Monitoring the Performance of a neuro-adaptive Controller

Johann Schumann[*] and Pramod Gupta[†]

[*]*RIACS / NASA Ames Research Center*
[†]*QSS / NASA Ames Research Center*

Abstract. We present a tool to estimate the performance of the neural network in a neural network based adaptive controller. Using a Bayesian approach, this tool supports verification and validation of the adaptive controller as well as on-line monitoring. In this paper, we discuss our approach and present simulation results using the adaptive controller developed for NASA's IFCS (Intelligent Flight Control System) project.

INTRODUCTION

Traditional control has proven to be ineffective to deal with catastrophic changes or slow degradation of complex, highly nonlinear systems like aircraft or spacecraft, and in areas such as robotics or flexible manufacturing systems. Adaptive control systems, which can adapt toward changes in the plant have been proposed as they offer many advantages (e.g., better performance, controllability of aircraft despite of a damaged wing). In the last few years, use of neural networks in adaptive controllers (neuro-adaptive control) has been studied actively [1, 2, 3, 4]. Neural networks of various architectures have been used successfully for on-line learning adaptive controllers. In one of the control architecture, the neural network receives as an input the current deviation between desired and actual plant behavior and, by on-line training, tries to minimize this discrepancy (e.g., by producing a control augmentation signal) [3].

Even though neuro-adaptive controllers offer many advantages, they have not been used in mission- or safety-critical applications, because performance and safety guarantees cannot be provided at development time—a major prerequisite for certification (e.g., by the FAA or NASA). Because of the requirement to adapt toward unforeseen changes during operation, design-time verification & validation (V&V) is not sufficient. Moreover, there are no established ways to verify and validate adaptive control systems due to (1) non-deterministic nature of controllers and (2) lack of established methods to determine basic adaptive control systems stability and performance. V&V of an adaptive controller requires the development of new analysis techniques which can demonstrate that the control system behaves safely under all operating conditions.

One important aspect of V&V is to analyze the *performance* of the system during design time and testing, and to monitor it after deployment. Since the neural network comprises the core adaptive element in a neuro-adaptive controller, it is important to obtain information about the learning and convergence behavior of the network. For example, in situations where the network is subjected to novel data, or the network has been over trained, the output of the neural network can be far from the desired value, possibly leading to uncontrollability of the system. Thus, a performance metric on the network training and generalization behavior is highly valuable. We have developed a tool ("Confidence Tool") which dynamically estimates the network performance (confidence intervals), based upon the current inputs $\mathbf{x}$ and the training history D. The confidence measure is the variance of the neural network output o, which is obtained by calculating the probability density $p(o|\mathbf{x}, D)$ using a Bayesian approach [5]. We have extended this approach to handle specific neural network architectures (e.g., single hidden layer, or Sigma-Pi networks [6]) and to accommodate the dynamic weight update. This tool allows real-time assessment of controller performance together with handling qualities of the aircraft's performance.

In this paper, we discuss this approach and present the tool, which is used in a simulation environment for controller design, tuning, and V&V, and is currently being implemented as a monitoring tool on the flight computer of a manned F-15 aircraft within NASA's IFCS project.

In the rest of this paper, we give a short background on the IFCS adaptive controller and our approach to verification and validation (V&V) of neural networks. Then we briefly explain the mathematical derivation for our confidence tool and discuss some experimental/simulation results.

BACKGROUND: THE IFCS ADAPTIVE CONTROLLER

We will illustrate our approach with an adaptive flight control system (FCS) which has been developed within the IFCS project at NASA. The target aircraft for this controller is a specifically configured F15 jet aircraft. It has additional actuator surfaces, so-called canards, that are located in front of the wings. By moving them, the airflow over the wing can be modified in a wide range. Thus, this aircraft can be used to simulate failures like wing-damage during test flights. The FCS (Fig. 1) is a straight-forward dynamic inverse controller: the pilot stick and pedal commands are mixed with the current sensor readings (airspeed, angle of attack, and altitude) to form the desired behavior of the aircraft (measured as roll-rate, pitch-rate, and yaw-rate). The dynamic inverse model then calculates the required actuator movements (e.g., of aileron or rudder) to bring the aircraft into the desired state.

If the aerodynamics of the aircraft changes (e.g., due to a broken surface), there is a deviation between desired and actual state. The neural network is trained during operation to produce a correction signal U_{AD} to minimize this deviation. The inputs of the neural network are typically the current state of the aircraft (i.e., the sensor signals), the commanded input, and the correction signal of the previous time frame.

The controller (IFCS Gen-II, [3]) uses a Sigma-Pi neural network [6]. In this network (Figure 2), the inputs are subjected to basis functions (e.g., square, tanh, scaling). Then products (Π) of these function values are calculated. The final output of the network is a weighted sum (Σ) of these products—hence the name of this architecture.

The network is trained according to a given update rule [3]. The weight update rule is derived from a Lyapunov stability analysis of the entire controller. In this paper, we

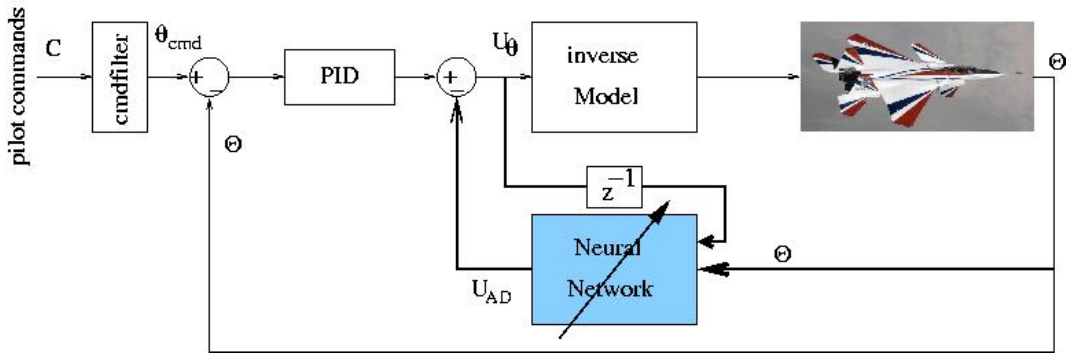

FIGURE 1. IFCS Adaptive Control Architecture

will not discuss this weight update rule (see [3] for details), because our approach is independent of the learning rule. In fact, our algorithm can easily be adapted for other network types.

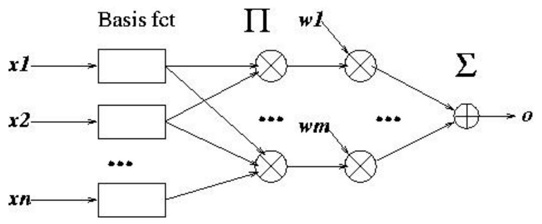

FIGURE 2. Architecture of ΣΠ network

APPROACHES TO NN VERIFICATION AND VALIDATION

Ensuring that a system meets its specifications, is a central goal of verification, validation, and certification procedures. For a system to be verified and certified, a unique set of specifications is required, describing the behavior of the system for every circumstance, within which it has to operate. If this approach is to be preserved for NN certification, the nature of the specification must be radically altered to accommodate changes of the underlying system. Instead of the explicit specification of normal operational scenarios and specific failure cases, possibly many (unforeseen) changes in the plant or in the environment must be addressed. Due to the special nature of the adaptive controller (and the involved numerical optimization routines), the *software process*, which covers all stages from initial requirements to deployment must be specifically tailored.

Based on the properties of the development process for NNs, we can specify requirements for a certification standard: 1) how are the goals or requirements for the NN 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 NN developer must provide regarding the way in which the NN module interfaces with the rest of the system, 5) what methods are to be used for quality assurance in the trained network, and 6) how does the on-line training behave during operation?

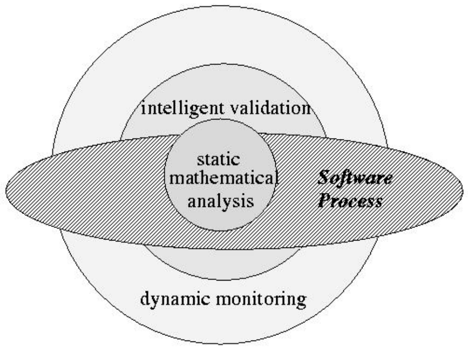

FIGURE 3. Layers of NN V&V methods

To address the problems discussed above, we use a multi-layered approach [7, 8] to V&V techniques and methods for NN based systems (Figure 3). 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-Chervonenkis-dimension arguments to reason about the NN's generalization abilities [9].

However, for safety-critical applications, much stronger guarantees are required, as, for example, about convergence within a 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 can be applied to reduce the number of required test cases in the nominal mode and pre-analyzed failure cases. This optimization can substantially reduce overall development costs, since the execution of test cases (in high-fidelity simulators or in test-flights) are a major cost driver.

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 NN and its behavior. Although it ultimately cannot provide the guarantee, it at least can return dynamic information on how the NN currently behaves and if the current state of the system is recoverable at all. This information then can be used by other systems, like intelligent vehicle health monitoring or recovery systems, to accordingly react in cases where the network's output does not have an acceptable quality.

All approaches and techniques on the different layers must be applied in a coordinated manner. To this end, we have developed a software verification and validation process guide [10] which extends standardized software processes toward specific requirements for neural-networks based controllers.

THE CONFIDENCE TOOL

Because of the adaptive nature of a neuro-adaptive controller, a dynamic monitoring of the performance of the neural network is important, to be able to detect critical situations.

Any feed-back controller can always handle small deviations between the model and the actual plant dynamics. However, this robustness is strictly limited by the design of the controller. In cases of (larger) deviations it is therefore necessary to know the current quality of the system's model in order to obtain a probability under which the controller exceeds it robustness limits. As mentioned before, this dynamic measure[1] does not prevent an uncontrollable situation, but it can provide information about the current safety margins. Therefore, our Confidence Tool calculates a measure for the network performance at each point in time. As discussed above, the variance σ^2 of the network output is used, assuming a Gaussian distribution. Thus, we do not consider the neural network as a function approximator which returns a number, but consider the probability distribution $p(o|\mathbf{x},D)$ when the network is getting a (noisy) input $\mathbf{x}$ and has been trained using training data D. o is the network output. Marginalizing over all possible weights $\mathbf{w}$ of the network, this probability can be calculated as:

$$p(o|\mathbf{x},D) = \int p(o|\mathbf{x},\mathbf{w})\,p(\mathbf{w}|D)\,d\mathbf{w} \tag{1}$$

We calculate the variance σ^2 of o following the derivation in [5]. The first term in the integral concerns the "query" phase of the trained network, when it is subjected to input $\mathbf{x}$. The second term $p(\mathbf{w}|D)$ describes, how the weights $\mathbf{w}$ of the network are influenced by training it using training data D. It can be calculated as a posterior using Bayes' rule, considering the distribution of the weights before and after the training data D has been seen by the network.

$$p(\mathbf{w}|D) = \frac{p(D|\mathbf{w})\,p(\mathbf{w})}{p(D)}$$

We use a simple Gaussian prior for $p(\mathbf{w})$, since we assume that the weights before training are Gaussian distributed. The probability of the data $p(D) = \int p(D|\mathbf{w})p(\mathbf{w})d\mathbf{w}$ is a normalization factor. For the further derivation (see [5] for details) two hyper-parameters α and β are introduced to obtain

$$p(\mathbf{w}|D) \propto exp\{-(\beta E_D + \alpha E_W)\}$$

where E_D is the (quadratic) training error and E_W the sum of the squares of the weights. A quadratic approximation of the exponent around the most probable weights and substitution into (1) finally yields (for details see [5])

$$\sigma_t{}^2 = \frac{1}{\beta} + \nabla_w^T \mathbf{A}^{-1} \nabla_w \tag{2}$$

where ∇_w is the gradient of the network output with respect to the weights at the current input $\mathbf{x}$, and $\mathbf{A} = \beta \mathbf{H}_D + \alpha \mathbf{I}$. $\mathbf{H}_D$ is the Hessian of the network with respect to its weights $\mathbf{w}$. This closed-form solution now enables the efficient calculation of our desired performance measure.

[1] In the architecture of Fig. 1, the NN does not represent the plant model. Since, however, the network produces the augmented control signal, the quality of the output directly influence the quality of the model.

There are various ways to estimate the parameters α and β (see [5]). Since our aim is not to obtain an optimal model (i.e., network architecture), we chose a coarse approximation for the hyper parameters, namely $\alpha = W/2E_W$ and $\beta = N/2E_D$ for $N >> W$, where W is the number of weights, and N the number of training data in D. Experiments with more elaborate mechanisms (see [5]) yielded similar results for our application area.

Eq. (2) provides a natural basis for the system architecture of the confidence tool. In a *preprocessing* step, we calculate the matrix **A** and the hyper-parameters α and β. This calculation requires training data D and the network weights **w**. The *monitoring* component calculates the confidence measure σ^2 based upon the current input **x** and the values calculated by the preprocessing component. Figure 4 shows a simplified version of this architecture.

This architecture of the confidence tool can be applied to a pre-trained neural network, where all the training is performed before deployment. A truly adaptive controller, however, requires that the network training is also performed during system operation. In this case, the network weights are updated after each system cycle. For such networks, our algorithm interleaves the preprocessing and the monitoring components of the tool. For further reduction of the computational needs (i.e., essentially, how often the matrix inverse is calculated), we use a simple sliding window technique which requires the matrix inverse to be calculated only every p steps. In our application, where the main update cycle is 12.5ms (80Hz), we are using $p = 40$ in order catch the important dynamics but not to overburden the CPU.

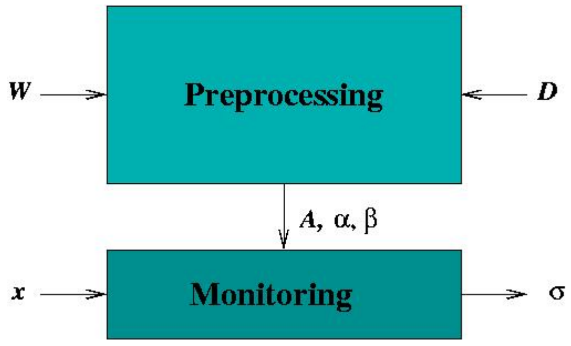

FIGURE 4. Simplified Architecture of Confidence Tool

EXPERIMENTAL RESULTS

The confidence tool has been implemented in C and Matlab for two system architectures. Architecture I uses a pre-trained network (single-hidden layer neural network), whereas architecture II uses a Sigma-Pi network which is trained on-line. Figure 5*(left)* shows the results of the confidence tool on data of an actual test-flight using a pre-trained neural network. The figure shows σ^2 over the entire duration of the flight (approx. 20 minutes). Before takeoff and after landing the aircraft is taxiing on the ground. It is observed that during the entire flight, the confidence of the network output is high (σ^2 is low). This is an indication that the network has been trained well for all operational conditions of this flight. Short spikes resulted from isolated bad data points. However, the network performance is substantially worse at the beginning and the end of the test flight. The reason for this is that the dynamic behavior of the aircraft on the ground is quite different from that in the air, and that the network has not been trained for these operating conditions.

Figure 5*(right)* shows the results of a simulation experiment carried out with the on-line adaptive controller of Figure 1. Here again, σ^2 is shown over time (small portion of a simulated test flight). At time $T = 1.0s$, the pilot issues a specific command, a so-called doublet (fast stick movement from neutral into positive, then negative and back to neutral position; Fig. 5*(lower right)*). Shortly afterwards ($T = 1.5s$), one control surface of the aircraft (stabilizer) gets stuck ("failure"). Because the system dynamics and the model behavior do not match any more, the neural network produces an additional control signal to compensate for this deviation. The network weights are updated according to the given weight update rule. Initially, the network confidence is very high, but as soon as the damage occurs, the network has to adapt. Here, σ^2 of the network outputs increases substantially, indicating a large uncertainty in the network output. Due to the dynamic training of the network, this uncertainty decreases very quickly.

A second and third pilot command which is identical to the first one is executed at $T = 11s$, and $T = 17s$, respectively. During that time, the network's confidence is still reduced, but much less than before. This is a clear indication that the network has successfully adapted to handle this failure situation.

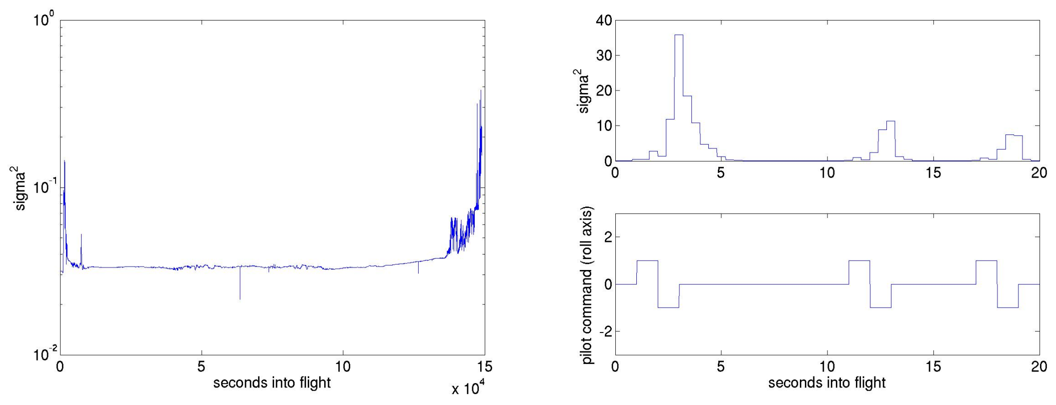

FIGURE 5. *(left)* Output of the confidence tool (architecture I) for a pre-trained neural network on test-flight data. Takeoff is at $T = 70s$, landing at $T = 1800s$. *(right)* Confidence value σ^2 over time *(top)* and pilot commands for roll axis (*bottom*). A failure has occurred at $T = 1.5s$.

CONCLUSIONS

In this paper, we have presented a tool to dynamically calculate the performance of the output of a neural network. The performance measure is the variance σ^2 of the network output. The tool has been developed for single hidden layer and Sigma-Pi networks which are pre-trained or trained during operation. In this paper, we have shown results of experiments for both architectures. The confidence tool for the on-line adaptive network is currently being implemented on the flight control computer of a NASA F-15 aircraft and will be test-flown later this year.

The confidence tool provides a statistical performance measure of the neural network. However, this measure cannot provide full information about the following questions: How fast will the network converge to a final solution? and How does the network performance relate to the overall performance of the controller? We are currently investigating the relationship of our performance measure to the overall handling quality of the aircraft. This relationship will finally enable the Confidence Tool to provide a reliable monitor for on-line adaptive neural network based controllers.

Acknowledgments This work is in part sponsored by the NASA IFCS project and the NASA Office of Safety and Mission Assurance (OSMA). We are thankful for the comments and suggestions provided by the anonymous reviewer.

REFERENCES

1. Norgaard, M., Ravn, O., Poulsen, N., and Hansen, L. K., *Neural Networks for Modeling and Control of Dynamic Systems*, Springer, 2002.
2. Ge, S. S., Lee, T., and Harris, C. J., *Adaptive Neural Network Control of Robotic Manipulators*, vol. 19 of *World Scientific Series in Robotics and Intelligent Systems*, World Scientific, 1998.
3. Rysdyk, R., and Calise, A., *AIAA American Institute of Aeronautics and Astronautics*, **AIAA-98-4483**, 1722–1728 (1998).
4. Calise, A., and Rysdyk, R., *IEEE Control Systems Magazine*, **21**, 14–26 (1998).
5. Bishop, C. M., *Neural Networks for Pattern Recognition*, Clarendon-Press, Oxford, 1995.
6. Rumelhart, McClelland, and the PDP Research Group, *Parallel Distributed Processing*, MIT Press, 1986.
7. Schumann, J., and Nelson, S., "Toward V&V of Neural Network Based Controllers," in *Proceedings WOSS (Workshop on Self-Healing Systems, 2002*, ACM Press, 2002, pp. 67–72.
8. Gupta, P., and Schumann, J., "A Tool for Verification and Validation of Neural Network Based Adaptive Controllers for High Assurance Systems," in *Proceedings High Assurance Software Engineering (HASE)*, IEEE, 2004.
9. Reed, R., and Marks, R., *Neural Smithing*, MIT Press, 1999.
10. Mackall, D., Nelson, S., and Schumann, J., Verification and validation of neural networks of aerospace applications, Tech. Rep. CR-211409, NASA (2002).