighth Conference on Computer Generated Forces and Behavior Representation. Orlando, FL., May 11-13, pp. 9-15.

Jaszlics, S.L. (1993). Artificial Intelligence in Tactical Command and Control Applications: Architecture and Tools. In Proceedings of the Third Conference on Computer Generated

Forces and Behavior Representation. Orlando, FL., March 17-19, pp. 367-373.

Lin, K., and Ng, H. (1993). Coordinate Transformations in Distributed Interactive Simulation (DIS). Simulation, vol. 61, No. 5, Nov, 1993, pp. 326-331.

Modjtahedzadeh, A., and Hess, R.A., (1993). A Model of Driver Steering Control Behavior for Use in Assessing Vehicle Handling Qualities. Transactions of ASME Journal of Dynamic Systems, Measurement, and Control, vol. 115, no. 3, pp. 456-464.

Nechyba, M.C. and Xu, Y. (1997). Learning and Transfer of Human Real-Time Control Strategies. Journal of Advanced Computational Intelligence, vol.1, pp. 137-154.

Pentland, A. and Liu, A., (1995). Toward Augmented Control Systems. Proceedings of Intelligent Vehicles, vol. 1, page 350-55.

Pomerlau, D., Thorpe, C., Longer, D., Rosenblatt, J.K., and Sukthankar, R., (1994). AVCS Research at Carnegie Mellon University. Proceedings Of Intelligent Vehicle Highway Systems America 1994 Annual Meeting, p. 257-262.

Pew, R.W., and Mavor, A.S., eds. (1998). Modeling Human and Organizational Behavior: Application to Military Simulations. Washington, DC: National Academy Press, 418 pages.

Ready, R., and Garrett, R., (1995). Future Technology Challenges in Distributed Interactive Simulation. Proceedings of the IEEE, August, p. 1188-1194.

Rummelhart, D., and McClelland, J. (1986). Parallel Distributed Processing. MIT Press, Cambridge, MA.

Schricker, S.A., Franceschini, R.W., and Mukherjee, A. (1998). Feasibility of Hardware-Based Computer Generated Forces for Embedded Training, Proceedings of the 1998 Interservice/Industry Training, Simulation and Education Conference, Orlando, FL, November 30-December 3, 1998.

Smith, J. (1994). Near-term Movement Control in ModSAF. In Proceedings of the Fourth Conference in Computer Generated Forces and Behavior Representation. Orlando, FL: University of Central Florida Institute for Simulation and Training.

Smith, S., and Petty, M. (1992). Controlling Autonomous Behavior in Real-Time Simulation. In Proceedings of the Second Conference in Computer Generated Forces and Behavior Representation. Orlando, FL: University of Central Florida Institute for Simulation and Training.