A Framework for the Analysis of Deep Neural Networks in Aerospace applications using Bayesian Statistics

1st Yuning He
NASA Ames Research Center
Moffett Field, USA
yuning.he@nasa.gov

2nd Johann Schumann
KBR, NASA Ames Research Center
Moffett Field, USA
johann.m.schumann@nasa.gov

***Abstract*—Deep Neural Networks (DNNs) have gained tremendous popularity in many application areas over the recent years. Safety-critical applications as found in the aerospace domain require that the behavior of the DNN is validated and tested rigorously for system safety.**

In this paper, we present a framework to support testing of DNNs. Our framework employs techniques from statistical modeling and active learning to effectively generate test cases for DNNs used in Aerospace systems and also supports a comparison between different DNNs. In this paper, we will describe our statistical framework, the algorithms for model construction and the metric guiding the test case generation process.

We will present a case study on a physics-based Deep recurrent residual neural network (DR-RNN), which has been trained to emulate the aerodynamics behavior of a Boeing 747-100 aircraft.

***Index Terms*—Deep Neural Network, Verification and Validation, Bayesian Statistics, Active Learning**

I. Introduction

Deep Neural Networks (DNNs) have gained tremendous popularity in recent years and new relevant applications are continuously detected. DNN applications have even found their way into the aerospace domain, where Neural Networks are started to be used for navigation for autonomous drones [15], obstacle and collision avoidance, as well as compact and efficient representation of Aircraft Dynamics [21], just to mention a few examples. Review articles like [2] or [4] discuss numerous applications.

Most of these applications share an important commonality: they are highly safety- and mission critical. This means that a malfunction of the DNN could cause loss of human life, cause substantial damage on the ground, or at least can jeopardize the mission. Therefore, when applied to safety-critical tasks, the behavior of the DNN and its software implementation must undergo rigorous V&V (Verification and Validation) to ensure safe operations at all times.

The Aerospace industry and agencies have developed very stringent safety requirements and standards with regards to safety-critical software systems (e.g., DO-178C [17]). Depending on the level of safety-criticality, rigorous software development processes, certified tool chains, as well as different levels of structural code testing are required.

For example, for level A software, software testing must occur according to the strict MC/DC (Modified Condition/Decision Coverage) metric that requires numerous test cases for conditional code (if-then-else, or case statements).

However, the structure of the code for the use of a pre-trained DNN is extremely simple as it only consists of nested loops, but no elaborate conditional code. All the trained information is stored as weights in numerical arrays. Therefore, DNNs render traditional V&V techniques like code-coverage testing unsuitable for validation of DNN behavior.

In general, approaches for V&V and certification of DNN applications are still in their infancy [1]. Important V&V tasks for such systems include comprehensive testing the DNN after training, understanding the model as represented by the DNN, and assessing the DNN configuration and structure.

Understanding a DNN model is usually difficult because of the complexity of the DNN architecture and its distributed representation of information. Also, a DNN application is often constructed using "out of the box" configurations like the preset number, type, and size of layers without any prior study or understanding if these configuration parameters are good or even suitable in the given situation.

Usually, a DNN is trained off-line, i.e., before the deployment of the system. During operation, the model is not adjusted to new data. The situation gets more complicated for systems or missions that make it necessary to adapt toward novel and unexpected situations. An example might be operation of a spacecraft in an unknown environment, like a mission to the Europa moon. In such situations, the network must be trained on-line during the mission, but retraining of the DNN from scratch is often not feasible because it is very resource consuming. Furthermore, the suitability of the DNN architecture and model parameters needs to be monitored and, if necessary, improved. Otherwise, the network setup might lead to a potentially poorly performing neural network with slow convergence, limited scalability, or bad generalization behavior. There are only few recent approaches on effective on-line training of DNNs [12], [18], which try to address slow convergence and scalability issues.

In this paper, we present a framework to support testing and

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

analysis of a Deep Neural Network. We employ techniques from Active Learning and Computer Experiment Design to iteratively construct new test cases to exercise the DNN. Here, we focus on testing for deviations in the behavior of the neural network compared to the truth as obtained by a high-fidelity system simulator.

System requirements for such a system define acceptable deviations of behavior with respect to the truth. For example, for a DNN, which is learning the dynamics of an aircraft, acceptable deviations would be given as a maximal allowable error in the roll, pitch, and yaw rates. A test case, as generated by our framework would prescribe a parameterized scenario that is executed by the DNN and, in parallel, by the system dynamics simulator. Then, the deviation is calculated and compared against the limits given by the system requirements to determine if the deviation is acceptable or not.

This information is used to incrementally construct our statistical model. For the representation and construction of the model, we are using Dynamic Regression Trees (DynaTrees) [5], [19], a dynamic Gaussian process model based upon Particle Filters. DynaTrees are regression and classification learning models with complicated response surfaces in on-line application settings. DynaTrees create a sequential tree model whose state changes over time with the accumulation of new data, and provide particle learning algorithms that allow for the efficient on-line posterior filtering of tree-states. A major advantage of DynaTrees is that they allow for the use of very simple models within each partition. The models also facilitate a natural division in sequential particle-based inference: tree dynamics are defined through a few potential changes that are local to each newly arrived observation, while global uncertainty is captured by the ensemble of particles.

This surrogate model is initialized with available training data and incrementally refined using candidate data points that are produced by our active learning module. It evaluates the current surrogate model using a customized active-learning heuristics and suggests candidate data points that provide most information for model refinement. For these candidate points, the ground truth is obtained by using a system simulation or real data.

Our framework features customizable heuristics that allow the active learning to focus on particular characteristics of the model. Classical algorithms like ALM [13] or ALC [3] focus on under-explored regions in general of the domain space. Inspired by [9] and work on contour finding algorithms, we loosely follow [16] and define our boundary-aware metric boundary-EI [7], [8] that puts the focus of the search into regions, which are close to boundaries that separate regions, where deviations are acceptable from regions of high, and potentially unsafe deviations.

For the purpose of *analysis and testing of a DNN*, our framework offers several advantages with respect to test coverage and model understanding. Due to the high dimensionality of the input space, a combinatorial exploration testing is not practical. Random testing of a DNN via "Monte Carlo" testing is a popular incremental testing method. However, requires large numbers of test cases and is therefore not applicable for most applications.

Our framework, which is based upon active learning explores the input space in an intelligent manner, focusing on finding new data points in "interesting" and potentially "troublesome" areas. Here, our surrogate model therefore exhibits substantially more details. This exploration is guided by the selected active learning heuristics and is able to cover the entire input space with a low number of data points.

In this paper, we will describe our framework, the underlying statistical models and learning algorithms, as well as our specific boundary-EI metric. We will illustrate and assess our framework with a case study, where we consider a physics-based deep recursive residual neural network (DR-RNN), which has been set up to learn the aerodynamics of an aircraft. The DR-RNN is trained on time-series data and learns to model the aircraft dynamics to a high level of accuracy, enabling efficient trajectory prediction, diagnosis, and prognostics [21]. Since this DR-RNN application is obviously safety-critical, its analysis using our framework can provide valuable insights for the V&V of deep neural network architectures.

The reminder of the paper is structured as follows: Section II briefly describes our application domain of aircraft dynamics and Deep Residual Neural Networks (DR-RNN). Then we present an overview of our framework architecture in Section III, before we discuss in detail the statistical models and algorithms of our framework (Section IV) as well as our boundary-aware active learning method in Section V. We then present our Bayesian shape estimation for the boundaries (Section VI). Section VII presents the application of our framework for the quality analysis for Deep Neural Networks. Section VIII focuses on our case study, a DR-RNN for the emulation of flight dynamics for a Boeing 747-100 aircraft. Finally, Section IX presents future work and concludes.

II. Background

A. Aircraft Dynamics

The dynamics of an aircraft is typically described as a set of differential equations, which are integrated to obtain, given an initial condition, the state vector x of the aircraft. In this paper, we only present a minimal context and refer the reader to standard text books in this area, e.g., [14]. In our paper, we are mainly interested in the aircraft attitude, i.e., the roll angle ψ, the pitch angle θ, and yaw angle ϕ.

B. Deep Residual Neural Networks

Deep Residual Neural Networks (DR-RNN) have been introduced in [10] for modeling of dynamical systems and model reduction. This approach has been used in [21] for learning and prediction of aircraft trajectories for a large commercial transport aircraft.

The physical system under consideration and its dynamics are defined using differential equations

$$\dot{x} = \mathbf{A}x + \mathbf{B}u + \nu \tag{1}$$

where x is the state vector, u the command input, and ν Gaussian noise. For example, for an aircraft, the state vector $x = [\psi, \theta, \phi]$ with roll angle ψ, pitch angle θ, and yaw angle ϕ would describe the attitude of the aircraft. $\mathbf{A}$ and $\mathbf{B}$ are matrices, into which numerous parameters concerning the aircraft structure and design are incorporated.

The residual function for Eq. (1), based upon Euler integration and fixed discrete time steps Δt is given by

$$r_{t+1} = x_{t+1} - x_t - \Delta t(\mathbf{A}x_{t+1} + \mathbf{B}u_{t+1})$$

A DR-RNN with K network layers is defined to minimize the residual. [10] defines the network as

$$\begin{array}{rcll} x^k_{t+1} & = & x^{k-1}_{t+1} - \mathbf{w} \circ \tanh \mathbf{U} r^k_{t+1} & \text{for } k = 1 \\ x^k_{t+1} & = & x^{k-1}_{t+1} - \frac{\eta_k}{\sqrt{G_k + \epsilon}} r^k_{t+1} & \text{for } k > 1 \end{array}$$

where $\mathbf{U}, \mathbf{w} \in \Re^N$ with N the dimension of the state vector x, and η_k are parameters of the DR-RNN, which are adjusted during training. Furthermore,

$$r^k_{t+1} = x^{k-1}_{t+1} - x_t - \Delta t(\mathbf{A}x^{k-1}_{t+1} + \mathbf{B}u_{t+1})$$

Finally, G_k is an exponentially decaying squared norm of the residual:

$$G_k = \gamma ||r^k_{t+1}||_2 + \zeta G_{k-1}$$

where γ and ζ are the decay factors, usually $\gamma = 0.1$ and $\zeta = 0.9$, and ϵ a small positive number.

The DR-RNN can be trained using different learning algorithms, for example, a modified Gradient descent [10], or the stochastic Adam algorithm [11], [21] with respect to the loss function

$$L = \frac{1}{N_t \cdot T \cdot N} \sum_{c=1}^{N_t} \sum_{t=1}^{T} \sum_{i=1}^{N} |\hat{x}_{i,t} - x^{true}_{i,t}|$$

where N_t is the number of training examples, T the length of the training time series, N the dimensionality of the state vector, $\hat{x}_t = x^N_t$ the predicted value, and x^{true}_t is the true state vector at time t.

III. Architecture of Our Testing Framework

Figure 1 shows the high-level architecture of our testing framework. On the right-hand side, we have our DR-RNN, which is the "system under test" (SuT). Its behavior is evaluated and compared against a system dynamics simulator and any deviations are recorded. The level of allowable deviations is given as a system requirement, against which our framework is testing the DR-RNN.

Each test case, which is used to trigger the DR-RNN assessment is given as a set of specific initial conditions or a scenario. For example, the aircraft dynamics is to be evaluated, given initial conditions of pitch angle θ, angle of attack α, and air-speed V_{ias}. The aircraft state over time is obtained by DR-RNN lookup or integration of the dynamic model. A comparison of these two time series and thresholding then yields the test result, stating if the system requirements have been met or not. Figure 2 shows the temporal development of the pitch angle θ over time, comparing the DR-RNN output

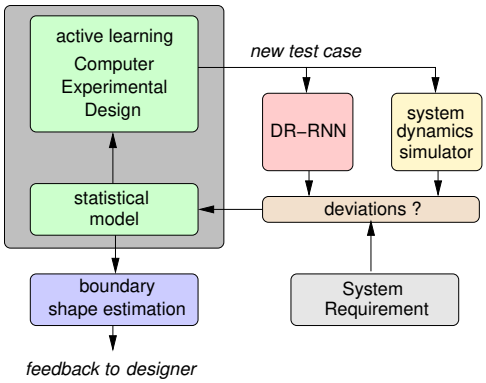

Fig. 1. High level architecture of our DNN testing framework

(red) against the results of the high-fidelity simulator (blue) over a run of 800 seconds. In the Figure 2A, there is almost no deviation between both time series and the requirements are met. In another scenario, however (Figure 2B) deviations show near the peaks of the oscillation, causing the requirement to fail.

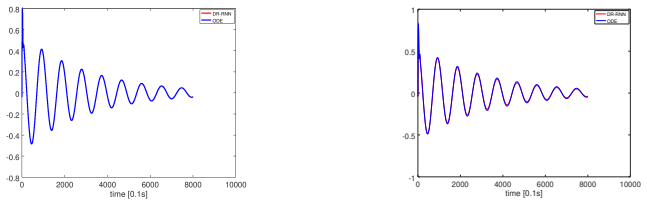

Fig. 2. Pitch angle θ over time after disturbance. Values obtained by the high fidelity simulator (including numerical integration) shown in blue; output of the DR-RNN shown in red. Example of initial conditions, which result in small deviations (left). Example of initial conditions, which produce large deviations, violating the safety requirement (right).

This result of the test run is then used to refine our statistical surrogate model (Figure 1 left). For the representation and construction of the model, we are using Dynamic Regression Trees (DynaTrees), which will be described in more detail below.

For the efficient generation of most informative test cases, we use active learning and Computer Experimental Design. Our boundary-EI metric (see below) proposes, based upon the information in our statistical model, new test cases that are close to "interesting" regions, but that also cover the entire space in order not to miss important areas in the high-dimensional state space.

Once the surrogate model has been constructed, geometrical shape estimation of the boundaries can take place. The boundaries are then characterized as parameterized geometric shapes taken from a library, thus facilitating feedback to the designer of the Neural Network.

IV. Methodology Overview

We propose a sequential method for the estimation of parameterized boundary shapes in high dimensional spaces.

A dictionary of shape classes is provided by the domain expert. Additional constraints on the parameters, e.g., parameter ranges and other prior information can be given. Typical examples for such shape classes include (hyper-)surfaces, polygons, spheres, or ellipses.

We represent our boundary problem as learning the response surface for the function f, where $f(x) = 1 + \epsilon$ if the experiment succeeds and $f(x) = 0 + \epsilon$ otherwise for some small $\epsilon > 0$. In this representation a boundary is determined by points x with $f(x) = 0.5$. This representation allows us to formulate powerful methods to select the next data point, which is explained in Section V.

Given an initial set of labeled data D_0, our approach builds a hierarchical Bayesian representation. Using active learning and computer experimental design, the number of required experiments and simulation runs can be kept minimal. The hierarchical representation provides information and confidence intervals for subsequent estimation of shape parameters Θ for the threshold surface. We present the shape estimation in Section VI below.

Our iterative modeling process, as depicted in Figure 1 uses active learning to build an initial classifier from the data set D_0. Our algorithm is based upon the sequential classification and regression framework as given by DynaTrees [6], [19]. It features dynamic regression trees and a sequential tree model. Particle learning for posterior simulation makes DynaTrees a good candidate for applications, where new data points are processed sequentially. In our architecture, the classifier is represented by a DynaTree at any given point in time. After adding a number of new data points, the current classifier is used to estimate a set of data points, which are close to the current prediction of the boundary. This is a subset of data points from a regular grid or a Latin hyper square, for which their entropy measure is high (classification representation) or the estimated response value is close to 0.5.

Since the execution of each new test can consume considerable resources, our framework aims to minimize the number of tests that need to be generated to characterize the boundaries in the high-dimensional space. For that, we use active learning with a specific boundary-sensitive metric, which will be discussed below.

V. Active learning and Expected Improvement

A. Finding Boundaries

Each data point describing one simulation run (experiment) is defined as $\mathbf{x} = \langle P_1, \ldots, P_p \rangle$, where P_i are the input parameter settings and the outcome $o(\mathbf{x}) \in \{success, failure\}$. Thus these data define a classification problem with $C = 2$ classes. Informally, a boundary can be found between regions, where all experiments yield success $p(\mathbf{x} = success) = 1$ and those, where the experiments do not meet the success criterion $p(\mathbf{x} = failure) = 1$. Therefore, we can define a point $\mathbf{x}$ to be on the boundary if $p(\mathbf{x} = success) = p(\mathbf{x} = failure) = 0.5$. Although this condition can easily be generalized to more than two classes, in this work, we will focus on $C = 2$.

A common metric to characterize points on the boundary is based upon the entropy. The entropy $entr = -\sum_{c \in c_1,..,c_C} p(\mathbf{x} = c) \log p(\mathbf{x} = c)$ becomes maximal at the boundary. In cases of more than two classes, [5] uses a BVSB (Best vs. Second Best) strategy. [20] defines a metric *advantage* as essentially $adv(\mathbf{x}) = |p(\mathbf{x} = success) - p(\mathbf{x} = failure)|$. Then [20] considers points with minimal advantage to be close to the boundary. In the general case with more than two dimensions, [20] proposed to use the difference between the two most likely classes.

In general, there are two basic methods: explicitly from knowledge of the classification function, or by treating the classifier as a black box and finding the boundaries numerically. For some classifiers it is possible to find a simple parametric formula that describes the boundaries between groups, for example, LDA or SVM. Most classification functions can output the posterior probability of an observation belonging to a group. Much of the time we do not look at these, and just classify the point to the group with the highest probability.

Points that are uncertain, i.e., have similar classification probabilities for two or more groups, suggest that the points are near the boundary between the two groups. For example, if a point is in Group 1 with probability 0.45, and in Group 2 with probability 0.55, then that point will be close to the boundary between the two groups. We can use this idea to find the boundaries. If we sample points throughout the design space we can then select only those uncertain points near boundaries. The thickness of the boundary can be controlled by changing the value, which determines whether two probabilities are similar to each other or not. Ideally, we would like this to be as small as possible so that our boundaries are accurate. Some classification functions do not generate posterior probabilities. In this case, we can use a k-nearest neighbors approach. Here we look at each point, and if all its neighbors are of the same class, then the point is not on the boundary and can be discarded. The advantage of this method is that it is completely general and can be applied to any classification function. The disadvantage is that it is slow ($O(n^2)$), because it computes distances between all pairs of points to find the nearest neighbors. In general, finding of the boundaries faces the "curse of dimensionality": as the dimensionality of the design space increases, the number of points required to make a perceivable boundary (for fitting or visualization purposes) increases substantially. This problem can be attacked in two ways, by increasing the number of points used to fill the design space (uniform grid or random sample), or by increasing the thickness of the boundary.

B. Boundary Expected Improvement

Finding a boundary between two classes can be considered as finding a contour with $a = 0.5$ in the response surface of the system response. Inspired by [9] and work on contour finding algorithms, we loosely follow [16], and define our heuristics by using an improvement function. In order to use the available resources as efficiently as possible for our contour/boundary finding task, one would ideally select candidate

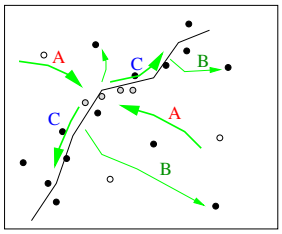

Fig. 3. Boundary-EI metric and its different components

points which lie directly on the boundary, but that is unknown. Therefore, new trial points x are selected, which belong to an ϵ-environment around the current estimated boundary. This means that $0.5 - \epsilon \leq \hat{y}(x) \leq 0.5 + \epsilon$ for $\epsilon > 0$. $\hat{y}(x)$ is the learned estimate of the response function at x. New data points should maximize the information in the vicinity of the boundary. Following [9] and [16], we define an improvement function for x as

$$I(X) = \epsilon^2(x) - \min\{(y(x) - 0.5)^2, \epsilon^2(x)\}$$

here, $y(x) \sim N(\hat{y}(x), \sigma^2(x))$, and $\epsilon(x) = \alpha\,\sigma(x)$ for a constant $\alpha \geq 0$. This term defines an ϵ-neighborhood around the boundary as a function of $\sigma(x)$. This formulation makes it possible to have a zero-width neighborhood around existing data points. For boundary sample points, $I(X)$ will be large when the predicted $\sigma(x)$ is largest.

The expected improvement $E[I(x)]$ can be calculated easily following [16] as

$$\begin{aligned} E[I(x)] = &- \int_{0.5-\alpha\sigma(x)}^{0.5+\alpha\sigma(x)} (y - \hat{y}(x))^2 \phi\left(\frac{y - \hat{y}(x)}{\sigma(x)}\right) dy \qquad (2) \\ &+2(\hat{y}(x) - 0.5)\sigma^2(x)\left[\phi(z_+(x)) - \phi(z_-(x))\right] \\ &+(\alpha^2\sigma^2(x) - (\hat{y}(x) - 0.5)^2)\left[\Phi(z_+(x)) - \Phi(z_-(x))\right], \end{aligned}$$

where $z_\pm(x) = \frac{0.5 - \hat{y}(x)}{\sigma(x)} \pm \alpha$, and ϕ and Φ are the standard normal density and cumulative distribution, respectively. Each of these three terms are instrumental in different areas of the space. Figure 3 sketches the situation: the first term summarizes information from regions of high variability within the ϵ-band (marked "A" in the figure). The integration is performed over the ϵ-band as $\epsilon(x) = \alpha\,\sigma(x)$. The second term is concerned with areas of high variance farther away from the estimated boundary (labeled "B" in Figure 3). Finally, the third term is active close to the estimated boundary (labeled "C"). After the expected improvement has been calculated, the candidate point is selected as the point, which maximizes the expected improvement.

VI. Boundary Shape Estimation

A. Notation

Suppose there are m shape classes $M_1, \ldots, M_m$ with $m \geq 1$, which are parameterized by $\Theta_1, \ldots, \Theta_m$. The task is to fit l shapes $S_1, \ldots, S_l$, $l \geq 1$, where $S_1 = (i_1, \Theta_1), \ldots, S_l = (i_l, \Theta_l)$ and i_j denotes the shape class for the j^{th} shape with $i_j \in \mathcal{M} = \{M_1, \ldots, M_m\}$. Several of the i_j can be the same to accommodate more than one shape belonging to the same class. The Θ_i should be different since we do not want to represent the same boundary shape twice. We also seek to determine the correct number of shapes l that represents the input point set X_n.

For example, we may consider the $m = 2$ shape classes $M_1 = \mathit{hyperplane}$ and $M_2 = \mathit{sphere}$ in $\Re^d$. Hyperplanes are represented as $a_1x_1 + \cdots + a_dx_d + a_{d+1} = 0$ with parameter vector $\Theta_1 = (a_1, \ldots, a_d, a_{d+1}) \in R^{d+1}$. In the same d-dimensional space, a sphere of radius r with center $c = (c_1, \ldots, c_d)$ is described by $(x_1 - c_1)^2 + \cdots + (x_d - c_d)^2 = r^2$ with parameter vector $\Theta_2 = (c, r) \in \Re^{d+1}$.

B. What is a Good Shape Set S for an Input Point Set X_n?

There are three conditions that specify when a shape set S provides a good fit to the data X_n:

(i) *Summary*: each point on a shape $S \in$ S is close to some classifier boundary point in X_n,
(ii) *Completeness*: each classifier boundary point in X_n is close to some shape point on some shape $S \in$ S, and
(iii) *Minimality*: the shapes in S are as different from one another as possible.

Condition (i) encourages each shape $S \in$ S to be a good *summary* of one of the parts of the boundary of classifier P_n. That is, the points of a shape should lie along high entropy areas of P_n.

Condition (iii) encourages that shape set S to be *minimal*; i.e., S will not use any extra shapes to form a complete summary of the boundaries of classifier P_n. A complete summary S (i.e., one satisfying (i) and (ii)) remains a complete summary if one of its shapes $S \in$ S is added to S either exactly or after a small perturbation. In fact, adding a small perturbation $\widehat{S}$ of S may actually improve completeness slightly since $\widehat{S}$ can be even closer to some high entropy points than S. And if S were a good summary, then so too would $\widehat{S}$. We need the minimality condition (iii) to be able to obtain the simplest (i.e., smallest) shape set that is a complete summary of the classifier boundaries.

C. Statistical Modeling

The shape set posterior is

$$P(\mathrm{S}|X_n) = \frac{P(X_n|\mathrm{S})P(\mathrm{S})}{P(X_n)} \propto P(X_n|\mathrm{S})P(\mathrm{S}).$$

We build the posterior model $P(\mathrm{S}|X_n)$ by modeling the likelihood $P(X_n|\mathrm{S})$ and the shape set prior $P(\mathrm{S})$. In the posterior $P(\mathrm{S}|X_n) \propto P(X_n|\mathrm{S})P(\mathrm{S})$, we will model the likelihood $P(X_n|\mathrm{S})$ to encourage *completeness* and the prior $P(\mathrm{S})$ to encourage distance between shapes and therefore *minimality*. It makes sense that the data likelihood accounts for completeness because completeness requires observed points to be close to a shape and the observed points arise from the ground truth shapes with the addition of noise. We will encourage good *summary* using a Bayesian loss function that grows with increasing distance of the shapes to the point set.

Finally, we determine the number of shapes l by minimizing the expected posterior loss.

1) Likelihood: Our likelihood will encourage completeness. For the completeness condition (ii), we are interested in making the average squared distance $\overline{D}^2_{X_n,\mathrm{S}}$ of boundary points in $X_n = \{x_1, \ldots, x_n\}$ to shapes in S small:

$$\overline{D}^2_{X_n,\mathrm{S}} = \frac{\sum_{x \in X_n} d^2_{X_n,\mathrm{S}}(x)}{|X_n|} = \frac{\sum_{j=1}^n d^2_{X_n,\mathrm{S}}(x_j)}{|X_n|}, \quad (3)$$

where

$$d^2_{X_n,S}(x) = \min_{s \in \mathrm{S}} ||x - s||_2^2 \quad (4)$$

is the minimum squared distance of a high entropy point x to a point on any shape in the collection $\mathrm{S} = (S_1, \ldots, S_l)$.

An observed point $x_j \in X_n$ is assumed to have been generated from a shape S_{z_j}, where z_j gives the shape number that explains x_j. Given z_j, we model the likelihood of x_j as a decreasing function of the minimum distance from x_j to S_{z_j}. The closer x_j is to shape S_{z_j}, the higher the likelihood of x_j. The observations x_j are assumed to be independent and modeled as

$$x_j = s_j + \varepsilon_j = s_j + r_j n_j, \quad r_j \sim N(0, \sigma_r^2),$$

where n_j is a unit normal to S_{z_j} at s_j and $r_j = (x_j - s_j) \cdot n_j$. Here the noise vector $\varepsilon_j = r_j n_j$ is along a unit normal n_j to the shape S_{z_j} at the closest shape point s_j to x_j. The scalar residual r_j is the signed distance along n_j from the shape S_{z_j} to x_j. We model the observation error ε_j by modeling the signed residual as a $N(0, \sigma_r^2)$ random variable.

Note that the squared residual r_j^2 is just the minimum distance squared from x_j to the closest point s_j on shape S_{z_j}:

$$r_j^2 = \min_{s \in S_{z_j}} ||x_j - s||_2^2,$$

where the minimum occurs at $s = s_j$. Let $Z = (z_1, \ldots, z_n)$. Assuming independence of points and that x_j depends only on shape S_{z_j}, then $P(X_n|Z, \mathrm{S}) = \prod_{j=1}^n P(x_j|z_j, S_{z_j}) = \prod_{j=1}^n N(r_j|0, \sigma_r^2)$. Since $r_j \sim N(0, \sigma_r^2)$, it follows that

$$P(X_n|Z, \mathrm{S}) = K\sigma_r^{-n} \exp\left(-\frac{1}{2\sigma_r^2} \sum_{j=1}^n \min_{s_j \in S_{z_j}} ||x_j - s_j||_2^2\right), \quad (5)$$

for a constant K. Note that if the observed point set X_n is close to the shapes in S, then $P(X_n|Z, \mathrm{S})$ is high. This statement assumes, of course, that the correct shape S_{z_j} explaining each point x_j has also been identified.

We can obtain the likelihood $P(X_n|\mathrm{S})$ by modeling $Z|\mathrm{S}$ and integrating out Z as in $P(X_n|\mathrm{S}) = \int_Z P(X_n|Z, \mathrm{S}) P(Z|\mathrm{S}) dZ$. We could, for example, model $Z|\mathrm{S}$ by modeling a count vector $C = (c_1, \ldots, c_l)$ which holds the number of observations c_i explained by shape S_i. Here $c_i = \sum_{j=1}^n 1_{z_j = i}$. We can encourage good summary by modeling $C \sim \mathrm{multinomial}(n, (1/l, 1/l, \ldots, 1/l))$ where each of the l shapes in S has the same probability $1/l$ of generating an observed point. This would make shape sets with any shapes that are from the data quite unlikely because we would expect to see points around each shape according to the given multinomial distribution.

It is difficult, however, to optimize over shape sets with the hidden random variables Z in our models. Instead, we make a simple but accurate and effective approximation in our models and assume that the shape S_{z_j} that explains observation x_j is the shape in S which is closest to x_j. Thus we replace the minimization in equation (5) over $s_j \in S_{z_j}$ with a minimization $s_j \in \mathrm{S}$ over the entire shape set to obtain the approximation

$$P(X_n|\mathrm{S}) = K\sigma_r^{-n} \exp\left(-\frac{1}{2\sigma_r^2} \sum_{j=1}^n \min_{s_j \in \mathrm{S}} ||x_j - s_j||_2^2\right). \quad (6)$$

From Equations (3), (4), we can see that the inner sum in equation (6) is just a scaled version $|X_n| \overline{D}^2_{X_n,\mathrm{S}}$ of our completeness measure. We can easily write our likelihood in terms of the completeness measure $\overline{D}^2_{X_n,\mathrm{S}}$. To do so cleanly, define $\sigma^2_{\mathrm{complete}} = \sigma_r^2/|X_n|$. Then

$$P(X_n|\mathrm{S}) = K\sigma^{-n}_{\mathrm{complete}} \exp\left(-\frac{1}{2\sigma^2_{\mathrm{complete}}} \overline{D}^2_{X_n,\mathrm{S}}\right),$$

where another constant factor has been absorbed into K.

2) Shape Set Prior: We build the shape set prior $P(\mathrm{S})$ based on the distances of points on each shape S_i to the rest of the shape set $\mathrm{S}_{-i} = \mathrm{S} \backslash \{S_i\}$. To keep shapes apart from one another, we want a large average squared distance from points on each shape to the rest of the shapes. Let $d^2_{S_i,S_j}(s_i)$ be the minimum squared distance of a point $s_i \in S_i$ to another shape S_j:

$$d^2_{S_i,S_j}(s_i) = \min_{s_j \in S_j} ||s_i - s_j||_2^2.$$

Then the squared distance of $s_i \in S_i$ to the shape set S_{-i} is

$$d^2_{S_i,\mathrm{S}_{-i}}(s_i) = \min_{S_j \in \mathrm{S}_{-i}} d^2_{S_i,S_j}(s_i),$$

which finds the closest point in the rest of the shapes S_{-i} to $s_i \in S_i$. Finally we average the inter-shape squared distances over all points on all shapes to get

$$\overline{D}^2_{\mathrm{S}} = \frac{\sum_{S_i \in \mathrm{S}} \sum_{s_i \in S_i} d^2_{S_i,\mathrm{S}_{-i}}(s_i)}{\sum_{S_i \in \mathrm{S}} |S_i|}$$

To keep the shapes apart a priori, we want $\overline{D}^2_{\mathrm{S}}$ to be large, indicating that on average the inter-shape distance is large. Equivalently, $1/\overline{D}_{\mathrm{S}}$ should be small. Therefore we model the prior for S using the normal distribution

$$\mathrm{S} \sim N(\overline{D}_{\mathrm{S}}^{-1}; 0, \sigma^2_{\mathrm{shapesim}}).$$

3) Bayesian Loss: Next we define a Bayesian loss function that encourages good summary. We can think of the summary condition (i) as requiring a small distance from each shape $S \in \mathrm{S}$ to the set of classifier boundary points X_n. Let $d^2_{S,X_n}(s)$ denote the squared distance from a shape point $s \in S$ to the point set X_n:

$$d^2_{S,X_n}(s) = \min_{x \in X_n} ||s - x||_2^2.$$

We capture the average squared distance $\overline{D}^2_{\mathrm{S},X_n}$ from the shape set S to the input points X_n by averaging over all points on all shapes in $\mathrm{S} = (S_1, \ldots, S_l)$:

$$\overline{D}^2_{\mathrm{S},X_n} = \frac{\sum_{a=1}^{l} \sum_{s \in S_a} d^2_{S_a,X_n}(s)}{\sum_{a=1}^{l} |S_a|}.$$

We define our Bayesian loss function as

$$\mathrm{loss}(\mathrm{S}, X_n) = \lambda_{\mathrm{summary}} \overline{D}^2_{\mathrm{S},X_n}$$

The smaller the distance from each shape in S to the point set X_n, the smaller the loss. Thus minimizing the loss will encourage good summary.

D. Shape Fitting Method

Our shape fitting method has two main steps:

Step 1: Minimize the expected posterior loss

$$g(l) = E[\mathrm{loss}(\mathrm{S}, X_n)], \quad |\mathrm{S}| = l$$

over l to obtain the number of shapes l^*

Step 2: Compute the MAP shape set S^{*,l^*} for sets of size l^*

1) Determining the Number of Shapes: We assume that we can a-priori limit the number of shapes l to some set $\mathcal{L}$. For example, if we know that there will not be more than five boundaries then we can set $\mathcal{L} = \{1, 2, 3, 4, 5\}$.

For each $l \in \mathcal{L}$, we compute the expected posterior loss

$$g(l) = E[\mathrm{loss}(\mathrm{S}, X)] = \int_{\{\mathrm{S}:|\mathrm{S}|=l\}} \mathrm{loss}(\mathrm{S}, X_n) \widehat{P}(\mathrm{S}|X_n) d\mathrm{S}.$$

Here we denote the shape set posterior probability distribution for shape sets with a fixed number of shapes as $\widehat{P}(\mathrm{S}|X_n)$. Then we choose the number of shapes to minimize the expected posterior loss:

$$l^* = \arg\min_{l \in \mathcal{L}} g(l).$$

E. Our Shape Set Posterior Sampling Method

For a fixed shape set size $|\mathrm{S}| = l$, we will draw samples from the posterior $P(\mathrm{S}|X_n) \propto P(X_n|\mathrm{S})P(\mathrm{S})$ using an iterative procedure. Shape set samples S with a small value for

$$-\log(P(X_n|\mathrm{S})P(\mathrm{S})) = -\log(P(X_n|\mathrm{S})) - \log(P(\mathrm{S}))$$

should be more likely to occur.

VII. Quality Analysis of Network Structure

Deep Neural Networks are characterized by numerous user-definable parameters, like number of layers, size and type of each layer, as well as transfer functions, just to name a few. For our DR-RNN, the number of layers K is of most importance, because it encodes the amount of historical data that is taken into account.

Finding optimal or even suitable parameter settings for the given task and the given training data is extremely difficult. In many cases, practitioners therefore just use the default parameter setting as provided with the learning software or library. This can result in numerous problems ranging from the use of unnecessary resources (too many layers or nodes) and potentially dangerous over-training to poor learning and generalization behavior. For V&V of DNN, a solid quality analysis of the network architecture is necessary. Only then, information about suitability about the chosen network parameters as well as the existence of "dangerous regions" is available to designers and quality engineers.

We have extended our statistical testing framework described in the earlier sections to support comparative quality analysis for DNN architectures. Figure 4 gives a high-level overview of the framework architecture. In this instantiation, our systems-under-test are different DR-RNN networks, $\text{DR-RNN}_1, \ldots, \text{DR-RNN}_N$, which have been set up using different configuration parameters and have been trained using potentially different training algorithms. Given new test cases in the same manner as describes above, the results of the DR-RNNs are compared *against each other* and deviations between them are compared against a given threshold.

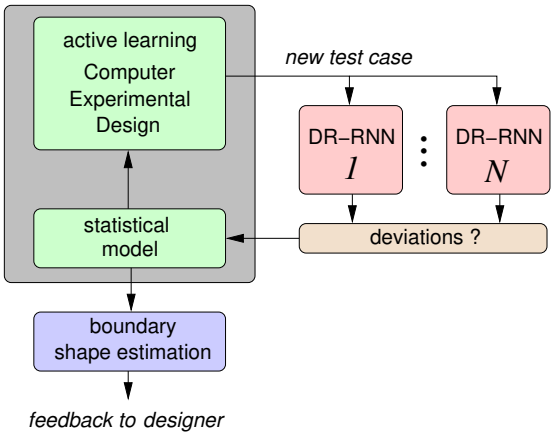

Fig. 4. High level architecture for DNN Comparison

In this setting, our framework iteratively constructs a statistical model, which learns the boundaries of areas, where the different DR-RNNs produce different results. Figure 5 illustrates this with a simple 3 dimensional dynamic system (from [10]): when using a DR-RNN with $K = 2$, we obtain results very close to the truth in a selected region, where the deviation is very small. Outside this region, the deviation grows very quickly, making the use of this DR-RNN problematic in those areas (Figure 5A). In contrast, a DR-RNN with $K = 3$ produces larger deviations throughout the entire space (Figure 5B) but shows a substantially better behavior in the outer parts. For a suitable threshold, we therefore characterize boundaries between the areas, where both DR-RNNs show a similar output, and potentially problematic areas. Figure 5C shows a 2D projection of the boundaries and regions as found by our algorithm. Due to our boundary-EI metric, new test cases are massed close to the boundary – two non-linear segments in this case.

If the behavior of the different DR-RNNs are very close to each other in the operational envelope, then this is an

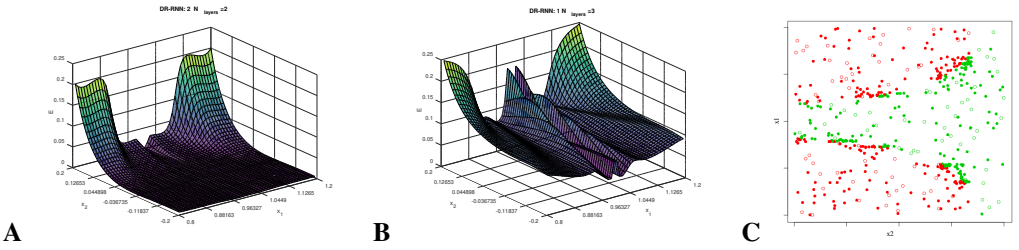

Fig. 5. Projection of error surfaces for DR-RNN with $K = 2$ (**A**) and $K = 3$ (**B**). Projection of result with $N = 400$ test cases generated by our framework. Circles correspond to test points in the initial data set $N_0 = 100$.

indication that the parameter settings are suitable for the task. This, of course, assumes that at least one of the DR-RNNs has been tested against the truth, obtained by the high-fidelity simulator as described above.

If, however, larger deviations exist, a deeper manual analysis will be necessary. In particular, if deviations exist in the "middle" of the operational envelope, this is an indication that the parameter settings of the corresponding DR-RNN is not set properly and might return, if used, unsafe results.

VIII. Case Study

We have applied our testing framework to a DR-RNN, which has been trained to model the six degrees-of-freedom aircraft dynamics of a Boeing 747-100. The DR-RNN and the application is described in [21] and we have been able to obtain the trained network and models from the authors. We are considering the DR-RNN as our "black-box" DNN-under-test.

The trained DR-RNN is adopted to predict the responses of the Boeing 747-100 aircraft under arbitrary disturbances and control inputs. Given initial states, disturbances, and control inputs (optional), the DR-RNN can efficiently provide predictions of the AC state over an extended period of time (800s in [21]) without costly numerical integration of the aircraft dynamics (Equation 1). Details of setup and results are described in detail in [21].

We used our testing framework to analyze the behavior of the DR-RNN against the true AC dynamics, given different initial disturbances of the state vector. For illustration purposes, we used the DR-RNN for the pitch aircraft axis and varied the disturbance with respect to pitch angle θ, the pitch rate $q = d\theta/dq$, angle of attack α, and normalized speed v/v_0.

As an example system requirement, we defined a passing accuracy if

$$\sum_{t=0}^{10s}(x_{dyn} - x_{DR-RNN})^2 < \Theta \tag{7}$$

for a given $\Theta = 0.001$ and x is the 4-dimensional state vector $[v/v_0, \alpha, q, \theta]$.

Using our testing framework using an initial uniformly distributed set of test cases D_0 with 200 test cases, we were generating 800 new test cases guided by the iteratively constructed surrogate model and our boundary-aware EI metric.

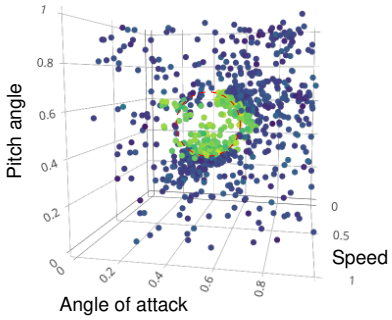

Fig. 6. Results of DR-RNN evaluation projected into axes of $\theta, \alpha, v/v_0$. Green points correspond to runs that fulfill the requirement (Equation 7), blue dots to violations of the requirement. The 4D surrogate model was initialized with $N_0 = 200$ test cases; active learning generated $N_t = 800$ new test cases.

Figure 6 shows a 3D projection of the test cases into the space of pitch angle, angle of attack, and speed. The parameter values are normalized to the interval $[0 \ldots 1]$. Green dots show test cases, which pass our requirement (Equation 7), blue dots represent failing test cases. As expected, all test cases near nominal values (i.e., 0.5 normalized) of θ and α yield DR-RNN output values that are close to the results obtained by the reference simulator. Further away from that center, the accuracy of the network is lower, causing the requirement to fail. Parameter variations of the speed v/v_0 within the given parameter range does not have a substantial influence on the network accuracy (not clearly visible in the shown perspective of Figure 6) Pitch rate q also does not show changes in fulfilling the requirement over the given parameter range (not shown in this projection).

Note that in this experiment, we chose extremely large parameter variations in order to illustrate the boundaries. Realistic flight envelopes are well within the "green area", indicating that the DR-RNN has been properly trained and can be used to efficiently emulate the aircraft dynamics [21].

For the analysis, our focus is on the *boundary* between regions, where the requirement (Equation 7) is met and where it fails, i.e., the boundary between green and blue dots. As Figure 6 shows, our boundary-aware active learning placed a

large majority of data points (= newly generated test case) near the boundary. Only few test cases (most of them from the initial data set) are deeply within the green zone or outside and away from the boundary. This makes it easy to visually recognize the boundary as a cylinder (along the speed axis). Our shape detection algorithm estimated the center as $(\alpha_c, q_c) = (0.415, 0.516)$ and $r = 0.22$ (shown as a red line in Figure 6).

Obviously, that boundary shape could be detected by systematic parameter exploration or Monte Carlo simulation. However, a hugely increased number or required test cases makes these approaches prohibitively expensive. Figure 7 shows a projection of a Monte Carlo run with the same number of test cases $N = 1000$ into the axes of $\theta, \alpha, v/v_0$. It is easy to see that the test points are spread out throughout the entire space, making the detection and characterization of the boundary extremely hard. A proper detection would require substantially more test cases.

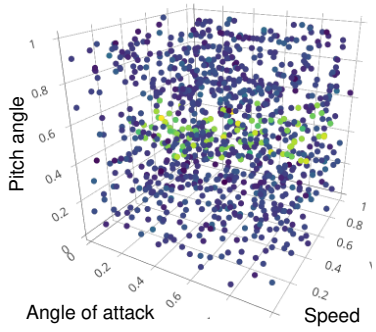

Fig. 7. Results of DR-RNN evaluation using a Monte Carlo run with $N = 1000$ projected into $\theta, \alpha, v/v_0$.

IX. Conclusions

In this paper, we presented a framework to support testing and analysis of DNNs. Using advanced techniques from statistical modeling and active learning, our framework efficiently generates test cases to model and characterize boundaries separating areas where safety requirements are met and not met. Geometric shape estimation of the boundaries can provide valuable feedback to the designer.

We presented results of a case study on a physics-based deep recurrent neural network, which has been trained to emulate the aerodynamic behavior of a Boeing 747-100 aircraft.

We also illustrated how our framework can be used to efficiently compare different DR-RNNs and characterize boundaries of regions where the network output deviates from each other. Such boundaries can provide feedback to the designer and help selecting suitable DR-RNN architectures. For future work, we will extend out framework to allow the active learning search among the structural parameters of the network (e.g., number of layers, kind of non-linearity, etc.). Here, a low number of required test cases is extremely important, since each test case requires a training operation.

References

[1] Markus Borg, Cristofer Englund, Krzysztof Wnuk, Boris Duran, Christoffer Levandowski, Shenjian Gao, Yanwen Tan, Henrik Kaijser, Henrik Lönn, and Jonas Törnqvist. Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. *CoRR*, abs/1812.05389, 2018.

[2] Adrian Carrio, Carlos Sampedro, Alejandro Rodriguez-Ramos, and Pascual Campoy. A review of deep learning methods and applications for unmanned aerial vehicles. *Journal of Sensors*, 2017(3296874):13, 2017.

[3] D. A. Cohn. Neural network exploration using optimal experimental design. *Advances in Neural Information Processing Systems*, 6(9):679–686, 1996.

[4] Paula Fraga-Lamas, Lucía Ramos, Víctor Mondéjar-Guerra, and Tiago M. Fernández-Caramés. A review on iot deep learning uav systems for autonomous obstacle detection and collision avoidance. *Remote Sensing*, 11(18), 2019.

[5] R. Gramacy and N. Polson. Particle learning of Gaussian process models for sequential design and optimization. *Journal of Computational and Graphical Statistics*, 20(1):467–478, 2011.

[6] Robert B. Gramacy and Herbert K. H. Lee. Bayesian treed gaussian process models with an application to computer modeling, 2007.

[7] Yuning He. *Variable-length Functional Output Prediction and Boundary Detection for an Adaptive Flight Control Simulator*. PhD thesis, University of California at Santa Cruz, 2012.

[8] Yuning He. Online detection and modeling of safety boundaries for aerospace applications using active learning and bayesian statistics. In *2015 International Joint Conference on Neural Networks, IJCNN 2015, Killarney, Ireland, July 12-17, 2015*, pages 1–8. IEEE, 2015.

[9] D. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[10] J. Nagoor Kani and Ahmed H. Elsheikh. DR-RNN: A deep residual recurrent neural network for model reduction. *CoRR*, abs/1709.00939, 2017.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[12] Max Kochurov, Timur Garipov, Dmitry Podoprikhin, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P. Vetrov. Bayesian incremental learning for deep neural networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018.

[13] D. J. C. MacKay. Information–based objective functions for active data selection. *Neural Computation*, 4(4):589–603, 1992.

[14] Duane T. McRuer, Dunstan Graham, and Irving Ashkenas. *Aircraft Dynamics and Automatic Control*. Princeton University Press, Princeton, NJ, USA, 1972.

[15] Ram Prasad Padhy, Sachin Verma, Shahzad Ahmad, Suman Kumar Choudhury, and Pankaj Kumar Sa. Deep neural network for autonomous uav navigation in indoor corridor environments. *Procedia Computer Science*, 133:643 – 650, 2018. International Conference on Robotics and Smart Manufacturing (RoSMa2018).

[16] Pritam Ranjan, Derek Bingham, and George Michailidis. Sequential experiment design for contour estimation from complex computer codes. *Technometrics*, 50(4):527–541, 2008.

[17] RTCA. DO-178C/ED-12C: Software considerations in airborne systems and equipment certification, 2012.

[18] Doyen Sahoo, Quang Pham, Jing Lu, and Steven C. H. Hoi. Online deep learning: Learning deep neural networks on the fly. In *IJCAI*, 2018.

[19] Matthew A. Taddy, Robert B. Gramacy, and Nicholas G. Polson. Dynamic trees for learning and design. *Journal of the American Statistical Association*, 106(493):109–123, 2011.

[20] H. Wickham. *Practical Tools for exploring data and models*. PhD thesis, Iowa State, 2008.

[21] Yang Yu, Houpu Yao, and Yongming Liu. Physics-based learning for aircraft dynamics simulation. In *Proc. Conference of the Society for Prognostics and Health Management (PHM)*, 2018.