LA-UR- 95-1391

Title: ADAPTIVE CAPTURE OF EXPERT KNOWLEDGE

Author(s): Christopher L. Barrett
Roger D. Jones
Un Kyong Hand

Submitted to: Informal distribution
Summary of research completed

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

Los Alamos
NATIONAL LABORATORY

MASTER

Form No. 836 R5
ST 2629 10/91

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

Adaptive Capture of Expert Knowledge

DRAFT: DO NOT RELEASE

Christopher L. Barrett[1], *Roger D. Jones*[1], *and Un Kyong Hand*[1, 2]

1.0 Abstract

A method is introduced that can directly acquire knowledge-engineered, rule-based logic in an adaptive network. This adaptive representation of the rule system can then replace the rule system in simulated intelligent agents and thereby permit further performance-based adaptation of the rule system. The approach described provides both weight-fitting network adaptation and potentially powerful rule mutation and selection mechanisms.

Nonlinear terms are generated implicitly in the mutation process through the emergent interaction of multiple linear terms. By this method it is possible to acquire nonlinear relations that exist in the training data without addition of hidden layers or imposition of explicit nonlinear terms in the network.

We smoothed and captured a set of expert rules with an adaptive network. The motivation for this was to (1) realize a speed advantage over traditional rule-based simulations; (2) have variability in the intelligent objects not possible by rule-based systems but provided by adaptive systems; and (3) maintain the understandability of rule-based simulations. A set of binary rules was smoothed and converted into a simple set of arithmetic statements, where continuous, non-binary rules are permitted. A neural network, called the expert network, was developed to capture this rule set, which it was able to do with zero error. The expert network is also capable of learning a nonmonotonic term without a hidden layer. The trained network in feedforward operation is fast running, compact, and traceable to the rule base.

2.0 Introduction

2.1 Objective

The specific goal of this project is to produce compact, fast running, and adaptive representations of rule-based logic used to control simulated intelligent actors and to demonstrate credible behavior of these "intelligent" simulation objects in a simplified simulation of a combat environment. This also has general applicability to any simulation domain where representation of human behavior is an essential element.

1. Los Alamos National Laboratory
2. U. S. Navy

2.2 Goals of Combat Simulation

Intelligent computer generated forces have long been discussed but have, as yet, remained an elusive and unobtainable goal. At the same time this deficiency in credible C3I (Command, Control, Communication, and Intelligence) representation within combat simulations is generally acknowledged, the modern battlefield places an even greater reliance on performance in this area. The credible representation of Command and Control functions and the incorporation of effective multi-source data fusion algorithms are central to the development of a viable, large scale synthetic battlefield environment.

Typically, behavior in combat simulations has been represented by rule-based, decision table, and predicate-action type logic structures which require, in many instances, supplemental "scripting" to ensure they are doctrinally and tactically sound, given a particular scenario context.

Such traditional Artificial Intelligence and decision theoretic approaches to the representation of reasoning have many beneficial properties which, are, unfortunately, often rendered less useful because of similarly many drawbacks. Two of the virtues of knowledge engineered rule-based systems are the immediate accessibility of the explicit representation of knowledge and the close interplay of rule construction, expert knowledge, and opinion.[3, 4] However, environments characterized by large knowledge structures and/or large numbers of interacting intelligent entities tend to be slow running and resist modification.[5, 6] Moreover, emergent behaviors of the global system (those that arise from interaction among intelligent entities) are often a central issue and are not illuminated in any important way by rule-based entity representations.[7, 8] Indeed, the computation problems impede adequate understandability of the overall systems dynamical behaviors.

2.3 Shortcomings of Traditional Neural Networks

Traditional neural networks have been limited in application by certain characteristics, some of which can be briefly described here. There is no "universal" network that can be practically applied to every problem. What a network can learn is dependent upon its architecture, and not simply the learning objective. Therefore, different networks have different natural applications. Moreover, it is impossible for most neural network architectures to learn arbitrary logical rules without a correspondingly complicated structure.[9, 10] Neural networks almost always learn behavior approximately; there is some degree of error associated with even fully trained networks. Finally, there exists a "chicken-

3. See Feigenbaum, 1981.
4. See Winston and Brown, 1979.
5. See Lenat, et. al., 1984, 1986, 1990.
6. See Holland, 1986.
7. See Rassmussen and Barrett, 1995.
8. See Baas, 1994.
9. See Rumelhart, et.al., 1986.
10. See Jones, et. al., 1991.

egg" dilemma. A neural network will learn the performance produced by a simulation, and therefore is only as "good" as the simulation. So a good simulation is required to train a neural network in the first place, but the characteristics of the simulation depend in part on the quality of the interacting intelligent objects.

2.4 Approach

Our work uses an Artificial Intelligence approach and adaptive systems (neural network) to represent the cognitive process in the belief that, if properly constructed and applied, tactical behavior will emerge rather than be forced as a result of state variables reaching some threshold condition. What is meant by a "hybrid" approach here is one in which we initially knowledge engineer rule-based logic for individual actors and then we directly acquire the input/output structure of the logical system by use of the neural network described in this paper. The representation of explicit individual actor logic in an adaptive form then allows the logic to be adapted and potentially even transformed via mutation and selection mechanisms using performance-based training by simulation of outcomes of interactions among the intelligent actors. The changed network can then be traced back to alternatives of the explicit rule-based system for a clearer understanding of the effects of performance-based learning.

This concept more closely represents real world decision making processes than the kind of "checklist mentality" represented by rule sets, decision tables, and expected value thresholds. Purely computationally, rule-based systems run slowly, have extreme memory requirements, and are very difficult to modify for any reason. By taking this hybrid approach, we hope to alleviate the problems, while retaining the explanatory power, of the rule-based approach.

This work represents a paradigm shift in the representation and behavior of the simulation entities or "actors." First, it takes advantage of the distinction between data structure and function in an object oriented software approach and distinguishes/compartmentalizes the behavioral function of an object into its physical and logical (e.g. Command/Control) components. It is the cognitive behavioral representation of the object which is the primary technical challenge addressed in this work. Secondly, most neural network approaches rely exclusively on performance-based training to learn. Our approach is designed specifically to first acquire explicit, knowledge-engineered, rule-based reasoning systems, and then to allow performance-based modification of those acquired rules. In addition, it learns the rules exactly.

The neural network developed for this project is called the expert network. A reasonable description of this network is that it is a multi-linear, logic acquisition network. Other networks were tried, but the expert network had the best performance on the task of direct acquisition of logical reasoning.

The three main issues that will be addressed in this paper are (1) Speed; (2) Variability; and (3) Traceability to Knowledge Engineering.

3.0 Rule Set

3.1 Rule Set for Capture

For the initial development of our approach, a rule-based representation of target selection reasoning by a tank was studied. This logic was a part of simulated tank tactics in an object-oriented land warfare simulation.

The rule set for the expert system examines a potential threat and evaluates its state to determine its threat value. This "threat value" refers to the priority a target has regarding the decision making process for weapon selection.

The three target types that are considered are a Tank threat, an Infantry threat, and an APC threat. Based on a combination of three parameters: whether the target is (1) in range, (2) damaged, or (3) moving, that target is assigned a threat value by which all the targets can then be prioritized and rank-ordered. These rules are binary in nature; that is, a threat is either in range, or out of range, damaged or not damaged, and moving or not moving. By this representation in the existing simulation there is no degree to which one of these parameters is true.

The threat values are assigned over an interval of [1,1000], where 1 is the greatest threat and 1000 is the least. A tank threat is the most dangerous, then an infantry threat, and an APC is the least dangerous. A threat that is out of range is simply ignored by giving it a threat value of 1000. If a threat is damaged, its threat value is decreased by 50 points, and if it is moving, 20 points are added to the threat value. The possible threat values assigned by the rules can be seen below:

TARGET TYPE	IN RANGE	DAMAGED	MOVING	THREAT VALUE
TANK	YES	NO	NO	1
INFANTRY	YES	NO	NO	2
APC	YES	NO	NO	3
TANK	YES	NO	YES	31
INFANTRY	YES	NO	YES	32
APC	YES	NO	YES	33
TANK	YES	YES	NO	61
INFANTRY	YES	YES	NO	62
APC	YES	YES	NO	63
TANK	YES	YES	YES	81
INFANTRY	YES	YES	YES	82
APC	YES	YES	YES	83
TANK	NO	YES/NO	YES/NO	1000+
INFANTRY	NO	YES/NO	YES/NO	1000+
APC	NO	YES/NO	YES/NO	1000+

Because this binary representation is arbitrary and results in an obvious loss of realism, we decided that a straightforward first step would be to smooth the input variables and rules to allow fuzzy reasoning.

3.2 Rule Smoothing

Since there is no method in the rule set to differentiate between a slightly damaged tank and a severely damaged tank, there is no way the threat value can reflect any difference in priority. As a result, it is possible for targets with unidentical characteristics to be assigned identical threat values, which, in turn, would result in unidentical threats having the same rank ordering. It is also common to unrealistically include or exclude a threat when it is very near a decision variable's boundary. This binary nature is unfortunately a common property of many types of expert rule sets.

In our approach, we generalize the rule set by smoothing the input space. Each input variable is allowed to be a continuous valued quantity on the interval [0,1], where 1 is the greatest threat and 0 is the least. For example, the input for a partially damaged threat could now have a value of 0.345, indicating a degree of damage, instead of just a value of 0 (damaged) or 1 (undamaged). In rescaling the output, the minimum threat value in the expert system was re-defined as 100 rather than 1000. This allows for a more even population of the output interval with the effect that the learning system will be more able to capture the behavior of the rules.

The threat value output of the network maps to the existing simulation values by the following relation, where T_s is the threat value of the existing simulation on the interval [1,1000], and τ is the threat value output by the expert network:

$$T_s = 101 - 100\tau \qquad \textit{Equation 1}$$

This relationship is obviously valid only for values of T_s between 1 and 100, since the maximum value of τ is 1. As can be seen in the table listing possible threat values in section 3.1, there are no threat values in the existing simulation with a value between 100 and 1000. Therefore, if the threat value is greater than 100, it is automatically assigned a "no threat" value of 1000.

The rules which describe the evaluation of a threat can be succinctly expressed in the arithmetic expressions listed below. τ is the threat value, and x_1, x_2, and x_3 are the input variables for range, damaged, and moving, respectively.

For a Tank threat:

$$\tau = (1 - x_1)[x_2x_3 + 0.7x_2(1 - x_3) + 0.4(1 - x_2)x_3 + 0.2(1 - x_2)(1 - x_3)] \quad \textit{Equation 2}$$

For an Infantry threat:

$$\tau = (1 - x_1)[0.99x_2x_3 + 0.69x_2(1 - x_3) + 0.39(1 - x_2)x_3 + 0.19(1 - x_2)(1 - x_3)] \quad \textit{Equation 3}$$

For an APC threat:

$$\tau = (1 - x_1)[0.98x_2x_3 + 0.68x_2(1 - x_3) + 0.38(1 - x_2)x_3 + 0.18(1 - x_2)(1 - x_3)] \quad \textit{Equation 4}$$

The input variables are defined as follows:

Range:
$x_1 = 0$: threat very close (minimum distance)
$x_1 = 1$: threat well out of range
$x_1 = 0.5$: range threshold

Damaged:
$x_2 = 0$: threat completely damaged
$x_2 = 1$: threat completely undamaged

Moving:
$x_3 = 0$: threat moving rapidly
$x_3 = 1$: threat completely still

These equations were derived directly from the rules and simply make calculation of threat values more simple, as well as all continuous variable values without rule modification.

By smoothing the input space, each unique combination of continuous input variables results in a unique real valued threat value, thereby preventing targets with different state characteristics from having the same threat value or rank ordering, as was possible in the binary expert rule set.

4.0 Expert Network

4.1 Expert Network Concept

Consider a real valued function f of binary input, x. The function can be expressed as a linear combination of x and $(1 - x)$. The quantities x and $(1 - x)$ can be thought of as two basis functions which span the binary space and the function f lies within the span. If f is a function of n inputs, there are then 2^n basis functions which span the space. For example, if f is a function of two binary variables x_1 and x_2, then f can be written as the linear combination of x_1x_2, $x_1(1 - x_2)$, $(1 - x_1)x_2$, and $(1 - x_1)(1 - x_2)$. Furthermore, if f is a function of three inputs then there are eight basis functions which span the space of the input variables. These basis functions correspond to the corners of the unit hypercube.

	x_1	x_2	x_3
$x_1x_2x_3$	0	0	0
$x_1x_2(1 - x_3)$	0	0	1
$x_1(1 - x_2)x_3$	0	1	0
$x_1(1 - x_2)(1 - x_3)$	0	1	1
$(1 - x_1)x_2x_3$	1	0	0
$(1 - x_1)x_2(1 - x_3)$	1	0	1
$(1 - x_1)(1 - x_2)x_3$	1	1	0
$(1 - x_1)(1 - x_2)(1 - x_3)$	1	1	1

Another way of looking at the functioning of the network is that it is effectively a component matching network. An input value of a variable can be compared to the rows that define the truth table entries that bound the input domain of the rules. If the components of a boolean input state vector are compared, component by component, with row entries in the truth table, obviously only one row can match. If the input vector components are real valued, i.e., smoothed or fuzzy inputs, then a "degree of match" will be apportioned over the rows as a function of how far an input is from different rows. For an input x_j and a row component of the truth table, c_{ij}, then we want the following truth table relationship to check for a match between x_j and c_{ij}:

x_j	c_{ij}	"match"
1	1	1
1	0	0
0	1	0
0	0	1

which is simply the biconditional relation of propositional logic. We develop such a relation below that will be used as the transfer function, u_i, in the expert network.

Consider the following, with variables a and b:

$$a - b = -(-a + b)$$

$$= -(-a + -(-b))$$

which, in the boolean case, is equivalent to

$$= \neg(\neg a \vee \neg(\neg b)) \ .$$

Then consider that by DeMorgan's Law,

$$a \wedge b = \neg(\neg a \vee \neg b)$$

as can be seen in the following truth table:

a	b	$\neg(\neg a \vee \neg b)$	$a \wedge b$
1	1	1	1
1	0	0	0
0	1	0	0
0	0	0	0

Therefore it follows that $\neg(\neg a \vee \neg(\neg b)) = a \wedge \neg b$.

Again, inspection of the truth table shows that, $a \wedge \neg b \equiv \neg (a \Rightarrow b)$:

a	b	$a \wedge \neg b$	$a \Rightarrow b$	$\neg (a \Rightarrow b)$
1	1	0	1	0
1	0	1	0	1
0	1	0	1	0
0	0	0	1	0

Now, given a function $f = a - b$, in the boolean case, $= a \wedge \neg b$, we can complete the component matching function.

By definition of absolute value,

$$|f| = \begin{cases} f, & f \geq 0 \\ -f, & f < 0 \end{cases} .$$

Since $-f$ corresponds to $\neg a \wedge b$, and a logical "and" is implicit in this definition, we can thus extend to continuous variables,

a	b	$\neg (a \wedge \neg b) \wedge \neg (\neg a \wedge b)$	$(1 - \lvert a - b \rvert)$
1	1	1	1
1	0	0	0
0	1	0	0
0	0	1	1

4.2 Expert Network Architecture

We then take a product over the component-by-component match, which corresponds to the logical "and" of a match for all components in a row:

$$u_i = \prod_{j=1}^{D} (1 - \lvert (x_j - c_{ij}) \rvert) \ . \qquad \textit{Equation 5}$$

The upper limit of the product, D, is the input dimension, x_j is the input variable and the c_{ij}'s are the corners of the unit hypercube which bound the input space. This product produces the exhaustive combinations of x_1, x_2, and x_3.

A network architecture can then be defined using the product u_i as the neuron transfer function:

$$\Phi(\vec{x}) = \sum_{i=1}^{N} w_i u_i \,, \qquad \textit{Equation 6}$$

where N is equal to 2^D.

4.3 Learning Algorithm

The learning algorithm is a function of the difference between the desired output and the actual output of the network. The training weights, w, are adjusted based on the error of the last training set. As error decreases the weights will converge.

The learning algorithm for the weights is:

$$w_i^{t+1} = w_i^t + \frac{\eta(\tau - \Phi(\vec{x}))\, u_i}{\sum_{i=1}^{N} u_i^2 + 5\eta} \,, \qquad \textit{Equation 7}$$

where:
η is a learning constant,
t indexes the training iteration,
τ is the desired threat value output,
$\Phi(\vec{x})$ is the output of the network for threat value.

4.3.1 Normalization of the Learning Algorithm

The denominator of the learning algorithm can be qualitatively understood as follows. Equation 6 can be written in vector form as

$$\Phi = \underline{w} \cdot \underline{u} \ .$$

Learning is described in the sense of $\underline{w}$ moving toward some desired $\underline{w}^*$ where it is assumed that

$$\Phi_{desired} = \underline{w}^* \cdot \underline{u}$$

exists.

The degree of learning is evaluated by calculating

$$(\Phi_{desired} - \Phi) \equiv \Delta\Phi \ .$$

The learning process is described by the trajectory of changes in $\underline{w}$, $\Delta\underline{w}$, as $\Delta\Phi \rightarrow 0$. So we write heuristically that

$$\underline{w}^{t+1} = \underline{w}^t + \eta f(\Delta\Phi)\,\underline{u}$$

$$= \underline{w}^t + \eta f\left(w^*\underline{u} - w^t\underline{u}\right)\underline{u}$$

where f is some function.

The units are obviously of interest here. $\underline{w}$ is unitless, so f must normalize the difference, $\Delta\Phi$. The appropriate normalization is

$$\underline{u} \cdot \underline{u} \ ,$$

such that

$$\Delta w^{t+1} = \Delta w^t + \frac{\eta\,(\underline{w}^* \cdot \underline{u} - \underline{w} \cdot \underline{u}) \cdot \underline{u}}{\underline{u} \cdot \underline{u}}$$

$$\Delta\underline{w}^{t+1} = \Delta\underline{w}^t + \frac{\eta\left(\underline{w}^* - \underline{w}^t\right)\underline{u} \cdot \underline{u}}{\underline{u} \cdot \underline{u}}$$

$$= \Delta \underline{w}^t + \frac{\eta \Delta \underline{w}^t \cdot \underline{u}}{\underline{u} \cdot \underline{u}} \cdot \underline{u}$$

$$= \eta \cos\Theta \cdot \frac{\underline{u}}{\sqrt{\underline{u} \cdot \underline{u}}}$$

$$= \eta \cos\Theta \cdot \underline{u}$$

where $\cos\Theta$ is the projection of the change in weights, $\Delta \underline{w}$, onto $\underline{u}$, the unit hypercube.

The additive term in the denominator is a translation that defines the scalar value of the components of $\underline{w}$.

5.0 Network Evaluation

5.1 Network Training

The network was trained exclusively on cornerpoints, or the rows of the truth table. Each Training Epoch consists of 10 sets of (8) cornerpoints as inputs, and their desired outputs, which are calculated from Equations 2-4. The network is trained on 10 epochs, which means it is shown a data set 800 times. Because the network trains only on boolean inputs, the learning constant is set to 1.0. The weights converge to exactly the value of the corresponding coefficients in Equations 2-4.

5.2 Network Testing

Testing of the expert network was accomplished by showing the network another set of randomly generated input data and the corresponding output data. The test data are real numbers between 0 and 1. This is important to notice primarily because the network is *trained* using only the unique boolean combinations that define the cornerpoints (e.g., the truth table rows). The network retrieves the trained weights and computes its own output. The RMS error is zero to four places, showing that the network learned the rules exactly.

6.0 Issues

6.1 Speed

6.1.1 First Order

A specific goal of using neural networks in combat simulation was to decrease the run time of the simulation. A comparison of both the neural network technique and the rule set was made and the difference in run times was not significant. It should be emphasized that the network code has not been optimized. For example, it currently uses many multiplies and divides, whereas the rule system is largely confined to adds and logical branching.

6.1.2 Second Order

The image size of the executable code for the neural network is smaller than that of the rules. Thus, more objects can be in the simulation before it begins to "thrash." Intermediate storage values during the rule trace is large and variable. The network has essentially no dynamic storage and minimal static storage of weights.

Rule systems have variable execution times. The variation in the run times for the rules is a result of the logical structuring of the rules in the actual code. For example, maximum run time was 62% longer than the minimum run time for the same rule set, with different inputs.

The neural network does have the advantageous characteristic of having a constant run time, whereas the rule set is dependent of the rules it must test. For example, depending on the parameters of the target (target type, range, damage, and motion) a different number of rules, or "if statements" will be executed. The neural network, however, is a mathematical computation that is independent of the logical structure. Therefore, neural networks will always have the same execution time. This is a very desirable characteristic to aid in load balancing for large scale simulations.

6.2 Flexibility/Variability

In a combat simulation it is often desirable to model variance in human behavior. Even though individuals may be trained by the same doctrine, each individual will not have the same response in a given situation. A rule-based simulation is only capable of executing what is specifically written out in rules. Any variation requires rewriting and recoding of the rules, which is not only tedious, but has the tendency to produce unexpected and untraceable changes in the simulation.[11]

11.See Lenat, ibid.

6.3 Range of Behavior

A more realistic simulation would have the capability to model this range of behavior. Therefore it would be useful to have a tool to measure the effect of varying degrees of compliance to doctrine. There are two possible approaches to this concept with this neural network. Because the network spans the rules exactly, it is possible to bias the training to model a certain type of behavior, or training could be stopped early to achieve, for example, 90% compliance to doctrine.

6.3.1 Performance-based Rule Adaptation

Performance-based training is another capability of this network. Adaptive elements can replace the rule-based elements in simulation. It is then possible to train the individual adaptive elements based on their performance in the large-scale simulation environment. This leads to the identification of a new or improved rule set.

Performance-based adaptation is achieved in two ways. The first amounts to "tuning" the network. That is, in a fixed logical structure, weights and constant values can be adjusted to improve performance. Second, entirely new rules can be discovered and tested using mutation and selection techniques (see section below on Nonmonotonicity). The expert network is capable of both kinds of learning.

6.4 Traceability to Knowledge Engineering

In a rule-based system it is apparent that the rules dictate the behavior of the intelligent objects. Neural networks have traditionally had the undesirable property of a "black box." That is, inputs are turned in to outputs and it is not clear what has transpired to produce these outputs. It is important to incorporate knowledge engineering into a simulation using adaptive elements. Knowledge engineering has important advantages over performance-based training, specifically in that transition rules are explicit and understandable.

The neural network developed for this project learns rules, and thereby intrinsically incorporates knowledge engineering. As a result, then, it is possible to derive an explicit prescription for behavior for later adaptive modification of the elements based on performance.

5.0 Network Performance

5.1 Network Performance on Smooth Rule Set

The output of the network is a threat value for each of the potential threats. The threat value is determined by the rule-based system given by Equations 2-4, and is actually a subjective value. The key operational quantity is the rank ordering of the threats. Thus, any systems with the same rank order-

ing are operationally equivalent. This allows a performance criterion or risk, ***R***, to be specified:

$$R = \frac{1}{T}\sum_{i=1}^{l}\tau_i \qquad \textit{Equation 8}$$

where ***T*** is the sum of the total number, ***L***, of threat values:

$$T = \sum_{i=1}^{L}\tau_i \qquad \textit{Equation 9}$$

L is the total number of threats, and ***l*** is a number less than or equal to ***L*** that represents the number of threats being considered. The summation in Equation 8 is over the ***l*** largest threat values as determined by a rank ordering. The actual threat values used are those determined by Equations 2-4 rather than those determined by any given network. This would represent "ground truth" risk factor. The ground truth risk is an upper limit on the risk value for each ***l***. If a network captures the ranking exactly, the risk calculated from that ranking will equal the ground truth risk. If, however, the ranking is different for the network than the ground truth ranking, then a high priority threat value within the summation set will be substituted for a lower ranking threat value. The network effect will be to lower the value of the summation. Figure 1 is this risk graphically displayed for four different neural networks, where the x- and y-axes are ***L*** and ***R***, respectively.[12]

12. See Jones, et. al., 1990.

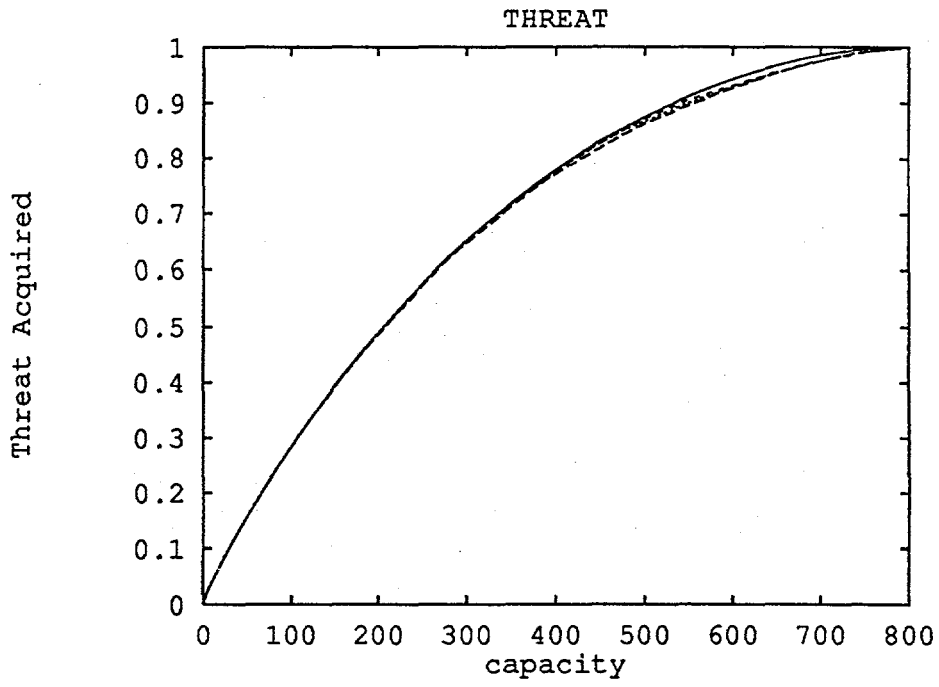

Figure 1. Prioritization Error: Deviations from the correct rank-ordering are idicated by departure from the ground truth curve. Expert network has zero error and therefore lies exactly on the ground truth curve; three other networks show error.

There are three curves that deviate from the ground truth curve. These are other networks that were tried. The expert network lies exactly on the ground truth curve, and so its rank ordering is exactly that of ground truth. Figure 2 is the error of Figure 1 graphically expanded. Again, it should be noted that there are three error curves that are visible because the error curve for the expert network lies exactly on the axis for zero error.

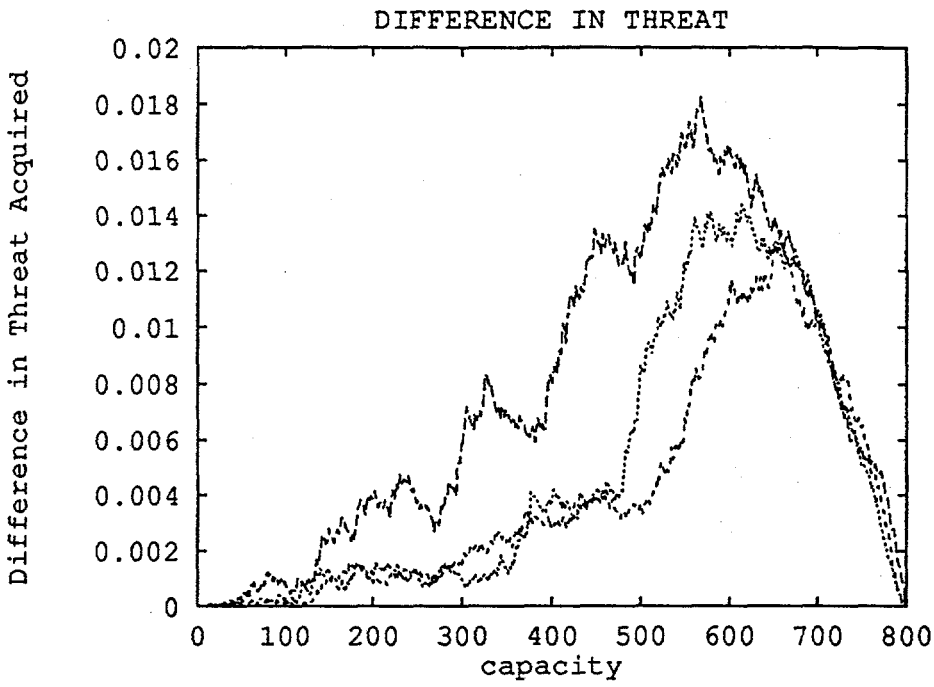

Figure 2. Prioritization Error Graphically Expanded.

6.0 Extensions

6.1 Nonmonotonicity

An interesting characteristic of this expert network is that it has the capability of learning a nonmonotonic function (such as the "exclusive or" XOR) without adding a hidden layer to the network architecture.[13] The range variable, x_1, for example, is not realistically linear or monotonic. There is, in most instances, a minimum effective range associated with weapons just as there is a maximum range.

To determine if the network would learn this, a "too close" scenario was added to the training set. This was done by explicitly including a nonlinear product, $(1 - x_1)(1 - x_1)$ in the learning rule. In order for this to be nontrivial, it is necessary to train on randomly generated, smoothed input data.

13. See Rumelhart, et. al., ibid.

The result was that the original network, which did learn maximum range, did not learn the more complex rule between minimum and maximum range. The error terms from a sampling of train epochs is listed in Table 1 below.

Table 1: **"Too Close": Nonmonotonicity not learned**

Train Epoch	Error	Sum Error	RMS Error
1	-0.044043	57.257866	2.024371
5	-0.036353	29.377436	1.038649
10	-0.035883	29.370567	1.038406
20	-0.035869	29.371565	1.038442
30	-0.035869	29.371566	1.038442
90	-0.035869	29.371566	1.038442

Table 1: A minimum range term was added without increasing the input dimension. Note that the error terms converge only slightly during the first several training epochs, but fail to converge any further.

However, if the input dimension is increased to include x_4 defined in terms of the range as $(1 - x_1)$, then a nonlinear term of $(1 - x_1)(1 - x_1)$ is *implicitly* formed. The the network did then learn the non-monotonic relation between minimum and maximum range without the addition of a hidden layer. The results are listed below in Table 2. Thus the nonlinear term emerges as in the dynamics of the interaction of the network and the relations in the training set. That is, the nonlinear term is nowhere explicitly represented in the network. Moreover, this is the basis for a mutation mechanism for the network that can be triggered when error convergence is not achieved, as described in the next section.

Table 2: **"Too Close": Nonmonotonicity learned**

Train Epoch	Error	Sum Error	RMS Error
1	0.009125	49.259193	1.741575
10	0.000015	0.766034	0.027083
20	0.000003	0.099998	0.003535
30	0.000000	0.015588	0.000551
50	0.000000	0.000761	0.000027
80	0.000000	0.000246	0.000009

Table 2: A minimum range was added by defining another input x_4, *the minimum range variable in terms of the range as* $x_4 = 1 - x_1$.

6.2 Relationship to more general Mutation and Selection Mechanisms such as Genetic Algorithms

As can be seen in the approach to learning a nonmonotonic relation in range above, this type of network has the potential for very powerful mutation and selection capabilities.

The "span of the boolean space" is simply a volume defined by the corner points of a truth table in the dimension of the number of input variables. When error convergence is not observed for a particular spanning boolean space, an implicit non-linear boolean space can be defined. If convergence in error is not achieved, the nonlinear term is implicitly defined as, for example, $x_4 = (1 - x_1)$. Thus the x_1x_4 product terms will contain x_1^2 implicitly. Inserting such terms as a result of nonconvergence can be automated as a random mutation mechanism.

Each new boolean term adds an infinite number of possible logical rules that contain the input relation to the system. The selection mechanism is simply the pruning process of setting network weights associated with nonessential corners in the spanning boolean space to zero. This can be seen in Table 3 below, in the case in which the network contains a nonlinear term but the training set is linear.

Table 3: **Truth Table with 4 dimensions.**

x_1	x_2	x_3	x_4	weight
0	0	0	0	0.2
0	0	0	1	0.2
0	0	1	0	0.4
0	0	1	1	0.4
0	1	0	0	0.7
0	1	0	1	0.7
0	1	1	0	1.0
0	1	1	1	1.0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Table 3. Exhaustive combination of four input variables, which represent the corners of the unit hypercube, and the associated weight for each corner point. Trained on three input variables in an architecture for four, the expert network prunes the fourth variable, and the weights are dependent only upon the original three variables on which it was trained.

7.0 Summary

This paper describes an adaptive system approach that permits direct acquisition of rule-based, or indeed, any logical formulation of, local state transition logic. There are many reasons to desire this capability for simulations, runtime efficiency and memory utilization among them. However, some of the most compelling reasons involve the ability to refine the rules using experience gained by observing behaviors generated as they are used in interaction with other entities. In order to refine the rules, we have described that this approach permits rule adaptive acquisition followed by adaptive performance-based modification. We have shown mutation criteria and mechanisms, as well as selection and pruning criteria that allow alteration of the structure of the network representation of the rules. The approach allows traceability of the learned changes back to the rules; that is, what improves the performance can be readily translated from the transformed network back to equivalent changes in the original rule system that was adaptively acquired.

During the course of this work, the approach was successfully applied by others to a weapon control problem.[14] After acquisition of tactical engagement logic in expert net, simulation-based perfor-

mance training resulted in 10-20% performance improvement and definable equivalent changes in the rules that would yield that improvement.

In all, this paper suggests a compelling direction for research and continued development of technology for the representation of simulated intelligent entities.

14. Personal communication with John Stroud, LANL staff member.

8.0 Bibliography

Baas, Nils A. (1994). "Emergence, Hierarchies and Hyperstructures," *Artificial Life III, Proceedings*. C. G. Langton, ed., Addison-Wesley/Santa Fe Institute Studies in the Sciences of Complexity.

Barr, Avron and Edward A. Feigenbaum. (1981). *The Handbook of Artificial Intelligence, vol I*. Los Altos, CA: William Kaufman, Inc.

Holland, John H., Keith J. Holyoak, Richard E. Nisbett, and Paul R. Thagard. (1986). *Induction*. Cambridge, MA: The MIT Press.

Jones, R. D., C. W. Barnes, Y. C. Lee, W. C. Mead. (1991). "Information Theoretic Derivation of Network Architecture and Learning Algorithms," LA-UR-91-325, *Proceedings of the International Joint Conference on Neural Networks*.

Jones, R. D., Y. C. Lee, S. Qian, C. W. Barnes, K. R. Bisset, G. M. Bruce, G. W. Flake, K. Lee, L. A. Lee, W. C. Mead, M. K. O'Rourke, I. J. Poli, and L. E. Thode. (1990). "Nonlinear Adaptive Networks: a Little Theory, a Few Applications," LA-UR-91-273, *Cognitive Modeling in System Control.*

Lenat, Douglas B. and J. S. Brown. (1984). "Why AM and Eurisko Appear to Work," *Artificial Intelligence* 23, 1984, pp. 269-294.

Lenat, Douglas B. and R. V. Guha. (1990). *Building Large Knowledge-Based Systems*. Reading, MA: Addison-Wesley Publishing Company, Inc.

Lenat, Douglas B., M. Prakash, and M. Shepherd. (1986). "Cyc: Using Common Sense Knowledge to Overcome Brittleness and Knowledge Acquisition Bottlenecks," *The AI Magazine*, vol. 6., no. 4, pp. 65-85.

Rasmussen, Steen and Christopher L. Barrett. (1995). "Simulatability, Computability, and System Representation: Elements of a Theory of Simulation," LA-UR-95-in preparation.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams. (1986). "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing, 1*. D. E. Rumelhart and J. L. McClelland, eds., pp. 318-362.

Winston, Patrick H. and Richard H. Brown, eds. (1979). *Artificial Intelligence: An MIT Perspective*. Cambridge, MA: The MIT Press.