On Verification & Validation of Neural Network Based Controllers

Johann Schumann†, Pramod Gupta‡ and Stacy Nelson⋆
† RIACS / NASA Ames, schumann@email.arc.nasa.gov
‡ QSS / NASA Ames, pgupta@email.arc.nasa.gov
⋆ Nelson Consulting Company, NelsonConsult@aol.com

Abstract. Artificial neural networks (ANNs) are used as an alternative to traditional models in the realm of control. Unfortunately, ANN models rarely provide any indication of accuracy or reliability of their predictions. Before ANNs can be used in safety critical applications (aircraft, nuclear plants, etc.), a certification process must be established for ANN based controllers. Traditional approaches to validation of neural networks are mostly based on empirical evaluation through simulation and/or experimental testing. For on-line trained ANNs used in safety critical applications, traditional methods of verification and validation cannot be applied, leaving a wide technological gap, which we attempt to address in this paper. We will describe a layered approach for ANN V&V which includes a V&V software process for pre-trained neural networks, a detailed discussion of numerical issues, and techniques for dynamically measuring and monitoring the confidence of the ANN output.

1. Introduction. During the recent past, the interest in research and application of universal model-free controllers has increased substantially. Considerable research has been done to use artificial neural networks for such controllers. A number of successful application areas in the realm of NASA has been identified: adaptive control of various types of manned and unmanned aircraft or space craft, space shuttle docking, or neural network based sensor failure detection, just to mention a few. In all cases, superior control behavior and high adaptability could be obtained together with drastically reduced development and operation costs. Practical feasibility of a neural network based controller for aircraft is currently being investigated at NASA for a manned F-15 Active aircraft based at Dryden Flight Research Center, as well as for a C-17 transport aircraft. Here, on-line trained neural networks within the flight controller can adapt to sudden damage of a control surface, a situation likely to cause fatal accidents.

The actual use of neural networks in safety critical areas is still severely limited. A major reason for that is that each piece of software in a safety critical application needs to undergo a software development process which calls out for extensive and rigorous *verification and validation* (V&V) of the software. Usually, a specific software *certification* procedure has to ensure that under all circumstances, the system works reliably and without failures. For example, all safety critical software for U.S. commercial avionics has to be certified by the FAA, in accordance with the RTCA DO-178B standard.

The aim of this paper is to develop requirements, which could then be adapted to design a safety critical ANN certification process appropriate to a particular standard. In this paper, we describe work done in the IFCS/V&V[1] project which develops methodology and software process for the V&V of neural network based aircraft controllers. Here, we focus on two different kinds of control architectures: one uses a pre-trained neural network (PTNN) to store a model of the plant, whereas in the other, the neural network is trained on-line (i.e., the weights are changed during the flight) to handle dynamic adaptation. A PTNN is a purely deterministic mapping, and it may be analyzed just as any other function. Hence, its verification should not be more difficult than of other well-accepted implementation of non-linear mappings, such as polynomials or look up tables. Yet, due to misconceptions surrounding neural networks, they have been treated with significant suspicion, and regarded as unsafe. One of the objectives of this paper is to clarify this issue and discuss methods to verify the performance of a network. In this paper, we present a software process guide which has been developed within the IFCS project and which has been specifically designed to aid V&V of PTNNs. In addition to the points mentioned above, we will also discuss

[1]IFCS (Intelligent Flight Control System) is a joint project between NASA DFRC, NASA Ames, ISR, and Boeing.

issues of off-line training (selection of training data, over-training, etc.) as well as numerical issues (e.g., scaling, condition numbers).

The main criticism against inclusion of on-line trained neural networks in safety critical applications has been their non-determinism. In on-line training, it is indeed difficult to predict, in a deterministic sense, the future connection weights and thus, the form of the neural network mapping. In addition, it was argued that it is difficult to assess the true performance of a network due to their time-variant characteristics. These are obviously valid concerns which need to be addressed in cases involving on-line adaptation and real-life training data. In this paper we describe the highlights of a process guide which is being developed and discuss advanced V&V techniques and tools which are important for handling the on-line adaptive case. We will illustrate the concepts with a simplified flight-control architecture, described in Section 2 below.

2. Application: Intelligent Flight Control. The IFCS is a flight control system, developed at NASA, which can efficiently identify aircraft stability and control characteristics using neural networks and utilize this information to optimize aircraft performance in both normal and (simulated) failure conditions. It aims at taking advantage of nonlinear mapping capabilities of neural networks to develop on-line a mapping of key parameters of the aircraft dynamics. The IFCS is tested on a highly modified configuration of a standard F-15 jet which includes canard control surfaces. In test flights, the canards are used to dynamically change the airflow over the wing, thus simulating a wing damage.

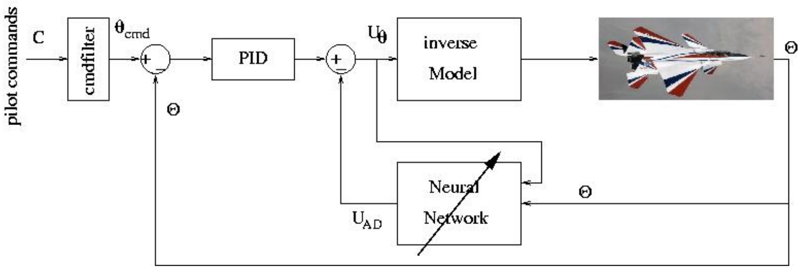

Figure 1: Overview of a simplified IFCS control architecture

Fig. 1 shows a block diagram of a simplified version of the controller [7,13] for one of the three aircraft channels (roll, pitch, and yaw). This controller is based upon a dynamic inverse model. The commands C issued by the pilot are pre-filtered (θ_{cmd}) and, in combination with the aircraft sensor readings (Θ), a desired command output U_θ is calculated. This value is then fed into the inverse model of the aircraft (realized using a linear mapping or a pre-trained neural net). This inverse model then provides the actual command values which are fed to the aircraft's control surfaces. In the nominal and idealized case, the inverse model is an exact inverse function of the aircraft dynamics. Due to modeling inaccuracies or changes in the aircraft dynamics, there are deviations. In this case, the neural network kicks in: it tries to learn from the sensor readings Θ and the previous command output U_θ how to minimize the deviation. Thus, the neural network's output U_{AD} is used to adjust the value of U_θ. The ANN is trained on-line. Within the project, a number of different neural network architectures (e.g., $\Sigma\Pi$, multi-layer perceptron) and training algorithms have been evaluated (e.g., as described in [13]).

3. V&V of Neural Networks. The main difference between ANNs and conventional implementations of a control system is that ANNs are trained by examples, taken from a set of training data representing the dynamics of the underlying system. For a conventional system to be certified, the specification is required to be unambiguous and, within reasonable bounds, complete. The conventional specification describes the behavior of the system for every circumstance, within which it is to operate. Ensuring that a system meets its specifications, is a central goal of conventional certification procedures. If this approach is to be preserved for ANN certification, the nature of the specification must be radically altered. A major pre-requisite to developing high-quality software is to use an appropriate *Software process* which covers all stages from initial requirements to deployment and which is specifically tailored toward adaptive ANN based systems (see Section 3.1). Based on the properties of the development process for an ANNs, we can specify requirements for a certification standard: 1) how are the goals or requirements for the ANN to be obtained, 2) what should be done to ensure that the training data adequately represent the system, 3) what type of networks can be used (i.e., number of nodes, number of layers, connectivity etc.), 4) what details the ANN developer must provide regarding the way in which the ANN module interfaces with the rest of the system, and 5) what methods are to be used for quality assurance in the trained network?

When it comes to actually perform V&V on a ANN-based system, the primary focus of the analysis is laid on the neural network specific tasks of training, learnability, convergence, and generalization. Quite often, however, an important aspect is overlooked: a neural networks is in essence a numerical method to approximate a function; its training method is usually an algorithm to solve an unconstrained quadratic optimization problem (minimization of the error $E = \sum_i (o_i - t_i)^2$ where o_i is the actual ANN output and t_i the desired output). Therefore, a process for V&V of neural networks also needs to investigate the numerical issues underlying the ANN and its implementation (see Section 4).

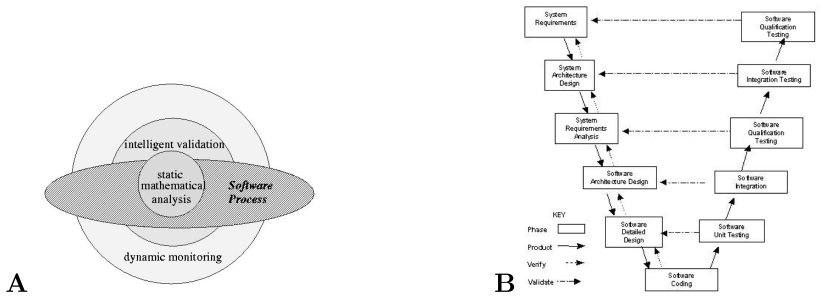
intelligent validation
static mathematical analysis
Software Process
dynamic monitoring
A
System Requirements
Software Qualification Testing
System Architecture Design
Software Integration Testing
System Requirements Analysis
Software Qualification Testing
Software Architecture Design
Software Integration
KEY
Phase
Product
Verify
Validate
Software Detailed Design
Software Unit Testing
Software Coding
B

Figure 2: (A) Layers of ANN V&V methods (B) Lifecycle.

To address the problems discussed above, we propose a multi-layered approach to V&V techniques and methods for ANN based systems (Figure 2A). The core-layer contains rigorous, mathematically sound results concerning robustness, stability, and convergence. Current state-of-the-art, however, only can provide relatively weak results, often in form of asymptotic guarantees. Typical examples here include Lyapunov proofs of (asymptotic) stability, or Vapnik-Cherenovis-dimension arguments to reason about the ANN's generalization abilities [12].

However, for our safety-critical applications, these results don't suffice, as, for example, they cannot provide convergence guarantees in the required short period of time (usually on the order of a few seconds). Thus, methods on the second layer need to address these issues by *intelligent testing*. Here, techniques to reduce the number of required test cases in the nominal and pre-analyzed failure case can be applied. Analysis of numerical issues, as well as tests for convergence and robustness of the system belongs to that layer. For truly adaptive systems, however, we still don't have an a-priori guarantee for performance. Here, the third layer comes into play: methods in this layer will *dynamically monitor* the ANN and its behavior. Although it ultimately cannot provide the guarantee, it at least can return dynamic information on how the ANN currently behaves and if the current state of the system is recoverable at all.

In the following, we will present the highlights of a software process which has been developed within the IFCS project [8]. Then we will discuss V&V issues for on-line adaptive ANN. In Section 4, we will discuss a number of important numerical and statistical aspects in detail.

3.1. A PTNN V&V Process. In order for a neural network to fly on spacecraft or aircraft, it must be certified. In this context, certification is the process of obtaining a certificate from an authority (typically the NASA and/or the Federal Aviation Authority (FAA)) to indicate conformance with airborne software standards. While very strict, these standards can be tailored for new software like ANN. During the IFCS project, a process guidebook [8] was written describing processes and methods specific to V&V of PTNN and work is underway on another guidebook for V&V of on-line adaptive ANN. These process guidebooks contain guidance on the required documents and activities (e.g., which kind of tests have to be carried out, etc.) required to meet the standards. In the following we summarize how to integrate V&V of PTNN into a typical software life cycle. First, a well-defined process with discrete Software Life Cycle phases (Fig. 2B) including documented work products (called deliverables) must be defined for the overall project. This V&V process must include analysis procedures established to ensure correctness of deliverables and scheduled reviews of major product releases. V&V of neural networks need to be applied to all phases of the Software Life Cycle. Thus all phases need to augmented and adapted specifically, most notably the following phases:

- *Systems Requirements*, *System Architectural Design*, and *Software Requirements Analysis*: Documentation for these phases must include the specification for the ANN and its architecture. This description needs to contain the type of ANN (feed-forward, Self Organizing Map, etc), the learning algorithm (gradient descent, Levenberg-Marquardt, Newton's method, etc.), and all parameters of the ANN architecture (e.g., number of layers and number of nodes, initialization of weights). Furthermore, a concise description of the inputs and outputs (including units and the expected and maximal range of values) and acceptable errors for PTNN must be provided. These design and requirements descriptions must be provided in detail, but not necessarily in a formal language. Software models (e.g., Simulink) are also acceptable; however all parameters which might be hidden in the simulator must be made explicit.
- *Software Detailed Design* must include a description of precise code constructs required to implement the ANN, including all data structures and algorithms (e.g., libraries for matrix operations).
- *Software Integration Testing* should verify that the architectural design is detailed enough so, when implemented, the ANN can interface with system hardware and software in various fidelity testbeds.
- *Software Qualification Testing* should verify that the system requirements are sufficient enough to ensure that, when implemented, the ANN will interface properly with the system in production.

3.2. V&V of on-line Adaptive Neural Networks. On-line adaptive systems in general, and learning neural networks in particular cannot be validated using traditional verification and validation techniques, because they evolve over time and past learning data influences their behavior. While they hold great technological promise, on-line learning systems pose serious problems in terms of verification and validation. Usually, methods for software product verification are generally classified into three categories: fault avoidance, fault removal, and fault tolerance [1]. Unfortunately, neither of these three methods is applicable as-is to adaptive systems, for the following reasons:

- *Fault Avoidance*: Formal design methods are based on the premise that we can determine the functional properties of a system by the way we design it and implement it. While this holds for traditional systems, it does not hold for adaptive systems, since their design determines how they learn, but not what they will learn.

- *Fault Removal*: Formal verification methods are all based on the premise that we can infer functional properties of a software product from an analysis of its source text. While this holds for traditional systems, it does not hold for adaptive systems, whose behavior is also determined by their learning theory. All testing techniques are based on the premise that the systems of interest will duplicate under field usage the behavior that they have exhibited under test. While this is true for traditional deterministic systems, it is untrue for adaptive systems, since the behavior of these systems evolves over time.
- *Fault Tolerance*: Fault tolerance techniques are based on the premise that we have clear expectations about the functions of programs and programs parts, and use these expectations to design error detection and error recovery capabilities. With adaptive systems, it is not possible to formulate such expectations because the functions of programs/program parts evolve.

4. Numerical and Statistical Aspects of ANN V&V. During V&V of an ANN-based system, the primary focus of the analysis is often laid on the neural network specific tasks of training, learnability, convergence, and generalization, often neglecting two important aspects: (a) a neural network is, in essence, a numerical algorithm to approximate a function that is trained using a non-linear optimization algorithm; (b) inputs to the neural network are usually subject to noise, thus a statistical analysis of the network and the probability density of the outputs can yield valuable information. In the following, we will discuss several of these issues in detail.

4.1. Numerical Issues

4.1.1. Choice of Input-Output Parameters. The preliminary step toward developing a neural network model is the choice/identification of the inputs and outputs of the problem to be modeled so that the corresponding training data can be generated. In general, the model outputs are determined based on the purpose of the model. Other factors influencing the choice of outputs are ease of data generation, ease of incorporation of the neural network model into the overall system and so forth.

4.1.2. Data Generation. Once the input(s)-output(s) have been identified, the next step in neural network development is the generation and collection of data for training the neural network. As the performance of the trained network ultimately depends on the quality of the data with which it was trained, great care must be taken in obtaining the data. Initially a neural network does not have any information about the problem; the neural network has to be trained with the corresponding data. So it is important that the training data sufficiently represent the original problem. Training data must be provided to characterize the component to be modeled over a desired range of operation and for different combinations of geometrical and physical model inputs. The data must be collected with enough samples around stationary points and near curvature while a relatively sparse sampling of points in flat regions is sufficient. The developer must properly document the collection of the data and must investigate the coverage of the input space achieved by the training data.

4.1.3. Scaling. Data scaling (i.e., multiplication by a constant) is an essential step to improve accuracy and learning of a neural network. There are two major reasons for this: First, neural networks are limited to the range that their output can attain. For example, the common feed forward network with logistic activation function is theoretically limited to an output range between 0 and 1. Secondly, scaling gives equal importance to each input, and prevents extremely large or small weights, leading to low numerical accuracy. A typical example for unfavorable ranges of data is when one input of the ANN represents the current altitude in feet (range: $0 \ldots 50{,}000$ ft) and the other a small angle, e.g., $\alpha \in \{0, \ldots, 0.09\text{rad}\}$. Without scaling, the ANN will almost never learn the dependency upon α. Obviously, scaling can be performed on the input, output or both. Not so obvious, however, is the fact that scaling can affect the behavior of *some* training algorithms. For example, the well-known gradient descent algorithm is influenced by scaling, whereas a Newton method is not (for a detailed discussion see [2,6]).

4.1.4. Sensitivity and Conditioning. In order to assess the ANN's behavior with respect to changes in the input values or the weights, *sensitivity analysis* can be performed. For a sensitivity analysis of the neural network during training, it is often helpful to consider, how small changes in the *weights* affect the error function (and thus how fast and good the network can be trained).

The quadratic error function E can be—in the neighborhood of the (local) minimum $\mathbf{x}_0$—be represented by $\Phi(\mathbf{x}) = \mathbf{g}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T \boldsymbol{G}\mathbf{x}$ where $\mathbf{g} \in \mathbb{R}^n$ is the gradient $\nabla E|_{\mathbf{x}_0}$, and $\boldsymbol{G}$ is the Hessian matrix of E. When we plot Φ in the vicinity of the minimum, we obtain surfaces shaped like a bowl or an elongated valley; the contour lines are ellipses. The behavior of Φ (and thus also of E) in the vicinity of $\mathbf{x}_0$ is determined by the eigensystem of $\mathbf{G}$ The ratio of the largest and the smallest eigenvalue of $\mathbf{G}$ is defined as the condition number $\mathcal{C}$. $\mathcal{C}$ also determines the eccentricity of the ellipses: a small condition number corresponds to an almost circular error surface; a large $\mathcal{C}$ results in a steep and long "valley" which can lead to serious problems during training. Thus, the condition number of the problem is an important metric for the behavior of the ANN with respect to training and convergence. Thus, all tests which are performed during ANN V&V should, in addition to demonstrating convergence, calculate and assess the condition number.

4.2. Statistical Estimation of ANN Reliability. As discussed earlier, on-line learning neural networks cannot be validated using traditional V&V methods. Because these systems are developing over time and have to adapt toward potentially unknown situations, V&V must guarantee that under no circumstance output values of the neural network can cause damage to the process and maneuver the aircraft into an unrecoverable state. Unfortunately, the ANN itself does not provide any indication of accuracy or reliability of its predictions. We therefore develop a dynamical monitor which checks all input and output values of the neural network and determines if the result of the neural network is reliable. This dynamical monitor is based upon a statistical model of the system. During the operation of the system (i.e., during the flight), we calculate a confidence measure for the outputs of the ANN. This confidence measure is related to a novelty measure and allows to dynamically check the neural network's behavior: nominal conditions (or conditions seen before) result in a high confidence value. In extreme cases of the parameter space (e.g., due to damage of control surfaces) error margins need to be much larger, resulting in lower confidence. In that case, the behavior of the neural network will deviate from the ideal one (which might yield some roughness in handling). With on-line learning, the neural network tries to adapt to the modified behavior, thus increasing confidence in the output again.

Figure 3 shows the a graphical representation of the dynamic monitor. In each graph, the output of the ANN over time is shown as a solid line. At time $t = 1.5s$, the pilot issues a command. In the nominal case (A), the neural network produces some output to accommodate for modeling inaccuracies. During that time of adaptation, the confidence decreases. In our figure, we show the variance (dashed lines), which is proportional to the inverse of the confidence. Within this band, the actual (true) value is located with a probability of 95%. Thus, a broad band corresponds to a low confidence value. Figure 3B and C depict scenarios when a failure occurs (a control surface got stuck at different angles at $t = 1.5s$). In these cases, the ANN's output needs to be substantially larger to counteract the damage. Also, the width of the error band is much larger, indicating a low confidence in the results of the network. The damage in Figure 3C was found to be already very close to the stability limit; the large waves in the output and the very low confidence is a clear indication of that. This situation also illustrates that our confidence measure can be used as an early-warning indicator.

Our monitoring approach uses Bayesian techniques. It could be shown [9,3,10] that Bayesian techniques can be applied to assign a confidence interval to the predicted output of a trained network. The confidence value basically can be obtained if we do not only consider the output value of the ANN, but rather its probability distribution, based upon the posterior distribution of the weights [2]. We are adapting and substantially extending techniques which allow to efficiently calculate these distributions for on-line adaptive networks, given the network and a statistical

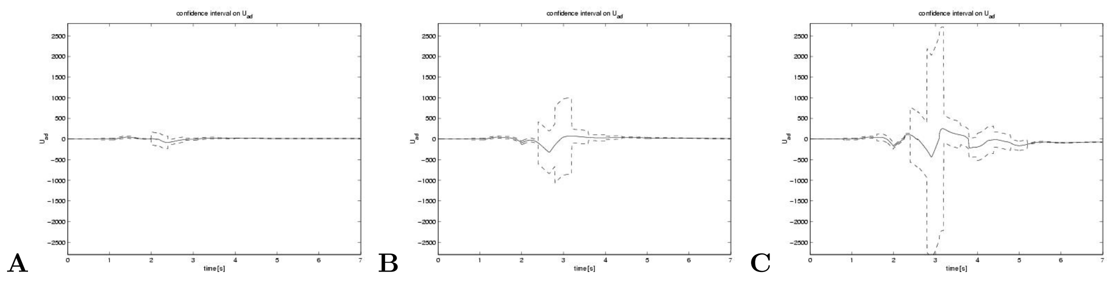

Figure 3: Output of the on-line adaptive ANN (solid); dashed lines define confidence interval

model of the data.

5. Related Work. Verification and certification of neural nets has attracted the attention of only a few researchers in the recent past. [9] proposes a Bayesian approach for the estimation of parameters in neural nets. The parameters in the neural network are considered as being drawn from a statistical distribution which is characterized by a set of hyperparameters (e.g., mean and variance). The network predictions are then simply a summary (typically of mean or mode) of the posterior distribution of the parameters. The Bayesian approach provides a natural framework for estimating prediction intervals. [5] discusses verification and validation of neural nets, where he interprets verification to refer to correctness and validation to refer to accuracy and efficiency. He establishes correctness by analyzing the process of designing the neural net, rather than the functional properties of the final product. [14] presents a comparison of nonlinear regression and Bayesian estimation methods of calculation of prediction interval for neural networks. [11] presents a comparison of three methods (ML,Bayesian and Bootstrap methods) on artificial and real data. In their study, the authors considered both data noise and model uncertainty. [15,4] treat neural nets as a special case of adaptive flight controllers. Those papers discuss the very generic problem and therefore give only top-level analysis of the conditions to be satisfied by the verification process, but do not offer any specifics on how exactly the process should be carried out. In [16], verification of a neural net that approximates a look-up table defined over a hyper-rectangular domain is addressed. It was shown how guaranteed deterministic performance bounds can be derived for such a net. That method is based on the assumption that for any input vector a reference output value can be easily evaluated using a look-up table.

6. Conclusions. In this paper, we have reported ongoing work on the development of techniques and a software process for the verification and validation of neural networks based controllers. We are confident that our layered approach, together with a strong and detailed software process is helpful to address this topic. Although this process has to deviate in important aspects from one for standard software, it is expected that V&V of a system with a pre-trained neural network can be performed with reasonable effort. However, for a control architecture with on-line adaptation additional aspects need to be addressed, most notably convergence and reliability of the neural network's output. In this paper, we focussed our discussion on the last item. We have developed an algorithm which can monitor the neural network's behavior dynamically and calculate a confidence interval, indicating how reliable the prediction of the ANN is.

These techniques, however, comprise only a first step. Classical techniques and techniques discussed so far only address the individual components (neural network and the controller), but not the entire system. It will be necessary to perform research on combining the different properties of the different components to obtain guarantees for the entire system.

The ultimate goal to provide guarantees for all unexpected cases can probably never be reached. So, for example, there always exists a point where the physical system is damaged/changed to such an extend that adaptation toward controllable behavior is simply not possible. In such cases, any attempt at adaptation would be futile. However, an adaptive system in a safety-critical application needs to quickly recognize this fact and should issue a warning. We are currently investigating on using the confidence measure as provided by our dynamic monitoring tool for such a purpose.

With a combination of these techniques which are to be applied during a traditional V&V-phase and also during deployment, more and more guarantees about the ANN's behavior can be given, thus facilitating the application of adaptive systems in safety-critical domains.

References

[1] A. Avizienis. The n-version approach to fault tolerant software. *IEEE Trans. on Software Engineering*, 12(11), 1985.

[2] C. M. Bishop. *Neural Networks for Pattern Recognition.* Clarendon-Press, Oxford, 1995.

[3] W. Buntine and A. S. Weigend. Bayesian back-propagation. *Complex Systems*, 5:603–643, 1991.

[4] V. Cortellessa et al. Certifying adaptive flight control software. In *Proc. of the 2nd International Software Assurance Certification Conference (ISACC2000)*, 2000.

[5] L. Fu. *Neural Networks in Computer Intelligence.* McGraw Hill, 1994.

[6] P. Gill, W. Murray, and M. Wright. *Practical Optimization.* Academic Press, 1981.

[7] C. Jorgensen. Direct adaptive aircraft control using neural networks. Technical Report TM-47136, NASA, 1997.

[8] D. Mackall, S. Nelson, and J. Schumann. Verification and validation of neural networks of aerospace applications. Technical Report CR-211409, NASA, 2002.

[9] D. MacKay. Bayesian interpolation. *Neural Computation*, 3(4):415–447, 1992a.

[10] R. M. Neal. *Bayesian Learning for Neural Networks.* PhD thesis, University of Toronto, Canada, 1994.

[11] G. Papadopoulos, P. Edwards, and A. Murray. Confidence estimation methods for neural networks: A practical comparison. In *Proceedings of the European Symposium on Artificial Neural Networks*, pages 75–80, 2000.

[12] R. Reed and R. Marks. *Neural Smithing.* MIT Press, 1999.

[13] R. Rysdyk and A. Calise. Fault tolerant flight control via adaptive neural network augmentation. *AIAA American Institute of Aeronautics and Astronautics*, AIAA-98-4483:1722–1728, 1998.

[14] L. H. Ungar, R. D. D. Veaux, and E. Rosengarten. Estimating prediction intervals for artificial neural networks. In *Proc. of the 9th Yale Workshop on Adaptive and Learning Systems*, 1996.

[15] W. Wen, J. Callahan, and M. Napolitano. Towards developing verifiable neural network controllers. In *Proc. of the Workshop on AI for Aeronautics and Space* , 1996.

[16] R. R. Zakrzewski. Verification of a trained neural network accuracy. In *Proc. of the International Joint Conf. on Neural Networks*, 1657–1662, 2001.