

## Chapter 2

# AUGMENTATION OF CURRENT VERIFICATION AND VALIDATION PRACTICES

Kareem Ammar, Laura Pullum, Brian J. Taylor  
*Institute for Scientific Research, Inc.*

### 1. INTRODUCTION

Many agencies, including the NASA IV&V Facility use the IEEE Standard for Software Verification and Validation (IEEE 1012-1998) [IEEE 1998] as a basis for verification and validation (V&V) activities. Standards like IEEE 1012-1998 are sufficient for traditional software development, but are inadequate for adaptive systems that can change their implementation and behavior over time. Faults may manifest themselves because of autonomous changes and this may introduce problems that were not present during design or testing.

Given the fault handling and other capabilities of neural networks, their use in the control systems of advanced avionics, aeronautics and other safety- and mission-critical systems is seriously being pursued. Along with the capabilities come challenges in the verification and validation (V&V) of these systems, as well as the need for V&V practitioner guidance. Neural networks that continue to learn while the system is in operation require additional steps for proper software assurance.

This chapter highlights research that was performed to determine the gaps in the traditional standards and guidelines for performing V&V when applying them to adaptive neural network systems. Previous work in the area of guidance for the V&V of neural network systems consisted primarily of efforts by NASA Ames Research Center and NASA Dryden Flight Research Center [Mackall 2002, 2003]. The NASA reports provide valuable inputs into what should be done to verify and validate adaptive aerospace systems. The authors of these reports align their guidance with the ISO/IEC

12207 [IEEE/EIA 1998], a standard that addresses the implementation of general software lifecycle processes, and offer guidance specific to adaptive aerospace systems.

Based on the research described in this chapter, the Institute for Scientific Research, Inc. (ISR) has developed a comprehensive guidance document [ISR 2005] aligned with the IEEE 1012-1998 to assist the V&V practitioner in evaluating general neural network systems. The goal of the guidance document is to provide relevant and applicable guidance for the V&V practitioner when faced with an adaptive system with a neural network component. This chapter will discuss the approach to the research, gaps found in the V&V standards when faced with a neural network system to V&V, and several augmentations to V&V processes to satisfy these gaps. Specifically, V&V methods are described in this chapter that address adaptive system requirements, neural network software requirements, neural network design, enhanced configuration management, modified lifecycle models, and neural network testing.

## **1.1 Combining Previous Research and Standards**

In order to identify what needed to be done to augment current practices to accommodate the V&V of neural networks, current standards and practices were examined. The documents examined during the research for this chapter included: IEEE 1012-1998, the IEEE ISO/IEC 12207, the NASA Ames and Dryden reports, and a sample software verification and validation plan (SVVP). (Note that before the release of this book the IEEE 1012 – 2004 became available. The guidance document produced from this research remains applicable to the majority of this updated standard.)

The first step in the research approach was to create a mapping between the ISO/IEC 12207 to IEEE 1012-1998. This bi-directional mapping allowed for gap analysis to be preformed with all available information.

The guidance developed by NASA Ames and NASA Dryden, though fairly top-level, provided an initial input into this effort. The NASA reports were aligned with ISO/IEC 12207 standard and so the mapping of ISO/IEC 12207 to IEEE 1012-1998 helped to map the Ames and Dryden guidance to IEEE 1012-1998. Also, in this step the sample Software Verification and Validation Plan was examined. The SVVP of the Airborne Research Test System II (ARTS II) of the Intelligent Flight Control System (IFCS) [Casdorph 2000, ISR 2003] provided insights into one implementation of a V&V plan for a neural network system. The IFCS was an excellent case study as it utilized safety- and mission-critical online adaptive neural networks. This text contains several examples from the IFCS Program to illustrate augmented V&V methods.

These mappings were then combined to determine the coverage of the V&V of neural networks problem space and to form a framework for the general V&V guidance document. The guidance document is the final product of the ISR's research initiative. The steps of this research plan, including intermediate and final results, are illustrated in Fig. 2-1.

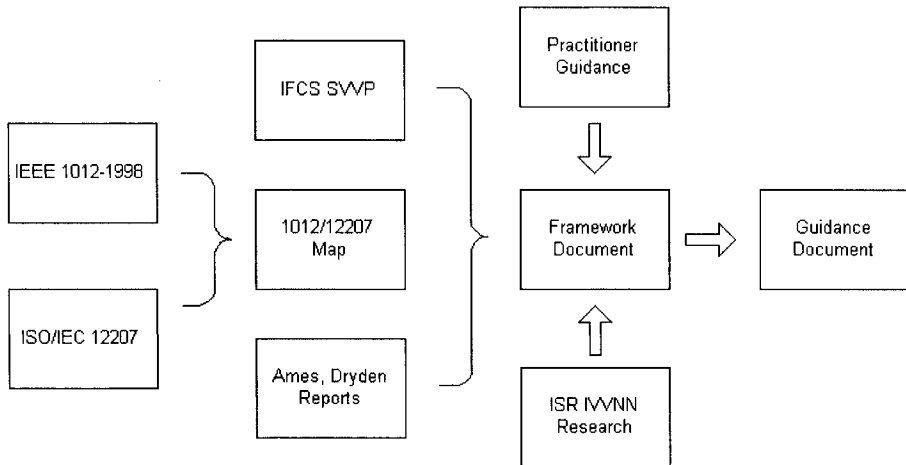


Figure 2-1. Research Plan – Interim and Final Results

## 1.2 Guidance for the Independent Verification and Validation of Neural Networks

The guidance document, *Guidance for the Independent Verification and Validation of Neural Networks* [ISR 2005], outlines general augmentations to lifecycle processes to assist the V&V practitioner. The guidance document provides a detailed listing of all IEEE 1012-1998 V&V tasking and provides augmentations for the V&V neural networks for every task. The guidance document also introduces novel V&V tasks when augmentations to IEEE 1012-1998 tasks are not sufficient. Finally, the guidance provides supporting examples from IFCS. In summary, the guidance document provides assistance to the V&V practitioner in designing V&V methodologies for neural networks.

The guidance document is generally aligned with the IEEE 1012-1998 in terms of section enumeration. The alignment provides the V&V practitioner the ability to quickly reference information on specific areas of interest. The document is organized into five sections. Section 1 is the overview of the document. Section 2 contains the important references for the document. Section 3 contains definitions, abbreviations, and conventions. Section 4

provides a summarized list of areas of consideration for neural network/adaptive systems. Section 5 provides detailed guidance for all lifecycle processes, activities and tasks related neural network and adaptive systems.

For a few tasks, no significant the guidance was needed. For many tasks, additional guidance is required to address the adaptive system. In some instances the guidance document provided additional V&V tasks that were not found in IEEE 1012-1998. These include Neural Network Design Evaluation, Adaptive System and Operational Monitoring Performance Assessment, Periodic Test Execution and Verification, Proposed Knowledge Change Assessment, and Configuration File Change Assessment.

## **2. AUGMENTATIONS TO CURRENT V&V STANDARDS FOR ADAPTIVE SYSTEMS**

The remainder of this chapter addresses several augmentations to the V&V processes. Discussions on specific techniques can be found in later chapters.

As a note, V&V processes are tailored to specific applications through the selection of software integrity levels to plan for appropriate V&V tasking. The additional methods outlined in this chapter are not intended to replace existing processes, but are designed to provide suggestions to the V&V practitioner to augment an established V&V process to address specific characteristics of neural networks. Use of the following techniques should be based upon assigned software integrity levels, and thus the safety- and mission-criticality level of the system. This section does not try to identify at what level these techniques should be employed. V&V practitioners should also be aware that many of these techniques mentioned here still maturing, and the cost associated with developing the technique to a technology readiness level usable by the project could highly influence how quickly the project might adopt the technique.

### **2.1 Enhanced Configuration Management for Neural Networks**

Typically, configuration management identifies configuration items, implements version control upon baselined configuration items, and describes processes to establish new baselines. A configuration item is usually software, documentation, or data. In IEEE 1012-1998, configuration management assessment of a project is conducted by determining the

completeness and adequacy of documentation produced that describe the configuration management process. Since adaptive systems have an added component of knowledge that is not present in conventional software, the configuration management processes must describe the methods neural networks use to obtain knowledge as well as the neural network knowledge itself.

The neural network design is often an iterative process. The process cycles through the steps of design-train-test multiple times. The neural network undergoes changes as it attempts to learn what the system designers intend. If this process is not tracked via configuration management, the project may lose the ability to repeat the design. Neural network validation techniques such as cross validation or bootstrapping should also be tracked through configuration management.

Neural network developers can also choose to modify the structure of the neural network. Based upon the evaluation of neural network performance, a designer might want to change the architecture, add additional neurons and neuron layers to the architecture, or change the algorithms the network uses to learn or grow. Alterations in structure, function, and architecture must be captured through configuration management.

Pre-trained neural networks rely on training sets to learn. After using a set of training data, the network begins to adjust internal parameters as it migrates to learning the features of the training data. The V&V practitioner should ensure that these training sets are tracked through configuration management. It is equally important to track the order and frequency in which training sets are applied to a neural network. Capturing the neural network during training is also useful. The partial learning in the neural networks allows for an evaluation of how the network is changing, and allows for developers to revert to an earlier state if the current neural network is lacking in performance.

The considerations outlined for pre-trained neural networks can be applied to online adaptive neural networks. Online adaptive neural networks are usually initialized with parameters that control variants such as learning rate and initial weight values. This configuration data should be under stringent version control, since slight alterations in parameters may considerably alter the way the neural network learns and performs.

## **2.2 Adaptive System Requirements V&V**

The V&V practitioner evaluates the system requirements early in the lifecycle. The system requirements are validated to ensure they can be satisfied by the defined technologies, methods and algorithms defined for the

project, and the system requirements verified for their consistency to the needs of the user.

The feasibility of the requirements determines if the defined technologies can fulfill the requirements adequately and efficiently. In order to evaluate if any specifications requiring the use of an adaptive system can meet the overall system requirements, the practitioner must first evaluate whether an adaptive system or a neural network is appropriate. There are numerous types of neural networks. Neural networks may be either supervised or unsupervised. They may have feed-forward architecture or feedback architecture. Different neural networks are applied to different problem domains. While no complete taxonomy exists that defines appropriate neural network architecture selection, there are general guidelines. These guidelines are described in Chapter 4.

To establish consistency with the needs of the recipient of the system, the needs must be clearly documented. For an adaptive system, the user needs should be represented as goals depicting system behavior and characteristics from a very high level of abstraction. The high-level goals are first stated in early documents and are then traceable through systems requirements and software requirements. To ensure complete coverage of user needs, system and software requirements are traced to high-level goals.

High-level goals of an adaptive can be difficult to write very early in the lifecycle. They should address two aspects of the system:

- How the adaptive system acquires knowledge and acts on that knowledge. (Control)
- What knowledge the adaptive system should acquire. (Knowledge)

Table 2-1 provides an example of high-level goals for an online adaptive neural network used in an intelligent flight control application. The table also shows traceability to system requirements. From high-level goals the developer will be able to understand the nature of the system. Once high-level goals are established for a project, consistency of the system requirements to the user needs can then be easily established.

## **2.3 Neural Network Software Requirements V&V**

Neural networks are based on statistics and mathematics. Requirements describing neural networks should be written in formal mathematical notation to describe the functionality of the neural network and intended knowledge. The V&V practitioner will need to make sure software requirements are readable and it may be necessary to provide appendices to

software requirement documentation that includes a brief mathematical background explaining some of the specifications.

Table 2-1. Example of Adaptive System High-Level Goals

High-Level Goal	Classification	Software Requirement ID	Software Requirement Description
The IFCS system shall include a safety monitor that safely limits the online learning neural network commands	Control	1.1.1.1[01]	The system shall[01] include a safety monitor that safely limits the online neural network commands and outputs or reverts to the conventional (non-research) flight control system in the case of a command that could damage the vehicle, precipitate departure from controlled flight, or injure or incapacitate the pilot.
		1.1.1.1[02]	The safety monitor shall[02] limit the authority of the experimental control laws to ensure that the control commands remain within the allowed loads limits of the NASA vehicle.
The IFCS system shall use an online learning neural network to adjust the feedback errors to achieve desired system behavior in the presence of off nominal behavior or failures.	Knowledge	2.2.2.2[01]	The system shall[01] use an online learning neural network to adjust the feedback errors to achieve desired system behavior in the presence of off nominal behavior or failures.
		2.2.2.2[02]	When the neural network is off, the adjustment to the feedback shall[02] be zero, and the adaptation shall be disabled.

The notion of *control* requirements and *knowledge* requirements discussed within Section 2.2 is used in the next two section to further explore neural network requirement analysis.

2.3.1 Neural Network Control Requirements

The following paragraphs outline some of the more common areas that the V&V practitioner may address when assessing completeness of the software control requirements specifications for an adaptive system or a neural network.

The convergence time and precision of a neural network should be outlined in the software requirements. Convergence refers to a global minimization of the neural network error. The developers should be able to prove that the adaptive system can converge within a specified amount of time. This is a necessity for safety-critical systems that adapt in real time.

Many neural networks increase the amount of memory use when running in real-time. The increase in memory use may be attributed to an increase in the number of nodes that account for new operational conditions or an increase in connections or associations between nodes. The software requirements should specify the precise conditions under which a neural network will be allowed to grow and constraints on the growth such as maximum memory size.

In a safety- or mission-critical system, developers should use operational monitors or run-time monitors to periodically check the conformance of the adaptive system to the requirements. If an operational monitor is used, then high-level descriptions of operational monitors must be included in the requirements specifications. Different types of operational monitoring approaches are discussed in Chapter 10.

Input and output requirements for neural networks are crucial since input data has the greatest influence on adaptation. Requirements that describe input scaling, gains, and limits are common for neural networks. The V&V practitioner should also verify the input and output data and the amount of acceptable error.

The learning algorithm used within the neural network should also be defined with the requirements. A learning algorithm is used to adapt the neural network based on an update law. The V&V practitioner should verify the description of the learning algorithm. Examples of learning algorithms include calculating the mean squared error between the desired output and the actual outputs, Newton methods, and Levenberg-Marquardt [Bishop 1996].

### **2.3.2 Neural Network Knowledge Requirements**

With an adaptive neural network system the states of neural network are unknown prior to operation. One of the motivations for employing an adaptive control system is the versatility to react to unknown operating conditions. Requirements related to what the neural network must and must not know are important in determining valid operation of the system. However, specifics of neural network knowledge are difficult to establish before the input domain is clearly understood and a preliminary neural network is constructed. As a result, software requirements should depict general knowledge of an adaptive neural network.



A neural network is initialized before it begins to learn upon input data. This method should be captured in the software requirements. Neural networks may be initialized from a data file, software module, or another software system. Purely adaptive online neural networks are initialized by randomizing the weights, or by setting the weight values to some constant.

For fixed neural networks or neural networks with knowledge prior to deployment, the requirements should specify the type of training data used. The requirements should depict the source of the training data, data ranges and attributes, and metrics of expected performance of the neural network upon the training data.

Software requirements specifications should impose limitations on neural network knowledge. These requirements should set specific knowledge limits or describe undesirable behavior. Software requirements may describe what the adaptive system shall not do as well as what the adaptive system must do.

A method to developing requirements depicting neural network knowledge is to describe the knowledge in the form of rules. Initially, domain experts can compose symbolic knowledge from the target problem domain. Neural network developers may then transform this knowledge into rules that is used in rule extraction and rule insertion techniques. Information attained from these processes can provide a sufficient level of detail for neural network knowledge requirements. A more detailed explanation of rule extraction is provided in Chapter 8.

## **2.4 Neural Network Design V&V**

After the neural network is designed to some desirable state, the V&V practitioner must evaluate the design based on the criteria of correctness, consistency, completeness, accuracy, readability, and testability. Although these criteria may seem identical to traditional software design evaluation outlined by IEEE 1012-1998, the methods for evaluating a neural network system vary greatly from the methods used for traditional software. Consequently, a new task needs to be considered to describe methods to V&V the neural network design.

In evaluating the neural network design, the V&V practitioner must focus on areas such as neural network structure, neural network performance criteria, training and testing data, training processes, and operational monitors. Each of these areas should be described in sufficient detail in design documentation. Because of the extra significance given to the neural network design, it is recommended that the project team develop a special appendix to the software design document for the neural network. Another option is the creation of a stand-alone neural network design document.

Many people of varied backgrounds participate in a project, so the documented design should contain enough information to convey the purpose, rationale, mathematics, and development of the neural network that anyone not familiar with neural networks in general would be able to understand it. Essential information would include a summary of all terms used for the specific neural network, as well as visualization techniques to aid understand such as diagrams or flowcharts.

Many of the ideas discussed in this section should appear in the documentation. There should be sections describing the functions that comprise the neural network, how the neural network is to learn and grow, and specific design details for input pre-processing, output post-processing.

Based upon the system design, there may be other considerations for inclusion into the design. Some systems may require that the neural network perform data recording for later analysis. Others might make use of multiple neural networks in a form of N-version programming for neural networks. All of these specifics should be present in the documentation.

Neural networks have several design elements that must be evaluated for correctness. These include the neural network structure that is composed of the specific neural network architecture and the internal neural network parameters. Design documentation should include a clear description of the neural network nodes, the connection matrix between the nodes, weights, activation functions, growth functions, learning functions, hidden layers, inputs, and outputs.

While some of these elements can number into the hundreds and thousands, each individual element need not be documented. Instead, the focus should be on descriptive qualities like the number of nodes, the number of connections per node, what mathematical functions occur for each node, etc.

Neural network architectures such as multilayer perceptron, self-organizing map, radial basis function, recurrent, and Hopfield are used in very different problem domains. V&V must validate that the selection processes for the neural network architecture is based upon solid theoretical or empirical evidence. The selection process may also be based upon comparison studies, recommendation from experts in the field, or past experiences within a similar problem domain. V&V should ensure that the reasons are clearly expressed within concept documentation.

Neural network designs must have sufficient acceptance and rejection criteria well documented. System testers need to have metrics to determine the acceptability of the system. These criteria should describe specific knowledge, stability, and performance constraints. Other possible performance criteria may include necessary operating frequencies (i.e., the

neural network must produce an output at a rate of 40Hz), and acceptable variance qualities of the output signal.

V&V must evaluate the accuracy of the neural network design by ensuring that the training set conforms to system accuracy requirements and physical laws and that the knowledge acquisition has adequate precision for system requirements. Training and testing data need to be considered as design elements. Analysis on this data should address if this data is appropriate for the intended usage of the system and if it is precise enough for expected results. The origins of the training and testing data should be evaluated to remove any possible problems from corrupt data collection, inappropriate sources of data collection, and problems associated with combining previous separate data sets into a single larger set.

The training process itself must undergo evaluation for correctness. The training process should be clearly documented and describe why it is appropriate for meeting the project needs. This includes identification of relevant training data sets, justification for the choice of the data sets, and consideration of the correct data formats for the training data. If multiple data sets are used, and certain data sets are emphasized during training more than others, this should also show up in the documentation with justification. A description of configuration management as used on the training process should also be present. Information here can include the training data configuration items that were use, the procedures employed for applying the training data configuration items, and identification and tracking of evaluation metrics used throughout the process.

If the system makes use of operational monitors (see Chapter 10), their design needs to be a part of the documentation. The project can decide to include these within the neural network documents, to make the operational monitor design a separate document, or to include it within the overall software design document. Operational monitor design can influence the neural network design and vice versa, the neural network design can influence the operational monitor design. Because of this, the design documentation needs to contain detailed information on the performance and interface of any operational monitors in order to minimize problems during integration with the neural network.

## **2.5 Modified Lifecycle for Developing Neural Networks**

The neural network development lifecycle is different from traditional lifecycle models. It does not follow the basic waterfall methodology nor does it follow a pure spiral lifecycle model. Instead neural network development utilizes parts of both methodologies. The V&V practitioner should understand the details of the neural network lifecycle model used and

ensure conformance of development activities to the model. The following sections discuss three distinct models for neural network development.

### **2.5.1 Common Neural Network Development Model**

A common method for development of a neural network is an iterative cycle that is performed until the neural network has been proven adequate by some quantifiable measure. The stages in this process are the design, training, and testing of the neural network.

During the design stage, the neural network developer chooses the architecture, and the initial number of neurons and layers. The developer then establishes a connection matrix between all neurons, selects the learning algorithm and possible growing algorithms, and determines the initial values for internal parameters including weights and constants controlling the algorithms. Subsequent passes through the design stage may involve major overhauls of the design or may simply fine-tune neural network parameters.

During the training stage, training data is used by the neural network to learn. Depending on the nature of the problem, the neural network may be designed to approximate a function describing the training data, or may learn relationships between input and output data within the training set for classification. Training sets can be significant in size with several thousand training examples. After each example, the learning algorithm continues to adjust the network structure. The goal of the training stage is that after training the neural network to some state, the internal neural network parameters are developed enough to satisfy designated requirements and objectives

After completing a pass through the training data, a separate set of data called the testing data is used. This set of data has similar properties to the training data, but with examples the neural network has never seen. By using new data, the performance of the network can be measured without influence of examples that were used for training. Results from testing may indicate that another pass through the design-train-test cycle is necessary. Usually, the neural network developers have some target metric value they wish to achieve such as a 95% classification accuracy rate on new data. Until the metric values are met, the design-train-test cycle is iterated.

### **2.5.2 Rodvold's Neural Network Development Model**

Rodvold [1999] identified that many current neural network development processes tend to be developed through empirical processes rather than through precise construction and training methods. To rectify this problem,

Rodvold constructed the nested loop model for neural network development. The model was constructed from common neural network development processes, and contains elements from both the waterfall lifecycle development model and the spiral lifecycle development model. Rodvold's model, shown in Fig. 2-2, is composed of five steps.

Step 1: Develop a set of neural network requirements, goals, and constraints into a document labeled Network Performance Specification.

Step 2: Assemble the data that will be used for training the neural network including data sources, original format of data, and modifications performed on the data. This step results in the Data Analysis Document.

Step 3: Training and testing loops are an iterative process in which the neural network architecture is developed and trained. The first loop, Variations of Artificial Neural Network (ANN) Topologies, involves changing the neural network architectural parameters. The middle loop, Variations of ANN Paradigms, involves modifications to the type of neural network used. The outer loop, Selection and Combination of ANN Input Neurons, concerns altering neural network inputs. All design and training from this step is documented in the Network Training Summary.

Step 4: Network deployment is completed through commercial tools, automatic code generation provided by commercial tools, or by saving raw neural network data to file with code to load and execute the neural network. The deployment of the neural network is documented in the Network Integration Document.

Step 5: Independent testing and verification produces the Network Test Plan and the Data Analysis Document. This step also involves creating the Network Test Report, which summarizes of all tests performed.

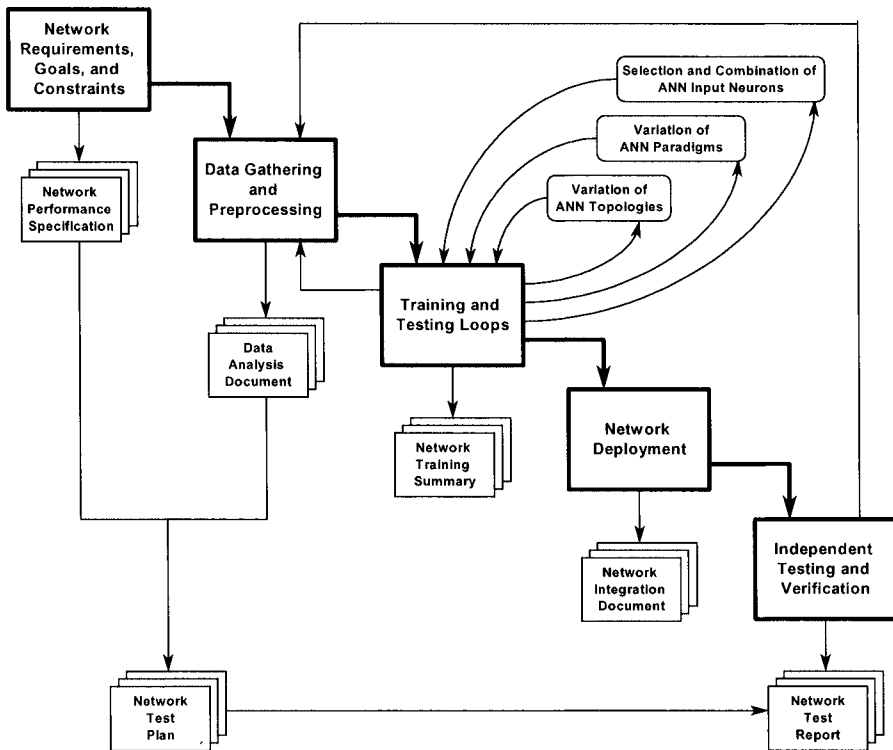


Figure 2-2. Critical Neural Network Development Process Model [Rodvold 2001]

### 2.5.3 Kurd's Neural Network Development Model

A major problem for the V&V practitioner, when faced with a safety-critical neural network, is the inability to effectively perform white-box analysis. The Safety Lifecycle [Kurd 2003] is a process for developing neural networks considering the safety criteria that must be enforced to justify safety operation of a neural network. This model ties hazard analysis into the development of the neural network's knowledge and specifically addresses neural networks developed for safety-critical applications. Fig. 2-3 illustrates the development and safety lifecycles. There are three levels in the diagram:

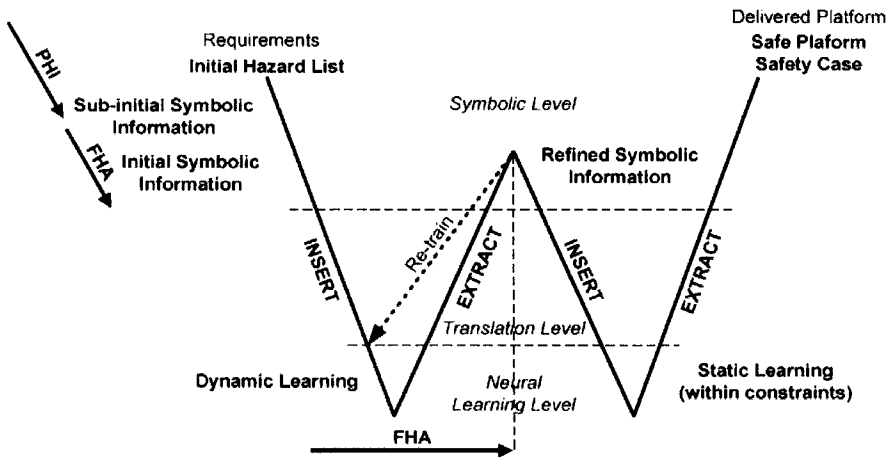


Figure 2-3. Safety Lifecycle for Hybrid Neural Networks [Kurd 2003]

The Symbolic Level is associated with the symbolic knowledge representations. The Translation Level is where the symbolic knowledge is modified into knowledge that is represented through rules for rule extraction and insertion. The Neural Learning Level trains the neural network through empirical data formulated from knowledge from the translation level.

Brief descriptions of the stages in the development model which traverse the above levels are as follows:

- Determine neural network requirements and goals,
- Collect symbolic sub-initial knowledge from domain experts,
- Compose initial knowledge for the neural network from sub-initial knowledge,
- Train the neural network on empirical data produced from the initial knowledge,
- Extract knowledge from neural network and refine this knowledge, and
- Train the neural network with a new refined knowledge set.

## 2.6 Neural Network Testing

Testing of an online learning neural network is difficult. Formal method testing techniques such as model checking and theorem proving are not practical given the characteristics of many neural networks. Theorem proving is too difficult in complex problem domains. For online adaptive neural networks, model checking would be inadequate since the neural network is non-deterministic at run-time. Instead the system tester is left to

augmenting normal testing practices as well as incorporating some new ones.

Like traditional software testing, the component or unit-level testing of a neural network system will focus upon determining the correct implementation of the system, but from a function or module perspective. This is true regardless of the neural network being adaptive or fixed.

Since a neural network is a statistical tool, the functions will be of a mathematical nature. Testing will need to ensure that these functions return values for all possible inputs and that the values returned are consistent with their mathematical nature. A project may make use of approximations or table lookups to implement these functions. Testing should concentrate on how accurate these functions are and how errors in the implementation can ripple through other software modules and the possible system level effects from this.

Interface testing, another aspect to testing that exists for non-adaptive software, can require more concern for adaptive software. Neural networks are very capable of performing an approximation to a solution given partially noisy and partially imprecise data. When analyzing the inputs and outputs, practitioners should inspect the pre- and post-processing of the inputs looking for poor performance or improper implementations. An example is a smoothing function that isn't smoothing the data as it was intended. Data interpolation algorithms may also contribute to generating poor data that is masked by the neural network. The system testers may know that the neural network is performing with a very minor error, but if an error in the pre-processing is corrected, the system performance could improve.

The robustness of the neural network may work as a disadvantage to the system tester. If allowed to adapt during system usage, neural networks can overcome incorrectly provided inputs. This can happen if one person or team develops a module that produces input for the neural network and another person or team develops the neural network module. If the data is switched between the module and the neural network, and the neural network adapts, it can compensate for this deficiency, possibly with only a minor error that the team regards as negligible.

Two new aspects of testing that are normally not considered are the neural network knowledge and structure. Structure testing determines whether the design is the most optimal at learning what is intended, compared to other designs. Knowledge testing would investigate what the neural network has already learned and how well it has learned it.

Typically, when the neural network undergoes design-train-test development, the network is tested after it passes through an iteration of training. There is typically a performance metric, usually an error function,



associated with how well the neural network handles the data found in the test set. Over time, this error metric should decrease towards zero representing the neural network has learned the exact function it is to perform.

System testers may want to make sure these iterative testing stages are tracked and the error values recorded. By looking at these error values over time, it can provide evidence that the neural network design is improving and is better than any of the previous other states the design was in. The tester may not actually be responsible for conducting this kind of test, they just make sure the results are collected and analyzed.

Since structure testing is testing if the current design is better than previous designs, the results do not need to prove that the current design is the best possible design achievable. It only needs to prove that the design is better than it was previously. While it would be very useful to know that the current design is the absolute best, trying to show this through testing would be impractical.

Knowledge testing is perhaps the hardest area for neural network analysis because it involves evaluating whether the neural network is learning what the designers want it to learn. This goes beyond the use of a performance metric like the error function mentioned before. An error metric only tests for one particular input set; it does not reflect how the neural network would perform for a larger set of inputs.

Testing knowledge may require the use of rule extraction techniques to translate the network knowledge into a set of symbolic rules. The rules present a more tangible representation; they can be individually tested, used within existing testing tools, and can facilitate other analysis methods. Further discussion of this approach can be found in Chapter 8.

Another method for testing knowledge, though limited and often times impractical, is to use a brute force approach that tests the neural network for a very large set of possible inputs. By looking at the performance over a greater range, some sense of how the neural network will perform for an input domain can be gathered. This kind of testing may also fall into reliability assessment or sensitivity analysis where minute differences within the input domain are studied to determine if the neural network could behave erroneously. The tester may need to rely upon test data generation techniques to assist with this kind of evaluation. Consult Chapter 9 for further details.

### 3. SUMMARY

This chapter has revealed several gaps in current V&V practices for adaptive systems, and has shown augmentations to traditional V&V processes to satisfy these gaps. Through the mapping of software V&V standards, process guidance, previous research, and IFCS documentation, a guidance document was formulated to augment current V&V practices to accommodate the characteristics of neural networks or adaptive systems. The guidance document [ISR 2005] embodies a comprehensive methodology for the V&V of neural networks and gives the V&V practitioner direction on how to augment established V&V processes to address neural networks or adaptive components within a system.

### REFERENCES

- Bishop, C. M. 1996. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, England.
- Casdorph, Van; Brian Taylor; et al. 2000. Software Verification and Validation Plan for the Airborne Research Test System II, Intelligent Flight Control Program. Institute for Scientific Research, Inc. IFC-SVVP-F001-UNCLASS-120100, Dec. 1.
- IEEE Std 1012-1998. 1998. *IEEE Standard for Software Verification and Validation*. New York, NY.
- IEEE/EIA 12207.2-1997. 1998. *IEEE/EIA Guide, Industry Implementation of International Standard ISO/IEC 12207:1995, (ISO/IEC 12207) Standard for Information Technology – Software Lifecycle Processes - Implementation Considerations*. New York, NY.
- Institute for Scientific Research, Inc. (ISR). 2001. DCS Design Report for the Intelligent Flight Control Program. IFC-DCSR-D003-UNCLASS-010401.
- Institute for Scientific Research, Inc. (ISR). 2003. Software Verification and Validation Plan for the Airborne Research Test System II of the Intelligent Flight Control Program. IFC-SVVP-F002-UNCLASS-050903.
- Institute for Scientific Research, Inc. (ISR). 2005. Guidance for the Independent Verification and Validation of Neural Networks. IVVNN-GUIDANCE-D001-UNCLASS-072505.
- 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.
- Mackall, Dale; Stacy Nelson; and Johann Schumman. 2002. Verification and Validation of Neural Networks for Aerospace Systems. NASA Dryden Flight Research Center and NASA Ames Research Center. June 12.
- Mackall, Dale; Brian Taylor; et al. 2003. Verification and Validation of Adaptive Neural Networks for Aerospace Systems, Version 1.2 (Draft without Appendices). NASA Dryden Flight Research Center and NASA Ames Research Center. March 31.
- Pullum, Laura L. 2003. Draft Guidance for the Independent Verification and Validation of Neural Networks. Institute for Scientific Research, Inc. IVVNN-GUIDE-D001-UNCLASS-101603. October.

- Rodvold, D.M. 1999. A Software Development Process Model for Artificial Neural Networks in Critical Applications. *In Proceedings of the 1999 International Conference on Neural Networks (IJCNN'99)*. Washington D.C.
- Rodvold, DM. 2001. Validation and Regulation of Medical Neural Networks. *Molecular Urology* 5(4): 141-145.

Methods and Procedures for the Verification and  
Validation of Artificial Neural Networks

Taylor, B.J. (Ed.)

2006, XII, 278 p., Hardcover

ISBN: 978-0-387-28288-6