NASA/TM–2016-219004

Decision Manifold Approximation for Physics-Based Simulations

Jay Ming Wong
University of Massachusetts Amherst, Amherst, Massachusetts

Jamshid A. Samareh
Langley Research Center, Hampton, Virginia

January 2016

NASA/TM–2016-219004

Decision Manifold Approximation for Physics-Based Simulations

Jay Ming Wong
University of Massachusetts Amherst, Amherst, Massachusetts

Jamshid A. Samareh
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

January 2016

Acknowledgments

This work was made possible by the NASA Internships, Fellowships, Scholarships (NIFS) program at NASA Langley Research Center. We thank Sriram Rallabhandi for FUN3D related help; Adam Weber for his help in computing the normal vector after trajectory convergence; Jing Pei and Jacqueline Clow for their support, interest, and help in this project; and Thomas Perry, Sherri Deshong, Richard Garfield, Rashinda Davis and Donna Mizell for their assistance throughout this project. We acknowledge the support of the Vehicle Analysis Branch of Systems Analysis and Concepts Directorate, the Big Data Analytics Group, and the Entry Systems Modeling technology development project. Lastly, we would like to thank Manjula Ambur and Ed McLarney for their continuous support of this activity.

Available from:

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

Abstract

With the recent surge of success in big-data driven deep learning problems, many of these frameworks focus on the notion of architecture design and utilizing massive databases. However, in some scenarios massive sets of data may be difficult, and in some cases infeasible, to acquire. In this paper we discuss a trajectory-based framework that quickly learns the underlying decision manifold of binary simulation classifications while judiciously selecting exploratory target states to minimize the number of required simulations. Furthermore, we draw particular attention to the simulation prediction application idealized to the case where failures in simulations can be predicted and avoided, providing machine intelligence to novice analysts. We demonstrate this framework in various forms of simulations and discuss its efficacy.

Contents

List of Algorithms

List of Figures

1 Introduction

With the recent success of machine learning methods, especially with large deep networks[1], the field of computer vision has undergone large advancements where learned features outperformed rule-based descriptors that have been the state of the art for over a decade [3–7]. The method of building large deep neural networks with massive training sets has soon propagated into other fields and applications as well, resulting in unprecedented performance [8–10]. However, in many of these cases, the success of these learning frameworks is largely dependent on having big datasets. For instance, many of the Imagenet frameworks training processes use large interconnected structures that rely on millions of human-labeled training examples [11–18]. In the case of the image classification problem, labeled training examples are abundant and the development of these approaches aims to leverage the massive collections of examples. Again, in the domain of other problems aside from image classification, large sets of labeled examples may be abundant or easily obtained, making techniques such as these extremely viable for learning.

For the context of this paper, we point to a unique application: the learning of simulation results. Machine learning algorithms in the context of learning simulations discussed in various works have been shown to be both feasible and effective [1, 2, 19, 20]. Some of these approaches draw connections to image-related problems, building massive network architectures for learning when provided with large sets of training examples [1, 2]. The traditional big-data approaches require enormous training sets that in some cases may be infeasible to obtain, especially for complex simulations requiring immense computational power. The complexity—both in the underlying simulation and large input dimensionality—makes naive learning methods infeasible.

In this paper, we propose a reformulation of a deep learning framework that is suitable for complex simulations requiring long execution time. Rather than relying on a massive collection of training examples, we investigate the notion of intelligent selection of target parameters. The approach consists of a reinforcement learning approach that quickly drives states along some learned trajectory towards the surface of the decision manifold. Afterwards, a support vector machine with radial basis kernel expansion is used for linear separation in infinite dimensional space.

2 Knowledge Bot

2.1 Definition and Mechanics

The concept of the *knowledge bot* is first introduced as an autonomous classifier which learns by continuously sampling a specific domain and refining its underlying resulting model from a particular simulation [1, 2]. An analyst who wishes to perform a simulation provides input parameters to the simulation for execution. Rather than executing the simulation, the knowledge bot intercepts these parameters and provides fast predictions regarding the success of the simulation. The prediction was shown to be over 99% accurate in the previous works[1, 2]. In essence, the knowledge bot serves as an abstraction of the simulation software providing accurate feedback as to whether the actual execution will result in success or failure, avoiding simulating problems that are predicted to result in failure. The bot encapsulates the expertise of a trained analyst with many years of experience and is familiar with parameter settings that result in failure through learning the underlying simulation idiosyncrasies. There are two major motivations for developing the knowledge bot. First, a trained analyst has learned the ability to detect failing parameters for simulation tools. After years of experience, they are quite good at having the insight to know what input parameters will cause the simulation to either fail or crash. The knowledge bot aims to encapsulate the learned expertise of a trained analyst into a predictive tool that a novice analyst may be able to query whether or not simulation parameters are valid for simulation. Second, instead of having analyst simulation parameters that are predicted to fail (even trained analysts sometimes run into this issue), the knowledge bot may be able to predict the simulation failure quickly, saving computational resources and analysis time. Moreover, these two motivations are connected by a single theme of allowing for rapid development and analysis without wasting time and resources on failed simulations. This is particularly important for the design of aerospace vehicles.

[1]Preliminary knowledge in this paper assumes our previous work in building knowledge bots, or simulation classifiers trained via large network structures [1, 2].

2.2 Extensions and Reworks

In the earlier works[1, 2], the knowledge bot learned from the complete input to output mapping of simulation parameters. However in practice, we are only concerned with states close to the boundary of the separation between success and failure. Because the problem is of a binary separation nature, states can easily be classified according to its relative position with respect to the decision boundary, or in the case of muti-dimensions, the decision manifold.

In this paper, we introduce extensions to the previously introduced concept of *knowledge bots*[1, 2] while maintaining a sense of abstraction from the individual application. The three major concepts we discuss are as: 1) autonomy, 2) knowledge-driven selection of target states, and 3) manifold approximation.

2.2.1 Autonomy – The autonomous behavior of the knowledge bot is achieved by considering some policy $\mathcal{P}$ that relates the tendency for exploration and exploitation of the candidate targets in the state space. Furthermore, it is important to produce solutions at any time that are refined as more time is allocated to learning. In other words, the knowledge bot must be able to operate with minimal simulation results while producing sufficiently accurate predictions. In addition, the bot must autonomously discover new states under some policy $\mathcal{P}$ that can be used to refine its learned model.

2.2.2 Knowledge-Driven Selection of Target States – In previous works[1, 2], the knowledge bot relied on pseudo-random numbers to determine candidate states to evaluate; although this approach results in a learned model that eventually converges to a model that predicts the outcomes with unity accuracy, it requires infinite exhaustion through all possible configuration of values in all possible state discretization schemes.

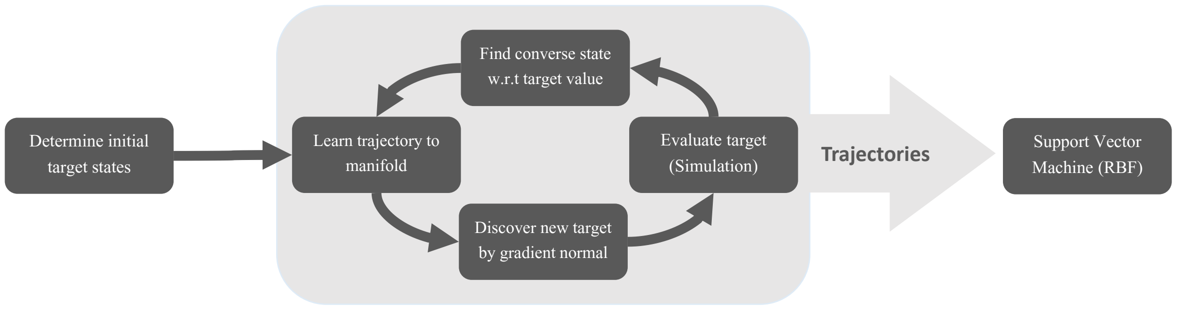
2.2.3 Manifold Approximation – Instead of attempting to learn the mapping of (possibly infinite) pairs of simulation input parameters to simulation termination results, in the proposed framework we consider only learning the deciding regions of the state space. In other words, this corresponds to the most interesting states along the decision manifold, which indicates the decision boundary between states are that extremely volatile to small perturbation to their configuration values. In the sense of the framework, these target states correspond to or have a high probability in being support vectors for the supervised support vector machine method to approximate the decision manifold.

3 Formulation

The authors have proposed a framework to capture analyst expertise in a *knowledge bot* by performing thousands of simulations as training examples for an artificial neural network[1, 2]. However, in some cases, acquiring thousands of simulation results may not be feasible for complex simulations. An intelligent method in determining simulation success or failure is necessary when these problems emerge. It is important to know that success and failure simulation results are highly structured. Simply speaking, generally small perturbations in the input parameters for a successful simulations will still tend to lead to success. This is not the case when the parameters approach the manifold of success and failure. In practice, learning the decision manifold that separates the two classes is sufficient.

This realization sparked our proposed framework that focuses on quickly identifying potential states (or configurations) close to the decision manifold using reinforcement learning to drive the state of the system sufficiently close to the manifold's surface forming a learned optimal trajectory that converges to the decision surface, and approximating the manifold through support vector machine with radial basis function (an infinite-dimensional kernel expansion).

The basic pipeline for the proposed framework is illustrated in Figure 1. The pipeline consists of an iterative step of trajectories of states $c_i \in \mathcal{C}$ or $d_j \in \mathcal{D}$ (Convergent and Divergent)that converge to the boundary of the manifold. The states close to the surface of the manifold can be thought of as support vectors that define the manifold itself. The states are extracted from the resulting trajectories and are used to train a support vector machine classifier with a radial basis function kernel approximating the true underlying decision manifold.



Determine initial target states
Learn trajectory to manifold
Find converse state w.r.t target value
Evaluate target (Simulation)
Discover new target by gradient normal
Trajectories
Support Vector Machine (RBF)

Figure 1. Proposed pipeline for our framework

3.1 Mathematics

3.1.1 Objective Function – The discovery of an unknown decision manifold of success and failure of some simulation can be thought of as an optimization problem where we define two sets $\mathcal{C}$ and $\mathcal{D}$, corresponding to the CONVERGENT and DIVERGENT simulation results given a particular set of input parameters $x_1, x_2, \cdots, x_n \in \mathcal{X}$. In the learning framework we can consider these input parameters as configuration variables $q_1, q_2, \cdots, q_n$ that define the state of the system with an output observation of $y \in \mathcal{Y}$. The output y determines whether the state of the system belongs to a convergent $\mathcal{C}$ or divergent $\mathcal{D}$ set. The decision manifold lies between the two sets. In other words, a particular configuration is near the decision manifold when it is a solution to the following objective function for a particular trajectory,

$$\underset{q_1,q_2,\cdots,q_n}{\arg\min} \; ||c_i - d_j||_2 \quad \text{for } i,j = 1,2,\cdots,k \tag{1}$$

Note that $c_i \in \mathcal{C}$ and $d_j \in \mathcal{D}$ consist of states along a particular trajectory. If we can drive the state to some solution of the above objective function following some trajectory, then that state is sufficiently near the decision manifold. The overall optimization problem can be framed as follows,

$$\begin{aligned} \underset{q_1,q_2,\cdots,q_n}{\text{minimize}} \quad & ||c_i - d_j||_2 \quad \text{for } i,j = 1,2,\cdots,k \\ \text{subject to} \quad & Q_1^- \leq q_1 \leq Q_1^+ \\ & Q_2^- \leq q_2 \leq Q_2^+ \\ & \quad\vdots \\ & Q_n^- \leq q_n \leq Q_n^+ \end{aligned} \tag{2}$$

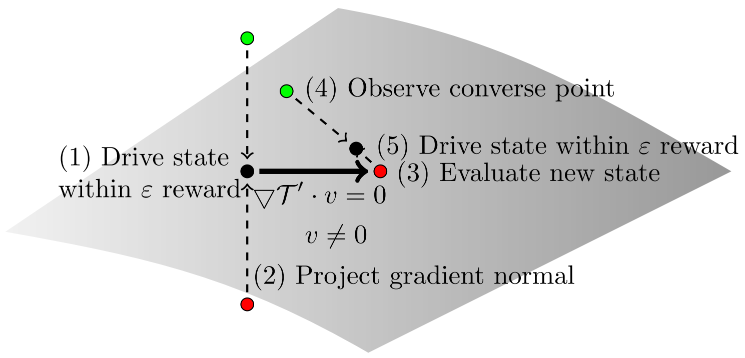
where Q is the set of configuration domains, and therefore Q_i^- and Q_i^+ correspond to the lower and upper bounds of the ith configuration variable. The set of configuration variables $q_1, q_2, \cdots, q_n$ defines some configuration or state.

3.1.2 Reward Function – A canonical reward function for any particular state or target configuration $t = (q_1, q_2, \cdots, q_n)$ can be written as,

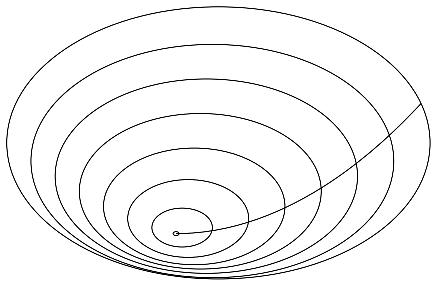
$$R_t(t) = \begin{cases} -\min ||t - d_j||_2 & \text{for } j = 1,2,\cdots,|\mathcal{D}| & \text{if } t \in \mathcal{C} \\ -\min ||t - c_i||_2 & \text{for } i = 1,2,\cdots,|\mathcal{C}| & \text{otherwise} \end{cases} \tag{3}$$

We modify the objective function such that the reward is monotonically increasing as target configurations are close to the decision manifold. The reward value for a particular target configuration is simply computed as the Euclidean distance between it and the closest observable converse state. We see that this reward function satisfies an underlying convex potential property where the reward value for states grow monotonically towards the optima which lies sufficiently close to the decision manifold.

3.2 Manifold Approximation


(4) Observe converse point
(5) Drive state within ε reward
(1) Drive state within ε reward
(3) Evaluate new state
$\bigtriangledown\mathcal{T}' \cdot v = 0$
$v \neq 0$
(2) Project gradient normal

(a) Policy for selecting candidate new targets



(b) Convex potential of objective function

Figure 2. Gradient-based policy converges manifold (left), guaranteed due to the convex property (right)

3.2.1 Trajectory Learning – The observation result y is fully deterministic given a set of configurations (due to the deterministic nature of simulations), and thus the reward function satisfies the convex property as shown in Figure 2, meaning that a greedy selection of actions will always lead to global extrema. This insight allows for an ε-greedy policy for selecting actions that drive the state of the system to extrema when convexity is satisfied.

Algorithm 1 describes the overall learning algorithm to drive the current state of the system toward the decision manifold. Since simulations are fully deterministic, the learning rate of $\alpha = 1$ is selected to guarantee optimality in the Q-value iteration step of the algorithm. An ε-greedy policy can be used for the action-selection policy, $\mathcal{P}$.

Algorithm 1 TrajectoryOntoManifold: Learns trajectory $\mathcal{T}$ that converges to the unknown manifold

```
1: procedure TOM($S, \varepsilon, \mathcal{I}, \mathcal{P}$, MaxIter)
2:     $\mathcal{T} \leftarrow \emptyset$
3:     for $i \leftarrow 1$ to MaxIter do
4:         if $\mathcal{I} \neq \emptyset$ then
5:             Initialize with and remove $(s, a)_i \in \mathcal{I}$
6:         else
7:             Choose action $a \in \mathcal{A}$ according to policy $\mathcal{P}$
8:         $s' \leftarrow (s, a)$
9:         $R(s, a) \leftarrow$ ComputeReward($s'$, Simulate($S$)$\big|_{s'}$)
10:        $Q(s, a) \leftarrow R(s, a) + \gamma \max Q(s, a)$
11:        $\mathcal{T} \leftarrow \mathcal{T} \cup s'$
12:        $s \leftarrow s'$
13:        if $R(s, a) \geq \varepsilon$ then
14:            break
15:    return $\mathcal{T}$
```

3.2.2 Gradient Normal Projection – When convergence is achieved, we determine new suitable candidate target states to learn new trajectories towards the unknown manifold by a gradient-based policy. To do this, we consider the previous trajectory and observe the gradient at the converged fixed point. This gradient indicates the direction ($\pm$) of trajectories from both $c_i \in \mathcal{C}$ to the manifold and $d_j \in \mathcal{D}$ to the manifold. Simply speaking, the gradient implies the two trajectories which when started initially at either a point in the convergent set $\mathcal{C}$ or divergent set $\mathcal{D}$ approach the surface of the manifold that separates the two sets. We can take some arbitrary normal to this gradient $\bigtriangledown\mathcal{T}'$ and project it outwards to approximate a state near the surface of the manifold. We then can evaluate this state by executing a simulation with the input parameters denoted by the states' configuration values. Lastly, we observe a converse point to the evaluated state and drive the state of the system again to some extrema between the two states used as new initialization parameters for a new trajectory. This process is illustrated in Figure 2.

3.2.3 Obtaining Gradient and Normal Vectors – An estimate of the gradient of the trajectory can be done by finite difference approximation using two sample targets along the trajectory that are sufficiently close.

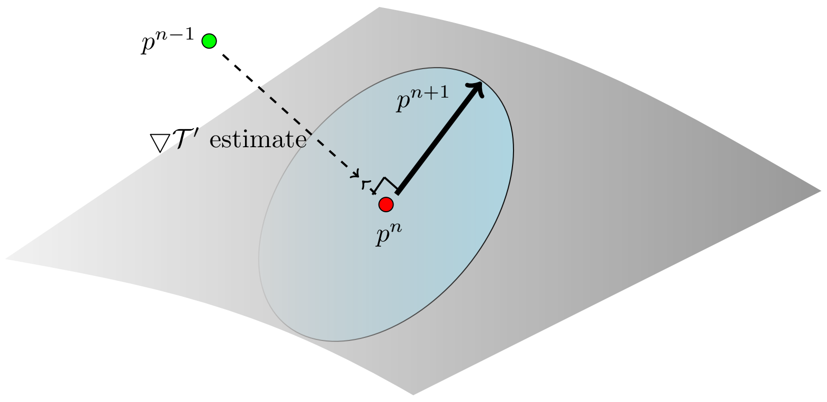


Figure 3. Infinitely many normal vectors are present for any given estimated trajectory gradient

As shown in Figure 3, there exists infinite many normal vectors for any given trajectory gradient estimate (shown as the dotted line), all of which lie on some hyperplane (as depicted in the image in light blue, where we consider only unit length vectors). Given some d-dimensional input parameter space, the space of the normal vectors resides in some $d-1$ space. In other words, the space to which normal vectors can be selected from grows on the order of dimensions. To solve for a particular normal vector we solve for the target p^{n+1} in the following,

$$\overset{\bigtriangledown\mathcal{T}'}{\begin{pmatrix} p_1^{n-1}-p_1^n \\ p_2^{n-1}-p_2^n \\ \vdots \\ p_{d-1}^{n-1}-p_{d-1}^n \\ p_d^{n-1}-p_d^n \end{pmatrix}} \cdot \overset{v}{\begin{pmatrix} p_1^{n+1}-p_1^n \\ p_2^{n+1}-p_2^n \\ \vdots \\ p_{d-1}^{n+1}-p_{d-1}^n \\ p_d^{n+1}-p_d^n \end{pmatrix}} = 0 \tag{4}$$

It is shown in Equation (4) that the next target parameter values that resides in the unit normal hyperplane can be found by solving a simple dot product between the estimated trajectory gradient near the manifold surface and the vector formed by the closest manifold state along the trajectory and the next target state. To solve for p^{n+1}, we simply assign $[p_1^{n+1}, p_2^{n+1}, \cdots, p_{d-1}^{n+1}]$ arbitrarily and solve for p_d^{n+1} resulting in a particular normal vector p^{n+1}. There is a subtlety however. We must enforce that p^{n+1} to be within the domain such that the following is satisfied,

$$Q_i^- \leq q_i \leq Q_i^+ \quad \text{for each } i = 1, 2, \cdots, n \tag{5}$$

Algorithm 2 TargetFromNormals: Returns target t' from unit normal space with maximal separation

```
1: procedure TFN($\mathcal{T}'$, $\mathcal{C}$, $\mathcal{D}$)
2:     $\mathcal{M} \leftarrow$ Build KD-tree using all states $s \in \{\mathcal{C}, \mathcal{D}\}$
3:     Extract $p^n$ and $p^{n-1}$ from trajectory $\mathcal{T}'$
4:     Compute estimated gradient $\bigtriangledown\mathcal{T}'$ from $p^n$ and $p^{n-1}$
5:     for $i \leftarrow 0$ to $N$ do
6:         Randomly assign $[p_1^{n+1}, p_2^{n+1}, \cdots, p_{d-1}^{n+1}]$
7:         Solve for $p_d^{n+1}$ resulting in complete $p^{n+1}$ that satisfies (4)
8:         Query $\mathcal{M}$ for minimal distance neighbor and store distance as $q_i$ and $\beta p^{n+1}$ as $t_i$
9:     return $t'$ as target state $t_i^*$ where $i$ corresponds to the index of $\max(q_i)$
```

The overall procedure is shown in Algorithm 2. The β term is a gain or a magnitude which is multiplied to the unit vector p^{n+1}. The parameter β influences the resolution of target states along the manifold. A high β results in spare coverage of states near the manifold, whereas a small β results in very dense clusters of target states near the manifold and as a results convergence is slower.

3.2.4 From Trajectories to Manifold – Given a set of trajectories $\mathcal{T}$, the configuration of each state is extracted and used as training data for a support vector machine classifier using radial basis kernel function, which projects the n-dimensional states to an infinite dimensional space for linear separation. The separating hyperplane is then used for classification. However, we can project this back into the n-dimensional configuration space and approximate the underlying decision manifold with the hyperplane. This procedure is further described in Algorithm 3.

Algorithm 3 ApproximateManifold: Learns a hyperplane approximation $\mathcal{H}$ that decides simulation S

```
1: procedure AM($S$, $\varepsilon$, $\mathcal{I}$, $\mathcal{P}$, MaxIter)
2:     Determine $\mathcal{I}$ from domain constraints
3:     $\mathcal{T} \leftarrow \emptyset$
4:     for each maximally separated tuple $(t_1, t_2) \in \mathcal{I}$ do
5:         do
6:             $\mathcal{T}' \leftarrow$ TrajectoryOntoManifold($S$, $\varepsilon$, $(t_1, t_2)$, $\mathcal{P}$, MaxIter)
7:             $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}'$
8:             $t_1 \leftarrow$ TargetFromNormals($\mathcal{T}'$, $\mathcal{C}$, $\mathcal{D}$)
9:             $s' \leftarrow$ Evaluate state $t_1$ via simulation
10:            $t_2 \leftarrow$ Select some state $t_2$ in $\mathcal{T}$ such that $t_2 \in \{\mathcal{C}, \mathcal{D}\}$ and $t_2$ not in the same set as $s'$
11:        while $\bigtriangledown\mathcal{T}'$ is valid and $(t_1, t_2)$ is unique
12:    Parse $\mathcal{T}$ s.t. $(x_i, x_j, \cdots, x_k) \in \mathcal{C}, (x_{i'}, x_{j'}, \cdots, x_{k'}) \in \mathcal{D}$
13:    $X \leftarrow (x_i, x_j, \cdots, x_k) \in \mathcal{C} \cup (x_{i'}, x_{j'}, \cdots, x_{k'}) \in \mathcal{D}$
14:    $Y \leftarrow$ Associate simulation result $y_i \in \{$Convergent, Divergent$\}$ to each $x \in \mathcal{X}$
15:    $\mathcal{H} \leftarrow$ SVM($X$, $Y$)
```

4 Demonstration

4.1 Isotropic Decision Boundary

4.1.1 Formulation – As a proof of concept, consider an isotropic decision boundary defining a simple classification scheme as follows,

$$F(x_1, x_2) = \begin{cases} \textsc{Success} & \text{if } x_1^2 + x_2^2 \leq 1 \\ \textsc{Failure} & \text{otherwise} \end{cases} \tag{6}$$

This defines a unit circle decision boundary about the origin. In practice, the isotropic decision boundary introduces challenges to gradient-based policies for selecting new candidate targets since at least twice the

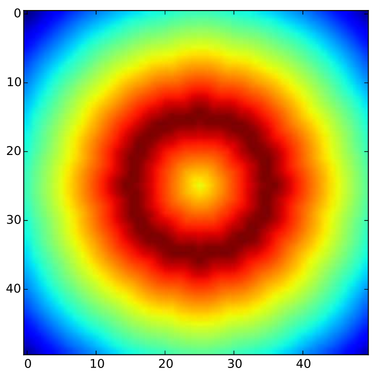


Figure 4. Reward function for isotropic boundary problem exhibits the convex property

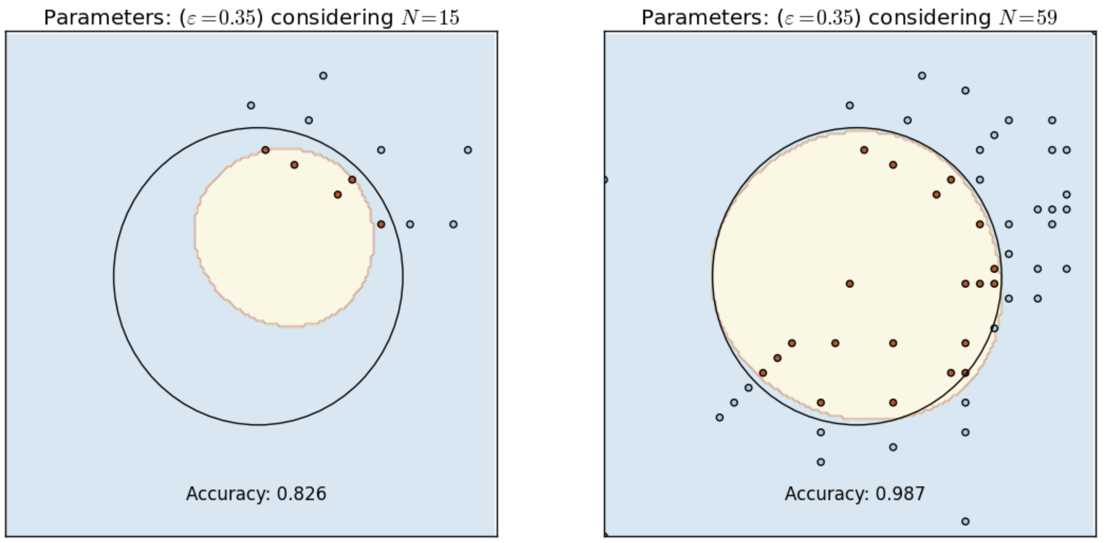


Figure 5. Support vector machine classification visualizations of isotropic decision manifold

gradient either diminishes or escapes to infinity when exploiting greedy gradient-based actions. However, the reward function itself, as outlined in (3), satisfies the convex property as illustrated in Figure 4, which indicates the fully observed Euclidean norm between states $c_i \in \mathcal{C}$ and $d_j \in \mathcal{D}$. However, due to states initially being unexplored, the resulting reward will not be identical to the idealized perfectly observed case. It is shown in Figure 4 that the reward function is a convex function where simply following the gradient of maximum reward will drive any state of the system to the global maxima for reward. In this context, it corresponds to the regions sufficiently close to the deciding boundary of the classification problem. The reward contour illustrated is a measure using the Euclidean norm with absolute knowledge about the entirety of elements in the convergent and divergent sets, $\mathcal{C}, \mathcal{D}$. The actual reward in practice may be much lower, although the greedy convex property still remains. The reward is an estimated value which computes the minimum Euclidean norm between all observed states in the two sets. In other words, the contour depicts the upper-bounds for the reward at any given state.

4.1.2 Learned Manifolds – Using the proposed framework, the learned manifold of the isotropic decision boundary problem is shown in Figure 5, which illustrates the result from using support vector machine classification with radial basis kernel function on states along all trajectories discovered by our proposed algorithms. The small red and blue nodes in the figures illustrate a single state along some trajectory towards the manifold. The true decision boundary is depicted in a solid black line, while the learned manifold approximation is demonstrated by the two distinct background colors that corresponds to the two

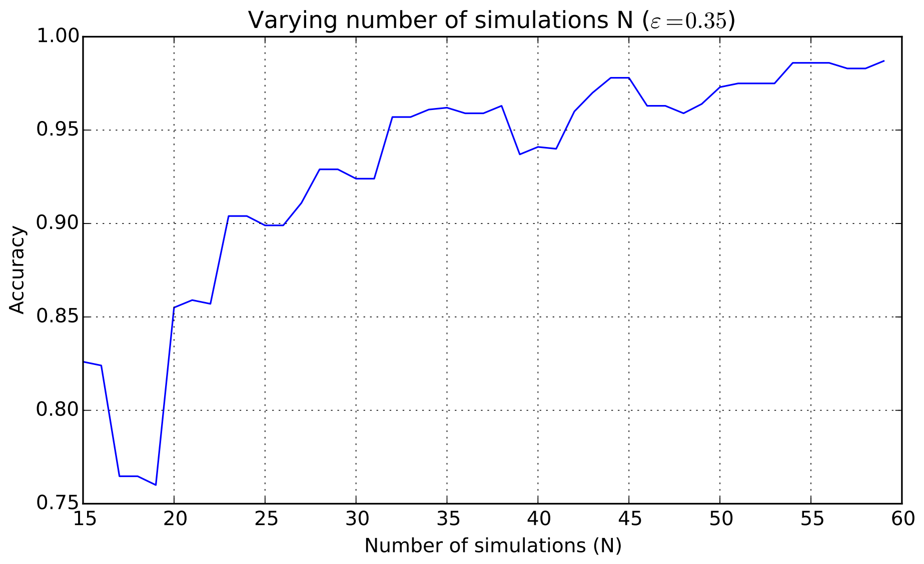


Figure 6. Accuracy vs. number of isotropic simulations

distinct classes $\mathcal{C}$ and $\mathcal{D}$.

4.1.3 Parameter Influences – We see that as the number of simulations, N, increases, the learned manifold approximation converges to the true decision boundary of the underlying simulation. When the number of simulation executions is low, for example $N = 15$ (Figure 5 left), the learned decision manifold holds bias towards the local region of explored trajectories and states, achieving an accuracy of 0.826. However, as the number of executions increases, the decision manifold converges to the true underlying decision boundary. This occurs somewhere near $N = 59$ (Figure 5 right) which obtains an accuracy of 0.987. The accuracy as the number of simulations is depicted in Figure 6.
In addition, the convergence threshold ε indirectly controls the number of simulation executions. A high ε results in fewer simulations, resulting in a coarse manifold approximation as trajectories quickly converge to a course region of the manifold. A low ε constrains the trajectories to terminate when states are close to the manifold, resulting in executing a larger number of simulations and obtaining a finer resolution manifold approximation. Note that ε can be made sufficiently small as long as it is within the magnitude of the discretized state space.

4.2 Disjoint Isotropic Boundaries

4.2.1 Formulation – we use the support vector machine with the radial basis kernel that has infinite dimensional kernel expansion property. We demonstrate this on an extended version of the simple isotropic decision boundary. This problem is again non-trivial in areas where gradients do not exist. We define the following classification scheme, generating two disjoint isotropic regions,

$$F(x_1, x_2) = \begin{cases} \textsc{Success} & \text{if } (x_1 + \alpha_{1,1})^2 + (x_2 + \alpha_{2,1})^2 \leq r^2 \text{ or } (x_1 - \alpha_{1,2})^2 + (x_2 - \alpha_{2,2})^2 \leq r^2 \\ \textsc{Failure} & \text{otherwise} \end{cases} \tag{7}$$

This defines two disjointed regions, and success is defined inside the two circles. The purpose of this demonstration is to show that the framework can adapt to learning multiple disconnected regions that are defined in the input parameter space. Note that when projected to a higher (or possibly infinite) dimensional space through the kernel basis expansion methods, these multiple regions may not exist and the targets may be linearly separable.

4.2.2 Learned Manifolds – Figure 7 illustrates the visualizations of the support vector machine classifications along with the true decision boundaries outlined. The state color differentiation is identical to the previous figures where blue points correspond to states in the convergent set (i.e., $s \in \mathcal{C}$) and red points correspond to states that are in the divergent set (i.e., $s \in \mathcal{D}$). We see that within $N = 20$ simulations, the learned decision manifold begins to discover two disjoint boundaries in the two-dimensional input space; meanwhile the manifold is approximately 82.3% accurate with respect to the true separation boundaries. The decision manifold converges to 98.4% accurate when the number of simulations increases to about $N = 150$, when using a convergence epsilon of $\varepsilon = 0.20$ for the trajectory learning step.

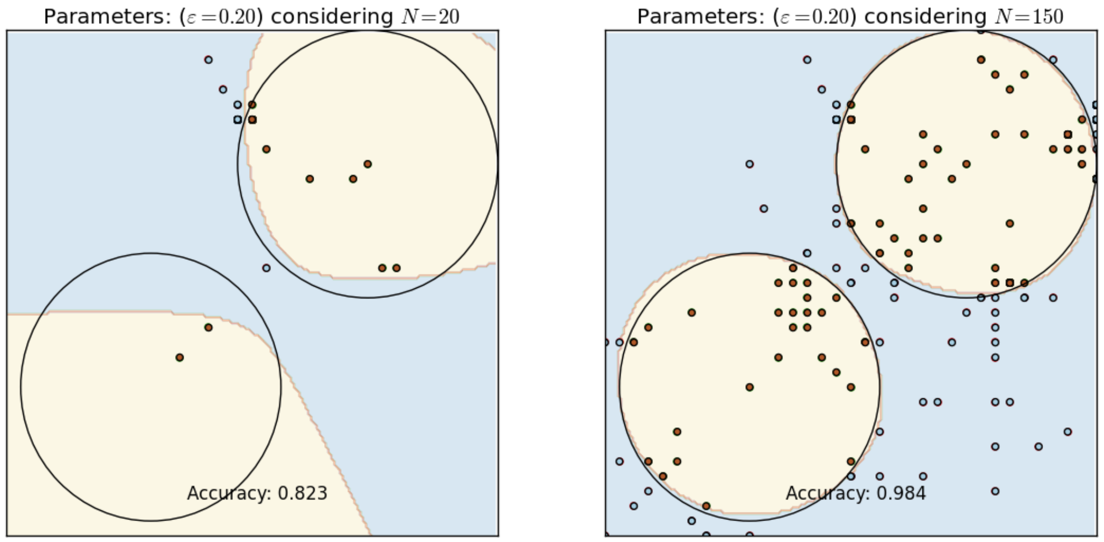


Figure 7. Support vector machine classification visualizations of disjointed isotropic decision manifold

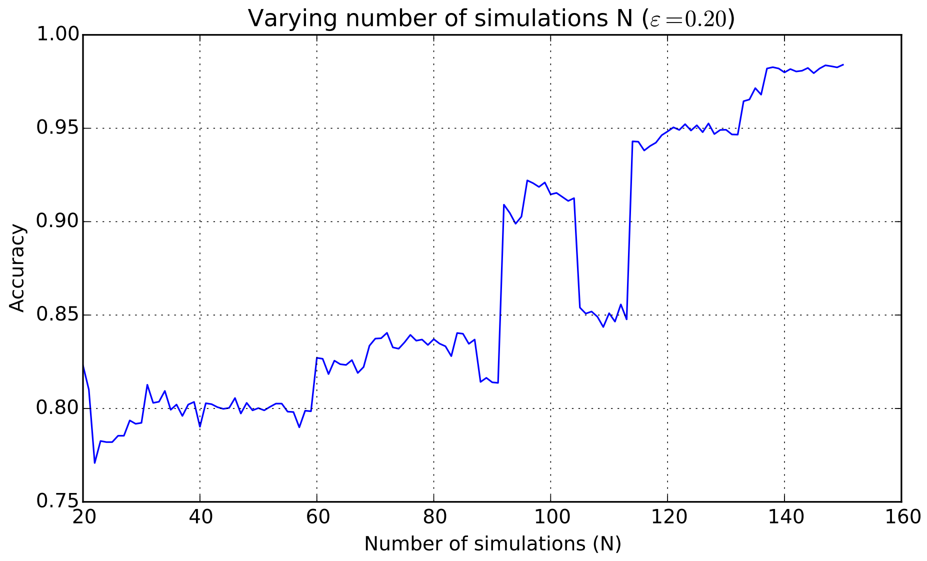


Figure 8. Accuracy vs. number of disjointed isotropic simulations

4.2.3 Parameter Influences Figure 8 illustrates the accuracy of the learned manifold as the number of simulation results are increased. It is interesting to note that there are several large jumps in accuracy which can be attributed to possible instances where new regions of the manifold are detected that force the support vector machine to encapsulate these new regions in a higher dimensional space resulting in a better fitted decision manifold. Furthermore the large accuracy decline at around $N = 110$ may be due to a point at which there exists a large concentration of a particular simulation result, either convergent or divergent, within a small radius that may pull or push the decision boundary from where it was before the addition of these new states. The decision manifold exceeds 98.0% accuracy within approximately $N = 135$ simulation results.

4.3 Numerical Methods: Hyperbolic Wave Equation

4.3.1 Preliminary Background – One of the numerical methods discussed in previous work [1, 2] is the hyperbolic wave equation that governs the simulation of vibrations by a string or rod. The equation is a differential partial equation and can be solved numerically using forward time central space (FTCS),

$$u_i^{n+1} = \rho u_{i+1}^n + 2(1-\rho)u_i^n + \rho u_{i-1}^n - u_i^{n-1} \tag{8}$$

where $\rho = (\Delta t/\Delta x)^2$ is the squared discretization ratio. Accuracy was tested against the following analytic solution:

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

where $f(\cdot)$ is the initial deflection given by $f(x) = u(x, 0)$. We see that a simple sinusoid satisfies this with $f(x) = \sin(\pi x)$. The analytic solution was used to determine the exact solution in [1, 2]. Results from using a simple neural network architecture achieved over 96.2% accuracy with 10000 training instances.

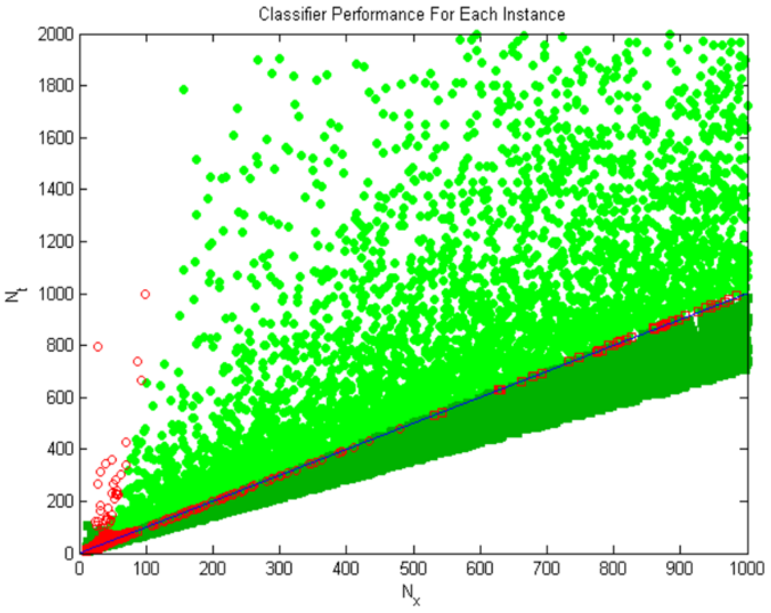


Figure 9. Classification results of the hyperbolic wave equation solver in previous work[1, 2]

The full legend of Figure 9 can be seen in [1, 2], though the following will list the most interesting aspects of the plot: The circles and squares in the plot correspond to simulations that succeeded and failed, respectively. The green represents correctly classified simulations. Given a set of inputs for the four major parameters, the learned model was successfully able to predict the outcome of the simulation. Red corresponds to failure in prediction. Therefore, a dark green square means that the simulation is a failure and the knowledge bot was able to successfully predict the failure. Likewise, a green circle means it was able to successfully predict success. Conversely, red squares are then instances that failed but were incorrectly predicted as successful simulations and red circles are the instances that were successful but were incorrectly predicted as failing parameters. In this numerical methods section, we recreate the experiment presented but instead incorporate our proposed framework and discuss the efficacy of this framework.

4.3.2 Learned Approximations – Our methods resulted in the following learned manifolds for the hyperbolic wave equation numerical simulation,

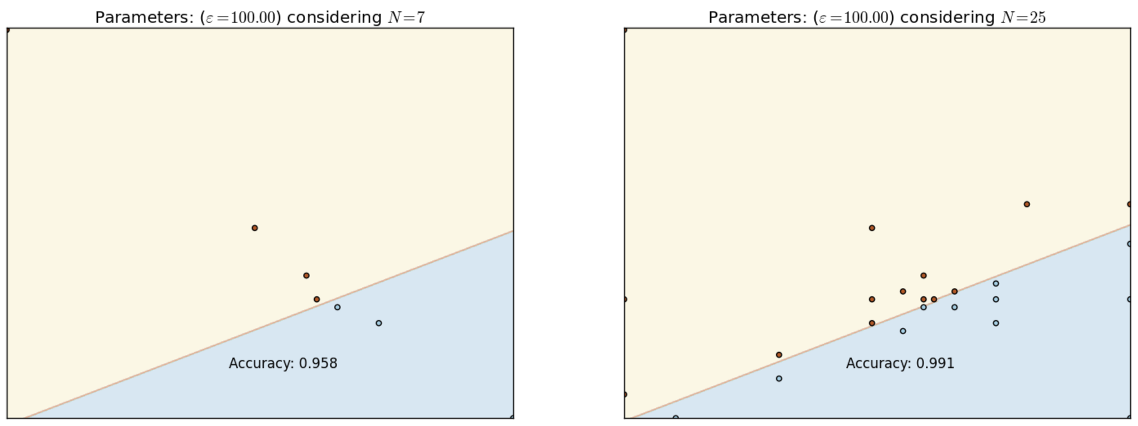


Figure 10. Support vector machine classification visualizations of the decision manifold for hyperbolic wave equation

Using the proposed method, we show that with only about seven simulation results (which corresponds to only a single trajectory), we can approximate the true decision manifold with 95.8% accuracy. This finding is significant since the previous results[1, 2] were only able to achieve 96.2% accuracy using 10000 training instances. We demonstrate these results in Figure 10. Note that using the methods proposed in this paper, we were able to actually far exceed the accuracy discussed in some of the previous works [1, 2]. The linear kernel easily encapsulated the linear separation in the input-space and resulted in over 99% accuracy within $N = 25$ simulation trials. Furthermore, it is important to note that many of numerical problems are well defined in the sense that the decision boundary can be described in a moderately low dimensional space. In other words, numerical problems tend to not be overly complex in the decision surface allowing for high accuracy and low simulation counts with using the proposed approach. The isotropic boundary, as shown previously serves as a much higher order problem and as a result is described by a more complicated manifold; although in principle the boundary itself seems simple, learning the manifold is nontrivial.

4.3.3 Discussion – We demonstrated the manifold approximation framework on the hyperbolic wave numerical simulation from previous works[1, 2]. Interestingly, the algorithm proposed here exceeded the accuracy of the trained knowledge bot trained in previous work by more effective selections of target states for simulation. Furthermore, the problem is inherently easy since the input parameter space is two dimensional and also linearly separable in the input parameter space itself, though this does serve as a confirmation and verification that the manifold approximation framework quickly and greedily learns the manifold with minimal simulation results through driving state trajectories near the manifold to find likely support vectors for separation.

5 Simulations

Previously we have shown that the proposed manifold approximation method is effective through several demonstrations and through a case of a numerical simulation of the hyperbolic wave equation. We now move on to complex simulations: POST2 and FUN3D.

5.1 Planetary Direct Entry Simulations with POST2

5.1.1 Preliminary Background – Consider the capture problem of atmospheric-assisted direct entry of an aerospace vehicle into the planet Venus. The simulation itself is governed by the complex dynamics of systems of nonlinear differential equations implemented in POST2 (Program to Optimize Simulated Trajectories II), which is based on a point mass, discrete parameter targeting and optimization program generally used for trajectory and targeting simulations[21]. A simplified version of the problem is presented in previous works that incorporates four important parameters: the diameter, mass, entry velocity, and entry flight path angle of the capsule [1, 2].

5.1.2 Problem Formulation – The POST2 simulation problem can be formulated as a learning problem where the mapping of the four dimensional input space to corresponding values of convergent or divergent simulation results is learned. In other words, given some four dimensional input values corresponding to each of the important input parameters, a model learns to predict the convergence of the simulation result. We demonstrate using the greedy manifold approximation method to quickly identify target states near the decision manifold that decides the boundary between convergence and divergence.

5.1.3 Learned Classification Boundaries – The learned classification boundaries of a particular model takes the form of some decision manifold approximated in previous works by nonlinear regression in the form,

$$\begin{aligned} J(x_1, x_2, \cdots, x_{n-1}) &= \alpha x_1^{\beta_1} x_2^{\beta_2} \cdots x_{n-1}^{\beta_{n-1}} \\ J(d, m, v_e) &= 0.0219 m^{0.0096} v_e^{0.6646} d^{-0.0017} \end{aligned} \tag{10}$$

As shown previously, this nonlinear regression was computed through use of the support vectors post-classification. The classification boundaries with respect to the regressed boundary described by Equation (10) is depicted as follows,

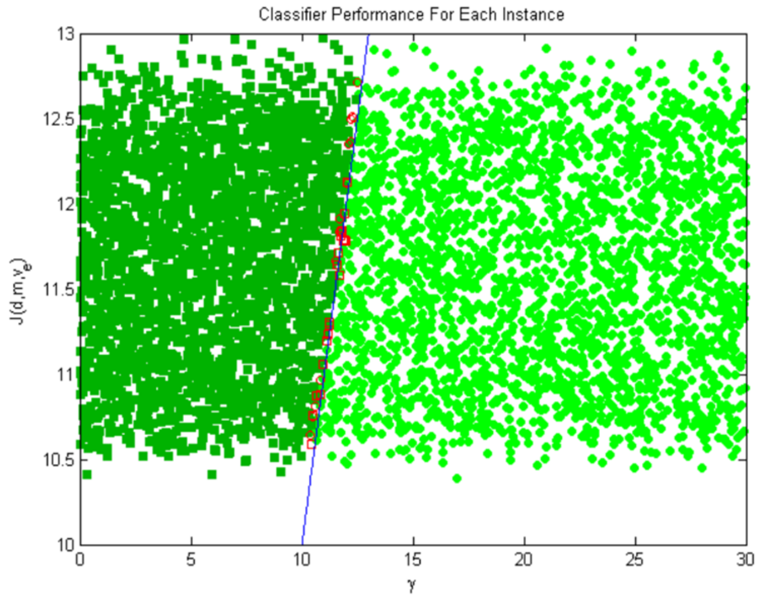


Figure 11. Classification results in previous works for POST2 Venus direct entry

The classification plot for the POST2 Venus direct entry problem is shown in Figure 11, where γ is the flight path angle in degrees. The decision boundary as described by Equation (10) is depicted as the solid blue line that separates the two shades of green points. The legend for the plot is identical to the previously shown classification diagram in Section **??**.

5.1.4 Statistical Analysis of Results – We demonstrate our proposed framework with varying numbers of simulation results on the planetary direct entry simulation problem using POST2 through a large scale experiment where we learn an individual model at every simulation maximum count and observe its classification accuracy. Each point in Figure 11 requires several minutes of simulation and iteration time, and each point in the following plot is an individual model that looks similar or has similar classification characteristics (although with varying accuracy) as the one particular model shown in Figure 11.

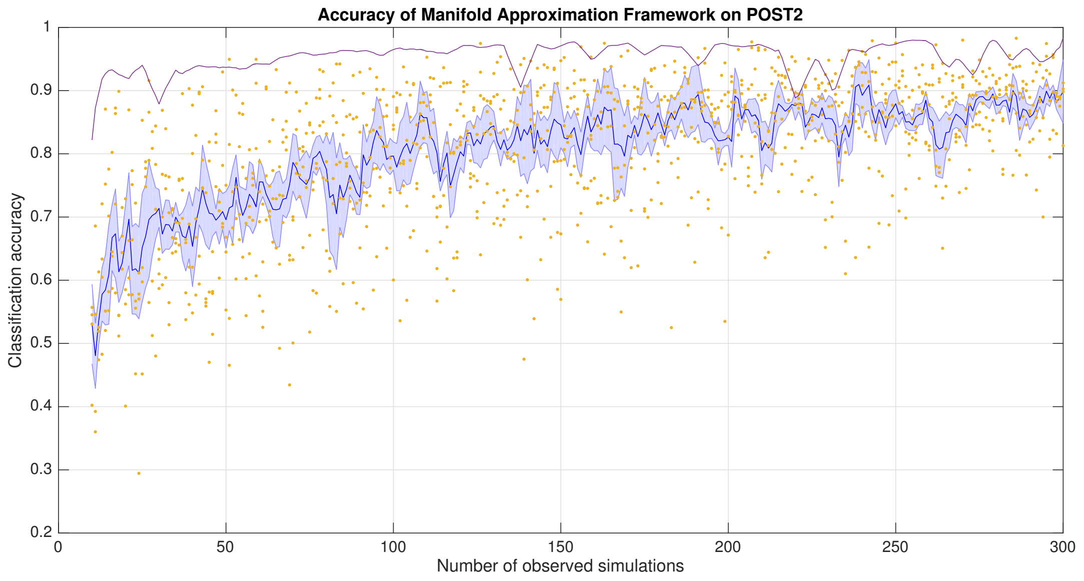


Figure 12. Accuracy of manifold approximation framework on POST2 simulations

Figure 12 shows the mean and one standard deviation (smoothed and scaled for clarity) for classification accuracy using a particular number of observed simulations of POST2. The fine dots in the image illustrates each trial of simulation that corresponds to a single execution of the proposed framework with a particular maximum number of simulations allowed. The purple solid line illustrates an approximate upper-bound for classification accuracy in this experiment. The upper-bound is shown because the simulations were not performed online, enabling a larger numbers of simulations and analytics on the proposed framework. However, this also imposes an upper-bound on the accuracy of the resulting model. A large set of 20000 POST2 simulation results for randomly sampled parameter values was completed and used for selecting target configuration parameters when using our framework (this set is actually the exact training and testing sets used in the previous works [1, 2]). When evaluating a particular target, we select the value of the minimum norm instance in the offline dataset. This generally works well and allows for speedy experiments (esp. the experiment from Figure 12 where there were over 1500 experiment trials with varying number of maximum observed simulations). The accuracy is only approximately as good as the accuracy of the underlying offline set we generate which is shown in Figure 12.

5.1.5 Discussion – We cannot directly compare these results to those presented in [1, 2] because the target selection methods in this paper are highly stochastic and have varying effects on the resulting accuracy of final model. Many of the results using a low number of observed simulations have high variance in classification accuracy, however, in many cases actually exceed the accuracy in previous work[1, 2], though the mean accuracy is slightly less due to the fact that 1) the accuracy is upper-bounded by the accuracy of the underlying offline dataset (it may be possible to exceed the accuracy by running simulations in real time, which results in slower experiments), 2) the methods here are highly greedy where after an arbitrary direction for the normal is selected it continues in that direction selecting new targets and trajectories (possibly random resets will aid in effectively covering the space), and 3) the parameter-space for this

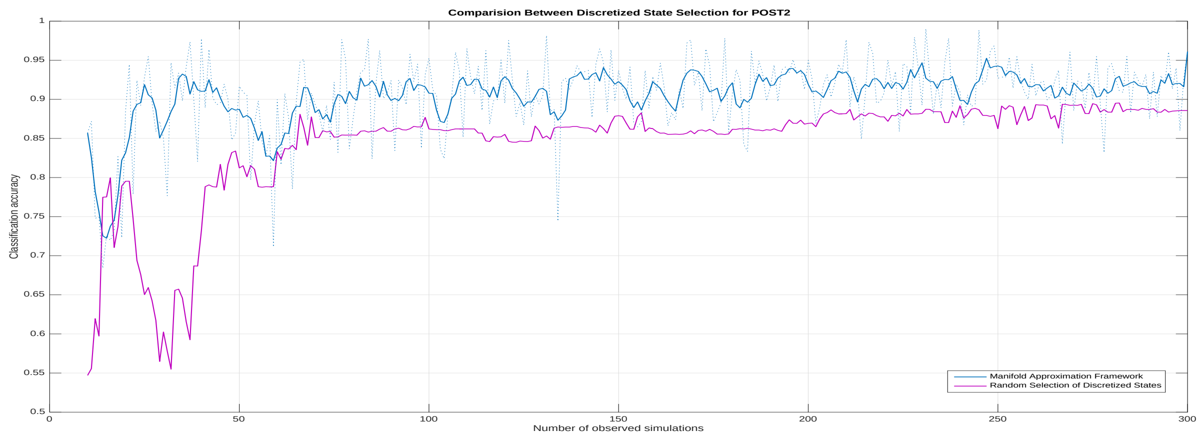


Figure 13. Comparison between discretized state selection for POST2 simulations

problem is multi-dimensional, resulting in not a single unique normal but actually infinite possible normals to select from. Compared to the results in the previous work where ten simulations resulted in an accuracy of 78.6% and twenty exceeded 90.0%, the results here showed that a number of experiments outperformed these baselines. Furthermore, the mean and standard deviations demonstrated in Figure 12 have been smoothed and scaled for clarity, hence, in reality in many cases, many of the means are either significantly exceeding the baselines or if not, slightly under by less than 2-3%.

5.1.6 Influence of Discretized State Space – To accurately compare this framework with probabilistic space sampling methods (Monte Carlo or Latin Hypercube) used in previous works, we must consider state space discretization where accuracy is limited by the granularity of the state discretization (in the case of this demonstration, the domain of each input variable was discretized using $N = 100$). For comparison, we demonstrate both methods here where input parameters are no longer continuous but discretized such that random selection uses Latin hypercube sampling to select randomly a particular state cell.
As shown in Figure 13 that the manifold approximation framework demonstrates a marginal increase in classification accuracy compared to the random selection of target states in the case of a discretized state space. Note that the experiment was shown in using a discretization parameter $N = 100$, where each dimension is discretized by N. The granularity of the discretization has a large influence on the classification accuracy as discretization of the state space results in an upper bound for accuracy as shown in Figure 12. Coarse discretization schemes tend to result in a lower upper bound and as a result lower classification accuracy while finer discretization schemes will generally achieve higher classification accuracy.

5.2 Inviscid Flow of OM6 Wing Using FUN3D

5.2.1 Preliminary Background – The next piece of simulation software used to demonstrate our algorithms on is FUN3D (Fully Unstructured Navier-Stokes [22]) which is a simulation software for unstructured-grid computational fluid dynamic simulations spanning incompressible flow to compressible flow used to study airframe noise, space transportation vehicles, flow control devices using synthetic jets, and the design of wind tunnel and flight experiments (amongst other applications).[2] We performed the learning framework using the flow solver for an ONERA M6 (OM6) wing with a symmetry boundary condition imposed at the wing root using a coarse grid for demonstration. The purpose of the knowledge bot in this case is to detect simulation failures in the OM6 wing inviscid flow solver by observing the convergence of the residual (R1).

5.2.2 Dataset Properties – We run OM6 wing flow solver simulations using a maximum of 5000 iterations while having a convergence epsilon of 10^{-15}. The parameters of interest in this problem are angle of attack α,

[2]NASA Langley's Fun3D code `http://fun3d.larc.nasa.gov`

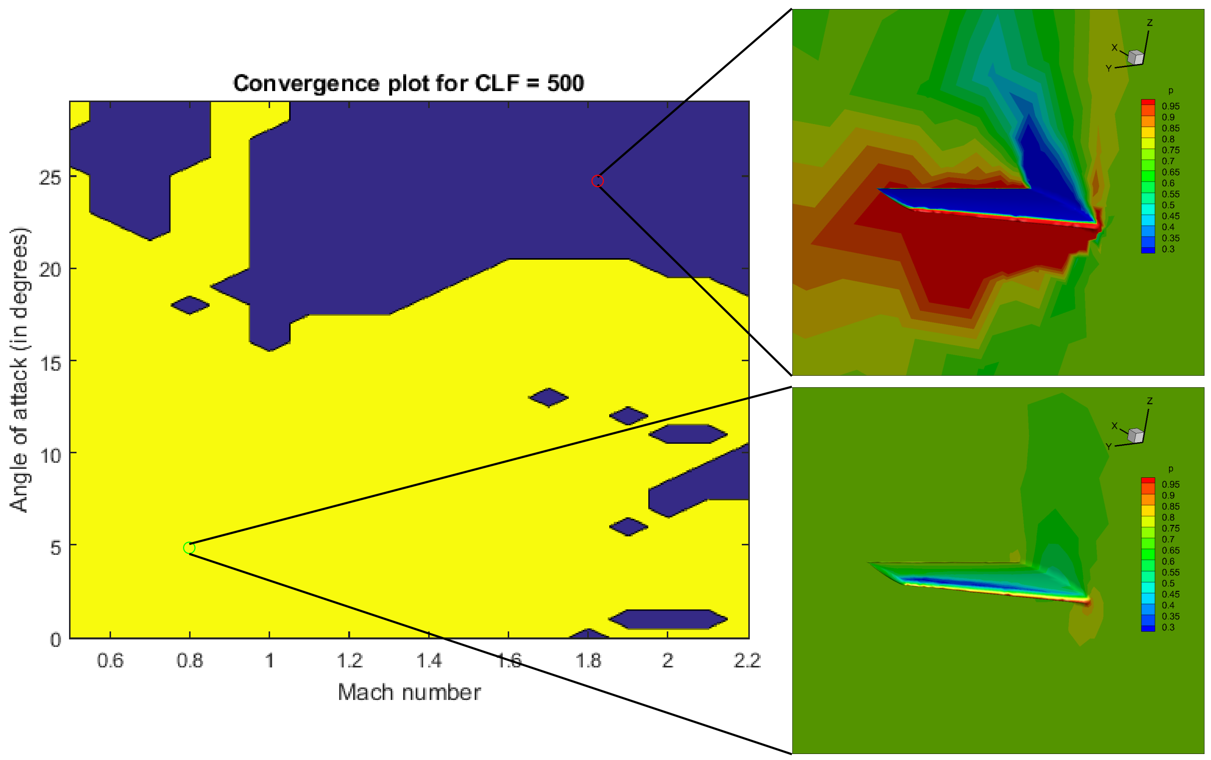


Figure 14. Decision boundary demonstration for inviscid flow on ONERA M6 (OM6) wing with FUN3D

Mach number, and the Courant-Friedrichs-Lewy (CFL) number, C_{max}. For our preliminary demonstrations, we fix the CFL number $C_{max} = 500$.

5.2.3 Decision Boundary and Physical Interpretation – The following illustrates a coarsely sampled convergence plot that indicates the parameter values of Mach number and angle of attack that resulted in convergence within epsilon and 5000 iterations.
The decision boundaries of the OM6 inviscid flow problem is shown in Figure 14. The yellow region in the image depicts the parameter settings that resulted in convergence, where convergence is defined by the residual (R1) of the FUN3D simulation. The dark blue region corresponds to the instances that do not converge within 5000 iterations. Most of these occur at high angles of attack as (shown in top right), where the coarse grid in this simulation is not able to appropriately capture the physical phenomenon resulting in failure of convergence. Shown in the top right and bottom right are two simulations of flow that 1) did not converge and 2) converged within some epsilon of the residual R1 respectively. Illustrated as contour lines are the (normalized) nondimensionalized coefficient of pressure. The decision boundary shown in Figure 14 is what we aim to learn using the framework proposed in this paper.

5.2.4 Accuracy of Framework – We demonstrate our framework on the OM6 wing using the flow solver, FUN3D. Approximately 3000 models were learned and evaluated against a coarse contour grid as shown in Figure 14. All parameters for learning these models were standardized and left identical to the POST2 examples from Section 5.1. Figure 15 illustrates the accuracy of each of these learned models trained with a particular number of training states. Each individual learned model (one of the 3000 points illustrated, top) and its associated accuracy with the coarse discretized grid is shown in Figure 15. As seen in the top image, each point in the plot corresponds to a single model and the classification accuracy it achieved. The bottom image illustrates the mean and one standard deviation for the set of learned models. In both images the upward trend for classification accuracy is present when the number of observed simulations is increased. This is simply due to the fact that the support vector machine model encompasses more and more states or training instances, however, we see that in both images the classification accuracy plateaus at around

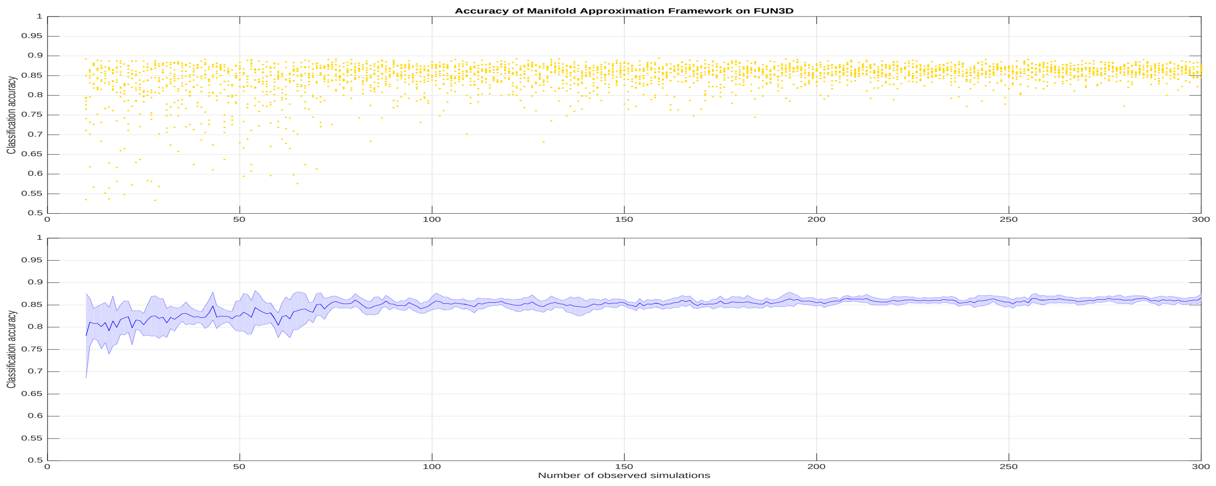


Figure 15. Accuracy with coarsely discretized offline set for demonstration of FUN3D OM6 wing flow

90% accuracy. This is a result of the underlying offline dataset as the plot illustrates preliminary findings. Furthermore, this is an artifact of the discretization done for each dimension, thus unable to capture some underlying properties of the decision boundaries. We do see that around $N = 100$ observed simulations, the knowledge bot achieves accuracies sufficiently close to 85%.

5.2.5 Larger Offline Simulation Set – We increased the offline simulation set to a set of 10000 instances, which in the two dimensional space is very fine, thus can simulate actually running the simulations (though for accuracy running the simulations themselves will result in a more accurate model, but for the purpose of demonstration, we need a validation set to confirm that predictions made by the model are indeed true and also for debugging purposes, using an offline set is simpler).

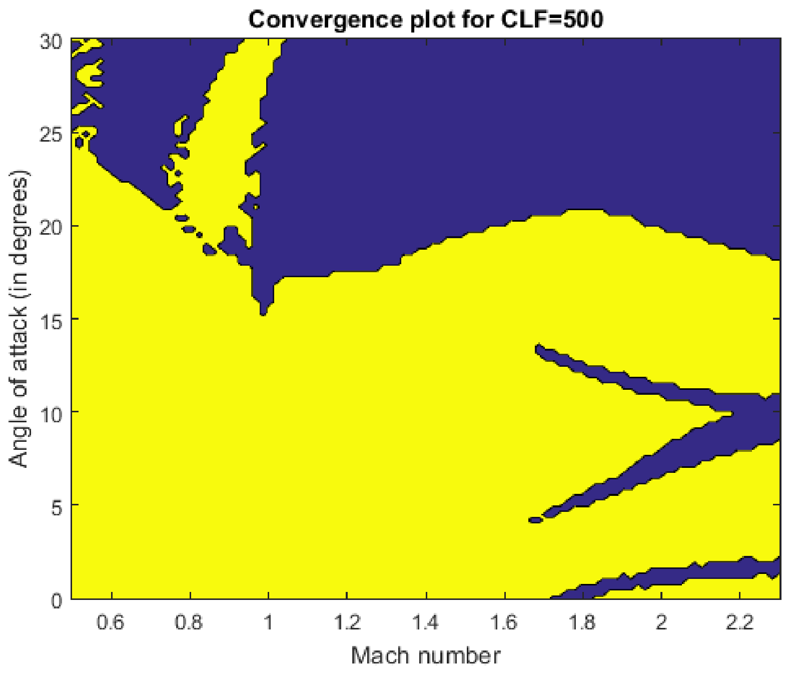


Figure 16. FUN3D decision boundaries with finer grid and more simulations

With more simulations and a finer grid sampling frequency, we see that the decision boundary (as shown in Figure 16 begins to exhibit a smoother shape and some discontinuities that were present before in the coarse sampling grid (from Figure 14). With the finer offline set, we perform more analysis on the classification accuracy of learned models with our framework. Similar to the discretized state selection shown in Figure 13 for the POST2 simulations and results, we show the classification accuracy of discretized state selection of the FUN3D simulations,

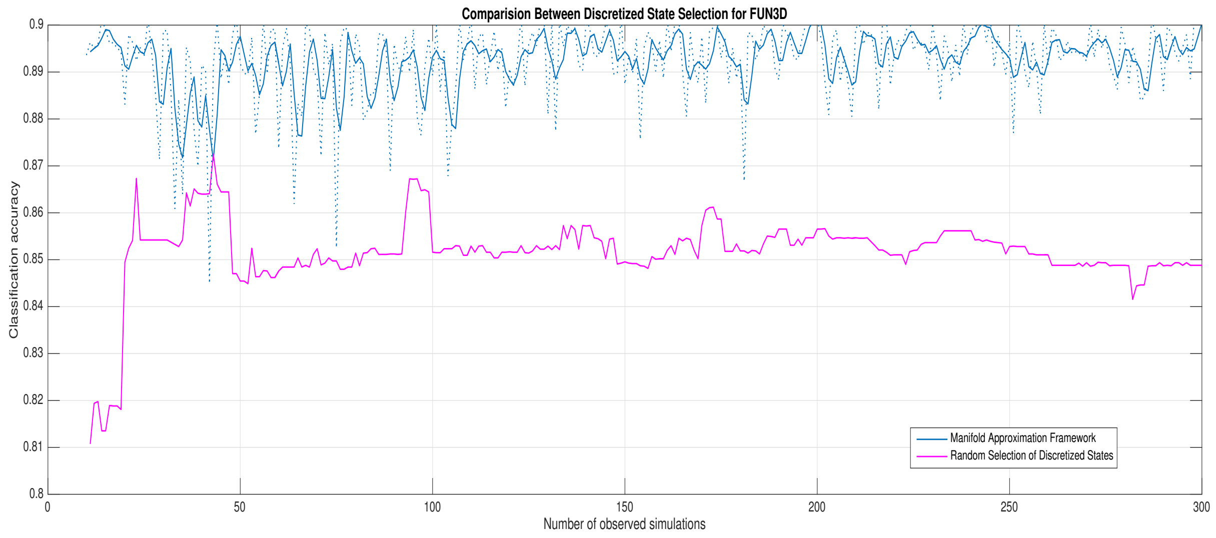


Figure 17. Comparison between discretized state selection for FUN3D simulations

Note that the classification accuracy upper-bound is at 90% accuracy, reasoned by the previously discussed artifacts of discretization and offline examinations. It is evident that the framework achieves higher classification accuracy than randomly selecting discretized states as interested simulation parameters as shown in Figure 17.

5.2.6 Discussion – We demonstrated that the proposed framework learns models with sufficiently good classification accuracy. We showed that this works well on the simulation software FUN3D. It is easy to imagine that the random selection method becomes largely infeasible as the simulations become more and more complex spanning several days or weeks for a single simulation. Therefore, we emphasize the intelligent selection and exploitation of underlying structure in the decision manifold to help alleviate this and render the problem more feasible.

6 Conclusions

Summary – The main contribution of this paper is a learning framework that attempts to minimize the number of simulation results that are necessary to effectively train a knowledge bot. In this paper, we present the framework that effectively learns manifolds that decide the simulation success of various simulation software. We showed that this framework works well on several demonstration problems that include single and disjointed isotropic decision boundaries. Furthermore, we extended previous work, showing that a knowledge bot can be trained using only 25 simulation results as opposed to the previous $10,000$ for the hyperbolic wave problem. Although this is a simple problem, with linear separability in the input space, we demonstrated the efficacy of this framework on two real world simulations: POST2 and FUN3D, showing that sufficient accuracy can be achieved by training only on a small subset of training instances, intelligently selected via the framework presented in this paper.

Discussions – Many of the comparisons in this paper were by means of a discretized state space using $N = 100$ for each input dimension. As a result this limits the maximum accuracy such that the most accurate model that can be learned is proportional to the granularity of the discretization of the input parameter space. The discretization can be refined, allowing for more accurate convergence towards the manifold, although this may increase the number of simulation counts before the trajectory converges slightly. Furthermore, we may consider setting the convergence epsilon to be proportional to the maximum domain values or based on how discretized the input parameter state space is for potential extensions to alleviate excessive parameter tuning.

Future Work – For future extensions, it is important to consider both the exploration of the state space and the exploitation of the underlying structure of the problem similar to many reinforcement learning related algorithms. Extensions may consider using information theory by maximizing information gain in addition to Gaussian processes to better select new target positions for future simulations. In addition, after tuning for the correct parameters, we would like to move away from offline test sets to integrated simulations and port this work into other simulation software and domains as a systems analysis tool. Future considerations may to involve this framework not directly in the input parameter space but to first project the states into some higher dimensions where there are fewer disjoint regions, fewer discontinuities, and where decision boundaries are smoother.

References

1. J. M. Wong and J. A. Samareh, "Training knowledge bots for physics-based simulations using artificial neural networks," NASA Langley Research Center, Hamption, Virginia, Tech. Rep. NASA/TM-2014-218660, November 2014.

2. J. M. Wong and J. A. Samareh, "Knowledge bots: Approach to learning idiosyncrasies of physics-based simulations," Tech. Rep., unpublished manuscript.

3. H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning.* ACM, 2009, pp. 609–616.

4. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

5. D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012, pp. 3642–3649.

6. C. Theriault, N. Thome, and M. Cord, "Dynamic scene classification: Learning motion descriptors with slow features analysis," in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, June 2013, pp. 2603–2610.

7. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

8. A. L. Maas, A. Y. Hannun, C. T. Lengerich, P. Qi, D. Jurafsky, and A. Y. Ng, "Increasing deep neural network acoustic model size for large vocabulary continuous speech recognition," *arXiv preprint arXiv:1406.7806*, 2014.

9. A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates *et al.*, "Deepspeech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

10. B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, R. Cheng-Yue, F. Mujica, A. Coates *et al.*, "An empirical evaluation of deep learning on highway driving," *arXiv preprint arXiv:1504.01716*, 2015.

11. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

12. Q. Le, "Building high-level features using large scale unsupervised learning," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, May 2013, pp. 8595–8598.

13. A. G. Howard, "Some improvements on deep convolutional neural network based image classification," *CoRR*, vol. abs/1312.5402, 2013.

14. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 675–678.

15. T. Xiao, J. Zhang, K. Yang, Y. Peng, and Z. Zhang, "Error-driven incremental learning in deep convolutional neural network for large-scale image classification," in *Proceedings of the ACM International Conference on Multimedia*, ser. MM '14. New York, NY, USA: ACM, 2014, pp. 177–186.

16. M. Kümmerer, L. Theis, and M. Bethge, "Deep gaze I: boosting saliency prediction with feature maps trained on imagenet," *CoRR*, vol. abs/1411.1045, 2014.

17. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, pp. 1–42, April 2015.

18. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

19. Z. Zografski, "A novel machine learning algorithm and its use in the modelling and simulation of dynamical systems," in *CompEuro'91. Advanced Computer Technology, Reliable Systems and Applications. 5th Annual European Computer Conference. Proceedings.* IEEE, 1991, pp. 860–864.

20. J. Vorba, O. Karlík, M. Šik, T. Ritschel, and J. Křivánek, "On-line learning of parametric mixture models for light transport simulation," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 101:1–101:11, Jul. 2014.

21. G. L. Brauer, D. E. Cornick, and R. Stevenson, "Capabilities and applications of the program to optimize simulated trajectories (post)," Martin Marietta Corporation, Tech. Rep. NASA CR-2770, 1977.

22. R. T. Biedron, J.-R. Carlson, J. M. Derlaga, D. P. H. Peter A. Gnoffo, W. T. Jones, B. Kleb, E. M. Lee-Rausch, E. J. Nielsen, M. A. Park, C. L. Rumsey, J. L. Thomas, and W. A. Wood, "Fun3d manual: 12.7," NASA Langley Research Center, Hamption, Virginia, Tech. Rep. NASA/TM-2015-218761, May 2015.

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

1. REPORT DATE *(DD-MM-YYYY)* 01- 01 - 2016	**2. REPORT TYPE** Technical Memorandum	**3. DATES COVERED** *(From - To)*

4. TITLE AND SUBTITLE Decision Manifold Approximation for Physics-Based Simulations	**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 388496.04.01.02
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Langley Research Center Hampton, VA 23681-2199	**8. PERFORMING ORGANIZATION REPORT NUMBER** L-20657
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-2016-219004

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 63
Availability: NASA STI Program (757) 864-9658

13. SUPPLEMENTARY NOTES

14. ABSTRACT

With the recent surge of success in big-data driven deep learning problems, many of these frameworks focus on the notion of architecture design and utilizing massive databases. However, in some scenarios massive sets of data may be difficult, and in some cases infeasible, to acquire. In this paper we discuss a trajectory-based framework that quickly learns the underlying decision manifold of binary simulation classifications while judiciously selecting exploratory target states to minimize the number of required simulations. Furthermore, we draw particular attention to the simulation prediction application idealized to the case where failures in simulations can be predicted and avoided, providing machine intelligence to novice analysts. We demonstrate this framework in various forms of simulations and discuss its efficacy.

15. SUBJECT TERMS

Big data; Deep learning; Machine learning; Reinforce learning

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	28	(757) 864-9658

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