Monitoring techniques for an online neuro-adaptive controller

Yan Liu a, Bojan Cukic b, Edgar Fuller c,*, Sampath Yerramalla b, Srikanth Gururajan d

a Motorola Labs, Motorola Inc., Schaumburg, IL 60196, USA
b Lane Department of CSEE, West Virginia University, Morgantown, WV 26506, USA
c Department of Mathematics, West Virginia University, Morgantown, WV 26506, USA
d Department of Aerospace and Mechanical Engineering, West Virginia University, Morgantown, WV 26506, USA

Available online 12 June 2006

Abstract

The appeal of biologically inspired soft computing systems such as neural networks in complex systems comes from their ability to cope with a changing environment. Unfortunately, adaptability induces uncertainty that limits the applicability of static analysis to such systems. This is particularly true for systems with multiple adaptive components or systems with multiple types of learning operation. This work builds a paradigm of dynamic analysis for a neuro-adaptive controller where different types of learning are to be employed for its online neural networks. We use support vector data description as the novelty detector to detect unforeseen patterns that may cause abrupt system functionality changes. It differentiates transients from failures based on the duration and degree of novelties. Further, for incremental learning, we utilize Lyapunov functions to assess real-time performance of the online neural networks. For quasi-online learning, we define a confidence measure, the validity index, to be associated with each network output. Our study on the NASA F-15 Intelligent Flight Control System demonstrates that our novelty detection tool effectively filters out transients and detects failures; and our light-weight monitoring techniques supply sufficient evidence for an insightful validation.

Keywords: Neural network; Adaptive system; Run-time monitoring; Dynamic cell structure; Support vector data description; Validity index

1. Introduction

Adaptive systems are those systems whose functionality evolves over time in response to changes in the environment in which it operates. If learning and adaptation are allowed to occur after the system is deployed, the system is called online adaptive system Mili et al. (2003). The use of biologically inspired soft computing systems for online adaptation to counter the changes in system environments has revolutionized the operation of real-time automation and control applications. Neural networks are one of the most popular learning paradigms employed in online adaptive systems. Because the learning algorithms behind these computational architectures are usually derived from error/risk minimization theories, the computations are complex and the learning process is non-linear.

Applications of adaptive computing in safety critical systems are on the rise. These applications provide fault-tolerant control capabilities, automated maintenance of distributed networks, or enhance implementations of high security devices. Different research communities use different terms to describe online adaptive computing. For example, in computer networks automated modification of internal variables in traffic routing, possibly through judicious application of machine learning algorithms, is called the parameter adaptation paradigm McKinley et al. (2004). The Flight Control has become one of the most promising emerging applications of real-time adaptive control. The mechanisms and algorithms for achieving fault tolerant system features through adaptivity are termed software-enabled control, adaptive control augmentation or intelligent flight control.
When learning software is used in online adaptation, its behavior has direct consequence on the performance of systems into which it is embedded. Therefore, it is necessary to understand, predict and assure the behavior of the adaptive learner before its actual deployment. And while many experiments, especially in the aerospace domain, have demonstrated the potential of online adaptive computing, the lack of verification and validation procedures represents a serious problem barring their widespread use.

Our previous research using formal methods on certain families of neural networks suggests that environmental changes (learning data) have a significant impact on system behavior Mili et al. (2003). Abrupt and abnormal environmental changes are likely to generate previously unseen learning behaviors, because judicious enumeration of all environmental conditions is impossible. Therefore, we believe that the most promising paradigm for validating the performance of adaptive systems is their continual monitoring. The monitoring data collected in real time is then used to analyze system properties of interest, most typically related to convergence and stability of the learning algorithm Yerramalla et al. (2003).

Therefore, the most meaningful questions related to the monitoring approaches to validation of online adaptive systems are the following two:

- What to monitor?
- How to analyze monitored data?

This paper provides our answers to these questions. The data stream entering the adaptive computational element, such as the neural network, is monitored to detect anomalous data patterns which indicate significant environmental changes. The internal structure of the neural network is monitored and analyzed with respect to the convergent behavior that usually indicates successful learning. Furthermore, the network predictions, i.e., its output data stream, is monitored and statistically analyzed for variations as an indication of their volatility over time.

Our research further indicates that the choice of the methods for system validation is phenomenological. In other words, we encountered systems where, depending on the design choices, not all the above mentioned techniques are feasible. Therefore, while these techniques are complementary, the feasibility of their application in the analysis of adaptive learning varies. This paper presents three monitoring and analysis techniques, demonstrates their partial applicability to two types of learning modes, both investigated in the context of an experimental adaptive flight control scheme.

1.1. The Intelligent Flight Control System

The Intelligent Flight Control System (IFCS) was developed by NASA with the primary goal to “flight evaluate control concepts that incorporate emerging soft computing algorithms to provide an extremely robust aircraft capable of handling multiple accident and/or an off-nominal flight scenario” Boyd et al. (2001) and Jorgensen (1991). The diagram in Fig. 1 shows the architectural overview of NASA’s IFCS implementation using Online Learning Neural Network (OLNN). The control concept can be briefly described as follows. Notable discrepancies from the outputs of the Baseline Neural Network and the Real-time Parameter Identification (PID), either due to a change in the aircraft dynamics (loss of control surface, aileron, stabilator) or due to sensor noise/failure, are accounted for by the OLNN.

The primary goal of OLNN is to accomplish in-flight accommodation of discrepancies, commonly known as Stability and Control Derivative errors. Derivative errors indicate conditions that fall outside the scope of traditional (linearized) control gain look-up tables. When OLNN performs adaptation, its behavior has a direct consequence on the performance of the flight control system. In such a safety-critical application, it is necessary to understand and predict the behavior of the NN. The goal of validating NN-based online component is to provide a means to detect novel (abnormal) conditions entering the OLNN, to investigate the NN’s stability behavior during adaptation, and to interpret the results of the analysis so that it ensures safe operation.

One concern in this process is the presence of noise in sensor data as well as temporary variations in environmental data which we may wish the neural network to tolerate without disruption of the training process. In any validation method, it is necessary to distinguish between such transient disturbances which do not persist or may not impact performance and true failures which do persist or have a major impact on the system. The transient nature of a disturbance may not become apparent except over time and the true impact may only emerge after an adaptive learner has been trained, but it is still necessary to respond to extreme variability as quickly as possible. The primary contribution of the current work is to observe that an integrated approach to validation is needed in which monitoring occurs at multiple time scales so that both immediate analysis of real-time learning and buffered long-term failure indicators are used.
1.2. Paper overview

The rest of the paper is organized as follows. In Section 2 we give a summary of the adaptive component of the IFCS. This section also reviews of the underlying structure of the neural network, describes the flight simulator used to generate test data and lists the failure modes used in the ensuing experiments. Section 3 introduces novelty detection techniques which discriminate between permanent failure events and transient failure events before their data representations reach the input layer of the neural network. The same section discusses the accuracy of this method. Sections 4 and 5 outline two techniques for analyzing the IFCS. Each technique addresses a different type of the learning algorithm. We also describe experimental results by applying the techniques to two simulated failure modes. Finally, we conclude with a summary and list our future research directions in Section 6.

2. The online learning neural network

In IFCS, a special type of neural network, the Dynamic Cell Structure (DCS), is deployed for online learning. The DCS neural network is a dynamically growing structure. It can be seen as a special case of Self-Organizing Maps (SOMs), originally introduced by Kohonen in 1981 as biologically inspired adaptive vector quantization algorithms suitable for unsupervised feature extraction. An SOM essentially constructs a structured representation of the presented input data using prototypes, called weight vectors. The weight vectors are \( \mathbf{w} \) as a function of neuron coordinates. They form a feature map. A scheme for generating topology-preserving feature map is characterized by assigning weight vectors such that nearby data patterns in the input data are mapped to neighboring neurons of the network. The DCS network utilizes a Kohonen update for neuron placement and a competitive Hebb rule to construct connections between neurons.

The data patterns presented to the DCS neural network can be divided into different regions, known as the Voronoi regions. Each Voronoi region is represented by a neuron of the neural network whose reference weight vector is closest to a given input data pattern than any other neuron with the network. Such neuron is called the “best matching unit (BMU)”. Further, a “second best matching unit (SBU)” is defined as the neuron whose reference weight vector is the second closest to a given input data pattern. Euclidean distance metric is commonly used for defining a BMU and SBU for a given input data pattern. In the DCS, the neurons are laterally interconnected. The strength of the connection can vary between 0.0 and 1.0. Neuron(s) connected to the BMU of a given input data pattern are considered as the BMU’s neighbors, and denoted by NBR. In the DCS, the weight vectors of neurons and the lateral connections between neurons are updated based on the presented input data pattern. In addition, DCS adds neurons as needed to model the given data as accurately as needed. The resulting DCS network adopts a self-organizing structure that dynamically evolves with the presented data. The graph network so created is topologically equivalent to the data set in the sense that network training causes this graph to converge to a network that has a neighborhood preserving correspondence with the Voronoi regions of the data set. The Kohonen rule

\[
\Delta \mathbf{w} = \epsilon ||m - \mathbf{w}_{BMU}||
\]

is used for updating the weight vectors \( \mathbf{w} \), and the Hebb rule

\[
C_{ij}(t+1) = \begin{cases} 
1 & (i = BMU) \wedge (j = SBU) \\
0 & (i = BMU) \wedge (j \in NBR - SBU) \wedge (C_{ij} < 0) \\
C_{ij}(t) & (i = BMU) \wedge (j \in NBR - SBU) \wedge \left(C_{ij} \geq 0 \right) \\
C_{ij}(t), & i, j \neq BMU 
\end{cases}
\]

is used for updating lateral connection values \( C_{ij} \) between neurons. For complete implementation details, the reader is referred to Bruske and Sommer (1995).

The DCS learning algorithm is depicted in Fig. 2. \( N \) is the number of training examples. Resource values are computed at each epoch as local error measurements associated with each neuron. They are used to determine the sum of squared error of the whole network. Starting from two

```
Initialization;
Repeat until stopping criterion is satisfied;
{
    Repeat N times
    {
        Determine the BMU and SBU;
        Update lateral connections;
        Adjust the weights;
        Update resource values;
    }
    If needed, a new neuron is inserted;
    Decrement resource values;
}
```

Fig. 2. A brief description of the DCS learning algorithm.
connected neurons randomly selected from the training set, the DCS learning evolves into topologically representative structure that satisfies the requirements. The evolution process is a combination of adjustments of weight vectors, and lateral connections that is usually followed by addition of neurons.

2.1. Two types of online learning

The role of DCS in the IFCS is to learn and represent in real-time the difference between the real-time parameter identification and the pre-trained component (refer to Fig. 1). DCS tracks the differences and provides estimates that can be utilized in subsequent flight segments. In the given flight control system, a data point consists of a vector in seven-dimensional Euclidean space. Each seven-dimensional point corresponds to four aircraft sensor measurements and three flight control derivative parameters. Twenty data points are presented to the DCS every second, which corresponds to a frequency of 20 Hz. The architecture of the system gives rise to two types of online learning as described below.

- **Type I:** Incremental learning. This type of online learning is common in online learning applications. The neural network receives a different data point every training cycle, i.e., a data point is presented to the DCS at each iteration of the control system. An input data point represents the latest sensor readings and the ensuing derivatives. After a data point is presented to the online learning neural network, the network adjusts its parameters based on its learning rules. For this type of online learning, a cycle assumes the presentation of a single (new) data point to the DCS network.

- **Type II:** Quasi-online learning. This type of online learning allows the neural network to train on a data set while incoming data is buffered for later presentation to the learner. It is based on learning from a data buffer of fixed size that is updated at the rate of the system clock. During learning, the DCS network fetches data from the data buffer and keeps learning using the algorithm in Fig. 2. The DCS network adjusts its structure to map the set of data points which remains constant over a certain number of training iterations. After the neural network has been exposed to training data, before fetching a new training set from the buffer, it provides outputs for a given test pattern, if needed. Using the machine learning terminology, the neural network now operates in the recall mode. In the recall mode, the DCS network is employed to recall parameter values at any chosen dimension. It should be noted that the computation of an output is different from that during training. When a DCS network is in recall, its output is computed based on two neurons for a particular input. One is the BMU of the input, the other is the closest neighbor of the BMU. In the absence of any neighbors to the BMU, the output is calculated using the BMU only.

2.2. Validating the online learning

For a safety-critical system like IFCS, validating the online learning is essential. However, relatively little research has been done for verification and validation of neural network-based control systems. NASA developed a software verification process guide NASA Guidebook (1996) addressing these issues. Different online monitoring techniques have been proposed to validate the learning process. In ISR Report (2001), Taylor et al. focus their effort on the DSC learning. They propose a prototype for real-time rule extraction in order to verify the correctness of DCS learning performance. In Darrah et al. (2004), present rule extraction from DCS network learning and suggest future examination of performance based on such rules. Practical hurdles associated with this approach include determining the frequency of rule extraction and impracticality of near real-time model checking of complex systems. Gupta and Schumann (2004) discusses an approach toward verification and validation of such systems, using Bayesian techniques to estimate neural network quality for a different type of online neural networks, namely the Sigma-Pi networks.

In our attempt to validate the online learning, we noticed that it is very important to differentiate the short term, transient changes in the system environment from the failure conditions. In principle, the online neural network is deployed for the purpose of failure accommodation. When a failure occurs the system needs a prompt recovery through effective adaptation. Therefore, we need a novelty detector that can filter out transients and provide failure detection capabilities based on the duration and the degree of the novelty in the input data stream. We use novelty detection to better observe and understand the impact of failure conditions on the learning behavior of the online neural network. Efficient failure detection ensures that the system implements failure accommodation when necessary.

The behavior of adaptive software systems depends implicitly on the manner in which adaptation is allowed to occur. In any given system, if the performance of the adaptive system is to be analyzed and validated, it is important to understand first what type of learning is being used and second how the system behaves under this type of learning. Sections 4 and 5 analyze the aforementioned Type I and Type II of learning seen in the NASA IFCS implementation of the DCS neural network based controller, respectively. For Type I learning, validation of the adaptive system comes from the confidence level of the actual DCS run-time behavior shown by the Lyapunov-like monitors. For Type II learning, a validity index is assigned to the neural network output as it responds to changing stream of data points. Both methods provide an insight into the response of the DCS network to anticipated failure conditions.

2.3. Tested flight conditions

In our experiments, the input data to the DCS neural network is collected from an F-15 flight simulator that
represents the functionality of the flight control system shown in Fig. 1. The simulator approximates a non-linear model of the F-15 aircraft based on the data distributed at the 1990 AIAA GNC Design Challenge. Readers are referred to Napolitano et al. (1998) for a detailed description of the simulator. Primary control surfaces of the aircraft consist of elevators, ailerons and rudders. The following two types of control surface failures have been modeled:

- The first type of failures are actuator failures resulting in locked surfaces. In this scenario the user has the option to have the failed surface stay locked at the current position or in a pre-defined position.
- The second failure type corresponds to the case of a damaged control surface resulting in a drop in the efficiency of the control surface starting from the instant of failure. The user of the simulator can select to fail any individual control surface: left or right stab, aileron, and rudder.

In order to obtain sufficient nominal flight data, we run the simulator for 40 s at the simulation rate of 20 Hz and collect 800 data points. We use these 800 data points as target-class (nominal) data to train our novelty detection algorithms. Simulated failure conditions usually follow the nominal data stream. The DCS algorithm trained on the nominal and failed data streams from such simulations. The NASA project selected seven typical failure modes, as follows:

- **Mode 1**: a locked control surface failure (locked left stab, stuck at 0°).
- **Mode 2**: a locked control surface failure (locked left stab, stuck at +3°).
- **Mode 3**: a locked control surface failure (locked left stab, stuck at -3°).
- **Mode 4**: a locked control surface failure (locked right stab, stuck at +3°).
- **Mode 5**: a locked control surface failure (locked right stab, stuck at -3°).
- **Mode 6**: a loss of control surface (50% missing surface of left stab).
- **Mode 7**: a loss of control surface (50% missing surface of right aileron).

In Section 3, we first present the novelty detection approach using Support Vector Data Description algorithm. Sections 4 and 5 describe two DCS monitoring approaches, related to the two types of learning. The data allowed to enter the DCS network follows the novelty detection step, which excludes (most) transient failures. While we applied our monitoring and analysis techniques to all the seven failure modes described above, due to space limitation, this paper will report results from the failure mode 3 and the failure mode 6.

### 3. Novelty detection

A novelty detection technique serves as a filter that disallows the transient conditions from entering the DCS network. In general, novelty detection techniques require beforehand knowledge of both nominal and off-nominal flight domains. However, for a flight control system it is impossible to anticipate all possible failure situations. As a one-class classification tool, Support Vector Data Description (SVDD) technique is derived from Support Vector learning theory by Tax and Duin (1999a,b, 2001). Differing form general support vector classifiers that decide the maximum margin hyperplane to separate two classes, SVDD method tries to find a decision boundary for a given one-class data set by minimizing the distance from the boundary to the center of the data set. It provides a sound representation of the target-class and offers inferences that can be used to detect the outliers. For our purposes, the target-class represents nominal flight conditions, i.e., the “safe region”.

SVDD is developed from the concept of finding a sphere with the minimal volume that contains all data items Tax and Duin (1999a). Given a target data set, the SVDD’s task is to minimize an error function containing the volume of this sphere. Flight control systems produce high-dimensional data, characterized by non-linearity and, consequently, inseparability by a linear hyperplane. This makes the data description more difficult to obtain. Similar to the Support Vector Machine (SVM) Vapnik (1998), by employing a kernel function, we are able to map the data from a high dimensional space onto a Hilbert space, also referred to as the feature space, so data classification is achieved with reduced computational complexity Bennet and Campbell (2000). In our experiments, we use the well-known Gaussian kernel function. By applying the SVDD method, we obtain a sound representation of the target class. The ROC curve given in Fig. 3 reflects...
only when there is an evidently "novelty detector will alarm the system of possible failure being used for DCS learning and system adaptation. The detection tool filters these data points to exclude them from being noticed. Because of their short duration, our novelty criterion should be inferred from empirical testing or predefined thresholds. In our experiments, more than 1 s of consecutively high values of novelty measure is considered a failure condition. With a frequency of 20 Hz, it also means a count of 20 consecutively high novelty measures.

Fig. 4 presents the novelty measure reported by SVDD during a specific flight simulation. The failure occurs at $t = 30$ s. Prior to the failure, several transients can also be noticed. Because of their short duration, our novelty detection tool filters these data points to exclude them from being used for DCS learning and system adaptation. The novelty detector will alarm the system of possible failure only when there is an evidently “long” period of novel data points. Since the evaluation of test data points involves “support vectors”, a relatively small fraction of the data set, the detection of novelties is computationally efficient.

3.1. Experimental results

A decision boundary for separating the nominal flight data from novelties is needed before we can use it for online transient filtering and failure detection. We first simulated several runs of nominal flight conditions, 40 s segments with 800 data points. Using the nominal data sets, SVDD algorithm forms a sound data description of nominal flight conditions. This data description is then used to filter the transients and detect failures. The crosses in both Figs. 5(a) and 6(a) represent the nominal data points which have been used by SVDD to define the boundary. Circles in Figs. 5(a) and 6(a) represent simulated failure mode data points.

We then used the boundary formed by SVDD on two different failure mode simulations. Novelty measures are plotted in Figs. 5(b) and 6(b) for these two failure modes. Evidently, the appearance of non-nominal data causes dramatic spikes in novelty measures. Transients can be seen in both plots before the failure occurs at $t = 30$ s. After that, SVDD detects continual large values of novelty measure in the incoming data sets. Due to its long duration, the measure meets the count of 20 cycles (i.e., a second) and declares a failure condition. A series of experiments with simulations of seven different failure modes demonstrated the effective and accurate failure detection capability of our SVDD based novelty detector.

4. Validating Type I online learning

Following the elimination of transients, DCS neural network is exposed to training data. The goal of the monitoring techniques is to extract numerical indicators of the quality of learning. Of course, learning quality is a very abstract term. But, if the DCS network can be considered a dynamical system, techniques that analyze convergence and stability of its adaptation can help. This is the motivation behind the approach described in this section. To remind the reader, Type I learning implies that the data points (sensor and derivative values) are immediately added into the training data set of the DCS network. Therefore, the network will be able to react to the emerging changes quickly. This creates the need to design a monitoring technique capable of tracing rapid changes in the behavior of the DCS algorithm.

A standard methodology for evaluating the performance of the DCS network is to compute its total and quantized errors. The total error, $E$, is defined to be the sum of the Euclidean distances between each data element of the presented input (training) data pattern $m \in M \subset \mathbb{R}^D$ and the closest node of the neural network, $w_{BMU}(m) \in \mathcal{W} \subset \mathbb{R}^D$ corresponding to each $m$. The quantized error, $Q$, is this sum divided by the number $N$ of nodes in the network. Symbolically, they are computed as

$$E = \sum_{m \in M} ||m - w_{BMU}(m)||$$

$$Q = \frac{1}{N} \sum_{m \in M} ||m - w_{BMU}(m)||$$

A necessary part of implementing NNs in safety or mission critical applications is determining whether or not the NN is performing as expected or desired NASA Guidebook (1996), ISR Report (2001). The following four error values numerically reflect the network’s success in learning the structure of the data set being trained.
Definition 1 (NN Monitors)

\[ M_1 = \sum_{m \in M} \| m - w_{BMU}(m) \| \]
\[ M_2 = \sum_{m \in M} \| m - w_{SBU}(m) \| \]
\[ M_3 = \sum_{m \in M} \frac{\| m - w_{NBR}(m) \|}{\#(NBR)} \]
\[ M_4 = \sum_{m \in M} \frac{\| m - w_{Non-NBR}(m) \|}{\#(Non-NBR)} \]

(4)

\( M_1 \) is the total error of the network. \( M_2 \) is the Euclidean distance between each data element of the presented input (training) data pattern \( m \in M \subseteq \mathbb{R}^D \) and its second closest neuron in the neural network, \( w_{SBU}(m) \in \mathbb{R}^D \). \( M_3 \) is the mean Euclidean distance between each data element of the presented input (training) data pattern \( m \in M \subseteq \mathbb{R}^D \) and the set of neighborhood neurons of the BMU of neural network, known as the NBR-set \( w_{NBR}(m, BMU) \in \mathbb{R}^D \). \( M_4 \) is the mean Euclidean distance between each data element of the presented input (training) data pattern \( m \in M \subseteq \mathbb{R}^D \) and the set of laterally connected, non-neighboring neurons of the BMU of neural network, known as the non-NBR-set, \( w_{Non-NBR}(m, BMU) \in \mathbb{R}^D \). Each of these expresses an aspect of the neural network performance and represents a function whose values are Lyapunov-like in the sense that they model Lyapunov functions for that aspect of the system of differential equations which govern the behavior of DCS although they are not try Lyapunov functions for the total system Zubov (1964).

As shown in Fig. 2, DCS learning involves several steps including Kohonen learning, Hebb update, resource update, growing neurons and pruning neurons. Through experiments, we observed that having a single monitor will not be sufficient for system assurance analysis. This is noticeable in cases where the DCS encounter rapidly varying, divergent data manifolds Yerramalla et al. (2003). Therefore, our online monitoring system is composed of four monitors. Each monitor is essentially a Lyapunov-like function that is designed to analyze and capture instable behavior from a particular aspect of online learning. These monitors provide an observation of how well a set of associated neural centers of the DCS are being overlayed over corresponding relative elements of presented training data.
Failed flight conditions were simulated by running a flight under nominal (no failure) conditions for the first 5 s. Thereafter, at the 5th second, a failure is injected into the simulation. The failure changes the operating condition of the aircraft into a failed flight condition. This means that the first 99 frames of data represent the data generated under nominal flight conditions. The data points 100–200 are generated under failed flight conditions. For this case study, we utilized data from the flight simulator for the two failed flight conditions described in Section 2.

The stability monitoring system proposed here comprises of four monitors that have been specifically developed to detect unstable learning conditions from different aspects of online learning. A spike (abrupt increase) in the values of any or all-four monitors indicates unstable online adaptation conditions in the neural network. Since there may be noise associated with the monitor values (either due to the neural network input or otherwise) that could raise a false alarm, a spike in the monitor value is deemed predominant if it is above a certain threshold value. For this case of online learning, the threshold is set to value of 10σ, where the term σ represents the standard deviation of the initial values of the monitor. The initial value of the monitor is referred here as the baseline value. For the current analysis, the first 20 monitor values (corresponding to the first 20 cycles of flight data) are collectively considered as the monitor baseline. The monitor baseline can be thought of as the startup values before the beginning of detection. Note that the value of the threshold gain is set to a value of 10 in order to reduce false alarms, while increasing the probability of detection. The gain value that is specified here is based on observations from several runs of stability monitoring experiments that were previously conducted.

4.1.1. Failure mode 3: a locked control surface failure

A stuck control surface failure condition is simulated in the experimental flight simulator by locking the left stabilator during flight at an angle of −3°. The data after 99 cycles can be regarded as stressed (failed) performance conditions. As the online learning neural network is embedded in the adaptive system, during the flight failure condition, it will likely deviate to an unstable learning condition. The goal of the proposed stability monitoring system is to detect this bifurcation away from stable online learning in real-time. The stability monitors work in parallel to the online learning, and provide an indication of the stability in the current condition of online learning. Fig. 7 shows the values of the 4 stability monitors for this case. The mathematical formulations of the stability monitors are given in Eq. (4). Pre-dominant spikes in monitors #2 and #3 can be noted around the 100th cycle. The spikes indicate that the stability monitoring system has detected the state of online learning that has bifurcated away from stable learning behavior. The behavior of the stability monitors is consistent with the simulated failed flight condition, the aircraft’s left stabilator locked during flight at an angle of −3°, injected after 4 s. Using a 20 Hz sampling frequency, the end of the 4th second corresponds to data points emerging after 99 cycles.

4.1.2. Failure mode 6: a loss of control surface failure

A missing control surface failure condition is simulated in the experimental flight simulator by restraining the left stabilator during flight to lose its functionality by 50%. As in the previous case, the data after 99 cycles can be regarded as the stressed performance conditions. Fig. 8 shows the values of the four stability monitors for this case. Pre-dominant spikes in monitors #2 and #3 can be noted around the 100th cycle. The spikes indicate that the stability monitoring system has detected the state of online learning that has bifurcated away from stable learning behavior.
zation of values from the stability monitors should be performed such that the minimum and maximum monitor-values after normalization lie between 0 and 1. The idea behind normalization is to transform the stability measures into probabilistic stability beliefs. In order to obtain the stability beliefs, the maximum value of a monitor is recorded and stored at each time frame. At any time frame, every monitor has an associated current maximum. The monitor values are then divided by their corresponding current maxima at each time frame.

4.2.2. Data-fusion scheme

In order to apply Murphy’s rule for combining evidence from distinct sources of information, we create a frame of discernment $\Theta$ Murphy (1996, 1998). Based on the premise that each monitor provides an independent belief of the stability conditions in online learning, the following propositions are made.

- $E \rightarrow$ how unstable is the current online learning condition.
- $C \rightarrow$ how much trust can be asserted in the current online learning condition.

The beliefs for proposition $E$ are first combined and used in calculating the belief for proposition $C$. While combining evidence, if a belief with a value 0 is combined with any other non-zero belief the resultant combined belief value is 0. Similarly combining any belief with a belief of value 1 will result in a combined belief of value 1. To avoid this ambiguity, all beliefs with values 0 are assigned small values close to 0. Similarly, all beliefs with value 1 are assigned values close to 1. Consequently, a belief at any instance of time represents a real number in the open interval $(0,1)$.

Let $m_1(E), m_2(E) \ldots m_k(E)$ represent the assigned beliefs for proposition $E$ from Monitor #1, Monitor #2, \ldots Monitor #k, respectively, where $k$ represents the total number of monitors. There are many possible combinations of error values from $k$ monitors. For example, they can be combined first in groups of two and the resultant can then be combined again in groups of two and so on. This method applies to cases where the value of $k$ is even. Another approach is to combine the first two beliefs and combine the resultant belief with the third belief and so on. This method of combination of beliefs, called a cascade strategy, is shown in Fig. 9. Note that when combining more than two beliefs using Murphy’s rule, the order in which beliefs are combined affects the outcome. In other words, using the cascade combination method, different orderings of beliefs result in different values of the combined belief.

Fig. 7. Online Stability Monitoring System for a locked control surface. (a) Monitor #1, BMU Error; (b) Monitor #2, SBU Error; (c) Monitor #3, NBR Error; (d) Monitor #4, Non-NBR Error.
The desirable goal is to find maximum and minimum values of combined beliefs. The maximum and minimum values of combined beliefs would then result in a bound for the interval that includes combined beliefs from all possible combinations. In a cascade it is evident that the last monitor (Monitor \(k\)) has the highest influence on the final result. An approach for determining the minimum and maximum combined beliefs is to generate combined beliefs from all possible orderings of individual beliefs so that every belief has the highest influence on the combined belief. But in this case, \(k!\) possible orderings of beliefs need to be evaluated in order to determine the minimum and maximum value of combined beliefs. This would be computationally too expensive and would prevent the application of this methodology to realtime systems.

It is shown in Mladenovski (M.S. Thesis) that combining beliefs in increasing order of their values significantly reduces the computational complexity of combining stability belief functions from individual monitors, as the most expensive operation becomes the sorting of \(k\) beliefs, and that the result is the maximum of all combined beliefs. An easily interpretable stability result can, therefore, be computed in \(O(k \cdot \log k)\), where \(k\) is the number of monitors (in our case, \(k = 4\)).

4.2.3. Experimental results

To demonstrate the practicality of the proposed fusion technique for stability monitors, we used it in the analysis of online learning of the intelligent flight control system under failed flight conditions. Details of the simulated failed flight conditions are provided in Section 2. For consistency with the stability monitoring results, we provide results here from applying Murphy’s rule to combine the four stability monitors shown in Figs. 7 and 8. Recall that the stability monitoring system in Fig. 7 is from online learning from a failed flight condition, locked left stabilator stuck at \(-3^\circ\), and stability monitors in Fig. 8 are from online learning from another type of failed flight condition, left stabilator looses 50% of the surface. Further details on the individual failed flight conditions can be obtained by referring to Section 2.3.
Fig. 10(a) shows the result from applying Murphy’s rule to combine the four stability monitors (given in Fig. 7) for the failed flight condition, a stuck control surface (locked left stabilator stuck at $\alpha/3$). In Fig. 7, monitors 2 and 3 indicate changes in online learning due to the failed flight condition. Using Fig. 10(b), this information of instability in online learning can be easily interpreted by observing a single value.

Fig. 10(b) shows the result from applying Murphy’s rule to combine the four stability monitors (given in Fig. 8) for the failed flight condition, a 50% loss in functionality of a control surface (left stabilator). In Fig. 8, monitors 1 and 2 indicate changes in online learning due to the failed flight condition. Using Fig. 10(b), this information of instability in online learning can be easily interpreted by observing a single value.

5. Validating Type II online learning

For Type II online learning of the DCS network, we propose a different monitoring technique. It is important to notice here that stability monitors would not provide an effective monitoring mechanism for Type II learning. In Type I, every cycle of the control loop introduces a new data point on which the DCS network trains. Therefore, stability monitors are designed to be very sensitive and capture rapid changes in the behavior of the learning algorithm. In Type II learning, the buffer which contains training data changes its content once in a while, usually every second. Stability monitors would likely indicate a departure from convergent learning every time the buffer undergoes a significant change. Therefore, it became obvious to us that we needed a different approach to cope with Type II learning.

This section introduces a mechanism to generate the measures of trustworthiness for DCS predictions. We define the validity index (VI) in DCS networks as an estimated confidence measure of a network output. The VI can be used to model the accuracy of the DCS network prediction and thus provide inferences for validation activities. Based on the rules of DCS learning and properties of the network structure, the computation of a validity index for a given input consists of two steps: (1) compute the local error associated with each neuron, and (2) estimate the standard error of the DCS output for the given input using information obtained from step (1). These steps are described below.

Step 1. The final form of DCS network structure is represented by neurons as centroids of Voronoi regions. Since the selection of the best matching unit must be unique, only the data points mapped to the same BMU will be contained in the same region. Therefore, all Voronoi regions are non-overlapping and cover the entire learned domain. The data points inside each region significantly affect the local fitting accuracy. The local estimate of variance of the network residual in a particular region can be calculated over the data points contained in the region and then be associated with its representative neuron. Because the adjustment by competitive Hebbian learning rule concerns connections between the BMU and its neighbors only, the further update of weight values by Kohonen learning rule is performed only on the BMU and its neighbors. Therefore, we also consider the proportional contributions made by training data points covered by the neighboring neurons of the BMU. As a result, the influence of all related data points is taken into account based on connections, referred to as $C_{ij}$ between the BMU and its neighbors.

Step 2. Recall that the output produced by DCS is determined by the BMU and its closest neighbor.
(CNB) of the given input. Thus, the local errors associated with these two neurons are the source of fitting inaccuracy. Provided with the local estimate of variance for every neuron from Step (1), we define the confidence measure as an estimate of the fitting inaccuracy using the local errors of the BMU and the CNB.

A very detailed description of the computation of VI can be found in Liu et al. (2005). Because all needed information for calculating VI is already available in each training cycle, minor changes to the algorithm imply a negligible computation effort added to the DCS network learning. When the DCS network is in recall/learning, the validity index is computed based on the local errors and associated with every network output. In the case of IFCS, a domain specific threshold can be pre-defined to help verify that the accuracy of the validity index is acceptable in the system context.

5.1. Experimental results

We again simulated different IFCS flight conditions at the rate of 20 Hz. We start training the DCS network under nominal flight conditions with 200 data points, as they become available. After that, every second, we set the DCS network in recall mode and calculate the derivative corrections for the incoming 20 data points and their validity index. Based on the values of validity index, we evaluate how well the neural network is able to respond to unforeseen conditions and accommodate such conditions. Then, we set the DCS network back to the learning mode and include the incoming data points in it. While updating the data buffer, we discard the earliest 20 data points and add the most recent 20 data points to maintain the buffer size, i.e., 200. The DCS neural network continues learning and repeats this recall/train procedure.

Figs. 11 and 12 show the experimental results of our simulation on failure mode 3 and failure mode 6, respec-

Fig. 11. Testing on failure mode 3 in real-time (running at 20 Hz, failure occurs at 100th data frame). (a) The final form of DCS network structures. (b) Validity Index shown as error bars for each DCS output.

Fig. 12. Testing on failure mode 6 in real-time (running at 20 Hz, failure occurs at 100th data frame). (a) The final form of DCS network structures. (b) Validity Index shown as error bars for each DCS output.
tively. In both figures, Plot (a) shows the final form of the DCS network structure at the end of the simulation. As a three-dimensional demonstration, the x-axis and y-axis represent two independent variables, z and β, respectively. The z-axis represents one derivative correction, ACzz. The 200 data points in the data buffer at the end of the simulation are shown as crosses in the 3-D space. The network structure is represented by circles (neurons) connected by lines. In both figures, Plot (b) presents the validity index, shown as an error bars chart. The x-axis represents the time/data frames. The failure occurs at the 100th data frame. We compute the validity index for the data points that occurred within the window of 5 s before and 5 s after the failure occurs (10 s, and thus 200 frames).

A trend revealed by the validity index in our simulations is the increasingly wider error bars after the failure occurs. Wide error bars indicate reduced confidence in the accuracy of the DCS network outputs. Shortly, after the occurrence of the failure the error bars start narrowing, while the DCS network starts adapting to the new domain. In essence, narrower bounds indicate that the network performs failure accommodation successfully. After the failure occurs, the change (increase/decrease) of the validity index varies. This depends on the characteristics of the failure, the position in the flight envelope, as well as the characteristics and parameters of the DCS network. Nevertheless, the validity index explicitly indicates how well and how fast the DCS network accommodates the novel data, thus helping out in the overall aircraft failure recovery process. It should also be pointed out that the detection of error in the output from DCS occurs approximately 20 data cycles after failure injection in the simulator. This is due to the data buffering for 20 frames and results in a delay of approximately 1 s in the VI indication of error.

6. Summary and discussion

In our experience, the architectures and design of many adaptive systems suggest a multi-tiered approach to their analysis. Due to their complexity and the dynamics of their change, the problem of performance validation is complicated. No single aspect of the adaptive system is likely to be a reliable indicator of correctness of operation in nondeterministic environments.

This paper outlines the development of three types of functional performance monitoring techniques that can be used to validate a dynamically developing, self-organizing neural network such as DCS. During the development of these tools, a pattern emerged that indicates a natural stratification of these validation techniques. Two modes of learning were observed in the implementation of DCS used in the NASA Intelligent Flight Control System. In each case, a different set of tools seems more appropriate for analysis. Specifically, tools such as SVDD and Lyapunov methods are more suited to true real-time analysis of emergent behavior in systems where fast response is necessary at the cost of sensitivity to transient effects. A measure of prediction error such as the validity index, on the other hand, is more applicable in situations where data is buffered and output is analyzed after training. This approach has the benefit of being less sensitive to transient changes in environment or sensor networks. When used in tandem, these approaches may offer the best of both worlds.

Computational efficiency and scalability of both methods presented in this paper inspire our confidence that the proposed monitoring techniques can be generalized to other type of online adaptive systems. We are also inclined to believe that the analyses enabled by our monitoring techniques are the meaningful early steps towards the verification and validation of online adaptive systems. However, at this point in time, it is very difficult to envision the types of applications, machine learning algorithms and systems which will require and/or utilize online adaptive behavior. There is no doubt that this research area will become increasingly important in the near future.

References

Mladenovski, M., Information fusion schemes for real-time risk assessment in adaptive control systems. M.S. Thesis, Lane Department of Computer Science and Electrical Engineering, West Virginia University.


Yan Liu is a senior research scientist in the Software and System Engineering Research Lab at Motorola Inc. She received the BS degree in computer science from Wuhan University, China, and the MS and Ph.D. degrees in computer science from West Virginia University. Dr. Liu’s research interests include software verification and validation, software process improvement, adaptive systems, and intelligent computing.

Srikanth Gururajan is currently a Graduate Research Assistant and a Tycho Brahe awardee for 2004 and a co-recipient of Tycho Brahe award from NASA Office of Safety and Mission Assurance. His research interests include Digital Signal Processing, Artificial Intelligence, Adaptive Systems, and Control Theory. Yerramalla earned his Ph.D. at West Virginia University in Electrical Engineering, and holds a Masters Degree in Control Systems Engineering from West Virginia University Institute of Technology.

Bojan Cukic is an associate professor in the Lane Department of Computer Science and Electrical Engineering at West Virginia University, where he chairs Biometrics curriculum. He also serves as the co-director of the Center for Identification Technology Research, a National Science Foundation Industry – University Research Center. Dr. Cukic’s research interests include software engineering for high-assurance systems, fault tolerant computing, information assurance and biometrics. He received a US National Science Foundation CAREER award and a Tycho Brahe award from the NASA Office of Safety and Mission Assurance. Dr. Cukic is a member of the editorial board of Empirical Software Engineering journal and served as a guest editor of IEEE Software magazine in 2005. He received his Ph.D. in Computer Science from the University of Houston.

Edgar Fuller is an Assistant Professor of Mathematics at West Virginia University. He currently works on applications of topology and geometry to the study of autonomous learners, especially those in mission critical applications. He was the co-recipient of the Tyco Brahe award from the NASA Office of Safety and Mission Assurance in 2005. His other interests include applications of computational geometry to knot theory, the analysis of non-deterministic systems, bioinformatics, and mathematics education. He currently is the Principal Investigator for a US Department of Education funded Math and Science Partnership in the state of West Virginia. He received his Ph.D. from the University of Georgia in the area of differential geometry and topology.

Brijesh Solaiman is currently with Sensor Research and Development Corporation (SRD-Corp.) at Maine, where he is a Senior Research Scientist. He is a leading researcher in the area of Sensor Signal Processing at SRDCorp, and serves as the project lead for a US Department of Defense, DoD sponsored program on sensor development for detecting material breakthrough applications. He is an IEEE Student Travel awardee for 2004 and a co-recipient of Tyco Brahe award from NASA Office of Safety and Mission Assurance. His research interests include Digital Signal Processing, Artificial Intelligence, Adaptive Systems, and Control Theory. Yerramalla earned his Ph.D. at West Virginia University in Electrical Engineering, and holds a Masters Degree in Control Systems Engineering from West Virginia University Institute of Technology.