NASA/TM–2014-218660

Training Knowledge Bots for Physics-Based Simulations Using Artificial Neurals Networks

Jay Ming Wong
University of Massachusetts, Amherst, Massachusetts

Jamshid A. Samareh
Langley Research Center, Hampton, Virginia

November 2014

NASA/TM–2014-218660

Training Knowledge Bots for Physics-Based Simulations Using Artificial Neurals Networks

Jay Ming Wong
University of Massachusetts, Amherst, Massachusetts

Jamshid A. Samareh
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

November 2014

Available from:

NASA STI Program / Mail Stop 148
NASA Langley Research Center
Hampton, VA 23681-2199
Fax: 757-864-6500

Training Knowledge Bots for Physics-Based Simulations Using Artificial Neural Networks

Jay Ming Wong and Jamshid A. Samareh
NASA Langley Research Center
Hampton, Virginia 23681-2199

Abstract—**Millions of complex physics-based simulations are required for design of an aerospace vehicle. These simulations are usually performed by highly trained and skilled analysts, who execute, monitor, and steer each simulation. Analysts rely heavily on their broad experience that may have taken 20-30 years to accumulate. In addition, the simulation software is complex in nature, requiring significant computational resources. Simulations of system of systems become even more complex and are beyond human capacity to effectively learn their behavior. IBM has developed machines that can learn and compete successfully with a chess grandmaster and most successful jeopardy contestants. These machines are capable of learning some complex problems much faster than humans can learn.**

In this paper, we propose using artificial neural network to train *knowledge bots* to identify the idiosyncrasies of simulation software and recognize patterns that can lead to successful simulations. We examine the use of *knowledge bots* for applications of computational fluid dynamics (CFD), trajectory analysis, commercial finite-element analysis software, and slosh propellant dynamics. We will show that machine learning algorithms can be used to learn the idiosyncrasies of computational simulations and identify regions of instability without including any additional information about their mathematical form or applied discretization approaches.

I. Introduction

Traditionally, individuals perform millions of simulations during the course of a design process by providing inputs, analyzing the simulation outputs, and identifying patterns that lead to successful simulations. Trained and experience analysts have learned the underlying simulation idiosyncrasies, and they are able to predict the simulation behavior given a set of input parameters, prior to actually performing the simulation. Humans are great at identifying complex patterns for events with small number of variables and given sufficient time to process the data and develop an expertise. Machine learning algorithms, such as artificial neural networks, are rapidly catching up and exceeding human capacity to identify these patterns that leads to knowledge creation. This is especially useful for complex simulation as the underlying implementation may be too difficult to decipher. However, to obtain such knowledge requires years of simulation experience. Furthermore, the software itself may not only be extremely complex, but may require a long period of time for execution.

Jay Ming Wong was supported through the Langley Aerospace Research Student Scholars (LARSS) program. He is studying Computer Science at the University of Massachusetts Amherst.

We propose an approach for a trained classifier, in which we refer to as a *knowledge bot* that learns the underlying properties of the black-box, much like a trained expert, and provide a quick response to whether or not a set of input parameters will result in failure before actually executing the black-box with inputs. Therefore, designers may be able to produce initial designs given the information of the *knowledge bot*, meanwhile validating the information by actually executing the black-box software. We define the term *knowledge bot* as a piece of software that has been trained using supervised machine learning techniques that interrogates simulation software and captures their underlying properties. The goal is to provide an accurate model of black-box physics-based simulations via training such a *bot* using artificial neural networks.

II. Relevant Work

The computational model of artificial neural networks has been widely popular in the early 1940s up until the late 1950s for mathematics and algorithms [1], [2]. However, a new spark of interest emerged in the 1990s and 2000s due to its roots in biological relevance, where human brain computation and conventional digital computation differ. Artificial neural systems are often analogized as "physical cellular systems which acquire, store, and utilize experiential knowledge [3]." These cellular systems have been modeled by a network of interconnected nodes, generalizable to an input layer, one or more hidden layers, and an output layer. The network formally defined as a Multilayer Perceptron (MLP) neural network, where each node takes in a set of input parameters and computes an output value feeding to nodes further down the layer hierarchy as input. Eventually, at the output layer a value is returned as a prediction [4]. This model of the network acts as a foundation for numerous machine learning techniques using artificial neural networks. It was shown in Ref. [5] that multilayer feed-forward networks are a class of universal approximators, making the use of artificial neural networks sufficiently good for a large number of varying applications. For example, neural networks have been used in areas of robotics and control dynamics for the controls of robot manipulators and non-linear systems [6], [7] and motion planning [8]; in areas of computer vision for face detection [9], [10]; in areas of chemistry to predict secondary structures of proteins [11]; in areas of clinical medicine [12]; and even in areas of finance to predict financial markets and bankruptcy [13], [14], [15]. Therefore, there has been a large amount of research using these networks in the form of machine learning

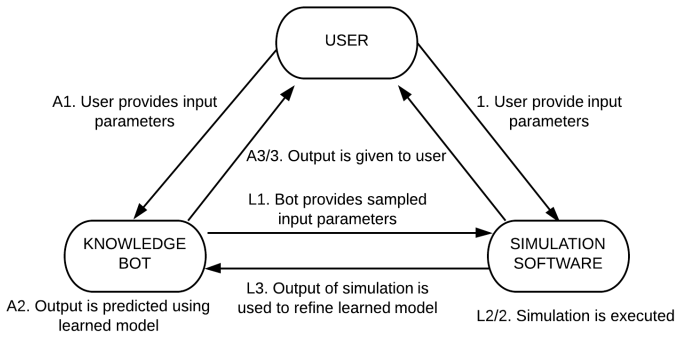

USER
A1. User provides input parameters
1. User provide input parameters
A3/3. Output is given to user
L1. Bot provides sampled input parameters
KNOWLEDGE BOT
SIMULATION SOFTWARE
L3. Output of simulation is used to refine learned model
A2. Output is predicted using learned model
L2/2. Simulation is executed

Fig. 1. The role of the *knowledge bot* in system analysis and design process

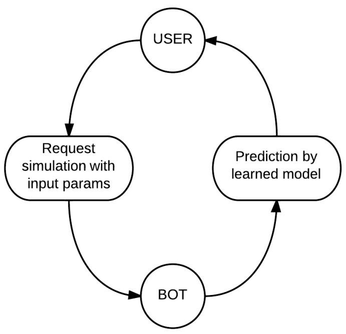

USER
Request simulation with input params
Prediction by learned model
BOT

Fig. 2. Abstract model of bot use-case, user provided with abstraction of bot

to capture specific patterns that can be exploited in making predictions, similar to humans, although in a much larger scale. GPU-based implementation of using artificial neural networks was shown in [16] to be largely successful in pattern recognition, allowing for massively parallel implementation. Various means of training neural networks to visualize and understand complex datasets has been demonstrated in [17]. However, for the purpose of our approach, we are more concerned with feasibility, precision, and accuracy rather than visualization because dimensionality reductions may remove essential components of our problems.

III. Learning Physics-Based Simulations

The *knowledge bot* described in Sec. I plays an important role in the design and system analysis process. Firstly, let's assume that the *knowledge bot* does not exist in the process and we look at the main cycle of the design process illustrated in Fig. 1. The design process without the *knowledge bot* is simply the traditional design process of input-output relationship with the simulation software.

A. Design Process without Knowledge Bot

1) User wishes to evaluate the outcomes if several parameters are modified, so user requests the simulation of the system with a set of input parameters
2) The black-box software executes the simulation and provides a set of output
3) Execution is complete and output is returned to the user
*) Repeat. Eventually the user will need to re-evaluate the system with another set of modified input parameters, thus returning to Step 1.

Now observing this cycle, we see that the most time consuming event is Step 2. The physical simulation itself, may take a long time to execute. Furthermore, the user, if lacking expert intuition, may not be able to proceed with the design process until the outcome of the simulation is known. Additionally, if given sufficiently bad input parameters, the simulation may result in failure. However, only after the simulation is finished executing does the user finally discover such essential piece of information. The purpose of the *knowledge bot* is to alleviate such ineffectiveness by providing an instantaneous prediction of whether or not the simulation will execute without failure, such that unnecessary failing cases do not have to be simulated in the design process.

B. Design Process with Knowledge Bot

A1) User wishes to evaluate the outcomes if several parameters are modified, so user requests the simulation of the system with a set of input parameters. However, the input is given to the *knowledge bot*.
A2) The *knowledge bot* make a prediction whether simulation will be successful or failure
A3) If *knowledge bot* predicts successful simulation, user will perform the simulation. Otherwise the input parameters are adjusted and return to Step A2
A4) Predictive output is returned to the user
*) Repeat. Eventually the user will need to re-evaluate the system with another set of modified input parameters, thus return to Step A1.

The role of the *knowledge bot* is to provide a speedy prediction of the outcome of simulation without executing the simulation itself, using the given parameters. Thus, this response is significantly faster than having to execute the simulation. The user has a prediction of the outcome of the system, thus the design process can proceed knowing the general outcome provided from the *knowledge bot*. The training process for a *knowledge bot* is defined as:

L1) The *knowledge bot* uses a Monte Carlo sampling technique to select candidate input parameters. The technique uses pseudo-random numbers to select samples from a probability distribution. The Monte Carlo technique has some similarity to the Latin Hypercube technique, where samples are selected in a stratified domain with input probability distributions.
L2) The software finishes executing the simulation with the specified parameters and provides output.
L3) The *knowledge bot* uses the output to refine the learned model.
*) Repeat or stop. The *knowledge bot* either continues to learn from the simulation software or user decides that the learned model is sufficiently accurate and stops the learning process.

The purpose of the *knowledge bot* is to learn the underlying properties of physics-based simulations that govern the physical world to help accelerate the analysis and design process.

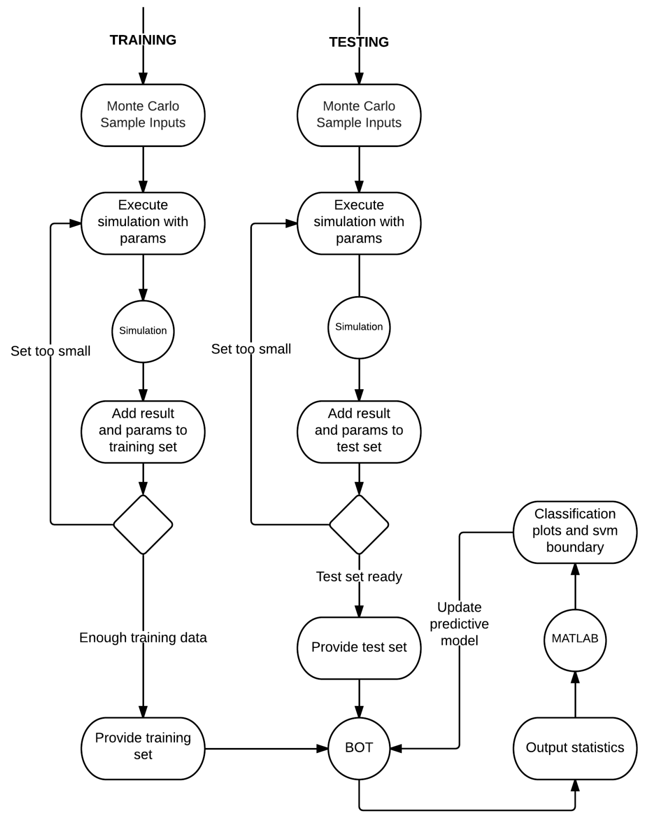

TRAINING
TESTING
Monte Carlo Sample Inputs
Monte Carlo Sample Inputs
Execute simulation with params
Execute simulation with params
Simulation
Simulation
Set too small
Set too small
Add result and params to training set
Add result and params to test set
Classification plots and svm boundary
Test set ready
Update predictive model
Enough training data
Provide test set
MATLAB
Provide training set
BOT
Output statistics

Fig. 3. Underlying mechanics of the knowledge bot, abstracted away in Fig. 2

Rather than having an analyst wait until after the simulation to realize that the input parameters results in a failed simulation, the knowledge bot is capable of providing a rapid prediction.

Partial differential equations are generally used in physics-based analysis of the physical world. The proposed *knowledge bot* is capable of learning the idiosyncrasies of various numerical methods that are used to solve such partial differential equations as well as capturing underlying properties of complex physics-based simulations.

Nominal classifiers were developed from a Multi-layer Perceptron neural network that is trained via the error back-propagation training algorithm. The problem formulation is based on a set of input and output vectors and a collection of hidden layers. The weight associated with links between layers are adjusted as learning rules propagate backwards from the output to input layer [3]. Furthermore, the learning rate for all sessions, unless specified otherwise is generally $r = 0.3$.

Weka[1] was used for training our neural network in this paper; Weka is a Java-based general purpose machine learning software developed at the University of Waikato, New Zealand.

IV. Knowledge Bot Mechanics

The knowledge bot represents an abstraction of simulation software, where to the user, the bot is simply a trained expert providing accurate predictions of simulation software behaviors based on user input parameters. The underlying predictive model, mechanics, and properties are abstracted, as illustrated in Fig. 2. However the *knowledge bot* creation encompasses various processes as shown in Fig. 3. The *knowledge bot* can be in a state of either training or testing. In the training state, the *knowledge bot* uses sufficiently large input data set and executes the simulation software. The outputs along with the sampled parameters are then collected in a training set. In our case, sufficiently large data set was denoted as 10000 instances. As it will be shown later, it is possible to create an accurate *knowledge bot* with less than 100 instances of simulations. The training set is then provided to a Java machine learning tool, Weka to develop the predictive model that is then fed back to the *knowledge bot* to make predictions. Similarly in the case of training, the bot again uses candidate samples input parameters, generating another sufficiently large set as the test set. The set is then provided to Weka, which is tested against the current predictive model of the knowledge bot. The resulting statistics are then given as output. MATLAB scripts is used to develop the classification results of each instance in the test set. We then used support vector machine algorithm to determine the boundary between simulation successes and failures. We also used MATLAB to collapse the resulting multidimensional classifications into two dimensions for not only visualization but also to describe in a single function the behavior of the classification boundary.

The abstraction of the *knowledge bot* allows for multipurpose simulation learning, as separate bots are trained to capture knowledge in various physics-based simulations. In fact, the concept of the *knowledge bot* can be implemented for any arbitrary physics-based simulations allowing for predictions of a wide range of physical problems and simulations.

V. Differential Equation Classification

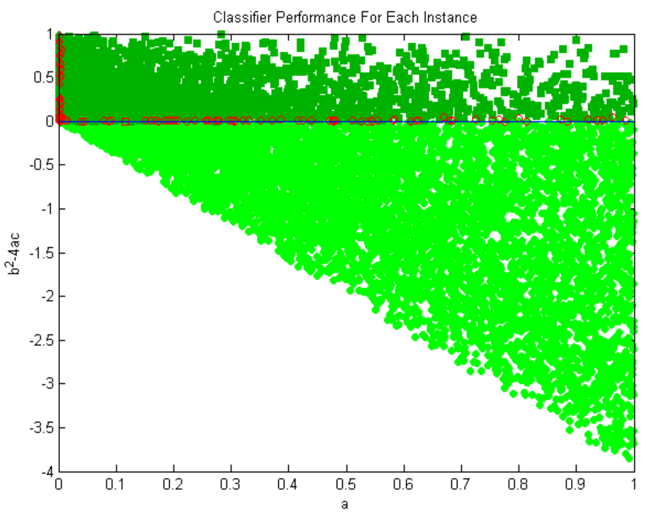
Most physics-based simulations can be accurately modeled by a set of differential equations. Especially in the case of aerospace vehicle design, fluid mechanics, structural mechanics, and heat transfer are some of the most important concepts taken into account in vehicle systems analysis. Furthermore, these important concepts are governed primarily by a set of differential equations that share a similar form. The below simple partial differential equation in two dimensions is used to describe the classifications.

$$a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + d\frac{\partial u}{\partial x} + e\frac{\partial u}{\partial y} + f = 0 \qquad (1)$$

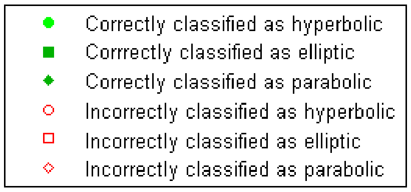
This can be classified into more specific families or classes of differential equations [18]. Given the set of coefficients a, b, c, it has been shown that $r = b^2 - 4ac$ can be used to classify these classes of equations based on the sign of r as follows,

1) parabolic differential equation, if $r = 0$

[1]http://www.cs.waikato.ac.nz/ml/weka/



(a) Classification of each instance classified described by (1)



●	Correctly classified as hyperbolic
■	Corrrectly classified as elliptic
◆	Correctly classified as parabolic
○	Incorrectly classified as hyperbolic
□	Incorrectly classified as elliptic
◇	Incorrectly classified as parabolic

(b) Classification legend

Fig. 4. Classification and legend

2) hyperbolic differential equation, if $r < 0$
3) elliptic differential equation, if $r > 0$

Given a simulation governed by a differential equation of the form described in (1), a method that classifies the equation into three distinctive classes and provides whether the input to the simulation will result in failure or success, similar to the acquired knowledge of a trained expert is demonstrated. The applications of numerical methods for these differential equations are treated as a piece of software with unknown underlying properties. The goal is to train *knowledge bot* to learn the underlying solution patterns and classify whether input to each of these three forms of differential equations will result in success or failure in numerical convergence and solution accuracy.

Provided that a partial differential equation is in the form described in (1), it can be classified into the three classes described earlier. This problem of classifying the type of differential equation is isomorphic to classifying whether the solutions to a *quadratic equation* is undefined, real, or complex (however, note that the a, b, c in the case of the quadratic classification corresponds to the coefficients of an equation in the form $ax^2 + bx + c$). A classifier was trained using 10000 instances generated by Monte Carlo sampling where $-1 \leq a, b, c \leq 1$. The artificial neural network model used for training consisted of a single hidden layer with twenty-five sigmoid nodes. The input node of the network corresponds to the a, b, c.

When the classifier, after being trained by a set of training data, is given a test set to classify, it may classify each instance as either,

1) $r = 0$, depicted as a diamond in Fig. 4(a)
2) $r > 0$, depicted as a square in Fig. 4(a)
3) $r < 0$, depicted as a circle in Fig. 4(a)

The classification of the 10000 instances was 98.8% accurate using a single hidden layer network architecture with twenty-five sigmoid nodes. Furthermore in Fig. 4(a), an instance may either be colored solid green or outlined as red. The green denotes that the prediction is correctly classified. The red outline denotes that the prediction was incorrect. Each of these cases corresponds to the classification of a class of differential equation (parabolic, hyperbolic, or elliptic). Refer to Fig. 4(b) for a pictorial legend. Note that this classifier simply classifies a partial differential equation in the form described by (1) as a general class or form, whether it be parabolic, hyperbolic, or elliptic. It does not suggest anything about the equation's stability or convergence. Simply given a general form of a partial differential equation, we can classify it as one of three classes of equations. A classifier that learns the convergence and stability of each of these forms will be discussed in following sections.

VI. Parabolic Equation: 1D-Heat Equation

To understand the properties of parabolic differential equations, a distinguishable example problem was selected - the heat equation problem simplified to a one-dimensional space with initial temperature $v(x) = V_0$ and boundary conditions with the temperature at zero. Thus, the physics of the problem assumes that the temperature change propagates from the boundaries, governed by the differential equation,

$$\rho c \frac{\partial v}{\partial t} = K \frac{\partial^2 v}{\partial x^2} \tag{2}$$

It can be shown by numerical differentiation and finite differences (using the FTCS, forward time central-space method described in [22]) that the following numerical method follows,

$$v_i^{n+1} = v_i^n + \frac{K \Delta t}{\rho c \Delta x^2}(v_{i+1}^n - 2v_i^n + v_{i-1}^n) \tag{3}$$

The exact solution, however, described in [21] in the case of constant initial temperature $v(x) = V_0$, is,

$$v = \frac{4V_0}{\pi} \sum_{n=0}^{\infty} \frac{1}{2n+1} e^{-\kappa(2n+1)^2\pi^2 t/l^2} sin \frac{(2n+1)\pi x}{l} \tag{4}$$

In our case, we assume $l = 1$ and $\kappa = K/\rho c$. The exact solution is evaluated as described in (4) to the n^{th} term where we decided that any terms that exceed $\varepsilon = e^{-20}$ is no longer significant for the purpose of our computations. Thus, n_{max} is determined as follows,

$$n_{max} = \frac{\sqrt{20l^2/(\kappa\pi^2 t)} - 1}{2} \tag{5}$$

The purpose of computing both the numerical solution and the exact solution (up to the n^{th} significant term) is that we

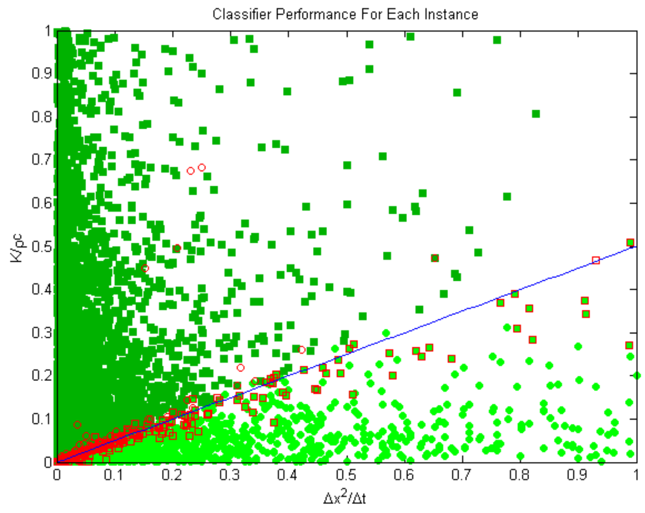
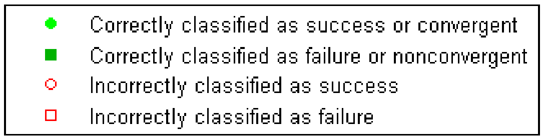


Fig. 5. Classification of each instance for parabolic 1D-heat equation



● Correctly classified as success or convergent
■ Correctly classified as failure or nonconvergent
○ Incorrectly classified as success
□ Incorrectly classified as failure

Fig. 6. Convergent-divergent classification legend

would like to treat the explicit numerical method as a black-box, such that it is a procedure that an individual without the knowledge of the heat equation or the procedure's underlying numerical methods can execute and produce the temperature distribution after sometime t for some positions $0 < x < 1$. Because the discretization in Eq. (3) is explicit in time, the algorithm will be unstable for some Δx and Δt combinations. Therefore, a classifier was trained to take the same input as the finite-difference procedure and to determine input validity and convergence of the numerical method such that an individual can query the classifier or *software bot* and determine whether a set of input will even execute correctly or fail on the black-box procedure prior to actually executing the code. The input parameters of the numerical method as well as the *software bot* are as follows,

1) N_x - number of discretized points in x $\Delta x = 1/(N_x - 1)$
2) N_t - number of discretized point in t $\Delta t = 1/(N_t - 1)$
3) V_0 - initial temperature of the substance (^oC)
4) K - conductivity of the substance $((cal/s)/(cm^{2o}C/cm))$
5) ρ - density of the substance (g/cm^3)
6) c - specific heat of the substance (cal/g^oC)

We used properties of platinum as an upper bounds for density, water for specific heat, and silver for conductivity for our training set and normalized the initial temperature, resulting in $0 < \rho \leq 22$, $0 < K, c, V_0 \leq 1$, and $0 < N_x, N_t \leq 101$. The training set for each of these networks, then is a seven-dimensional dataset encompassing the six input parameters and a convergent-divergent Boolean flag.

After training 10000 instances, the predictions of the classifier is illustrated in Fig. 5. The legend description is shown separately in Fig. 6, and teh following description will be used for the rest of the paper:

The classification of 10000 numerical simulations was over 99.2% accurate using a single layer network with twenty-five sigmoid nodes. It was shown by [23] that the explicit finite-difference numerical method has a convergence limit expression of $\alpha(\Delta t/\Delta x^2) \leq 1/2$, where in this case $\alpha = K/\rho c$. Thus, the deciding term $K/\rho c$ is plotted against the inverse of the step ratio resulting in $\Delta x^2/\Delta t$. The curve $K/\rho c = \Delta x^2/2\Delta t$ is depicted in Fig. 5.

Note, that the term *divergence* is used very loosely (in this section and in future sections); in this context, it is more alias to the term *non-convergence* which is denoted as the result as one of the following,

1) the input results in a failed execution or overflowed outputs
2) the input results in divergence (or oscillation) and results in arbitrarily large error, $\varepsilon \to \infty$
3) the input results in convergence but to a point that results in error that exceeds the maximum, $\varepsilon > \varepsilon_{max}$

VII. Hyperbolic Equation: Wave Equation

The wave equation can be used as an identifiable example of the class of hyperbolic partial differential equations. The one-dimensional wave equation, described in [19], which governs the simulation of vibrations of a string or rod, can be expressed as the partial differential equation,

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial^2 u}{\partial t^2} \tag{6}$$

where $u(x,t)$ represents the deflection at time t of a point x away from the origin. The hyperbolic wave equation is solved numerically using FTCS; the numerical method in [19] is shown to be,

$$u_i^{n+1} = \rho u_{i+1}^n + 2(1-\rho)u_i^n + \rho u_{i-1}^n - u_i^{n-1} \tag{7}$$

where,

$$\rho = \left(\frac{\Delta t}{\Delta x}\right)^2 \tag{8}$$

The analytic solution for wave equation is,

$$u(x,t) = \frac{1}{2}[f(x+t) + f(x-t)] \tag{9}$$

where $f(\cdot)$ is defined to be the initial deflection, thus $f(x) = u(x,0)$. In our case of training we assumed a sinusoid where $f(x) = sin(\pi x)$, since it encapsulates the boundary conditions $u(0,t) = u(1,t) = 0$. A Monte Carlo sampling of 10000 numerical solutions was used for training based on $0 < \rho < 2$ and varied $10 \leq N_x \leq 1001$ to compute $N_t = 1 + 1/\Delta t$, with $\Delta t = \sqrt{\rho}\Delta x$. The input parameters are then simply,

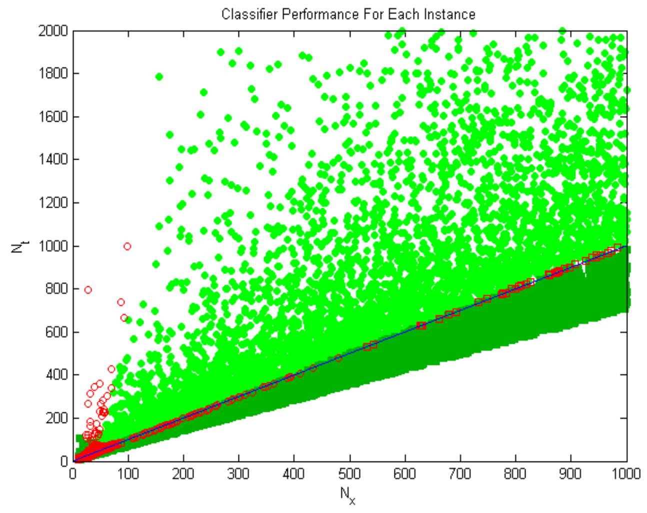


Fig. 7. Classification of each instance for hyperbolic wave equation

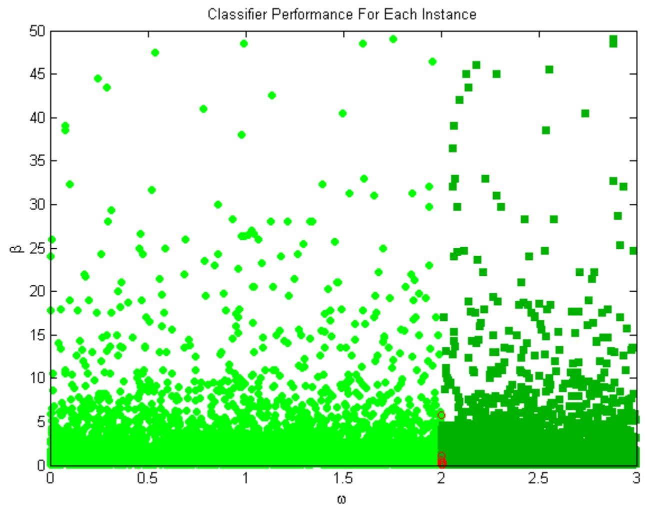


Fig. 8. Classification of each instance for elliptic Laplace's equation

1) ρ - the squared ratio of step sizes in t and x
2) N_x - the number of discretized x, $\Delta x = 1/(N_x - 1)$

The trained classifier was observed and compared to the the actual stability criteria defined to be $\rho \leq 1$, which can be interpreted as $N_x \leq N_t$.

After training 10000 instances, the predictions of the classifier is shown in Fig. 7. The legends are identical to the convergence predictions of the classifiers in previous sections (see Fig. 6). Recall that green denotes correct classification and circles denote that the classifier's prediction was convergent and squares correspond to divergent predictions. Note that the misclassifications lie along the $N_x = N_t$ curve. The classification of the 10000 instances for the hyperbolic wave equation problem was over 96.2% accurate using a single layer, twenty-five node network architecture.

VIII. Elliptic Equation: Laplace's Equation

Both Laplace's and Poisson's equations are important partial differential equations in mathematical physics and engineering described by [19] and are examples of elliptic equations. For the purpose of this paper, we used Laplace's equation as an example of an elliptic partial differential equation, which takes the form,

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \tag{10}$$

The analysis of steady conduction in a rectangular region with prescribed boundary temperatures as described in [20] and it is demonstrated that a scenario with boundary conditions where three of the four sides as well as the interior of the region are initialized to zero with the last side initially $T(x, y) = f(x)$, which in our case we select $f(x) = (\Delta u)sin(\pi x)$ since it satisfies the boundary conditions. The exact solution for a half-sine-wave boundary temperature distribution is indeed described for this particular problem,

$$u(x, y) = \Delta u \frac{sinh(\pi y/L)}{sinh(\pi H/L)} sin(\pi x/L) \tag{11}$$

where, Δu describes the amplitude of the sinusoid and assumed to be one in this study. Furthermore, the rectangular region is fixed as well for simplicity, thus, $H = L = 1$. The numerical solution to this elliptic partial differential problem can be expressed using the *successive over relaxation* (SOR) method denoted as,

$$u_{i,j}^{n+1} = u_{i,j}^{n} + \omega L(u_{i,j}^{n}) \tag{12}$$

where the function $L(\cdot)$ denotes the residual defined as,

$$L(u_{i,j}^{n}) = \frac{L_1(u_{i,j}^{n}) + \beta^2 L_2(u_{i,j}^{n})}{2(1+\beta^2)} \tag{13}$$

with $\beta = \frac{\Delta x}{\Delta y}$ and terms $L_1(\cdot)$ and $L_2(\cdot)$ are the central difference linear operators,

$$L_1(u_{i,j}^{n}) = u_{i+1,j}^{n} - 2u_{i,j}^{n} + u_{i-1,j}^{n+1} \tag{14}$$

$$L_2(u_{i,j}^{n}) = u_{i,j+1}^{n} - 2u_{i,j}^{n} + u_{i,j-1}^{n+1} \tag{15}$$

The term ω governs how aggressively the residual plays a role in the relaxation of $u_{i,j}^{n}$. Monte Carlo techniques were used to sample the input parameters, which for this particular problem are the discretizing quantity N_x, N_y and the weight for the residual ω. Therefore the input parameters are,

1) N_x - the number of discretized x, $\Delta x = 1/(N_x - 1)$
2) N_y - the number of discretized x, $\Delta y = 1/(N_y - 1)$
3) ω - the residual weight

The following input parameter domains were used in our demonstration, $0 < N_x, N_y \leq 101$ and $0 < \omega \leq 3$.

After training 10000 instances of the elliptic Laplace's equation, the classifier was over 99.9% accurate when classifying instances for convergence and non-convergence. The classification for each instance is demonstrated in Fig. 8, which demonstrates the predictions and whether they correspond to results of numerical method (see Fig. 6 for legend description). The network architecture of the classifier demonstrated is a single hidden layer, twenty-five node network. Note that the

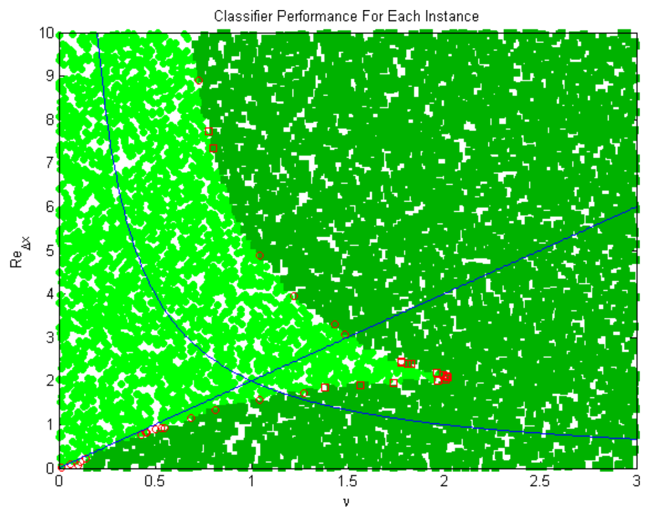


Fig. 9. Classification of each instance for linearized viscous Burgers' equation

misclassifications and boundary between convergent and non-convergent lies approximately at $\omega = 2$.

IX. Burgers' Equation

The Burgers' equation is a differential equation that describes gas dynamics behavior. It includes a convective term that adds some complexity in the formulation. We chose linearized viscous Burgers' equation for this study, and the equation is defined [23] as:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \mu \frac{\partial^2 u}{\partial x^2} \tag{16}$$

The exact steady-state solution of the linearized viscous Burgers' equation can be computed by the following,

$$u = u_0 \left\{ \frac{1 - e^{R_L(\frac{x}{L-1})}}{1 - e^{-R_L}} \right\} \tag{17}$$

where

$$R_L = \frac{cL}{\mu} \tag{18}$$

Using an explicit forward-time and central-space finite differences, the viscous linearized Burger's equation can be discretized as:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} + c\frac{u_{j+1}^n - u_{j-1}^n}{2\Delta x} = \mu \frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{(\Delta x)^2} \tag{19}$$

Because the method is explicit, it may run into stability issues. We trained a *software bot* to learn about the stability behavior of Burgers' equation for the following selected parameters: $\Delta x, \Delta t$, u_0, c, and μ.

We used a Monte Carlo technique to select the input parameters that are used solve the Burgers' equation. The input parameters and resulting solution are used to train an artificial neural network to predict stability behavior of discretized Burgers' equation. The parameter $L = 1.0$ was fixed as the specified domain of interest, thus the input parameters are as follows,

1) N_x - number of discretized x, $\Delta x = 1/(N_x - 1)$
2) N_t - number of discretized t, $\Delta t = 1/(N_t - 1)$
3) u_0 - the initial condition, initial velocity
4) c - wave speed
5) μ - viscosity of the fluid

Some parameters were normalized, resulting in the following ranges of our input parameters $0 < u_0, c, \mu \leq 1$. For the viscous linearized Burgers' equation, we selected $0 < N_x, N_t \leq 1001$. When adjusting the network architecture the input nodes are labeled corresponding to the input parameters N_x, N_t, u_0, c, and μ, respectively.

The stability criteria described in [23] for the explicit finite-difference numerical method holds a convergence expression in terms of a heuristic stability region of,

$$2\nu \leq Re_{\Delta x} \leq 2/\nu \tag{20}$$

where the mesh Reynolds number $Re_{\Delta x} = c\Delta x/\mu$ and stability constraint $\nu = c\Delta t/\Delta x$. To study the impact of $Re_{\Delta x}$ and ν on stability, we fixed $N_x = N_t = 101$ and $u_0 = L = 1$. We used Monte Carlo sample distributions for $Re_{\Delta x}$ and v. The domain of our problem then lies in the following area, $0 \leq Re_{\Delta_x} < 10$ and $0 < \nu \leq 3$.

After training with 10000 instances, using a single layer with twenty-five sigmoid node network, the classification to predict stability was over 99.6% accurate. Results are shown in Fig. 9, which also incorporates the heuristic stability defined by Eq. (20) illustrated as the two blue curves. See Fig. 6 for legend description. Note that the heuristic stability described by Eq. (20) is much more conservative than capability of the numerical method as shown in Fig. 9, where the method is stable in regions outside the heuristic stability. The artificial neural network predicts the stability regions far more accurately than the heuristic approach.

X. Error Behaviors in Numerical Methods

The numerical methods presented in previous sections are approximate solutions to the differential equations. In order to evaluate and learn the accuracy behavior of these numerical methods, *software bots* were trained to learn and predict how accurate numerical solutions for a given a set of input parameters. The same numerical methods and exact solutions as presented in the previous sections were used but we provide the error between the numerical method and the exact solution as opposed to a convergence-divergence Boolean flag in the training data. The error is computed as the l^2-norm or Euclidean norm of the difference between the two solutions. Furthermore, if the numerical method diverges, then the method may exhibit arbitrarily large error (until overflow occurs). Thus if divergence begins to occur, we assume that the error is $\varepsilon_{max} = 1.0$, which is an upper bounds on the error for our training set, therefore, the absolute error is normalized such that $0 \leq \varepsilon \leq 1$.

A numerical predictor was trained to learn the l^2-norm error behavior for each numerical method to predict the accuracy of convergence for specified input parameters. In other words,

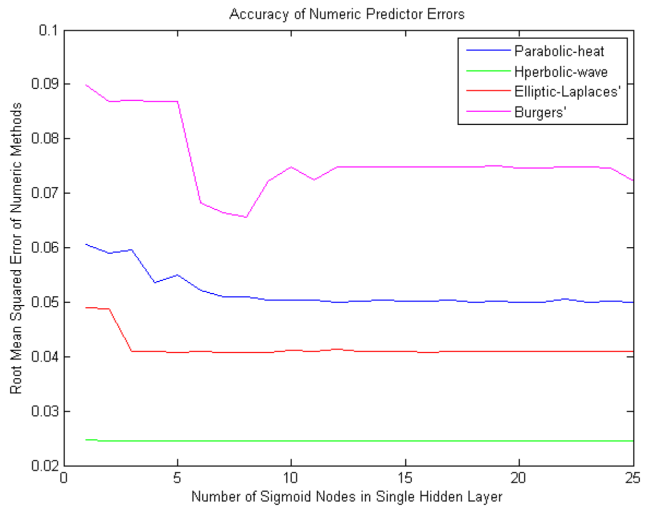


Fig. 10. Root mean squared error of the numerical predictor as network complexity increases for each numerical method *convergence error predictor*

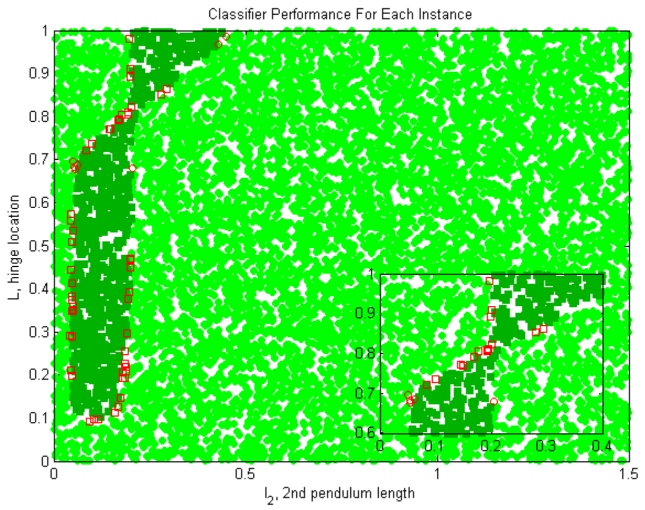


Fig. 11. Classification of each instance for the slosh propellant dynamics pendulum model problem

the predictor acts as a *convergence error predictor* for each numerical method, predicting the error between the numerical solution and the exact solution. The root mean squared error of the predictor is illustrated in Fig. 10 as the number of nodes in the hidden layer increases. For this demonstration, a single hidden layer network with varying sigmoid nodes is used. It is shown that the predictors are able to predict the error of the numerical method within $\varepsilon_{nm} = 0.05$ for methods in the form described in (1) of Sec. V. Burgers' equation, resulted in a numerical prediction error that was slightly higher with $\varepsilon_{nm} = 0.07$. We see that the true ε_{nm} varies from method to method, and that as the number of nodes increase in the network architecture, the predictors plateaus at their own respective ε_{nm}. The error can be further reduced by including additional network layers.

XI. Slosh Propellant Dynamics

It has been demonstrated in previous sections that artificial neural networks can be used to predict and classify stability criteria of numerical methods for computational fluid dynamics in addition to predicting the success and failure of simulations when given a set of input parameters for simulation software. However, the use of neural network training can be extended to learning other complex models.

For large launch vehicles, a significant portion of the gross mass is taken up by liquid propellant, thus fuel sloshing dynamics plays an important role in vehicle stability [24]. The issue of slosh dynamics is a crucial factor for spacecraft and various other large flight vehicles. Sloshing dynamics may results in anomalies in flight with partial loss of control due to oscillations resulting from propellant sloshing citeslosh-motiv and [26]. In [28], it was shown that proportional-integral-derivative (PID) control systems can be designed to provide further stability and "avoiding adverse interaction with the propellant slosh and structural dynamics." It was shown later that the slosh dynamics in a spacecraft can be modeled and simulated using a pole-cart system, in which the pole-cart acts as an inverted pendulum and attached to it is a second pendulum which acts as the slosh load [24]; the problem was simplified to observe the system stability in terms of the hinge location (the position where the second pendulum is attached) and the length of the second pendulum with the goal to establish an analytical expression that predicts the second pendulum's instability.

We used *knowledge bot* to classify whether parameters of this system may correspond to instability in the propellant slosh dynamics. The pendulum system is treated as a simulation with the parameters defined as,

1) L - hinge location (position 2nd pendulum is attached)
2) l_2 - the length of the second pendulum

The data given by [24] was interpolated to generate a denser model. Monte Carlo sampling technique was used to generate the training set for the artificial neural network; the sampling was within the domain of $0m \leq L \leq 1.0m$; $0m \leq l_2 \leq 1.5m$.

After training with a set of 10000 instances, the accuracy of artificial neural network classification was over 99.4% accurate. The classification details are illustrated in Fig. 11. Note that the instances classified as unstable (depicted as green squares) lie generally in the $l_2 < 0.5$ region. Furthermore, misclassifications lie along the boundary of stable and unstable instances. The results demonstrate that artificial neural network can accurately predict complicated stability region.

XII. Pressure Vessels Design Concepts

The same methodologies of training can be demonstrated with commercial simulation software. For this demonstration, we observe the design of pressure vessels using NASTRAN [30], a commercial finite element analysis simulation program initially developed for NASA. The design concept of pressure vessels for planetary probe missions depends on key parameters such as the external pressure and allowable stress the system undergoes. In terms of systems analysis, the

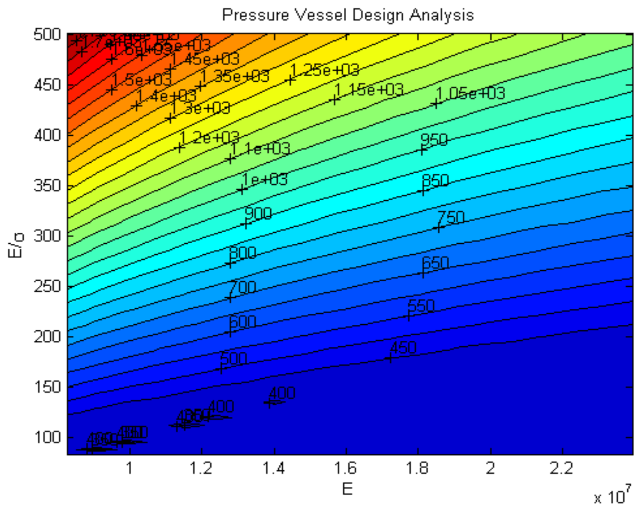


Fig. 12. Pressure vessel design contour ($m\sqrt{E}$) with σ and E-dominance

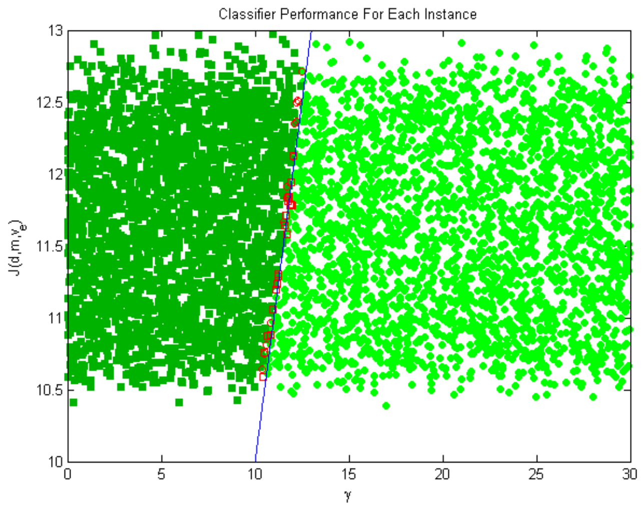


Fig. 13. Classification of each instance for direct entry simulations

design of the pressure vessel's mass is highly correlated to the maximum allowable stress σ and the modulus elasticity E of the material. Generally, when designing the vessel, a material with sufficiently high σ is selected in order to meet constraints, however, in reality, there exists instances in which E could dominate the design. Thus, a classifier or *knowledge bot* is used to classify and predict the instances for which the two governing parameters σ and E dominate in contribution to pressure vessel design. For this particular problem we used NASTRAN (MSC commercial Finite-Element Analysis tool [29]) for simulating optimal design mass when given the parameters σ and E.

The mass m of the pressure vessel varies inversely with the root of the modulus elasticity $\sqrt{E}$ as described by [31]. Therefore, observing the contour plots of $m\sqrt{E}$ in the material domain $0 < E \leq 2400000$, $0 < E/\sigma \leq 500$ demonstrated in Fig. 12, demonstrates that when $m\sqrt{E} \leq 450$ is constant, corresponding to the region of E-dominance. Conversely, the region described by $m\sqrt{E} > 450$ then corresponds to the region of σ-dominance. Therefore, the magnitude of , $\sqrt{E}$ was used as a identifier for whether the region was E-dominant or σ-dominant. By σ, E-dominances, we refer to the fact that either σ or E is the driving parameter in terms of design and mass contribution.

The training for the pressure vessel design concepts using NASTRAN will be discussed in a later section. But firstly, we will observe another type of simulation software and introduce a method to visually demonstrate the classifications of multi-dimensional results. In Sec. XIV, we will integrate this with the previously discussed training sets as well as the NASTRAN pressure vessel design classifications.

XIII. Planetary Direct Entry Simulations

The method of atmospheric-assisted *direct entry* for aerospace vehicles is a widely explored concept, governed by complex dynamics of systems of nonlinear partial differential equations. An analytic solution for a simplified version of the direct entry problem is described in [32]. We used POST2 (Program to Optimize Simulated Trajectories II), a point mass, discrete parameter targeting and optimization program. The goal for this simulation was to predict capture into Venus atmosphere, making atmospheric-assisted *direct entry* possible. Four potential input parameters that are most essential to the direct entry problem were observed,

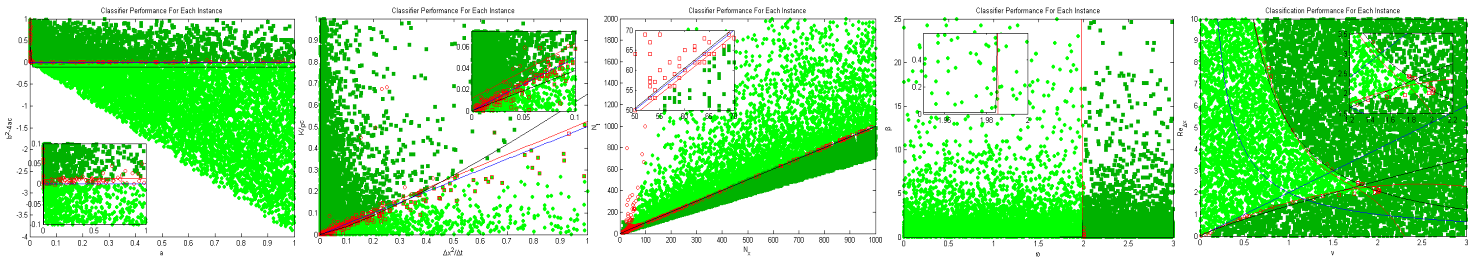
1) d - the diameter of the capsule
2) m - the mass of the capsule
3) v_e - the entry velocity of the capsule
4) γ_e - the entry flight path angle

The Monte Carlo sampling techniques was used in the domains of, $0.5m \leq d \leq 4.0m$; $500kg \leq m \leq 5000kg$; $10km/s \leq v_e \leq 13km/s$; and $-30^o \leq \gamma_e < 0^o$.

POST2 solves a system of nonlinear partial differential equations of motion and generates a trajectory for the capsule given a set of input parameters. We determine if the input parameters allowed for successful direct entry by observing the resulting trajectory. If the trajectory resulted in a final position within $\varepsilon \leq 1km$ of the surface of the body, then we consider the capsule to be captured into the atmosphere and as a result a successful direct entry trial. For the purpose of demonstration, a direct entry simulation by a 45^o sphere-cone capsule into the atmosphere of the planet Venus was observed by generating 10000 instances of training data via Monte Carlo sampling within the specified domains for input parameters to interrogate the POST2 software.

An artificial neural network was trained using the 10000 training instances. Furthermore, we used support vector machine (SVM) to identify the boundary between the trajectory cases with successful and failed direct entry parameter. The instances on the boundary was then used for nonlinear least squares regression to identify a nonlinear expression to classify successful direct entry cases in the following form,

$$J(d, m, v_e) = am^b v_e^c d^d \tag{21}$$



(a) Sec. V classifier (b) Sec. VI classifier (c) Sec. VII classifier (d) Sec. VIII classifier (e) Sec. IX classifier

Fig. 14. Estimated boundaries of numerical method classifiers using support vector machine regression on close-perspective classification plots

Solving for the exponents $a, b, c,$ and d using least squares, the resulting function $J(\cdot)$ can be used to classify the problem of atmospheric-assisted direct entry given the parameters $d, m, v_e,$ and γ_e such that,

$$J(d, m, v_e) = 0.0219 m^{0.0096} v_e^{0.6646} d^{-0.0017} \tag{22}$$

1) if $\gamma_e < J(d, m, v_e)$, predicted skipped entry
2) if $\gamma_e > J(d, m, v_e)$, predicted successful capture

Furthermore, note that $J(\cdot)$ is defined in terms of m, v_e, d. The reason behind this is that γ_e is found to the be most influencing parameter in successful direct entries, thus we chose to isolate this in two-dimensional space resulting in $J(\cdot)$ defined in terms of the remaining parameters.

The resulting classification on a separate set of 10000 Monte Carlo sample input parameters is depicted in Fig. 13. The curve described in (22) is illustrated as the boundary between instances that were success direct entry parameters and instances that resulted in failure (skipped entries). The legend is similar to the previous sections with convergence relating to the success of the direct entry. The classification for the success of direct entry simulations given $d, m, v_e,$ and γ_e using a single layer twenty-five sigmoid node network was over 99.6% accurate.

XIV. Support Vector Machine Regression

It was shown in the previous section that using the support vectors generated by support vector machine, combined with a nonlinear least squares regression, we can calculate an analytical approximate boundary between different classifications (e.g., successful and unsuccessful instances). Furthermore, this process enabled us to reduce our multidimensional problem into two dimensions for more effective visualization. In this section we revisit the numerical method classifiers described in the previous sections and capture the boundary instances via support vector machine to compare the curve generated by regressions of these instances against the known analytical and heuristic boundaries.

Support vector machine is a supervised machine learning technique that constructs one or more hyperplanes in multidimensional space. The algorithm was originally proposed by Vladimir N. Vapnik [34] in 1995. The algorithm has been widely used in various machine learning applications, including text classification [35]; image retrieval [36] and [36]; biology and bioinformatics predicting protein subcellular localization [38], protein primary, secondary structures [39] and [40]; and bankruptcy prediction [41]. Additionally, other applications of the technique demonstrates its classification ability in terms of brain state [42] and drug use classification [43]. Although support vector machine can produce an accurate classifier for a properly-selected dataset, the technique was chosen due to its ability to identify boundaries among discrete classifications. The identified boundaries can be used to develop an analytical expression representing the boundary between failure and success cases. Similar to the previous section, we used SVM to reduce our multidimensional classification results in two dimensions for more effective visualization.

Support vector machine was used with previous classifiers to identify the boundary of success and failure (e.g., convergence or *non-convergence*) such that an approximate analytic boundary can be developed and compared to the classifier's boundary. Both polynomial interpolation and nonlinear regression were used on the classification results demonstrated in the previous sections.

Let ∂_0 denote the mathematically derived boundaries (e.g., heuristic stability boundary) and $\dot{\partial}_1$ denote the boundary curve predicted by using polynomial fit and $\dot{\partial}_2$ denote the boundary curve determined by nonlinear regression on the instances identified by SVM. The nonlinear regression was completed using the model of the form,

$$Y_{param} = C + \alpha (X_{param})^{\beta} \tag{23}$$

where X_{param}, Y_{param} denote the expressions denoted in the x-axis and y-axis of the classification plot and the unknown constants $C, \alpha,$ and β was calculated using nonlinear regression. Furthermore, piecewise regressions and fitting was done on the classifier described in Fig. 14(e). Note in Fig. 14, ∂_0 is labeled by a blue curve, $\dot{\partial}_1$ by a red curve and $\dot{\partial}_2$ by a black curve, as a result the following approximations for the numerical method classifiers in the previous sections are identified,

1) Differential equation classifier, Fig. 14(a)
   ∂_0 $\quad b^2 - 4ac = 0$
   $\dot{\partial}_1$ $\quad b^2 - 4ac = 0.0117 + 0.0029a$
   $\dot{\partial}_2$ $\quad b^2 - 4ac = 0.444 - 0.555a^{\varepsilon}, \varepsilon = 1.675E\text{-}22$
2) Parabolic 1D-heat equation classifier, Fig. 14(b)
   ∂_0 $\quad K/\rho c = 0.5(\Delta x^2/\Delta t)$

$\dot{\partial}_1$ $K/\rho c = 0.5131(\Delta x^2/\Delta t) - 0.0016$
$\dot{\partial}_2$ $K/\rho c = 0.652(\Delta x^2/\Delta t)^{1.237}$

3) Hyperbolic wave equation classifier, Fig. 14(c)
∂_0 $N_t = N_x$
$\dot{\partial}_1$ $N_t = 1.0019(N_x) - 1.2713$
$\dot{\partial}_2$ $N_t = 1.025(N_x)^{0.996}$

4) Elliptic Laplace's equation classifier, Fig. 14(d)
∂_0 No analytic stability criteria
$\dot{\partial}_1$ $\omega = 1.9855 - 0.0002\beta$
$\dot{\partial}_2$ $\omega = 1.493 + 0.493\beta^{0.00036}$

5) Burger's equation classifier, Fig. 14(e)
∂_0 $Re_{\Delta x} = 2/\nu$
$\dot{\partial}_1$ $Re_{\Delta x} = 0.38\nu^4 - 6.4\nu^3 + 25.7\nu^2 - 41.5\nu + 26.9$
$\dot{\partial}_2$ $Re_{\Delta x} = 5.130\nu^{-1.306}$
∂_0 $Re_{\Delta x} = 2\nu$
$\dot{\partial}_1$ $Re_{\Delta x} = -0.39\nu^2 + 1.94\nu - 0.003$
$\dot{\partial}_2$ $Re_{\Delta x} = 1.402\nu^{0.855}$

The boundary predictions of both polynomial and nonlinear regression fitting, $\dot{\partial}_1, \dot{\partial}_2$ in cases 1-4, produced via regression on the support vector instances, were sufficiently good approximations to the analytical boundary. The insets in Fig. 14 demonstrate a closer view of the classification plots, demonstrating both the analytic boundary ∂_0 as well as the predicted boundaries $\dot{\partial}_1$ and $\dot{\partial}_2$.

Note that the regression for the classifier demonstrated in Sec. IX for Burgers' Equation is done via a split at $Re_{\Delta x} = 1.5$, resulting in two curves for the upper and lower constraints for the stability of the numerical method. The resulting equations for the upper and lower constraints demonstrated by polynomial interpolation ($\dot{\partial}_1$) are given as a fourth order and second order equation. The approximations for $\dot{\partial}_1$ and $\dot{\partial}_2$ provide a sufficiently good border estimations, illustrated in Fig. 14(e) as red curves and black curves respectively. The support vector machine polynomial interpolation ($\dot{\partial}_1$, red) curves and the nonlinear regression ($\dot{\partial}_2$, black) provides much closer fit to the stability border than the heuristic stability constraints ∂_0 described in (20) and illustrated in Fig. 14(e) as blue curves. Generally speaking via observations, the polynomial fitting technique producing boundary curves much closer to the true boundaries with well-behaved, mostly-linear boundaries. However, in the case of Burgers' stability domain, the nonlinear regression with powers performs slightly more accurate as demonstrated in Fig. 14(e) due to its highly-nonlinear boundary.

A. Pressure Vessel Design using NASTRAN

Fig. 15 illustrates the classifications results using 10000 instances for training and 10000 for testing. The resulting classifier, obtained by a single hidden layer network with twenty-five sigmoid nodes, was over 99.5% accurate. Note that the classification described here do not denote failure and success rather the regions of E-dominance (green circles) and σ-dominance (green squares) are illustrated (with previous conventions circle corresponded to success and square to failure, thus one may observe the plots with E-dominance

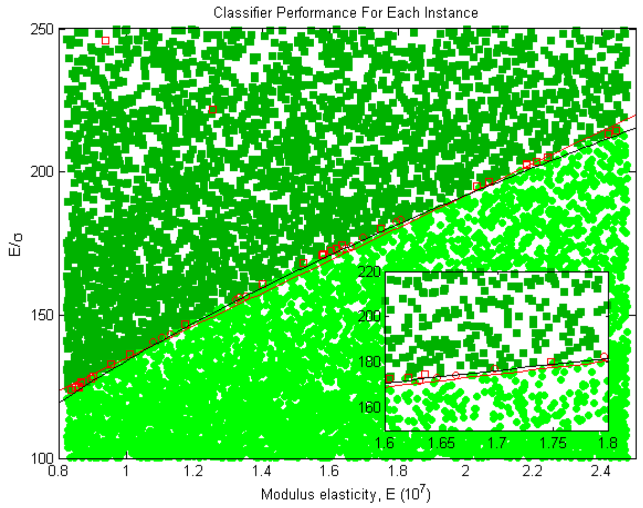


Fig. 15. Classification of each instance for the pressure vessel simulation

corresponding to success in mind). The red outlines represent incorrectly classified instances which were either identified as E and σ but were in reality the opposite.

The boundary equation between successful and failure cases was found via support vector machine and polynomial interpolation resulting in the following fourth-order equation relating the modulus elasticity E to the ratio E/σ at the boundary,

$$J_1(E) = -0.28E^4 + 1.4E^3 - 2.7E^2 + 58.9E + 77.6 \quad (24)$$

The technique of polynomial interpolation (red) was compared to nonlinear regression (black) illustrated in Fig. 15. The nonlinear regression of the boundary curve in terms of the power equation, described in (23) for the pressure vessel NASTRAN simulations is,

$$J_2(E) = .031977E^{0.51747} \quad (25)$$

Thus, $J(E) = E/\sigma$ corresponds to the instances along the boundary (illustrated as a red line for polynomial interpolation and black for nonlinear regression in Fig. 15). Furthermore, similar to the case described in the previous section, the function $J(\cdot)$ can be used to classify instances as either σ-dominant or E-dominant generally by the following guidelines,

1) if $E/\sigma > J(E)$, predicted σ-dominant
2) if $E/\sigma < J(E)$, predicted E-dominant

Figure 15 shows the boundary generated by nonlinear regression in the specified power form (black) provided a closer fit to the true boundary in the case of the NASTRAN simulation problem as opposed to the boundary generated by polynomial interpolation (red).

XV. Impact of Number of Instances on Accuracy

It should be noted that the training and testing performed in all previous sections with artificial neural networks used sets of 10000 instances. We studied the impact of training size on the accuracy of the learned artificial neural network model for

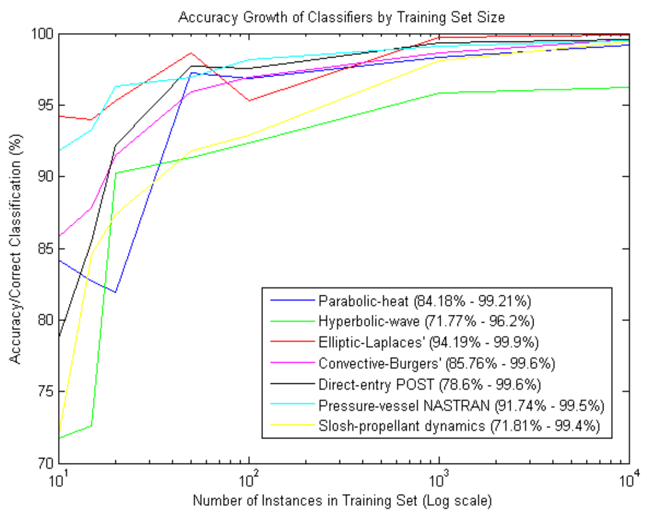


Fig. 16. Accuracy growth with set size increase for mentioned classifiers

the previous discussed problems at discretized set sizes for the training dataset $(10, 15, 20, 50, 100, 1000, 1000)$.

Figure 16 illustrates the accuracy of the classifiers as function of training set size. With a minimum of ten training instances, the classifiers have a mean accuracy of 80% with the minimum being the hyperbolic-wave equation classifier at 71.77%. At 10000 training instances, the resulting neural network classifier achieves over 99% accuracy, with the exception of the hyperbolic-wave equation (96.2%); the classifiers all exhibit similar behavior in achieving over 90% accuracy with less than 100 training instances and exhibits growth in accuracy, which corresponds to eventual convergence to all solutions of the black-box simulation software. Furthermore, some problems are presumably better suited and easier to predict with artificial neural networks. For example, the Laplace equation and pressure-vessel NASTRAN classifiers were over 90% accurate when trained with only 10 training instances. The legend depicted illustrates the maximum and minimum accuracy values between the range of 10 to 10000 instances in the training set. All classifiers demonstrated are achieved via an artificial neural network with a single hidden layer, twenty-five sigmoid node architecture.

XVI. Conclusion

We have shown that machine learning techniques can be used to learn the idiosyncrasies of computational simulations and identify regions of instability without including any additional information about mathematical modeling of partial differential equations. We have also demonstrated that the use of artificial neural network to learn the underlying properties of numerical method for solving differential equations is both feasible and accurate. The applications in this study included various types of physical simulation of fluid and heat flows important to systems analysis of an aerospace vehicle.

By training classifiers using neural networks, we develop *knowledge bots* to detect simulation failures prior to execution and identify idiosyncrasies of unknown black-box simulation software. Furthermore, we demonstrated that the use of support vector machines helps to reduce the multidimensional and highly nonlinear classification datasets by obtaining simple expressions describing the boundary between success and failure instances in the dataset.

Acknowledgements

Jay Ming Wong was funded by the Langley Aerospace Research Student Scholars (LARSS) program under the Virginia Space Grant Consortium. We would like to thank Manjula Amber for funding this activity. The authors would like to thank Charles Liles for his continued support and help with regards to the machine learning tool Weka. Furthermore, we would like to thank the Machine Learning Group at the University of Waikato for developing the Weka software which was used for a majority of the neural network training. Finally, we thank Jing Pei for his inspirations and datasets that contributed to the slosh dynamics section.

References

[1] McCulloch, Warren; Walter Pitts (1943). “A Logical Calculus of Ideas Immanent in Nervous Activity”. Bulletin of Mathematical Biophysics 5 (4): 115133.

[2] Rosenblatt, F. (1958). “The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain”. Psychological Review 65 (6): 386408.

[3] Zurada, J. Introduction to Artificial Neural Systems. 0-314-93391-3. West Publishing Co. 1992.

[4] I.H. Witten et al., Data Mining: Practical Machine Learning Tools and Techniques (3rd ed.). Burlington, MA: Elsevier Inc., 2011.

[5] Kurt Hornik, Maxwell B. Stinchcombe, Halbert White: Multilayer feedforward networks are universal approximators. Neural Networks 2(5): 359-366 (1989).

[6] Lewis, F. W., Suresh Jagannathan, and A. Yesildirak. Neural network control of robot manipulators and non-linear systems. CRC Press, 1998.

[7] Ge, Shuzhi S., and Christopher J. Harris. Adaptive neural network control of robotic manipulators. World Scientific Publishing Co., Inc., 1998.

[8] Yang, Simon X., and Max Meng. “An efficient neural network approach to dynamic robot motion planning.” Neural Networks 13.2 (2000): 143-148.

[9] Rowley, Henry A., Shumeet Baluja, and Takeo Kanade. “Neural network-based face detection.” Pattern Analysis and Machine Intelligence, IEEE Transactions on 20.1 (1998): 23-38.

[10] Golomb, Beatrice A., David T. Lawrence, and Terrence J. Sejnowski. “SEXNET: A Neural Network Identifies Sex From Human Faces.” NIPS. 1990.

[11] Holley, L. Howard, and Martin Karplus. “Protein secondary structure prediction with a neural network.” Proceedings of the National Academy of Sciences 86.1 (1989): 152-156.

[12] Baxt, William G. “Application of artificial neural networks to clinical medicine.” The lancet 346.8983 (1995): 1135-1138.

[13] Azoff, E. Michael. Neural network time series forecasting of financial markets. John Wiley & Sons, Inc., 1994.

[14] Odom, M.D.; Sharda, R., “A neural network model for bankruptcy prediction,” Neural Networks, 1990., 1990 IJCNN International Joint Conference on , vol., no., pp.163,168 vol.2, 17-21 June 1990.

[15] Ahn, B. S., S. S. Cho, and C. Y. Kim. ”The integrated methodology of rough set theory and artificial neural network for business failure prediction.” Expert Systems with Applications 18.2 (2000): 65-74.

[16] D. C. Ciresan, U. Meier, J. Masci, J. Schmidhuber. Multi-Column Deep Neural Network for Traffic Sign Classification. Neural Networks, 2012.

[17] Liles, C. Neural Network Machine Learning and Dimensional Reduction for Data Visualization. NASA/TM-2014-218181. 2014.

[18] Loret, Benjamin. “Classication of partial dierential equations into elliptic, parabolic and hyperbolic types”. 12 Dec 2008.

[19] W. Chenney and D. Kincaid. Numerical Mathematics and Computing. 2nd ed. Pacific Grove, California: Brooks/Cole Publishing Company. 1985.

[20] F. M. White. Heat and Mass Transfer. Reading Massachusetts: Addison-Wesley Publishing Company. 1988.

[21] H. Carslaw and J. Jaeger. Conduction of Heat in Solids. 2nd ed. Oxford at the Clarendon Press. 1959.

[22] Patrick J. Roache. Computational Fluid Dynamics (1st ed.). Hermosa. 1972.

[23] D. A. Anderson, J. C. Tannehill, R. H. Pletcher. Computational Fluid Mechanics and Heat Transfer. New York: Hemisphere Publishing Corporation., 1984.

[24] Jing Pei. “Analytical Look at Launch Vehicle Propellant Slosh Instability”. Vehicle Analysis Branch Tech Talk. Presentation. Jun 12, 2014.

[25] Reyhanoglu, M. “Maneuvering control problems for a spacecraft with unactuated fuel slosh dynamics.” Control Applications, 2003. Proc 2003 IEEE Conference. Volume 1, 23-25 June 2003, pp695-699.

[26] Veldman, A.E.P. et al. “The Numerical Simulation of Liquid Sloshing On-Board Spacecraft.” J. Comp. Phys. 224 (2007) 82-99.

[27] “Falcon Demo Flight 2 Flight Review Update.” SpaceX. June 15, 2007.

[28] Jing Pei, John Wall. Estimation of Aerodynamic Stability Derivatives for Space Launch System and Impact on Stability Margins. NASA/TM-2014-0002405. 2014.

[29] MacNeal, Richard H., page i., “The NASTRAN Theoretical Manual”, December 1972.

[30] “NASA Press Release 2008.” http://www.nasa.gov/centers/dryden/news/X-Press/stories/2008/10_08_technology.html. 2008.

[31] Jamshid Samarah and Sasan Armand. “Pressure Vessel Design Concepts for Planetary Probe Missios.” 11th International Planetary Probe Workshop. Pasadena, CA. Jun 16-20, 2014.

[32] F. J. Regan. S. M. Anandakrishnan. Re-Entry Vehicle Dynamics (AIAA Education Series). New York: American Institute of Aeronautics and Astronautics, Inc. ISBN 0-915928-78-7. 1984.

[33] Brauer, G. L., Cornick, D. E. and Stevenson, R., “Capabilities and Applications of the Program to Optimize Simulated Trajectories (POST), NASA CR-2770, Feb. 1977.

[34] Cortes, C.; Vapnik, V. (1995). “Support-vector networks”. Machine Learning 20 (3): 273.

[35] Tong, Simon, and Daphne Koller. “Support vector machine active learning with applications to text classification.” The Journal of Machine Learning Research 2 (2002): 45-66.

[36] Tong, Simon, and Edward Chang. “Support vector machine active learning for image retrieval.” Proceedings of the ninth ACM international conference on Multimedia. ACM, 2001.

[37] Zhang, Lei, Fuzong Lin, and Bo Zhang. “Support vector machine learning for image retrieval.” Image Processing, 2001. Proceedings. 2001 International Conference on. Vol. 2. IEEE, 2001.

[38] Hua, Sujun, and Zhirong Sun. “Support vector machine approach for protein subcellular localization prediction.” Bioinformatics 17.8 (2001): 721-728.

[39] Cai, C. Z., et al. “SVM-Prot: web-based support vector machine software for functional classification of a protein from its primary sequence.” Nucleic acids research 31.13 (2003): 3692-3697.

[40] Hua, Sujun, and Zhirong Sun. “A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach.” Journal of molecular biology 308.2 (2001): 397-407.

[41] Min, Jae H., and Young-Chan Lee. “Bankruptcy prediction using support vector machine with optimal choice of kernel function parameters.” Expert systems with applications 28.4 (2005): 603-614.

[42] Mourao-Miranda, Janaina, et al. “Classifying brain states and determining the discriminating activation patterns: support vector machine on functional MRI data.” NeuroImage 28.4 (2005): 980-995.

[43] Byvatov, Evgeny, et al. “Comparison of support vector machine and artificial neural network systems for drug/nondrug classification.” Journal of Chemical Information and Computer Sciences 43.6 (2003): 1882-1889.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YYYY)* 01- 11 - 2014	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Training Knowledge Bots for Physics-Based Simulations Using Artificial Neural Networks	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Wong, Jay Ming; Samareh, Jamshid A.	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER 470883.04.07.01.03
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-20493
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, DC 20546-0001	**10. SPONSOR/MONITOR'S ACRONYM(S)** NASA
	11. SPONSOR/MONITOR'S REPORT NUMBER(S) NASA/TM-2014-218660

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 63
Availability: NASA CASI (443) 757-5802

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Millions of complex physics-based simulations are required for design of an aerospace vehicle. These simulations are usually performed by highly trained and skilled analysts, who execute, monitor, and steer each simulation. Analysts rely heavily on their broad experience that may have taken 20-30 years to accumulate. In addition, the simulation software is complex in nature, requiring significant computational resources. Simulations of system of systems become even more complex and are beyond human capacity to effectively learn their behavior. IBM has developed machines that can learn and compete successfully with a chess grandmaster and most successful jeopardy contestants. These machines are capable of learning some complex problems much faster than humans can learn. In this paper, we propose using artificial neural network to train knowledge bots to identify the idiosyncrasies of simulation software and recognize patterns that can lead to successful simulations. We examine the use of knowledge bots for applications of computational fluid dynamics (CFD), trajectory analysis, commercial finite-element analysis software, and slosh propellant dynamics. We will show that machine learning algorithms can be used to learn the idiosyncrasies of computational simulations and identify regions of instability without including any additional information about their mathematical form or applied discretization approaches.

15. SUBJECT TERMS
CFD; Machine learning; Neural net

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	18	(443) 757-5802

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18