Rule Extraction as a Formal Method for the Verification and Validation of Neural Networks

Brian J. Taylor, Marjorie A. Darrah
Institute for Scientific Research, Inc.
Fairmont, WV 26555-2720
E-mail: btaylor@isr.us, mdarrah@isr.us

***Abstract*—The term *formal method* refers to the use of techniques from formal logic and discrete math in the specification, design, and construction of computer systems and software. These techniques enable the formalization of software for development and testing so that it may be verified and validated in a more thorough way. Although not specifically identified in the literature as a verification and validation (V&V) formal method technique, neural network rule extraction fits the basic definition by using techniques from formal logic to formalize neural network software so that it may be examined more completely. This paper identifies several areas where rule extraction can be an effective tool for the V&V of neural networks.**

I. INTRODUCTION

One example of an adaptive neural network used in a safety- and mission-critical system comes from the Intelligent Flight Control (IFC) program. This program is a collaborative effort of the NASA Dryden Flight Research Center, the NASA Ames Research Center, Boeing Phantom Works, and the Institute for Scientific Research, Inc. (ISR). This program seeks to flight-demonstrate, on board an F-15 aircraft, a research flight control system that can adapt to accommodate changing aerodynamic conditions, including loss of aircraft surfaces or improper surface control. The first two generations of the IFC system have utilized neural networks. Since this experiment is flown with a human pilot, the system is both safety- and mission-critical.

Safety- and mission-critical uses of neural networks are not limited to flight control systems. Neural networks are used in vehicle health monitoring, power generation and transmission control systems, fault detection and identification in industrial processes, and medical diagnosis systems [1]. They have the potential for use in space exploration as part of a decision and control process for planetary rovers or for improving autonomous systems. Yet while neural networks are slowly being used in a few fields requiring high assurance, the limitations associated with their certification restricts their widespread acceptance.

Consider the IFC program's use of an adaptive neural network. Since the flight control system is a research, and not commercial, system, the certification process to approve the adaptive neural network is not as rigorous as it would be for Federal Aviation Administration (FAA) approval. Existing NASA standards, documentation and testing procedures, and a physical isolation of the neural network software from critical aircraft systems have thus far been enough to qualify as adequate assurance. However, as these adaptive control systems show promise, it is hoped such systems can eventually be given greater roles and eventually adopted in commercial airliners to improve their safety. The FAA review procedures for adaptive system use in a civilian aircraft will likely be very stringent and inhibit their use.

Developers in general of neural network systems have been cautious about expanding their use into safety- and mission-critical domains due to the complexities and uncertainties associated with these complex, adaptive software systems [2]. Since adaptive neural networks are beginning to be used within high-assurance systems, like the IFC program, the NASA Independent Verification and Validation (IV&V) facility has encouraged research in the area of neural network V&V to answer the question: How can we be sure that any system that includes neural network technology is going to behave in a known, consistent and correct manner?

Common formal method techniques like model checking and theorem proving appear unable to completely address the V&V requirements of neural networks. Model checking starts from an initial state and repeatedly applies the transition relation to search all reachable states for a property violation, while remembering explored states to avoid looping [3]. Model checking seems less applicable when the state space is infinite or extremely large, a possibility with an adaptive neural network. In [4], the idea of using model checking to verify properties of recurrent neural networks is discussed. The system presented in that paper, by the author's own conclusion, was clearly undecidable and therefore could not be automated.

Theorem proving is the use of logical induction over the execution steps of the program to prove system requirements. System requirements are translated into complex mathematical equations and solved by verification experts to prove the system is accurate [5]. Like model checking, theorem proving in the traditional sense does not seem to be applied to adaptive neural networks. Rather than an approach where the proof of requirements is done by logical induction over the structure of the program, the

approaches for adaptive neural networks deal with proving convergence and stability. Lyapunov or stochastic methods can be used and take the place of theorem proving for neural networks.

A formal approach that seems more suited for neural network V&V is rule extraction. Rule extraction has been researched by the neural network scientific community for at least the past two decades and fits the basic definition for a formal method because it uses techniques from formal logic to formalize neural network software so that it may be examined more completely. In essence, it changes a black box system into a white box system by translating the internal knowledge of a neural network into a set of symbolic rules. These rules have the potential for several uses in the V&V and certification of neural networks, many of which are discussed in the following sections.

A. Rule Extraction Overview

By design, neural networks change while training on a data set. After training, some networks are fixed while others are allowed to adapt during operation. It is a challenge to understand how the network will handle additional input. Testing can give some level of confidence but may not provide a satisfactory level in safety- or mission-critical cases.

A solution, rule extraction, is the process of developing natural language-like syntax that describes the behavior of a neural network [6]. These techniques can convert the neural network structure to a prepositional if-then format offering the possibility of requirements traceability. Without this representation, it becomes difficult to identify design aspects of the neural network since the internal knowledge does not undergo traditional design.

The same techniques used to map rules from the neural network in rule extraction can also be used in two additional ways: rule initialization and rule insertion.

Rule initialization [7] is the process of giving a neural network some pre-system knowledge, possibly through early training or configuration. A system developer may have improved confidence if the starting condition of the neural network is known, which may lead to a constrained path of adaptation.

Rule insertion [8] is the method of moving symbolic rules back into a neural network, forcing the knowledge to incorporate rule modifications or additional rules. A system developer can use this scheme to exert a condition or reinforce conditions within an adaptive neural network. Examples of this include restricting the neural network to a region of the input space or instructing it to deliberately forget some data it has already seen.

B. Rule Formats

There are several main rule formats. Rule extraction algorithms will generate rules of either conjunctive form or subset selection form, commonly referred to as M-of-N rules named for the primary rule extraction that makes use of the form. All rules follow the natural language syntactical if-then prepositional form.

Conjunctive rules follow the format:

IF condition 1 **AND** condition 2 **AND** condition 3
THEN *RESULT*

Here the RESULT can be of a binary value (TRUE/FALSE or YES/NO), a classification value (RED/WHITE/BLUE), or a real number value (0.18).

The condition can be either discrete (flower is RED, ORANGE or YELLOW) or continuous ($0.25 \leq$ diameter ≤ 0.6). The rule extraction algorithm will search through the structure of the network, and/or the contents of a network's training data, and narrow down values across each input looking for the antecedents (conditions) that make up the rules.

Subset rules, or M-of-N rules, follow the format:

IF (M of the following N antecedents are TRUE)
THEN *RESULT*

Cravin and Shavlik explain that the M-of-N rule format provides more concise rule sets in contrast to the potentially lengthy conjunctive rule format [9]. This can be especially true when a network uses several input parameters.

C. Rule Extraction Categories

Andrews [7] identifies three categories for rule extraction procedures: decompositional, pedagogical, and eclectic.

Decompositional rule extraction involves the extraction of rules from a network in a neuron-by-neuron series of steps [10]. This process can be tedious and result in large and complex descriptions. The drawbacks to decompositional extractions are time and computational limitations. The advantages of decompositional techniques are that they do seem to offer the prospect of generating a complete set of rules for the neural network.

Pedagogical rule extraction [11] is the extraction of a network description by treating the entire network as a black box. In this approach, inputs and outputs are matched to each other and the rule extraction algorithm is a machine-learner approach. The decompositional approaches can produce intermediary rules that are defined for internal connections of a network, possibly between the input layer and the first hidden layer. Pedagogical approaches do not result in these intermediary terms. Pedagogical approaches can be faster than the decompositional, but they are somewhat less likely to accurately capture all of the valid rules describing a network's contents.

The eclectic approach is merely the use of those techniques that incorporate some of a decompositional

approach with some of a pedagogical approach. The Rule-Extraction-As-Learning (REAL) method [9], for example, is designed such that it can use either technique.

II. RULES FOR V&V

For purposes of V&V, rule extraction is used to model the knowledge that the neural network has gained while training or adapting [2, 12]. The rules give insight into the workings of the neural network and may also be used to check against basic system requirements. The rules extracted are generally represented by a set of if-then statements that may be examined by a human. If the neural network is fixed after training, then the extracted rules model the way the neural network will handle other data that is processed with a confidence level based upon the rule extraction technique used. If the neural network is a real-time adaptive neural network, then rule extraction can be done for one point in time to establish what the system looks like at that instance. Repeated application of rule extraction will yield an understanding of the progression of the network during adaptation.

Rule extraction, rule initialization, and rule insertion can all be used for V&V purposes throughout the development life cycle, including the activities of Concept, Requirements, Design, Implementation, and Testing.

Requirements for adaptive systems are difficult to write because the system changes during training and operation and developers may be at a loss as to how to define the intended system knowledge. If a rule-like syntax is used to create knowledge requirements, then extracted rules from the neural network structure can be used for direct comparison against the requirements.

The extracted rules can also undergo design team review and analysis to detect improper network behaviors or missing knowledge. A system analyst might be able to ascertain novel learning behaviors that had not been previously recognized. By translating these features into comprehensible logical statements, the analyst can gain not only a better understanding of the network's internal knowledge, but perhaps of the input domain as well.

Rule extraction algorithm uses in several specific areas of V&V are discussed below. Specific examples showing rule extraction in V&V can be found in [10].

A. Symbolic Rules for Knowledge Requirements

As mentioned before, creating adaptive system requirements can be difficult. The initial requirements for an adaptive system may be intentionally incomplete (hence the need to include an adaptive system). The requirements should address the two aspects of the system shown in Fig. 1, namely control and knowledge.

The *control* type requirements are the type that would typically be generated for any software system. The *knowledge* type requirements are more difficult to write and must fully address the adaptive nature of the system and how it can evolve over time.

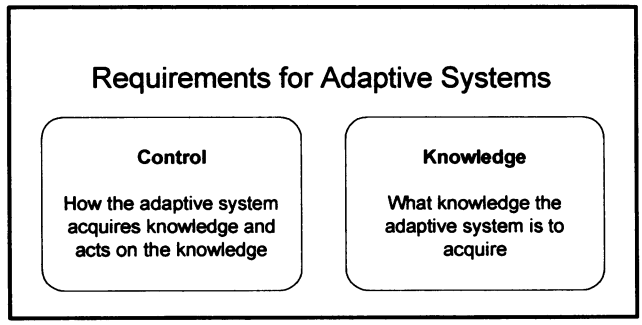

Fig. 1. Requirements for Adaptive Systems

Early in the process of developing adaptive system requirements, the adaptive behavior must be clearly defined and documented. Project documents at early stages should contain *high-level goals* for the adaptive system behavior and knowledge. These high-level goals should be stated in early documents, such as the Project Plan, and then be traceable through Systems Requirements, Software and Interface Requirements, and Design documents. The high-level goals are a representation of the problem to be solved, possibly in the form of informal descriptions.

From the high-level goals, the system level requirements can be developed. One approach to developing requirements related to knowledge in neural networks, and other adaptive systems, is to model, or describe the knowledge to be acquired in the form of rules [13]. Kurd's model is summarized in the next section. Experts, or those involved in development, can translate knowledge requirements into symbolic rules labeled as *initial knowledge*.

In Fig. 2, the top row shows the typical requirements developed for the system. The second row of boxes shows how this initial knowledge can lead to refined knowledge as the system is trained and then either to intermediate knowledge given to the adaptive system before deployment

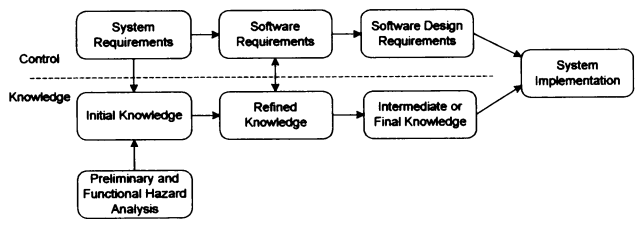

Fig. 2. Requirements for Adaptive Systems (Control and Knowledge)

or final knowledge if the system is fixed. These two rows represent the difference between the control and knowledge type requirements.

B. Rule Insertion for Hazard Mitigation

Kurd, Kelley, and Austin [13] describe a model that uses rule initialization, extraction, and insertion and ties hazard analysis into the development of the neural networks' knowledge and specifically addresses neural networks developed for safety-critical applications.

The following information is summarized from [13]. The authors describe a process for developing neural networks considering the safety criteria that must be enforced to justify the safety of the neural network in operation.

Fig. 3 illustrates the development and safety life cycle. The description of the three main levels and the major stages in the diagram are described below.

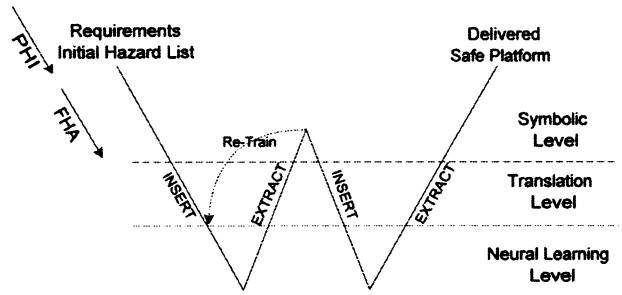

Fig. 3. The Safety Life Cycle for Artificial Neural Networks [13]

Symbolic Level: This level is associated with the symbolic information. At this level the gathering and processing of initial knowledge occurs, as well as evaluating extracted knowledge gathered post-learning.

Translation Level: This level is where the symbolic knowledge and the neural architectures are combined or separated through the use of rule insertion and rule extraction techniques.

Neural Learning Level: This level uses neural learning to refine the symbolic knowledge through the training of the neural network using learning algorithms.

The major stages that outline the process follow the "W" model in the development life cycle. These stages are

- Determination of requirements: These requirements describe the problem to be solved in informal terms and are intentionally incomplete.
- Sub-initial knowledge: Knowledge given by domain experts during a Preliminary Hazard Identification (PHI) is translated into logical rules.
- Initial knowledge: The rules that describe the sub-initial knowledge are converted into symbolic forms during Functional Hazard Analysis (FHA). These forms are compatible for translation into the neural network structure.
- Dynamic learning: Suitable learning algorithms and training sets are used to refine the initial symbolic knowledge and add new rules to reduce the error in the output. This may result in topological changes to the neural network.
- Refined symbolic knowledge: Knowledge refined by the learning is extracted using appropriate algorithms. This results in a new set of rules that can be analyzed.
- Static learning: Refined symbolic knowledge may be modified by domain experts and re-inserted into the neural network. This further refines the knowledge in the neural network but does not allow topological or architectural changes.
- Knowledge extraction: This can be performed at any time during static learning to get a modified rule set.

C. Rule Extraction for Traceability

Neural network rule insertion/extraction can facilitate requirements traceability of top-level system requirements, in the form of knowledge requirements, to software requirements in the form of more specific rule bases.

For the neural network knowledge the traceability from source code to design specifications is more difficult. As the neural network learns and adapts, the changes that occur in the internal parameters and structure act like source code modifications. However, implementation traceability can be achieved by running rule extraction algorithms from a trained neural network. The extracted rules can be compared directly against the design rules, or the extracted rules can be compared to higher-level software and system knowledge rules.

The refinement of requirements into software rules and design rules are presented here and shown in Fig. 4.

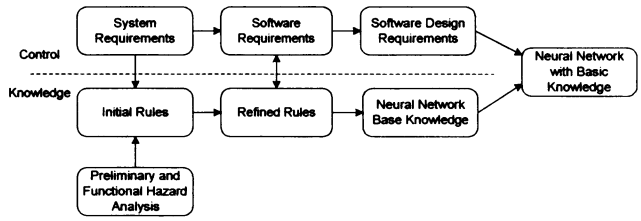

Fig. 4. Use of Rule Extraction to Develop System Knowledge

One approach to developing the functional requirement related to the neural network knowledge may be to model, or describe the knowledge to be acquired, in the form of rules. This would allow the neural network requirements to have a sufficient level of detail so they can be traced throughout the life cycle documentation. The following three-step process outlines how this can be achieved.

Step 1. Translate System Specifications dealing with the function, basic knowledge, and constraints of the neural network into initial symbolic information in the form of a rule base for the system. These rules can also include information from the PHI and the FHA to address what the neural network is not allowed to do or learn. These initial

rules should be translated into a format that can be inserted into the neural network (architecture specific).

Step 2. Refine initial rules and translate them into software requirements. The process of refining the rules may include prototyping the neural network, inserting the initial symbolic knowledge, training the neural network to allow the rules to adapt to the training sets, and extracting the rules.

Step 3. Translate refined rules into a format (neural network architecture specific) that can be inserted into the neural network.

For projects that lack well defined knowledge requirements, another method for performing traceability may be had by using machine learners on training data in conjunction with rule extraction techniques on trained neural networks. After the training sets have been developed to reflect requirements, the neural network is trained and a set of rules is extracted. Machine-learning techniques, like a decision tree, are then used on the same training set to produce a second set of rules. The neural network extracted rules can be compared against the machine learning results, enabling a kind of traceability between design intent (the machine learning rules) and implementation (the neural network rules.)

D. Rule Extraction for Testing

Rule extraction can be used in several ways to improve the testing V&V activities for trained neural networks, whether they be fixed prior to system usage or just given a basic set of knowledge before deployment and online adaptation. Proof of correct operation (and thus correct learning) can be evidenced through knowledge evaluation criteria achieved via extracted rules. These rules can be assessed with their own pass/fail criteria like the number of rules, complexity of the rules (e.g., number of antecedents), or rule size (e.g., graphical dimensions – too large/too small).

Extracted symbolic rules can direct test data generation and highlight specific areas within an input domain for testing. An example is the use of symbolic rules to act as a form of data generation. If the rules are of a mathematical equation form, randomly generating inputs for the antecedent part of the rule allows computation of a consequent. Consequents can be collected, combined with the randomly generated antecedents, and formed into test data sets.

System testing can be difficult due to a lack of well-specified knowledge requirements. If there are only few system requirements related to knowledge, the system test design should focus on paying special attention to boundary data. In this circumstance, extracted symbolic rules describing the neural network can guide the generation of data near the system knowledge boundaries.

If the neural network uses real-time V&V in the form of an operational monitor, the symbolic rules of the neural network can aid in testing the monitor. The symbolic rules expose the behavior of the neural network that in turn facilitates selection of input domain regimes to exercise the monitor.

Assertion testing is another area within testing that is possible once rules are extracted. An assertion is a statement describing a specific condition against the system. Assertions can be stated that represent system or software requirements and possible hazards.

Once an assertion is created, it can be checked against the extracted symbolic rules allowing domain experts the ability to analyze the rules to identify discrepancies from the intended design. Testers can use assertions to determine if certain scenarios like specific input combinations or possible outputs can occur given the neural network knowledge.

E. Rule Insertion/Refinement for System Stability and Fault Tolerance

Online adaptive neural networks continue to learn and change during system operation. In some cases, a neural network may be pre-trained prior to deployment and additional learning directs the network away from this original knowledge foundation. If the neural network is to be initialized with some *a priori* knowledge that represents the design specification, then this knowledge can be written in rule format and techniques used for rule insertion can be repeated during operation to steer the neural network back to this basic set of knowledge.

This method provides a sense of system stability because the network can be allowed to adapt to whatever the operational environment encounters but the network can also be reminded of important core knowledge.

Fault recovery could also be accomplished via symbolic rules like rule refinements, rule re-insertion, or some combination. If a system is somehow flagged as being erroneous, basic knowledge can be re-inserted back into the network to try to recover from the problem. Another option is the complete reloading of a neural network with knowledge that is deemed acceptable from a previous system state or from the bare knowledge given at system delivery and installation.

F. Rule Extraction for Operational V&V

A prevalent technique to evaluate correct system behavior is the use of operational monitors, sometimes referred to as run-time monitors. These separate software modules exist as oracles, continuously judging the neural network performance based upon some pre-defined criteria established by the system designers. These observations lead to a system specific decision-making process or control, especially if a problem is found.

One possible operational monitor design is the real-time extraction of rules from a neural network for comparison

against a set of acceptable rule conditions. Allowed conditions can be based on system requirements, hazard limits, or perhaps boundary conditions.

The feasibility of this approach is still under investigation. Rule extraction offline is well documented and easily accomplished. Rule extraction on a complex, continually adapting neural network at high computation speeds remains to be established.

III. CONCLUSIONS

Neural networks lack the ability to explain how they reached a specific output. This is one of the main reasons neural networks are not trusted. In most applications users want to know the reasoning behind the conclusion of the learning system or expert system.

Rule extraction algorithms provide a means for either partially or completely *decompiling* a trained neural network. This is seen as a promising vehicle for at least indirectly achieving the required goal of enabling a comparison to be made between the extracted rules and the software specifications.

One of the remaining issues with rule extraction is the lack of tools to assist with extracting and using rules. Naïve rule extraction could result in the creation of large sets of rules that are too unwieldy for human comprehension. Automated tools, like the one mentioned in [10] could reduce that problem.

Rule extraction from neural networks may have greater utility for fixed neural networks than for dynamic neural networks. Fixed neural networks proceed through the steps of training and testing until they reach an acceptable error threshold and only then are they used within a system. The knowledge of the domain is considered embedded inside the weights and connections of the network. If the network is no longer encouraged to adapt, the symbolic rules extracted to describe it can be a useful tool to validate that network at the time of extraction.

With a dynamic neural network, symbolic rule extraction may be required at intermediate stages in the learning. At some intermediate points symbolic rules would be extracted and passed through an oracle or system monitor to confirm that the network is still "correct." It may be that the maximum benefits for dynamic systems lie with rule insertion or rule initialization.

ACKNOWLEDGMENT

This research was sponsored by NASA Goddard Research Center through the NASA Independent Verification & Validation Facility, under Research Grant NAG5-12069.

REFERENCES

[1] Lisboa, P. 2001. Industrial use of safety-related artificial neural networks. Health and Safety Executive Contract Research Report 327.

[2] Taylor, Brian J., Marjorie Darrah, Laura Pullum, et. al. 2004. Methods and Procedures for the Independent Verification and Validation of Neural Networks. Technical Report prepared by Institute for Scientific Research, Inc. for NASA Independent Verification and Validation Facility under grant NAG5-12069.

[3] Pecheur, Charles. 2000.Verification and Validation of Autonomy Software at NASA. NASA/TM 2000-209602.

[4] Rodrigues, Pedro, José Félix Costa, and Hava T. Siegelmann. 2001. Verifying Properties of Neural Networks. IWANN (1) 2001: 158-165.

[5] Pecheur, Charles and Stacy Nelson. 2002. V&V of Advanced Systems at NASA.

[6] Tickle, Alan B.; R. Andrews; M. Golea; and J. Diederich. 1998. The truth is in there: directions and challenges in extracting rules from trained artificial neural networks.

[7] Andrews, Robert; J. Diederich; and A. B. Tickle. 1995. A Survey and Critique Of Techniques For Extracting Rules From Trained Artificial Neural Networks. Knowledge Based Systems 8:373-389.

[8] Tan, A. 1997. Cascade ARTMAP: Integrating Neural Computation and Symbolic Knowledge Processing. IEEE Transactions on Neural Networks, 8 (2) 237-250.

[9] Craven, Mark; and J.W. Shavlik. 1994. Using Sampling and Queries to Extract Rules from Trained Neural Networks. In Proceedings of the 11th International Conference on Machine Learning 37-45.

[10] Darrah, Marjorie, Brian Taylor and Michael Webb. A Geometric Rule Extraction Approach used for Verification and Validation of a Safety Critical Application. 2005 Florida Artificial Intelligence Research Society Conference, Clear Water Beach, FL, May 16-18, 2005.

[11] Nayak R., R. Hayward and J. Diederich. 1997. "Connectionist Knowledge Base Representation by Generic Rules from Trained Feedforward Neural Networks", Proceeding of Connectionist Systems for Knowledge Representation and Deduction Workshop, Townsville, Australia, July 13-19.

[12] Taylor, B. J., Marjorie Darrah, Spiro Skias. 2004. Weaving it All Together - A Methodology for the Verification and Validation of Adaptive Neural Networks. NIPS-2004 Workshop on Verification, Validation, and Testing of Learning Systems. Whistler, British Columbia, December 2004.

[13] Kurd, Zeshan, and Tim Kelley. 2003. Safety Lifecycle for Developing Safety Critical Artificial Neural Networks. In Proceedings of 22nd International Conference on Computer Safety, Reliability, and Security (SAFECOMP'03) 23-26 September 2003.