Stability Monitoring and Analysis of Learning in Adaptive Systems

Sampath Yerramalla, Martin Mladenovski, and Bojan Cukic
sampath,cukic@csee.wvu.edu
Department of Mathematics
West Virginia University
Morgantown, WV 26506

Edgar Fuller
ef@math.wvu.edu
Department of Mathematics
West Virginia University
Morgantown, WV 26506

Abstract
Increasing interest in adaptive computing systems implies the growing need for safe and reliable learning platforms. The ability to assure reliability of adaptation is especially important in safety-critical applications. Traditional software V&V techniques cannot account for the time-evolving nature of a system, making them inapplicable for adaptive computing system assurance. We propose considering stability of adaptation as a heuristic measure of reliability. This paper presents a stability monitoring system for detecting unstable learning behavior during online operation of adaptive system. The stability monitoring system comprises of several Lyapunov-like functions that detect distinct states in learning that bifurcate away from stable behavior. Murphy’s rule based on Dempster-Shafer theory is applied for combining stability information provided by individual monitors into an easily interpretable stability belief representation.

The proposed analysis technique is evaluated using online learning experiments based on data generated by an actual adaptive flight control system. Results indicate that the stability monitoring system detects unstable learning conditions. Furthermore, they provide an insight into understanding if and when the learning algorithm will converge back to a stable state after experiencing unstable learning conditions, which correspond to aircraft failures.

Keywords: Verification and Validation (V&V), Adaptive Systems, Safety Critical Systems, Flight Control, Stability Monitoring, Neural Networks, Lyapunov Theory.

I. INTRODUCTION

Autonomy and ubiquity are two important application paradigms that may require the system to adapt dynamically to its environment. Two general approaches exist in implementing software adaptation, the compositional approach and parameter adaptation [27]. Compositional approach is based on structural reconfiguration of software components at run time. In the parameter adaptation paradigm internal software variables that determine system’s behavior are modified automatically. These parameter modifications are usually achieved through judicious application of machine learning algorithms.

In this paper we deal with parameter adaptation and use its more specific definitions that emerged in the aerospace community. We call a system an adaptive system if it has a means to sense its environment, process information in a timely (usually real-time) manner, monitor and optimize its performance by adjusting its variables using closed-loop actions. Online adaptive systems are characterized by the potential for learning during system’s operation, i.e., following the deployment. Neural networks or other machine learners are typically the components of online adaptive systems.

Safety-critical applications of online adaptive systems include providing fault-tolerant control capabilities, maintaining distributed networks, developing unmanned vehicles, autonomous agents and high security devices. There is a growing need for providing safe and reliable learning software for adaptation in safety-critical systems [7],

- Contact author: Sampath Yerramalla; sampath@csee.wvu.edu
- Submission category: Regular paper
- Approximate word count: 5,500
[21], [22]. Verification and Validation (V&V) research is the key missing link in the wider deployment of adaptive systems, especially safety critical ones. When learning algorithm is used for performing online adaptation, its behavior has direct consequence on the adaptive system performance into which it is embedded. Therefore, it is necessary to understand, predict and assure the behavior of the adaptive learner before its actual deployment into the safety critical system.

From the surface, systems ability to adapt seems appealing. However, changes in system behavior lead to uncertainty, making most traditional V&V techniques inapplicable. Paper’s contribution is in presenting a non-conventional validation technique for stability monitoring and analysis of learning behavior in adaptive systems.

Unlike in other software programs, online learning process strongly depends on the nature of the inputs. The system evolves stochastically over time in a seemingly unpredictable manner. Theoretical analysis of online learning underlying adaptive systems proves to be a complex task. The stochastic nature of online learning also makes defining and distinguishing stable learning from unstable or unsafe learning difficult. V&V experts cannot easily predict the data patterns that need to be learnt, as they change to reflect the application environment.

Lyapunov theory is a well-established mathematical theory of stability analysis. It is often used for the stability analysis of dynamical systems. Our approach is based on the idea of extending its use for testing and analyzing online learning. We have seen some success in providing theoretical stability results that guarantee the correct behavior of an online learning neural network using Lyapunov stability theory. But the delineated stability boundaries are confined for cases of online learning where target data sets remain consistent [8], [9]. In order to be tractable, analytical analysis must ignore continual changes in the environment. But these changes are essential, as the learning algorithm may encounter drastically different data representations over time. Constant adaptation to inconsistent representations leads to stressed learning conditions and, generally, a loss of stability properties [8], [9]. But experiments show that learning is actually stable most of the time. The short periods of instability are, however, a major safety concern. The real V&V challenge, is to be able to monitor, analyze and detect unstable learning behavior during the operation of adaptive system under changing environmental conditions. A technique that would be able to predict the “return to stability” state is highly desirable too.

The proposed stability monitoring approach comprises of monitors that detect anomalies in different aspects of online learning. Monitors detect distinct states in learning that bifurcate away from stable behavior. As the final result, the proposed monitoring system provides a visual interpretation of the stability conditions of learning during the online operation of adaptive system. Making an easily-interpretable stability result available in realtime can overcome the difficulty of analyzing the complex stability information provided by individual monitors. Considering the information provided by monitors as beliefs of stable learning, we applied Murphy’s rule and fuzzy logic techniques for combining beliefs from individual monitors. Murphy’s rule of evidence combination is based on Dempster-Shafer theory and was developed for information fusion in autonomous robots [28], [29].

The paper describes the development of a stability monitoring system for detecting unstable learning conditions during the online operation of adaptive system. Our research was performed in the context of an adaptive flight control system project [6]. Adaptivity is required to accommodate unusual and arguably rare system conditions, such as the malfunction or loss of an aircraft’s control surface, thus providing a robust fault-tolerance mechanism. Such system-level fault tolerance application requires thorough validation. The context of our research is a very specific flight control system. The learning algorithm we studied is a specific type of a self-organizing neural network, the Dynamic Cell Structure (DCS) [1–3], [6]. However, we strongly believe that the concept of stability monitoring, which we applied to an adaptive neural network, is new and generic. It can be applied to many different learning algorithms and used to safeguard their application in safety critical adaptive systems.

This paper presents novel concepts in adaptive system assurance that may require familiarity with the online learning neural networks and a generic understanding of Lyapunov theory. Therefore, Section 2 contains a brief description of online learning and the limitation of theoretical stability analysis. Section 3 describes the development of a stability monitoring system. Section 4 describes a data-fusion technique for combining stability-information from individual monitors into a stability-belief. Section 5 describes the case study and results from evaluating the stability monitoring technique for online learning in an adaptive flight control system. Conclusions are drawn in Section 6 with a proposition to extend the validation approach to other adaptive systems.
II. ONLINE LEARNING

The material presented in this section directly relates to the learning concepts related to neural networks. Neural networks are by far the most frequently used adaptation tool currently utilized in practical applications.

A. Stable Online Learning

The difficulty in imposing strong stability restrictions for nonlinear systems was realized as early as a century ago by a Russian mathematician A. M. Lyapunov. Details on Lyapunov’s stability analysis technique for nonlinear discrete systems can be found in [26]. Lyapunov’s direct method or Lyapunov’s second method can be easily and systematically applied to validate the existence (or non-existence) of stable states in a system.

Without presenting a formal definition, a system is said to be stable near a given solution if one can construct a Lyapunov function (scalar function) that identifies the regions of the state space over which such functions decrease along some smooth trajectories near the solution. In simple terms, the system is stable if all solutions of the state that start nearby end up nearby. A good distance measure of nearby must be defined by a Lyapunov function \( V \) over the states of the system. By constructing \( V \), we can guarantee that all trajectories of the system converge to a stable state. The function \( V \) should be constructed keeping in mind that it needs be scalar \( (V \in \mathbb{R}) \) and should be non-increasing over the trajectories of the state space.

In mechanical systems a Lyapunov function is considered as an energy minimization term, in economy and finance evaluations it is considered as a cost-minimization term, and for computational purposes it can be considered as an error-minimization term. Though this concept was intended for mathematics of control theory, Lyapunov stabilization in a general sense can be used for stability analysis of many systems. To the best of our knowledge, we are the first to use it in the analysis of learning algorithms. In this case, Lyapunov function will be defined as an error minimization term over the successive epochs of neural network learning.

B. DCS

The online learning in the adaptive flight control system used later in the case study is realized by Dynamic Cell Structures (DCS), a self-organizing neural network. Figure 1 shows the block diagram describing the online learning algorithm using DCS neural network.

DCS is a topology preserving self-organizing neural network, used for unsupervised function approximation in a fault-tolerant adaptive flight control system [6], [21], [22]. DCS consists of a network of interconnected processing units (neurons). Neurons are positioned in the cartesian coordinate space and represented using their centers, \( w_i | i \in \{1, 2, \ldots, N\} \). Neurons are connected to each other by weighted lateral connections with strengths, \( c_{ij} | i, j \in \{1, 2, \ldots, N\} \). \( N \) is the total number of neurons in the neural network. Discussing details relative to DCS algorithm and its properties is beyond the scope of this paper. For the description of online learning and DCS neural networks the reader is referred to [1–3], [5], [8], [9], [21]. Based on the existing literature describing DCS, the following basic definitions can be formulated. These definitions describe the process of DCS learning and are used in the later part of the paper to describe the development of a stability monitoring system.

**Definition 1 Best Matching Unit (bmu):** A neuron \( i = \text{bmu} \) of the network with center \( w_{\text{bmu}} \in W \subset O \subset \mathbb{R}^D \) is called the best matching unit for a given input element \( m \in I \subset \mathbb{R}^D \) if it is closer to the input element than any other neuron of the network.

\[
i = \text{bmu}, \quad \|m - w_{\text{bmu}}\| < \|m - w_i\|, \quad \forall i \in \{1, 2, \ldots, N\}
\]  

(1)

**Definition 2 Second Best Unit (sbu):** A neuron \( i = \text{sbu} \) of the network with center \( w_{\text{sbu}} \in W \subset O \subset \mathbb{R}^D \) is called the second best unit for a given input element \( m \in I \subset \mathbb{R}^D \) if it is second closest to the input element than any other neuron of the network.

\[
i = \text{sbu}, \quad i \neq \text{bmu}, \quad \|m - w_{\text{sbu}}\| < \|m - w_i\|, \quad \forall i \in \{1, 2, \ldots, N\}
\]  

(2)
**Definition 3 Neighbor and Neighborhood (nbr):** Two neurons $i, j$ of the network are said to be neighbors if they have a non-zero lateral connection strength $c_{ij}$ existing between them. Neuron $i$ is then said to be the neighbor of $j$ and vice versa. The set of all neighbors of a neuron $i$ is known as its neighborhood $\{\text{nbr}(i)\}$.

$$j \in \{\text{nbr}(i)\}, \ c_{ij} \neq 0, \forall \ j \in \{1, 2, \ldots, N\}$$  \hspace{1cm} (3)

$N$ represents the number of neural units in the neural network at any time during its adaptation. A euclidean metric serves as the basis for measuring closeness.

The map generated by neural network during its course of online learning can be represented using a center matrix, $W | w_i \in W \subset \mathcal{O} \subset \mathbb{R}^D$, and a weighted connection matrix, $C | c_{ij} \in C \subset \mathcal{O} \subset \mathbb{R}^D$. $\mathcal{O}$ is the output space in which the neural net generates its network map. The map generated can be considered as a bi-directional graph consisting of $N$ neural units, $G(W,C,N)$. From a dynamical system point of view, a neural network map generated in online learning can be defined as follows.

**Definition 4 OLNN Map:** An online learning neural network map, $G(W,C,N,t_n)$ for a given feature manifold, $\mathcal{I} \subset \mathbb{R}^D$, is an $N^{th}$ order network representation of $\mathcal{I}$ in the output space, $\mathcal{O} \subset \mathbb{R}^D$ that is generated by assigning $N$ neural units in $t_n$ steps of the training algorithm.
C. Analytical Stability Analysis

The goal of analytical stability analysis is to guarantee that learning algorithm converges to a stable state within a "reasonable" amount of time when exposed to a representative set of data. Several results on convergence of neural network learning have been proved so far [4], [20]. We were successful in showing that Lyapunov theory can be used for validating the existence (or non-existence) of stable states for online adaptation by DCS algorithm. This result holds for cases when the feature manifold (the input data) remains relatively consistent [8], [9]. The result has been obtained by considering online learning in adaptive systems as a dynamical system.

Most mathematical theories of stability for dynamical systems require a solution for the differential equations that govern the system's behavior. When adaptive systems alter their behavior to compensate for a change in environment, the model associated with it will likely change. Therefore, finding a solution for the differential equations governing learning behavior in adaptive systems would be a complex task. In most cases, there is no guarantee for the existence of a solution for higher order differential equations. On the contrary, Lyapunov's theory relies on the construction of system specific Lyapunov functions (a function of the system’s state) to account for system stability (or instability). It does not assume that system’s functionality is static, but rather dynamic. The application of Lyapunov theory to test the behavior of online learning opens a possible approach to adaptive software assurance. One of the most compelling factors for applying Lyapunov theory for validating the learning behavior in adaptive software is its ability to impose strong stability restrictions. Lyapunov’s second method is a highly respected mathematical theory of stability analysis that is often used to analyze stability behavior of nonlinear systems [24–26]. Our approach of validating learning behavior in adaptive software is based on the concept of extending the use of Lyapunov theory to monitor stability properties of online learning. We will use the stability and convergence results as heuristic measures of "correctness" of online learning. The following theoretical stability result was proved in our earlier work [9]. This stability result guarantees stable behavior of a DCS neural network when exposed to a specific (static) data representation.

\textbf{Theorem 1} Let $V(G(W,C), n)$ be a scalar function that is constructed for the map $G(W,C)$ generated by DCS learning from a static input manifold $M$. Given $\epsilon > 0$ there is an integer $K > 0$ such that for all $n > K$ we have $V(G(W,C), n) < \epsilon$.

\textit{Proof:} For a detailed proof, the author is referred to [9].

Such a proof of learning stability for a neural network guarantees its correct behavior. In other words, when presented with a fixed input data set, the DCS neural network will approximate it with the desired precision in a finite number of learning cycles (epochs). Here, the stability of learning is a heuristic measure of the correctness of learning. For self-organizing neural networks such as DCS the dynamics are complicated, prohibiting the extension of this proof to variable input data sets (the data sets which may change in between the successive learning epochs). The delineated stable learning conditions are confined to cases of online learning where data representations remain consistent over time [8], [9]. During the operation of adaptive system within a continually changing environment, online learning may encounter inconsistent and/or radically different input representations. Continual adaptation to inconsistent data representations leads to a loss of stability properties [8], [9].

III. Stability Monitoring System

The previous section established that stability of learning is an important consideration when working with online learning components in safety critical applications. In absence of an analytical proof, the following are the goals that must be achieved by any stability monitoring scheme for an online learning algorithm.

- During the operation of adaptive systems under constantly changing environmental conditions:
  - Does online learning tend to become unstable?
  - Is it possible to detect unstable learning behavior?
- After experiencing unstable learning conditions:
  - Does the online learning converge back to a stable state?
  - How long does it take for the learner to converge back to stability?
Due to model uncertainty, adumbrating stability boundaries or predicting the timing of learning conditions that diverge from stability is not currently possible. Therefore, we cannot provide answers to the above questions analytically. But this does bring the prospects for the validation of adaptive systems into a condition of despair. Based on our experimental success, we believe that developing Lyapunov-like functions to monitor and detect online learning errors in real time can resolve this problem. In the context of the analysis of adaptation, the goal is to develop an effective adaptation monitoring tool.

We designed and built such a stability monitoring system. It monitors and detects unstable learning behavior during the operation of the adaptive flight control system’s software. Based on our ongoing research, four monitors were developed in an effort to diagnose unstable behavior of neural network learning. Different monitors analyze the positions of BMU, SBU, NBR, and NON−NBR in the cartesian coordinates. The definition of these concepts was provided in equations 1,2, and 3. We defined learning stability monitors based on the following mathematical formulations.

**Definition 5 Monitor #1, BMU**: BMU monitor measures the Euclidean distance between each data element of the presented input (training) data pattern \( m \in M \subset \mathbb{R}^D \) and its closest neuron (node) of the neural network, known as the Best Matching Unit \( w_{BMU}(m) \in W \subset \mathbb{R}^D \).

\[
Monitor\#1 = \sum_{m \in M} \|m - w_{BMU}(m)\|
\]

**Definition 6 Monitor #2, SBU**: SBU monitor measures the Euclidean distance between each data element of the presented input (training) data pattern \( m \in M \subset \mathbb{R}^D \) and its second closest neuron (node) of the neural network, known as the Second Best Unit \( w_{SBU}(m) \in W \subset \mathbb{R}^D \).

\[
Monitor\#2 = \sum_{m \in M} \|m - w_{SBU}(m)\|
\]

**Definition 7 Monitor #3, Neighborhood (NBR)**: Neighborhood monitor represents the mean Euclidean distance between each data element of the presented input (training) data pattern \( m \in M \subset \mathbb{R}^D \) and the set of neighborhood neurons (connected nodes) of the BMU of neural network, known as the NBR-set \( w_{\{NBR\}}(m, BMU) \in W \subset \mathbb{R}^D \).

\[
Monitor\#3 = \sum_{m \in M} mean\{\|m - w_{\{NBR\}}(m)\|\}
\]

**Definition 8 Monitor #4, Non-Neighborhood (Non-NBR)**: Non-NBR monitor represents the mean Euclidean distance between each data element of the presented input (training) data pattern \( m \in M \subset \mathbb{R}^D \) and the set of laterally connected, non neighboring neurons of the BMU of neural network, known as the Non-NBR-set \( w_{\{Non-NBR\}}(m, BMU) \in W \subset \mathbb{R}^D \).

\[
Monitor\#4 = \sum_{m \in M} mean\{\|m - w_{\{Non-NBR\}}(m)\|\}
\]

The online learning algorithm consists of several parts, including Kohonen learning, Hebb update, resource update, growing neurons and pruning neurons, as shown in Figure 1. We noticed that it was not possible to capture stability information from all states of online learning using a single monitor. As an example, let us consider a typical situation in our case study of a failed flight control surface condition. The DCS is presented with 200 frames of data. The first 99 frames represent the data generated under nominal or no-failure conditions. The 100th frame of data represents data generated as the system fault had been injected. So, around the 100th frame of data, the environment will change from nominal to stressed. Because the DCS network is embedded in the flight control system, when it encounters inconsistent data representation (100th frame), it will likely deviate from a stable learning state. A stability monitoring system comprising of 4 monitors has been deployed to detect unstable learning conditions. An unstable learning condition is indicated by a spike (abrupt increase) in the values of monitors. Figures 2(a),2(b),2(c), and 2(d) show values from the 4 monitors after presenting the 100th frame of data. Notice a spike (abrupt rise) as
indicated by the monitors # 3 and 4 around the 100th time frame. At this instance of online learning, monitors # 3 and 4 detected errors (instability) in online learning. Monitors # 1 and 2 did not detect unstable states of learning.

This study demonstrates that stability monitoring can detect unstable learning behavior in changing environmental conditions. This can significantly enhance our ability to analyze and understand the performance of an adaptive software system. In our case study, we observed that no single monitor can diagnose all unstable learning behaviors. Our ongoing research indicates that stability monitoring successfully detects unstable learning behavior in tests runs using data generated under different nominal flight scenarios and failure conditions. This result is summarized in Table III, where a check mark (✓) indicates that a spike in the monitor is observed. A spike in any or all of the monitors indicates successful detection of unstable learning condition. The results in this table are based on thousands of test runs of stability monitoring system using simulated flight data.

Different monitors analyze different aspects of online learning. This complements and asserts our initial proposition of using a multiple monitors for detecting unstable learning behavior.

### IV. Combining Stability Information

In the previous section, we described a stability monitoring system comprised of four different monitors. Drawing a conclusion based on analyzing stability-information provided by each individual monitor would be a complex task.

![Online Data Frame](image)

Fig. 2. On-line stability monitoring system after presenting data from failed-flight conditions

**TABLE I**

<table>
<thead>
<tr>
<th>Mode #</th>
<th>Failed-Flight Conditions (change in environmental condition)</th>
<th>Stability Monitoring System</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Left stabilator, locked at 0°</td>
<td>✓</td>
</tr>
<tr>
<td>2</td>
<td>Left stabilator, locked at +3°</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>Left stabilator, locked at −3°</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>Right aileron, locked at +3°</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>Right aileron, locked at −3°</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>Left stabilator, missing by 50%</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>Right aileron, missing by 50%</td>
<td>✓</td>
</tr>
</tbody>
</table>

* ✓ represents that a predominant spike is noticed in the monitor value when presented with data generated under changing environmental conditions indicating successful detection of unstable learning behavior.
Making a single, easily-interpretable stability result available in realtime can overcome the difficulty of analyzing the stability information provided by individual monitors. Therefore, this section describes a monitor fusion technique. We utilize Murphy’s rule based on Dempster-Shafer theory. Before we adopted it, it was used for combining evidence from different sources of information in autonomous robots [28], [29].

In the literature a significant research effort has been placed into the development of data-fusion methods and algorithms. Some of the commonly used techniques for combining related information from different sources are based on Bayesian theory [30], Dempster-Shafer theory, [28], [29] and fuzzy logic [31], [32]. Due to realtime restrictions in our application domain, we had to modify existing data-fusion techniques and improve their computational efficiency. In order to apply Murphy’s rule, the stability information provided by monitors needs to be considered in terms of belief functions. This is accomplished by the normalization process.

A. Preprocessing Data

Prior to the implementation of Murphy’s rule for combining stability information from individual monitors, the data provided by the monitors needs normalization. The normalization of monitor-values must be performed in an online manner such that the minimum and maximum error-values after normalization lie between 0 and 1. The idea behind normalization is to transform the stability information into probabilistic measures of stability beliefs. In order to accomplish this 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. The normalized stability information obtained in this manner can be considered probabilistic measures of belief in stability of learning.

During the early epochs of learning, it is apparent that the values from the monitors are high. Our neural networks are not pretrained and it takes a certain number of learning cycles to be able to achieve a basic representation of the input data set. Over time, as learning progresses towards a better representation, the values from monitors will decrease. If online learning loses its stability properties, the current maximum of a monitor value will be close to its overall maximum. Therefore, the normalized values from monitors will spike (show sharp, abrupt increase) to a value close to 1. This can be used as an indication to show that the current conditions within the neural network are unstable. Note that the network will likely periodically lose its stability properties due to constant adaptation to inconsistent data representations in changing environmental conditions.

B. Data-Fusion Scheme

In order to apply Murphys rule for combining evidence from distinct sources of information, a frame of discernment \( \Theta \) needs to be created. If all monitors provide an independent belief of the stability conditions in online learning, the following propositions can be made.

- \( E \rightarrow \) how unstable is current learning state.
- \( C \rightarrow \) how much trust can be asserted in the current learning state.

These propositions represent opposite beliefs and represent disjoint and compliment sets.

\[
C = \overline{E}, \quad C \cap E = \Phi
\]

We consider the preprocessed stability-information from monitors to be the beliefs (basic probability measures) of the proposition \( E \), represented as \( m(E) \). Consequently, the assigned belief to proposition \( C \) is \( m(c) = 1 - m(\overline{E}) \). The total assigned belief for all propositions is 1. To demonstrate the concept of evidence combination from two monitors, let \( m_1 \) and \( m_2 \) be their respective error probability assignments. The following table shows the orthogonal combination of the two probabilities.

<table>
<thead>
<tr>
<th>( m_1/m_2 )</th>
<th>( E )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( E )</td>
<td>( m_1(E) \cdot m_2(E) )</td>
<td>( \Phi )</td>
</tr>
<tr>
<td>( C )</td>
<td>( \Phi )</td>
<td>( m_1(C) \cdot m_2(C) )</td>
</tr>
</tbody>
</table>
Based on this table, the combined belief functions for the propositions $E$ and $C$ can be written as follows:

$$m_{12}(E) = \frac{(m_1(E) \cdot m_2(E))^n}{(m_1(E) \cdot m_2(E))^n + (m_1(C) \cdot m_2(C))^n}$$

for the proposition $E$,

$$m_{12}(C) = \frac{(m_1(C) \cdot m_2(C))^n}{(m_1(E) \cdot m_2(E))^n + (m_1(C) \cdot m_2(C))^n}$$

for the proposition $C$.

There are implementations of Murphy’s rule where $n$ is adapted along the process. However, in the context of the system being validated, $n$ is chosen to be a constant and assigned a value of 0.5. 0.5 is a neutral value as we had no good understanding of the dependence between the monitors. Table II demonstrates that combined belief functions, $m_{12}$ for propositions $C$ and $E$ are related, $m_{12}(C) = 1 - m_{12}(E)$. The normalized values from monitors represent basic probability values for proposition $E$. Therefore, the beliefs for proposition $E$ are first combined and used in calculating the belief for proposition $C$. While combining evidence using the above rule, 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 $m_1(E), m_2(E), m_3(E) \ldots m_k(E)$ represent the assigned beliefs for proposition $E$ from Monitor # 1, Monitor # 2, Monitor # 3, …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 as shown in Figure 3(a). This method applies to cases where the value of $k$ is even. A more general 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, is shown in Figure 3(b). 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.

![Diagram](a) A binary tree strategy for combining beliefs (b) A cascade strategy for combining beliefs

Fig. 3. Methods of Combining Beliefs for use in Murphy’s Rule

In the context of analysis of learning, 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 confidence interval that includes combined beliefs from all possible combinations. During the combination of monitor values as shown in Figure 3(b) 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 too expensive computationally as it would limit the application of this methodology to realtime systems. In order to overcome this difficulty, we claim the following.
**Theorem 2** Combining beliefs in the increasing order of their values using a cascade combination approach of Figure 3(b) will result in the maximum of all combined beliefs. Similarly, cascade combination of beliefs in the decreasing order of their values will result in the minimum of all combined beliefs.

**Proof:** A proof is given in the Appendix of the paper.

This theorem significantly reduces the computational complexity of combining stability belief functions from individual monitors, as the most expensive operation becomes the sorting of \( k \) beliefs. A single, easily-interpretable stability result can, therefore, be made available in \( O(k \times \log k) \), where \( k \) is usually a small number of monitors.

V. **CASE STUDY**

This section demonstrates the practicability of the proposed stability monitoring and analysis techniques. The proposed technique is used for testing and analysis of online learning during the operation of an adaptive flight control system under failed flight conditions. The failed flight conditions represent the adaptive component of flight control system with unexpected (and possibly previously unseen) environmental condition.

In recent years, NASA conducted a series of experiments evaluating soft computing paradigms (neural networks, AI planners) for providing fault tolerance capabilities in flight control systems following sensor and/or actuator failures [6], [7], [21], [22]. One of these programs is **Intelligent Flight Control System** (IFCS). Figure 4(a) depicts the architecture of NASA’s first generation IFCS implementation. The system was developed to evaluate novel flight control concepts based on emerging soft-computing techniques for optimized system performance in multiple accident and/or off-nominal flight scenarios. The online learning neural network, OLNN, implemented by the previously described DCS algorithm, is the adaptive component requiring nonconventional validation techniques. It needs to accommodate for changing dynamics of the aircraft that exceeds traditional robustness limits in order to provide optimized system performance. The heuristic results presented here are based on the analysis of online learning in the context of IFCS. The proposed stability monitoring and analysis techniques were evaluated using experiments with data generated using a high fidelity F-15 adaptive flight control simulator, built at West Virginia University. Figure 4(b) shows the user interface of this simulator [23]. The simulator reflects the architecture and functionality of the IFCS system of Figure 4(a). The simulator is capable of simulating both nominal and failed flight conditions. The data generated by the simulator serves as the input to the online learning neural network. In actual system, there are 5 DCS neural networks operating in parallel on different subsets of sensor data and flight derivatives. In this paper, we report only on the performance of one of the 5 neural networks. Its input data set is 7-dimensional. Each of these 7 dimensions represents a specific control parameter that needs to be learnt and represented. 20 seven dimensional data frames are presented to this DCS neural network every second, corresponding to a frequency of 20 Hz.

During the simulation of failed flight conditions, the simulator generates 200 frames of data corresponding to 10 seconds of flying time. Initially the flight is run under nominal (no failure) conditions for the first 5 seconds. Thereafter, (at the 5th second), a failure is injected. Consequently, the first 99 frames of data represents nominal
flight conditions. The data following the 100th frame represents failed flight conditions. The simulated failed flight condition used in the experiments herein is the the loss of 50% of aircraft’s left stabilator. Apparently, after 99 frames system environment, sensed by sensor inputs and their derivatives, changes and the DCS network needs to accommodate the new conditions.

At the 99th frame, the DCS neural network is likely to deviate to an unstable state of learning. The goal of the proposed stability monitoring system is to detect instances in learning that bifurcate away from stable behavior. Figures 5(a), 5(b), 5(c), and 5(d) depict the normalized measures of learning stability obtained from the individual monitors. These monitors depict stability conditions in online learning under nominal conditions (prior to fault injection in frame 100). All 4 monitors show declining values of learning instability and indicate no spikes. In other words, following the initial dozen frames of learning, the monitors indicate no unstable behavior in the learning of DCS neural network.

For easy-interpretation, the stability information provided by individual monitors is combined into a single stability result using the discussed combination method. The combined stability result generated by Murphy’s rule is shown in Figure 6(a). Note that the maximum and minimum stability beliefs shown in Figure 6(a) are obtained by combining stability-information from individual monitors in the increasing and decreasing order of their values, respectively. A fuzzy logic inference system based on “if-then” Mamdani rules has been developed to provide a comparison with the stability results obtained by the Murphy’s rule. The fuzzy inference system is given the same information from individual monitors as the Murphy’s rule. Figure 6(b) shows the corresponding combined stability measure obtained using a fuzzy inference system. As shown in both Figure 6(a) and 6(b), the combined stability belief after the introductory learning phase remains steadily close to 1 thus indicating stable learning behavior.

Figures 7(a), 7(b), 7(c), and 7(d) also show the normalized stability measures obtained from the individual monitors. However, the monitors depict stability conditions associated with the DCS adaptation before and after presenting the 100th frame of data. Note that that point a failure is injected and system operation switches from nominal (no-failure) to a failed condition. Spikes in the monitors around the 100th time frame are obvious. This is an indication that online learning behavior is bifurcating to unstable conditions.

The combined stability result generated by Murphy’s rule is shown in Figure 6(a). Figure 6(b) shows the corresponding stability measure obtained using fuzzy inference system. As shown in both Figure 6(a) and 6(b), the stability belief obtained by combining stability information from individual monitors indicates a sharp decline to a
value close to 0 shortly after the $100^{th}$ frame. This decline is an indication that the online learning is bifurcating away from stable behavior. However, the learning algorithm soon settles into another stable state around $140^{th}$ time frame.

Our ongoing research indicates that the proposed stability monitoring and analysis techniques provide successfully detection and easy-interpretation of unstable learning behavior for data generated under various operational conditions of the adaptive system. These results are summarized in Table III. Results from this table indicate that the combined stability belief drops sharply in Modes # 1-7 when learning under stressed system conditions. Table III also provides heuristic comparison of combined stability beliefs obtained using Murphy’s rule to those obtained through the fuzzy inference system.

This study demonstrates the ability of the proposed stability monitoring technique to detect unstable learning conditions during online operation of the adaptive system under changing environmental conditions. We also demonstrated that having a stability monitoring system for detecting abnormal (unstable) learning behavior can significantly enhance system understanding and, consequently, provide a meaningful dependability attribute. Probably one of
Fig. 8. Combined stability beliefs in online learning using Murphy’s combination rule and fuzzy inference system after presenting data from failed flight conditions

TABLE III

TEST RESULTS FROM EVALUATING BELIEF OF STABILITY IN ONLINE LEARNING DURING THE OPERATION OF ADAPTIVE FLIGHT CONTROL SYSTEM UNDER CHANGING ENVIRONMENTAL CONDITIONS (FAILED-FLIGHT CONDITIONS).

<table>
<thead>
<tr>
<th>Mode #</th>
<th>Failed-Flight Conditions (change in environmental condition)</th>
<th>Stability Belief Using Murphy’s Rule Before Failure (Min, Max)</th>
<th>Stability Belief Using Fuzzy Inference Before Failure</th>
<th>Stability Belief Using Murphy’s Rule After Failure (Min, Max)</th>
<th>Stability Belief Using Fuzzy Inference After Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Left stabilator, locked at 0°</td>
<td>.97, .98</td>
<td>.91</td>
<td>.47, .66</td>
<td>.69</td>
</tr>
<tr>
<td>2</td>
<td>Left stabilator, locked at +3°</td>
<td>.98, .99</td>
<td>.92</td>
<td>.01, .13</td>
<td>.3</td>
</tr>
<tr>
<td>3</td>
<td>Left stabilator, locked at −3°</td>
<td>.98, .99</td>
<td>.92</td>
<td>.54, .66</td>
<td>.42</td>
</tr>
<tr>
<td>4</td>
<td>Right aileron, locked at +3°</td>
<td>.98, .99</td>
<td>.92</td>
<td>.22, .52</td>
<td>.5</td>
</tr>
<tr>
<td>5</td>
<td>Right aileron, locked at −3°</td>
<td>.98, .99</td>
<td>.92</td>
<td>.38, .49</td>
<td>.51</td>
</tr>
<tr>
<td>6</td>
<td>Left stabilator, missing by 50%</td>
<td>.96, .98</td>
<td>.91</td>
<td>.08, .32</td>
<td>.39</td>
</tr>
<tr>
<td>7</td>
<td>Right aileron, missing by 50%</td>
<td>.97, .98</td>
<td>.91</td>
<td>.29, .58</td>
<td>.65</td>
</tr>
</tbody>
</table>

the most interesting results of the case study was the ability to measure the time needed to stabilize the aircraft’s control system following the injection of a fault. This is one of the most important safety arguments presented at the flight readiness review board. Our monitoring technique provided one of the first known practical and meaningful systemic insights into the neural network adaptation process.

VI. Conclusions

In this paper, we described a set of techniques suitable for dependability analysis of adaptive software. The stability monitoring approach treats learning as a dynamical system and explores its stability during system’s operation. We demonstrated that building such stability monitoring systems is feasible and the results of stability monitoring are useful in system analysis and validation. Through the application of these concepts in a realistic and complex case study, we learned that no single monitor will likely be able to diagnose unstable learning behavior for all environment conditions in an adaptive system. Further, we proposed and implemented an information fusion algorithm that combines individual stability monitors into an easy-to-interpret stability belief function. The combined stability-belief obtained by the application of Dempster-Shafer theory provides answers to questions if and when will learning converge back to a stable state after experiencing unstable environmental conditions.

In practice, the stability analysis technique presented here can provide invaluable information to system assurance teams and assist them in the validation of learning in online adaptive systems. The proposed testing and analysis technique is not model-based, and it overcomes the difficulty of model-uncertainty associated with online adaptive system assurance. It can therefore be adjusted for use in validation of other adaptive systems. At this time it is likely that stability monitors similar to the ones described in this paper will be implemented in the next generation of the Intelligent Flight Control Systems program conducted by NASA.
**APPENDIX**

**Proof of Theorem 2**

Proof: For simplicity reasons, let the beliefs from the monitors be labeled as $a_1 = m_1(e), a_2 = m_2(e), a_3 = m_3(e), \ldots a_k = m_k(e)$. Murphy's rule for beliefs $x$ and $y$, $m(x, y)$ is then rewritten in the following manner.

$$m(x, y) = \frac{(x \cdot y)^n}{(x \cdot y)^n + (1 - x)^n(1 - y)^n}$$ (3)

Note that $x, y \in (0, 1), n \in [0, 1]$, and $m(x, y) \in (0, 1)$. Consider the following lemma that is later used in proving the theorem.

**Lemma 1** Let $a, b, c \in (0, 1)$ be real numbers so that $a \leq b \leq c$. Let $n$ be a real number, $n \in [0, 1]$. Suppose there exists a relation $\gtrless$ such that $\gtrless(a, b, c) = m(m(a, b), c)$, where $m()$ is given in 3. Then the inequality $\gtrless(a, b, c) \geq \gtrless(a, c, b)$ holds.

Proof: Using Murphy’s rule as given in 3 results in the following for function $\gtrless(a, b, c)$ and $\gtrless(a, c, b)$ after simplification.

$$\frac{1}{\gtrless(a, b, c)} = 1 + (\frac{1}{a} - 1)^n(\frac{1}{b} - 1)^n(\frac{1}{c} - 1)^n$$
$$= 1 + \mathcal{P}(a, b, c)$$

$$\frac{1}{\gtrless(a, c, b)} = 1 + (\frac{1}{a} - 1)^n(\frac{1}{b} - 1)^n(\frac{1}{c} - 1)^n$$
$$= 1 + \mathcal{P}(a, c, b)$$

$$\Rightarrow \frac{\gtrless(a, b, c)}{\gtrless(a, c, b)} = \frac{1 + \mathcal{P}(a, c, b)}{1 + \mathcal{P}(a, b, c)}$$ (4)

From 4 it is evident that in order to prove that $\gtrless(a, b, c) \geq \gtrless(a, c, b)$ it is sufficient if we show that $\mathcal{P}(a, b, c) \leq \mathcal{P}(a, c, b)$. Let us suppose that $\mathcal{P}(a, b, c) > \mathcal{P}(a, c, b)$, we then have the following implication.

$$\frac{\mathcal{P}(a, b, c)}{\mathcal{P}(a, c, b)} > 1$$

$$\Rightarrow \frac{(\frac{1}{a} - 1)^n(\frac{1}{b} - 1)^n(\frac{1}{c} - 1)^n}{(\frac{1}{a} - 1)^n(\frac{1}{b} - 1)^n(\frac{1}{c} - 1)^n} > 1$$

$$\Rightarrow \frac{(\frac{1}{a} - 1)^n}{(\frac{1}{c} - 1)^n} > 1$$ (5)

Since $n \in (0, 1)$ and $n - 1 > 0$, it implies from 5 that $b > c$. This contradicts our initial assumption that $a \leq b \leq c$. Therefore $\gtrless(a, b, c) \geq \gtrless(a, c, b)$.

This lemma shows that the theorem holds for a base case of $k = 3$ with beliefs $a, b, c$. We will prove that the theorem holds for the case of $k + 1$. Then by induction the proof will apply for any finite $k$. Combining beliefs $a_1, a_2, a_3, \ldots, a_k$ as shown in Figure 3(b) can be formulated as,

$$comb(1) = a_1$$
$$comb(i) = m(comb(i - 1), a_i), \quad i = 2, 3, \ldots, k$$

The final belief obtained by combining $k$ beliefs is represented by $comb(k)$. Since we are intending to prove that this theorem holds for $k + 1$, we need to show that combining $k + 1$ beliefs $a_1, a_2, a_3, \ldots, a_k$ using Murphy’s rule in the increasing order of their values results in a combined belief $comb(k + 1)$, which is the maximum of all possible orderings. Table IV shows possible orderings using $k + 1$ beliefs. These orderings are generated by placing each belief at the end and sorting the remaining $k$ beliefs in their increasing order.
TABLE IV

POSSIBLE ORDERINGS FOR \( k + 1 \) BELIEFS

<table>
<thead>
<tr>
<th>ordering #</th>
<th>sorted ( k - 1 ) beliefs</th>
<th>( k^{th} ) belief</th>
<th>( (k + 1)^{th} ) belief</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( a_1, a_2, \ldots, a_{k-1} )</td>
<td>( a_k )</td>
<td>( a_{k+1} )</td>
</tr>
<tr>
<td>i</td>
<td>( a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k )</td>
<td>( a_{k+1} )</td>
<td>( a_i )</td>
</tr>
<tr>
<td>k</td>
<td>( a_1, a_2, \ldots, a_k )</td>
<td>( a_{k+1} )</td>
<td>( a_k )</td>
</tr>
</tbody>
</table>

For each row \( i = 1, 2, \ldots, k \) of the table there is a corresponding \( \text{comb}_i(k) \) and \( \text{comb}_i(k+1) \) that can be expressed as follows.

\[
\begin{align*}
\text{comb}_i(k) &= m(\text{comb}_i(k-1), a_{k+1}) \\
\text{comb}_i(k+1) &= m(\text{comb}_i(k), a_i) \\
&= \mathcal{F}(\text{comb}_i(k-1), a_{k+1}, a_i)
\end{align*}
\] (6)

For row \( i = 0 \) of the table, the following relationship holds.

\[
\begin{align*}
\text{comb}_0(k) &= m(\text{comb}_0(k-1), a_k) \\
\text{comb}_0(k+1) &= m(\text{comb}_0(k), a_{k+1}) \\
&= \mathcal{F}(\text{comb}_0(k-1), a_k, a_{k+1})
\end{align*}
\] (7)

We need to show that \( \text{comb}_i(k+1) \leq \text{comb}_0(k+1) \) for each row \( i = 1, 2, \ldots, k \) of the table. Since \( a_i \leq a_{k+1} \), for all rows \( i = 1, 2, \ldots, k \) using Lemma 1.

\[
\text{comb}_i(k+1) \leq \mathcal{F}(\text{comb}_i(k-1), a_i, a_{k+1})
\]

from 6, \( \mathcal{F}(\text{comb}_i(k-1), a_{k+1}, a_i) \leq \mathcal{F}(\text{comb}_i(k-1), a_i, a_{k+1}) \) (8)

Using the induction hypothesis for \( k \) beliefs,

\[
m(\text{comb}_i(k-1), a_i) \leq \text{comb}_0(k) \\
\leq m(\text{comb}_0(k-1), a_k)
\]

Sine it is evident from 3 that \( m(x, y) \) is an increasing function,

\[
m(m(\text{comb}_i(k-1), a_i), a_{k+1}) \leq m(m(\text{comb}_0(k-1), a_k), a_{k+1})
\]

\[
\mathcal{F}(\text{comb}_i(k-1), a_i, a_{k+1}) \leq \mathcal{F}(\text{comb}_0(k-1), a_k, a_{k+1})
\]

from 7, \( \mathcal{F}(\text{comb}_i(k-1), a_i, a_{k+1}) \leq \text{comb}_0(k+1) \)

from 8, \( \text{comb}_i(k+1) \leq \text{comb}_0(k+1) \)

Since the claim is shown true for \( k = 3 \) and \( k + 1 \), by mathematical induction theory it holds for any finite \( k \). ■

REFERENCES
