COMBINING REACHABLE SET COMPUTATION WITH NEURON COVERAGE

By

Ulysses Yu

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

August 7, 2020

Nashville, TN

Approved:

Professor Taylor Johnson, PhD.

Professor Xenofon Koutsoukos, PhD.

ACKNOWLEDGMENTS

The material presented in this thesis is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) through contract number FA8750-18-C-0089 and the Air Force Office of Scientific Research (AFOSR) through award number FA9550-18-1-0122. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of AFOSR or DARPA.

Writing a thesis is difficult, and I would like to acknowledge those that have assisted me with this process. First, I would like to thank my advisor, Taylor Johnson PhD., for all of his helpful guidance and support, in addition to his technical expertise and research knowledge. I would like to acknowledge as well as Tran Hoang Dung, Neelenjana Pal, Diego Lopez and the rest of the VeriVITAL lab for the assistance they provided for understanding their research. In addition, thank you to Xenofon Koutsoukos PhD. and Chris Lindsey for their feedback during the thesis process. I also want to thank professors Graham Hemingway PhD. and Maithilee Kunda PhD. for all of their words of advice throughout my Vanderbilt career. I would also like to acknowledge Matthew Burruss and Jasper Lu for assisting me with questions over the dual degree thesis process.

I am also incredibly grateful to my family for all of their moral support throughout my education. They have supported me emotionally and financially throughout my entire life, and everything I have achieved, I owe to them. Special thank in particular to my parents, for their sacrifice and perseverance in ensuring that I obtain the best education possible. Finally, I would like to thank my friends at Vanderbilt for always being by my side during such a challenging process. I consider myself very lucky to have friends who believe in me and make me a better person, and I am grateful for everything they have contributed to my mental health and well-being.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACAS Xu Airborne Collision Avoidance System Unmanned

CNN Convolutional Neural Network

CPS Cyber Physical Systems

DNN Deep Neural Network

MNIST Modified National Institute of Standards and Technology

MPT Multi-Parametric Toolbox

NNV Neural Network Verification

ReLU Rectified Linear Unit

CHAPTER I

Introduction

Within the past decade, there has been an expansion of the use of machine learning in many applications, spurred by the rise of deep neural networks (DNN) [1]. One field where the usage of deep learning systems has increased is in cyber-physical systems (CPS). Some examples of this high use of DNNs can be found in the autonomous vehicle sector, where companies such as Uber, Tesla, and Alphabet have all employed Deep Learning as part of their systems [2]. One relatively extreme example is NVIDIA's work on steering a car using exclusively machine learning components [3]. While neural networks can be incredibly accurate, even reaching levels of human perception, they can fail if they are not properly tested, leading to fatal accidents [4,5]. As usage of deep learning systems continue to increase, especially in safety critical systems, they will likely lead to more accidents if techniques are not developed to ensure the safety of machine learning components.

Thus, it is important to be able to ensure that neural networks are safe. However, ensuring safety is complicated by the fact that neural networks are incredibly complex and difficult to understand. Neural networks are not directly coded, but rather they are learned through machine learning algorithms, so their performance is subject to the quality of the data presented. Since neural networks can oftentimes contain thousands, if not millions of parameters, there needs to be a way to scalably analyze the safety of neural networks [4].

To evaluate neural networks, we can test them by making sure that every single input matches the expected output. However, this approach has the problem of the input space becoming very large and unfeasible to evaluate. For example, the images of the Modified National Institute of Standards and Technology (MNIST) dataset, which is a simple dataset of handwritten numbers used for training networks, consists of $28 * 28$ gray scale images with 8 bits per pixel, leading to a color depth of 2^8 or 64. Quickly computing the total

possible images leads to a total number of $2^{28*28*8}$ or 2^{6272} images [6]. Because of this large state space, it becomes infeasible to evaluate every individual input, and this problem is only exacerbated on larger data sets such as Canadian Institute For Advanced Research (CIFAR-10) or ImageNet, which have more pixels and a higher color depth [7]. Thus, instead of trying to use brute force to compute all possible outcomes, we have to find alternative methods of verifying the overall safety of a neural network.

In addition to issues of scalability for testing DNNs, another threat is the existence of adversarial examples. It has been shown that there are many methods to manipulate the inputs to DNNS in order to induce false outputs. For example, [8] shows that changing even a single pixel can result in an image that is nearly identical to the naked eye but creates a completely different output in a machine learning classifier. This sort of attack is problematic because we would expect a small alteration to an image would not make a difference in performance of the system, but it can even lead to cases when the overall safety of the system is compromised. Thus, ensuring safety for DNNs also means preventing adversarial examples from causing the CPS as a whole to fail.

One way to analyze safety from adversarial attacks is by evaluating the robustness of a given neural network. To ensure an image is robust, we want to be able to ensure that any alteration to an image within a given range does not lead to an incorrect output or one that threatens the safety of the system. This sort of analysis has been done in a wide variety of ways. A general overview of techniques to verify neural networks is presented in [9].

One technique for implementing robustness analysis is through reachability, which takes in a set of images and propagates them through the neural network to determine if the output set is robust. Typically, this input set of images is continuous and consists of an input shape that is then output as an output shape. To verify the safety of this output, we want to then ensure that it fulfills certain safety properties that are defined for the overall CPS. If the output shape does not adhere to the safety principles, then the verifier should ideally output a counterexample showing how the safety property is violated. Otherwise,

the verifier should show a proof detailing how the output region is safe.

There are many possible input sets for reachability analysis, based on high-dimensional shapes such as hyperrectangles, zonotopes, and polytopes. Stars are a representation of polytopes, and any star input can be output as a star. In addition, stars can be extended for CNNs as ImageStars, which are well suited for verifying images. Both star and ImageStars are implemented in the Neural Network Verification (NNV) toolbox, as well as hyperrectangles and zonotopes for comparison [10].

In addition to robustness-based analysis, another form of ensuring safety is to examine what percentage of the neurons in the neural network have been tested. This is done by a metric known as neuron coverage, which is similar to code coverage used in traditional software applications [11]. The goal of neuron coverage is to look at which neurons have been activated by a given input or set of inputs, and use that to determine which neurons have been tested. One key element of neuron coverage is that it is currently only defined for sets of individual inputs, as opposed to continuous sets of inputs.

Neuron coverage is based on the code coverage verification technique, which allows the user to ensure that they have tested every part of their software and are not just repeatedly testing on certain sections of logic and ignoring corner cases. The reason why traditional code coverage is not used is because in certain cases, even one test input can achieve 100% code coverage without demonstrating complete coverage of the neural network. Thus, neuron coverage is used, because it can express which neurons have been tested for and which ones would require more extensive testing.

I.1 Contributions

In this thesis, our contributions towards advancing safety of DNNs in CPS are as follows.

- First, we propose a method for computing the volume of star sets through polytope volume computation.
- Secondly, we extend the definition of neuron coverage to include volumes of contin-

uous sets.

- Third, we implement this definition within the NNV toolbox and test it on a variety of DNNs, including the Airborne Collision Avoidance System (ACAS Xu) dataset.

CHAPTER II

Background Information

II.1 Neural Networks

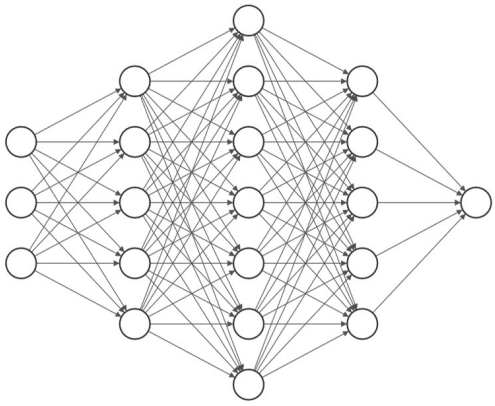
In this thesis, our goal is to propose a method for computing the volume of certain reachable sets, and to apply that technique to neuron coverage. We propose a mathematical framework for doing so and implement it within the NNV code base [10].

Neural networks can be described as layers of neurons, each of which feeds into another layer. The input is fed into the first layer and output in the final layer. To get from one layer to another, two steps occur. First, the input to a layer is multiplied by a set of weights and added to a bias, which is a linear transformation of the input. Next, a nonlinear activation transforms the result into the output. This nonlinear activation is essential in ensuring that the neural network does something other than just performing affine transformations from layer to layer. Examples of activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh. The ReLU function in particular is a function that outputs $max(0,x)$, as shown in II.1b. Since this ensures that the result is either 0 or a number greater than 0, we know that a neuron will be activated only if x is positive, otherwise the output from the neuron will be 0. More formally, a layer of a neural network can be defined as follows:

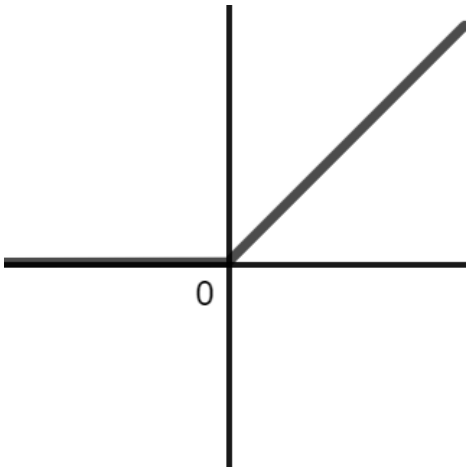
$$\hat{Y}_l = f_l(W_l * x_{l-1} + b_l) \tag{II.1}$$

where each $\hat{Y}_l$ is the output of the layer l, f_l is the activation function, W_l and b_l are the weights and biases, and x_{l-1} is the input from the previous layer. A deep neural network is just the composition of these layers where the previous output becomes the next layer's input. An example is shown in II.1a.

In addition to deep feedforward networks, such as the above, there is another type of neural network, the convolutional neural network (CNN). The convolutional neural net-



(a) A Deep Neural Network with one input layer, three hidden layers, and one output layer.



(b) The ReLU activation function. Note that the function is a piecewise linear function $y = x$ for positive values, and $y = 0$ for negative values.

Figure II.1: An example DNN along with the ReLU activation function.

work is a special case of neural network applied mainly to image recognition tasks. While CNNs have a lot in common with feedforward networks, they differ in that they also have added layers such as pooling layers, and they utilize convolutions to reduce the number of parameters needed to transition between each layer [1].

The traditional method of evaluating the effectiveness of a DNN is through its accuracy. However, it has been shown that accuracy is not a good metric [12] because of the prevalence of adversarial examples. In addition, accuracy is limited to the examples within the dataset, so when the DNN encounters previously unseen examples, we want to be able to ensure that the network is properly able to handle working with those new examples.

II.2 Reachability Analysis

Reachability analysis is a method of analyzing the safety and local robustness of a neural network. Local robustness is defined to be

$$\forall x \, ||x - x_o|| \leq \delta \implies N(x) = N(x_o) \tag{II.2}$$

[13], where N can be defined as the neural network function, x defines an element of the input space, x_o is the original input, and δ is a scalar value. Intuitively, this definition means that for a small δ around the input example of x_o, an adversarial perturbation would not be able to cause the example to be misclassified. Thus, using robustness as a measure of a network's performance can avoid one of the downsides of using accuracy, which is its poor performance on adversarial examples.

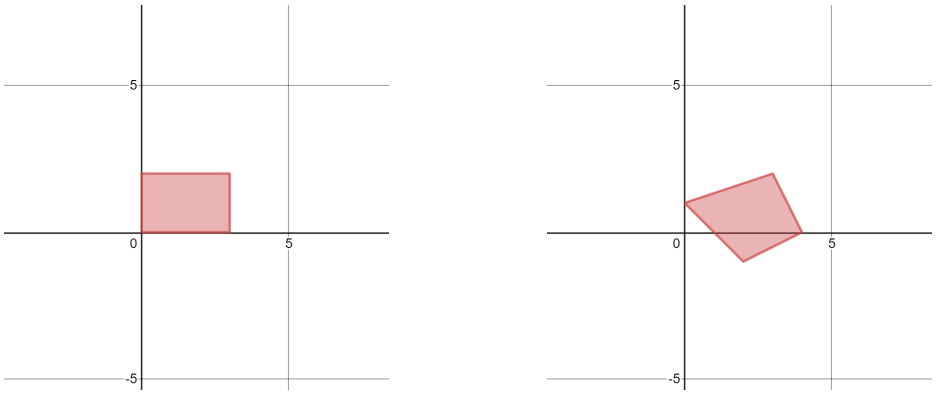
Reachability analysis takes an input and surrounding neighborhood and propagates it forward through each layer of a network, creating an output set for each layer. The output of the previous layer is used as the input set for the next layer, until the last layer of the network has an output set [9]. More formally, the reachable set for a layer l of a given neural network is defined as follows:

$$R_l = \{y_l | y_l = f_l(x_l) \wedge x_l \in R_{l-1}\} \tag{II.3}$$

where y_l are all of the elements of R_l, f_l is the function that represents the lth layer of the neural network, R_{l-1} is the reachable set of the layer prior to the current layer, and x_l are the elements of R_{l-1}.

When the output set is computed, safety analysis can be performed on it. If the output set intersects with an unsafe region or cannot fulfill a certain predefined safety property, we can claim that the network is not locally robust for the given input. However, if the output is shown to be safe, then we can claim that the input is locally robust and can provide proof that it has been verified.

There are many types of input sets that can be used for reachability analysis, such as hyperrectangles, zonotopes, and polytopes, as shown in figure II.2. Since there can be hundreds or thousands of neurons in a network, this can lead to very high dimensional shapes. Since polytopes use exact bounds for reachable set analysis, running polytopes through the input set will result in correct outputs for verification. The results will be



(a) Rectangle (b) Polygon

Figure II.2: Lower dimensional analogues of the hyperrectangle and polytope, respectively.

sound and complete [9]. Soundness means that when the verifier returns that the output is safe, it is actually safe. Completeness is defined to be when the solver never returns unknown, and that when the solver returns that a property is violated, that the property is actually violated.

However, this method can have a higher runtime, so over-approximate alternatives are used instead. Shapes used in over-approximate analysis include hyperrectangles and zonotopes, which can output incorrect responses, but have faster runtimes. When over-approximate analysis is used, it can lead to outputs that are sound but not complete. Since hyperrectangles and zonotopes can be incomplete, they can respond that a property is violated when it is not actually violated, but they will not return that a region is safe when the region is unsafe because they are sound. In this thesis, we will primarily focus on star sets, which are a representation of polytopes that have faster reachability computation times than just polytopes.

II.2.1 Star Sets

A star set is defined as a tuple (c, V, P), where c represents a given input to a neural network, V is a set of basis vectors, and P is a set of predicates [14]. The predicates in P are given

as a set of linear equations such that

$$P(\alpha) = C\alpha \leq d \tag{II.4}$$

where C and d represent a system of linear inequality constraints, and α corresponds to the predicates in P. Star sets are used in reachability analysis. Since sets can be split up due to the piecewise nature of the ReLU activation, we can potentially end up with more sets than we started with. Over approximate analysis of star sets also exists; this can lead to situations where the set's outputs are larger than what the exact analysis sets would be, but results in significantly faster computations and also reduces the number of reachable sets produced.

For the purposes of this thesis, there are several facts that are important to understand about star sets. First, since each star set represents a polytope, we can use the predicates in the representation to derive the original polytope as shown below

$$Polytope = C\alpha \leq d. \tag{II.5}$$

This equation also allows us to compute the volume of a given star set.

Second, any star set propagated through the layers of a neural network with piecewise linear activation functions will also create more star sets that can be input into the next layer. Theoretically, since a split can occur with any neuron that uses the ReLU activation function, this could lead a total of 2^n output sets, where n is the number of neurons in the entire network. Because of this, there are also approximate methods for computing the reachable sets that result in less output sets but could also overestimate the output region.

An ImageStar is an extension of the star set that applies to images [15]. It is often used to analyze convolutional neural networks as opposed to feedforward neural networks. ImageStars retain many of the same properties as star sets, but the anchor and basis vectors represent images instead of input ranges. Although their reachability analysis is more

complex, we only need to focus on the fact that they are also based on convex polytopes, which allows us to compute their volume.

II.3 Neuron Coverage

In the DeepXplore paper [11], neuron coverage for a given neuron is defined as

$$NCov(T,x) = \frac{|(n|\ \forall x \in T,\ out(n,x) > t)|}{|N|} \tag{II.6}$$

where $N = \{n_1, n_2, \dots\}$ represents all of the neurons, $T = \{x_1, x_2, \dots\}$ represents all of the test inputs, $out(n,x)$ is the output of neuron n with input x, and t is the activation threshold.

Practically, this measures if a neuron reaches above a certain activation for a given input. The neuron activation over an entire neural network can then be given by taking the total proportion of activated neurons over the total number of neurons in the network. Typically, the activation threshold used is 0, since it simply represents if a given neuron is activated.

Neuron coverage is a useful metric because it helps determine how a network will respond to previously unseen examples. If a network has not been fully covered by its tests, it is possible that the network then will fail on new examples that activate portions of its logic that have not been activated yet. By covering all of the neurons of a network, we can ensure that all aspects of its logic have been tested, which is an important step towards verifying the overall safety of the network.

Currently, this measure is only applied on sets of individual inputs. While this is useful, inputs to a DNN can be continuous in certain situations, so it would make more sense to test those inputs in a continuous manner. In this thesis, instead of using individual inputs, we will be extending the definition of neuron coverage so that it can include reachable sets.

CHAPTER III

Related Work

III.1 Adversarial attack examples

The problem of AI safety and especially adversarial attacks on neural networks is a major subject of ongoing research. Some notable adversarial attacks include the Fast Gradient Sign Method, which uses linearization to generate fast adversarial examples, and the Carlini Wagner attack, which is effective against defensive distillation [16, 17].

In Kurakin et al., the authors show that real-world images from a cell phone camera can be misclassified, showing that adversarial attacks do not have to occur when an attacker directly controls what is input to the model, but can occur through real-world scenarios [12]. This is furthered in the paper Eykholt et al., where the authors demonstrate how a user can generate real-life adversarial attacks. Using only black and white stickers, the authors are able to attack multiple road signs and cause misclassifications [18].

In the paper Dreossi et al., the authors are able to show how errors in a machine learning component are able to affect the overall performance of the CPS. They also provide a framework for searching for falsifying examples and demonstrate it on an automatic braking system [19].

III.2 AI Safety

The problem of AI safety is discussed in the paper Towards Verified AI. This paper discusses some of the key principles needed to solve the problems related to creating a verification system for AI safety. In particular, the fourth problem they discuss, the creation of computational engines to verify safety using formal methods, is pertinent to the topic of this paper. The authors of this paper also propose a software system for verification named VerifAI in another paper. This toolkit also helps provide a framework for testing and verifying machine learning components [20].

One component that is used heavily in the VerifAI paper is Scenic, a probabilistic programming language used to generate testing examples. Using probabilistic programming is a good way to generate a diverse variety of examples, especially for testing autonomous vehicle systems [21]. Another way of generating examples is presented in the paper DeepTest, which generates examples by performing affine transformations on the original images [22]. DeepRoad is a system which builds on the examples presented in DeepTest that uses Generative Adversarial Networks to transform images of sunny roads into snowy or rainy roads for testing autonomous cars [23].

Other areas of DNN verification where examples can be drawn from software verification include input space partitioning. In the paper Reachable Set Estimation [24], an algorithm is presented to bisect the input spaces based on the simulations, which saves computation time because it allows for use of larger intervals as inputs. One possible approach to partitioning of the input space is through the use of t-Distributed Stochastic Neighbor Embedding (t-SNE), which reduces the dimensions of the input space [25]. This method could potentially be used with convex hulls to define partitions on the input space.

III.3 Neuron Coverage

In the paper DeepGauge, an alternate definition of neuron coverage, k-multisection coverage, is presented as:

$$\frac{|\{S_i^n |\ \exists x \in T :\ out(n,x) \in S_i^n\}|}{k} \tag{III.1}$$

where k is the number of sections, S_i^n is the set of values in the ith section for $1 \leq i \leq k$, and the rest of the variables are the same as in DeepXplore. They also define an upper and lower neuron coverage for edge cases:

$$UpperNeuron = \{n |\ \exists x \in T :\ out(n,x) \in (high_n, +\infty)\} \tag{III.2}$$

$$LowerNeuron = \{n |\ \exists x \in T :\ out(n,x) \in (-\infty, low_n)\}, \tag{III.3}$$

where $high_n$ and low_n represent the upper and lower boundaries of the activation for a neuron selected during the training phase, and all other variables are the same as in DeepXplore [26].

This definition of neuron coverage focuses on splitting the output values of each neuron into sections. Then each of the sections are evaluated to provide a more detailed analysis of the neuron's outputs. The upper and lower neurons also allow for examining corner cases more closely.

In the thesis Is Neuron Coverage a Meaningful Measure for Testing Deep Neural Networks [27], the author examines the concept of neuron coverage to see if it leads to testing sets with greater defect detection, naturalness, and output diversity. The paper concludes that increasing neuron coverage does not benefit any of those metrics.

The paper Sun et al. [28], proposes another four criteria for testing coverage of neural networks, which can be used to help detect adversarial examples. The paper also demonstrates how their metrics are stronger than neuron coverage.

In Li et al. [29], the authors criticize structural coverage criteria, such as neuron coverage, for its inability to detect fine grain adversarial attacks. They test the techniques used in DeepXplore and DeepGauge to show that they are still vulnerable to adversarial attacks. In addition, they examine the amount of misclassification in natural examples and show that coverage does not necessarily correlate with less misclassification. However, this work is currently still in preliminary stages, so it would be worth examining if more experiments are conducted on the usefulness of neuron coverage in the long term.

III.4 Reachability

Xiang et al. [30] introduces the concept of polytope reachability analysis; however it runs into problems related to scale and computation time. Abstract Interpretation for AI is a method of analysis that uses abstract interpretation to create reachable sets [31]. The abstract domains that it deals with are polyhedrons, zonotopes, and boxes. It over approx-

imates the reachable sets, which has an added runtime benefit but can lead to incorrectly saying a property is violated. ReluVal uses a hyperrectangle box approach to perform reachability [32]. Unlike other box approaches, it uses symbolic intervals as opposed to numeric ones, which can lead to tighter bounds.

Other approaches to verifying neural networks include using optimization techniques. These methods treat the verification problem as a satisfiability problem and treat the neural network as one of the constraints to optimize. One example is the Reluplex algorithm which builds on the simplex algorithm by making it feasible on ReLU activation functions [33]. The Reluplex algorithm is able to verify properties on the ACAS Xu networks, making it relevant to the work presented here. The Marabou framework builds on Reluplex by optimizing it and extending it to any piecewise activation function, instead of just the ReLU function.

The paper Algorithms for Verifying Neural Networks [9] is a survey paper that addresses many of the papers listed above and provides more detailed information on other papers related to verification as well. It also classifies the verification techniques into satisfiability or reachability and discusses issues such as completeness that are not addressed here.

III.5 Polytope Volume Computation

A polytope is a higher dimensional analogue of a two-dimensional polygon or three-dimensional polyhedron, and a convex polytope is one in which all points in the polytope form a convex set. They can be represented in two ways. The first is the vertex representation, which can be obtained by taking the convex hull of a finite set of points. The second definition of a polytope is through its half space representation. This representation can be written as a linear inequality, as shown in the equation II.5

Common techniques for computing the volume of a polytope use triangulation of the vertices and a summation of the triangulations to compute the overall volume. It has been

shown that the complexity of computing the volume of a polytope is #P-hard [34, 35]. #P hardness is a category of problems that is a counting problem based off of the NP decision problem [36]. This means that computing the volume of a polytope efficiently would require solving the $P = NP$ equation. Some estimation techniques exist for volume computation, including some based off of Monte-Carlo algorithms, and these typically have polynomial runtimes [37].

CHAPTER IV

Volume Computation

In this thesis, we introduce a method of computing the neuron coverage of a given network using a reachable set. Currently, all methods of computing neuron coverage use individual inputs as opposed to continuous sets, which we do here. The sets that we focus on are the star sets and also the ImageStar set for CNNs.

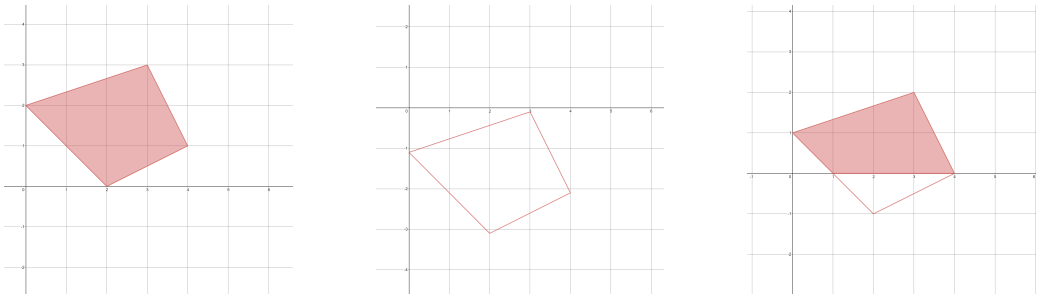
IV.1 Definition

First, we need to present a definition for volume of a set. The $Vol(x)$ of a set x is defined to be the volume of the set in the space $\mathbb{R}^n$ where n is the dimensionality of the input space for that layer of the network. Since this volume is occurring within a higher dimensional space, it can be treated as a hypervolume in the nth dimension. This volume assumes that the input space is continuous, and that the input set is also bounded. By introducing a volume over the input set, we are able to describe what portion of the overall input space has been included in a reachable set. For example, a point x_0 in $\mathbb{R}^n$ would have a volume of zero, but the set $\{x \mid x_0 - 1 \leq x \leq x_0 + 1 \ \wedge \ x \in \mathbb{R}^n\}$ would have a volume 2^n.

Now that we have formally defined the volume of a set, we present an extension of the current definition of neuron coverage. Intuitively, neuron coverage is a measure that checks if a certain test example activates a given neuron and aggregates that value over the entire network. We extend the definition in the equation below:

$$NCov(T,x) = \frac{\frac{\sum Vol(R_l \cap \alpha_{li} \geq t)}{\sum Vol(R_l)}}{|N|} \tag{IV.1}$$

where i represents the current neuron in layer l and R_l represents the reachable set for a given layer. The $\alpha_{li} \geq t$ represents the half-space defined by being greater than a certain activation t. Intuitively, the intersection of the two sets represents the set created by exam-



(a) A fully active reachable set

(b) A completely inactive reachable set

(c) A partially activated reachable set

Figure IV.1: An example of the three possibilities for set activation levels. The shaded areas represent the activated portions of the set.

ining only the part of the reachable set that is activated by the neuron.

This metric reflects the continuous nature of the set, and by aggregating it over every neuron, we obtain the full coverage for the network. This definition is very flexible, since it can be extended to any reachable set and any method of computing the volume.

There are three distinct possibilities of what can occur when a reachable set is intersected with a half space. First, if the resulting set is empty, it means that the intersection between the original set and the half space produced an empty set. The volume of the shape produced would be zero, and in this case the neuron coverage would be zero for that neuron. Second, if the volume stays the same after activation, this means that the intersection of the two sets did not change the sets themselves. This would mean that the neuron is activated by the entirety of the reachable set, or that any element of the reachable set would activate the neuron.

The final option is that the neuron is partially activated, which is a new term that we define. This occurs when the intersection of the reachable set and the half space has a lower volume than just the polytope, but is not 0, as shown below:

$$0 < Vol(R_l \cap HalfSpaceActivation) < Vol(R_l). \tag{IV.2}$$

In this case, we can see that the neuron is partially activated, and find the percent activation of the neuron by computing the volume of the new set and dividing that by the volume of the prior set. The three scenarios are portrayed in figure IV.1.

In the third case, we want to test the computation for star sets. In order to compute the volume of a star set, we need to convert it back to its polytope representation. To compute the volume of the polytope defined by a given star set, we use the following equation:

$$Vol(Star_l) = Vol(C_l\alpha \leq d_l). \tag{IV.3}$$

To compute the volume of the star set when it is activated, we intersect the polytope with the hyperplane defined by the inequality

$$Vol(C_l\alpha \leq d_l \cap \alpha_{il} \geq t), \tag{IV.4}$$

where l represents the current layer and i represents the neuron which is being activated. Since that intersection defines the shape of the activated polytope, by computing the volume of the activated polytope, we know what proportion of the reachable set was activated by the given neuron.

IV.2 Algorithm

To implement this new metric, we used the existing NNV code base [10]. We used the Multi-Parametric Toolbox (MPT) on Matlab to compute the volumes of the polytopes [38]. MPT draws from the Qhull library for implementation of methods related to polytopes and their volume computation [39]. First, we ran reachability analysis for a given input star/ImageStar. Second, when the output was computed for each layer, we computed the volume of the polytope for the star for each layer. Then, iterated over every neuron in the layer, finding the volume of the new polytope formed by intersecting the original polytope with the half space defined by the neuron. Finally, we compiled this data together and

obtained the neuron coverage for each neuron as a result.

Algorithm 1 Neuron Coverage Algorithm

NeuronCoverage$[R_1 \dots R_L]$

1: **for** all l in $1 \dots L$ **do**
2: $\quad Vol(R_l) = Vol(C\alpha \leq d)$
3: $\quad$ **for** all n_j neurons in l **do**
4: $\quad\quad cov(i,j) = \frac{vol(R_l \cap \alpha_{ij} \geq t)}{vol(R_l)}$
5: $\quad$ **end for**
6: **end for**
7: **return** cov

IV.3 Manual Example

To evaluate our metric and implementation, we tested our approach on several different networks. First, we tested on the manual example presented in the NNV package as proof that the metric works. The initial example utilized a two-layer feedforward network; the first layer had 3 neurons and the second had 2 neurons. The input star was a two-dimensional star with dimensions $x_1 \in [-1,1]$, $x_2 \in [-2,0]$. The coefficients C and d of the linear inequality used in the predicate P are defined in the following matrices:

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, d = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}. \tag{IV.5}$$

The center c and the generator V are also defined as follows:

$$c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, V = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{IV.6}$$

After running the analysis, we found that all 5 neurons were partially covered, as demonstrated in Table IV.1. We also visualized the coverage in diagram IV.2. For compar-

Layer	Neuron	Star Coverage	Center Point Coverage	Aggregated Single Inputs
1	1	.5	0	.484
1	2	.125	0	.138
1	3	.125	0	.142
2	1	.0656	0	.062
2	2	.1958	0	.186

Table IV.1: Comparison of coverage between stars and single inputs for the manual example

ison, we also tested the coverage on an individual point defined to be the center of the star, with coordinates $(0, -1)$. This individual example highlights how volume-based coverage can be revealing — if we only examine one example, we have a very narrow view of how much of the network is covered as opposed to if we examine a larger portion of the input space.

As another point of comparison, we computed the coverage over 1000 points selected from a uniform random distribution over the range of the input star. This is a good approximation for the value computed by the volume, as noted by their mean squared error value of $1.6477 * 10^{-4}$. Although this is a good approximation value, since the coverage computed is through an analytical method of volume computation, as opposed through generating inputs from a uniform random distribution, the volume coverage will be a more exact representation.

This example reveals to us that all of the neurons are partially covered. This is good, because it means that the overall network has been covered, so we can be assured that we tested for its logic in most cases. Although this example is small, it proves that the overall metric works and is a feasible way of computing neuron coverage.

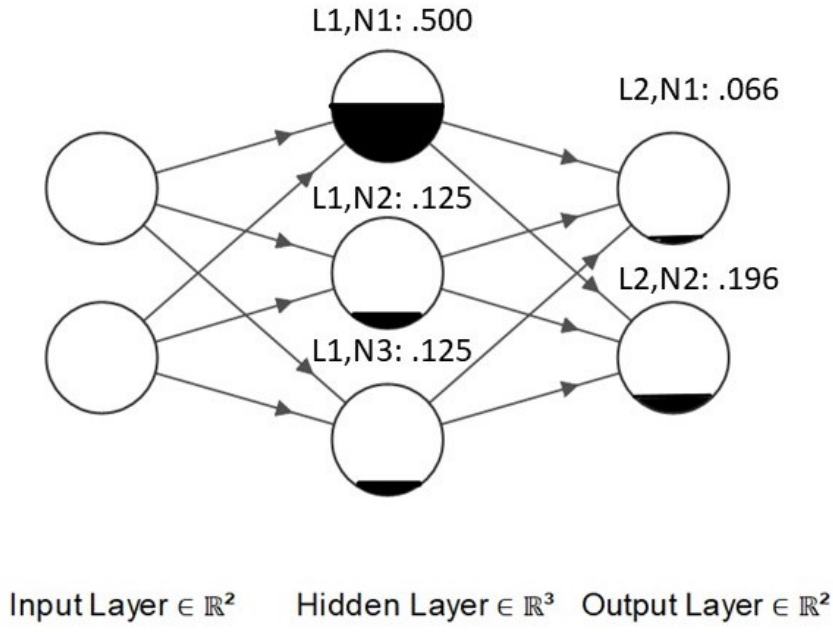


Figure IV.2: Percent coverage for each neuron of the manual example.

IV.4 ACAS Xu Experiments

IV.4.1 ACAS Xu Overview

In addition to the tests we ran on the manual example, we tested the volume computation on the ACAS Xu networks. ACAS Xu is a set of 45 networks for an unmanned aircraft to use in the situation where it detects another incoming aircraft, with the goal of preventing aircraft collisions. The set of networks replace a lookup table used for collision avoidance, and because of their importance in safety-critical applications are good targets for verification analysis.

There are five inputs for the ACAS Xu network $[\rho, \theta, \psi, v_{own}, v_{in}]$. The parameter ρ represents the distance between the aircraft, θ represents the heading of the ownship, ψ represents the heading of the intruder, and v_{own} and v_{in} represent the velocities of the ownship and the intruder, respectively. The parameters have the input ranges of $[0 \leq \rho \leq 62000, -\pi \leq \theta \leq \pi, -\pi \leq \psi \leq \pi,\ 100 \leq v_{own} \leq 1200,\ 0 \leq v_{in} \leq 1200]$. Each network consists of seven layers, with six layers of fifty neurons and a final output layer of five

neurons, for a total of 305 neurons.

The outputs of ACAS Xu are a set of five directions, which are $[COC, WL, WR, SL, SR]$. The W designations mean a turn of $1.5^\circ/s$, and the S designations mean a turn of $3.0^\circ/s$, and the L and R designations mean turn left and right, respectively. COC stands for clear of conflict and means that there will not be a collision between the two aircraft. To avoid collisions between the two aircraft, a certain set of properties have to hold for the entire input space, making it a useful target for reachability analysis.

IV.4.2 Experiment Overview

For our experiments, we computed the volume neuron coverage and compared it to the single input coverage. For networks 1-10, the inputs were a box with each side of the inputs taking up 1/25 of the input space, leading to a size of $1.024 * 10^{-7}$ of the original input space. For the networks 11-45, the input space was slightly smaller, with each edge taking up 3/100 of the input space, resulting in an input size of $2.43 * 10^{-8}$ of the original input space. We first computed the star reachability for the inputs, then computed the volume neuron coverage. For each network, we would take the average coverage and compare them to a set of test examples. To generate the test examples, we used a test example of 1000 single inputs for each network, selected from a uniform random distribution over the range of the input stars. To obtain the aggregate coverage for these 1000 inputs, we would simply just take the average of the coverage for each input and compare them directly.

IV.4.3 Results and Analysis

The results of the reachability are shown here in the table IV.3. Each row represents one of the 45 networks, and each column compares the percent coverage for each network. The percent coverage was computed by taking the sum of the coverages and dividing them by the number of neurons in the network. The sum includes the sum of the partial and the full neuron coverages.

For the first 10 networks, the average percent coverage was 28.88%, for the 11-45th

Network Number	Volume Coverage	Test Coverage
1	0.2076	0.2076
2	0.2747	0.2748
3	0.3145	0.3146
4	0.3438	0.3438
5	0.3289	0.3288
6	0.2800	0.2801
7	0.2584	0.2583
8	0.2805	0.2806
9	0.2949	0.2947
10	0.3049	0.3047
11	0.2200	0.2200
12	0.2454	0.2452
13	0.2654	0.2655
14	0.2526	0.2526
15	0.2767	0.2766
16	0.2948	0.2949
17	0.2351	0.2350
18	0.2564	0.2565
19	0.2097	0.2097
20	0.2702	0.2702
21	0.2079	0.2080
22	0.2738	0.2738
23	0.2532	0.2533
24	0.3005	0.3004
25	0.2703	0.2699
26	0.2574	0.2576
27	0.2493	0.2492
28	0.2576	0.2578
29	0.2542	0.2544
30	0.2558	0.2558
31	0.2716	0.2717
32	0.2762	0.2763
33	0.2649	0.2649
34	0.2694	0.2692
35	0.2614	0.2614
36	0.2583	0.2583
37	0.2607	0.2603
38	0.2533	0.2534
39	0.2333	0.2336
40	0.2492	0.2491
41	0.2783	0.2785
42	0.2776	0.2776
43	0.2914	0.2916
44	0.2796	0.2795
45	0.2504	0.2503

Table IV.3: Comparison between star sets and single inputs for ACAS Xu

Layer Number	1	2	3	4	5	6	7
Number of Stars	11.13	33.82	94.27	290.87	628.27	758.18	758.18

Table IV.4: Number of Reachable Sets Generated

input sets, the coverage was 25.95%. This shows that even with a relatively small input volume, a substantial amount of the overall neural networks can be covered. We also compared this measure to the testing coverage over the entire input range. This value can be computed by checking to see if a certain neuron has been covered by any of the testing examples and then summing the coverage. Using this, we saw that the average coverage of the network was 79.79%. On average, it means that the percent coverage achieved for the smaller input sets cover around 32.52% of how much the overall network can be covered.

One interesting aspect of the results to note is that the test neuron coverage is lower for some of the networks than the volume coverage (for example networks 2 and 6). This difference can be explained by the fact that because the test examples are chosen from a uniform random grid, which can lead to variance in which examples get activated. For certain neurons, it is possible that disproportionately more of the inputs end up in the activated portion of the output space as opposed to being evenly distributed through the output space, leading to a higher coverage than in the exact volume computation. Similarly, the coverage from the testing examples can be lower than the volume coverage if a disproportionate number of the inputs end up in the inactivated portion of the neuron.

Out of the 45 networks examined, on average 6.62% of the network was partially covered. This shows that by computing the volume of the neuron coverage, we learn more that we would have by just comparing the computation to a single input. However, when we compare the amount of coverage to a number of testing examples, we see there is a very small difference between the results. This indicates that while volume-based neuron coverage offers an alternative method of computing neuron coverage, there is not a serious advantage in using it compared to the testing coverage.

On average, the number of reachable sets generated was $2,574.7$. The number of sets generated are displayed in the table IV.4. Note that for the first 5 layers, the number of sets roughly doubles, but the last two sets do not have this behavior, due to the fact that the number of neurons decreases in the last layer, leading to less necessary splits. This highlights how the growth of the number of reachable sets is exponential because of the splits needed for each ReLU neuron.

After running the experiments, it took 1,338.1 seconds to perform volume computation per network. By comparison, the single input coverage took .1880 seconds to compute over 1000 inputs for each network. This highlights one downside of using volume computation, which is that it has a slow runtime. In fact, on average, it is 7,000 times slower than the single input coverage. This slow runtime can be very costly, as running volume neuron computation can take 60,216 seconds, or 16 hours, over all 45 ACAS Xu networks to cover a relatively small amount of the input space.

IV.5 ImageStar

However, problems occurred when we tried to run this example on larger networks with more complex inputs. When we used the small MNIST network presented in the NNV toolbox, the computation time grew to a point where it was not practical to compute the volumes of the polytopes. In our initial examples, we were working with relatively small stars, most of them had C terms with dimensions $[10, 2]$ and d of dimension $[10, 1]$. However, on the ImageStars used in the MNIST networks, the smallest stars had C of dimension $[18, 9]$ and d of dimension $[18, 1]$, with the largest having dimensions of $[219, 68]$ and d of dimensions of $[219, 1]$. For the smaller stars, computations could take upwards of 30 minutes, and for the largest, over two hours. Computing the volume of a polytope is a problem that belongs to the $\#P$ complexity class, which means that the problems have an exponentially increasing runtime.

In addition to the problem of increasing runtime for volume computation, the increasing

Perturbation Level	0	1	2	3	4	5
Percent Coverage	95.04	97.16	98.58	99.29	99.29	99.29

Table IV.5: Comparison of coverage and perturbation amount

number of generated stars also slows down the overall runtime. Recall that for a network that uses the piecewise ReLU function, each neuron could be the site for a potential split of the stars, leading to a maximum of 2^n reachable sets. Since the number of stars is increasing in an exponential manner as well, for deeper networks with upwards of hundreds of neurons, this can pose a scalability issue.

Computing the volume of the polytopes generated by C quickly became too computationally expensive and impractical. Since the ImageStars used in this example were relatively small, this could indicate that this is not a practical scalable way of performing neuron coverage analysis with sets. One way to avoid the problem with growing polytope volume computation time is through the use of volume estimation techniques, which have polynomial runtimes. In addition, we can avoid problems of increasing numbers of reachable sets through the use of over-approximate reachable sets as opposed to the precise polytope reachable sets.

In addition to the experiments we performed on the CNNs, we decided to look at a smaller DNN to evaluate if increasing the input space would lead to more coverage on a network. In order to do this, we evaluated the single input coverage for a set of images. Then, we extended the range of allowable perturbations in the test set to see if it would lead to increasing coverage. We evaluated this on a DNN presented in the NNV toolbox. To generate the test sets, we used 6 different levels of perturbation $[0,1,2,3,4,5]$. There were 1,000 images, with 100 from each of the classes $[0-9]$. For each perturbation level (except for the 0 level), we generated 10 images based off of each of the test images, leading to a test set of $10,000$ images per perturbation level. To aggregate the neuron coverage, we counted which neurons were covered by the test set instead of using an average, because it

would give us clearer information on if increasing the size of the input range would lead to an increase in coverage.

The results of the experiment are presented in the table IV.5. As evidenced by the table, increasing the perturbation level allowed leads to a higher coverage over the network. It appears that the effect is stronger for the first few perturbation levels, and drops off as the perturbation increases. However, this work is still mostly in preliminary stages, and more experiments will be needed to conclude a stronger relationship between the two metrics.

CHAPTER V

Closing Remarks

V.1 Future Work

Although the experiments presented here are promising, they are limited by the fact that computing the volume of a polytope is an expensive operation, so we present several different options to move forward. First, finding a way to cheaply compute the volume can be possible using estimation techniques instead of exact volume computation. Although estimation techniques are useful, if we are trying to find the exact amount of coverage for a network, we would need to resort to other techniques.

One such technique would be to leverage the discreteness of certain input spaces. For example, certain problems use images as their inputs, and because images have discrete color depths (as opposed to continuous input ranges), it could be worth it to use a counting approach. Currently, our approach is centered on computing a volume, but that information can be unnecessary, since we are more concerned with the cardinality of a set of images as opposed to a volume of overall images. One way to perform these computations is through examining the L_p norms of input spaces. For example, on the MNIST dataset, we can take a star with an upper bound of $+1$ to every pixel of the image and a lower bound of -1 to every pixel in the original image. This would give us a cardinality of 3^{28*28}, which would we could compare to the 256^{28*28} cardinality dataset.

Other reachable sets that we could feasibly compute the input size of in the future include hyperrectangles. This would have the advantage of being relatively cheap to compute, since computing the volume of a box only requires multiplying together the difference between intervals. Other shapes have been used in recent works, such as zonotopes, and it is worth examining the feasibility of computing the volume of those figures, although because they are over approximate, there could be problems related to their overall usefulness for

reachability analysis. Since the overall goal is to verify the safety of the entire system, we want to avoid increasing neuron coverage at the expense of decreasing the usefulness of the reachable set verification, and vice versa.

One major of computing volumes of reachable sets is to determine what proportion of the input space is covered, so comparing the sizes of input spaces would be an important problem for future research. If there is any way to narrow down input spaces, that could greatly reduce the amount of computation necessary for us to verify neural networks. One interesting side effect of this concept would also be to examine if there is a way to examine the sizes of the inputs to a particular layer of a DNN as opposed to just examining the overall network. This could be useful because it allows us to focus on verifying individual layers of the network instead of going through an entire DNN.

One area of potential research in the future would be examining the relationship between neuron coverage and reachable set size. We touched on this work earlier in the MNIST section, but due to time constraints, we were unable to explore it more fully. Some notable questions to ask would be, does increasing the reachable set size increase coverage, and by how much. In addition, it would be useful to know if the size of the network affects how much of the input set get covered, since it can allow us to balance between using structural coverage and input coverage.

Finally, once it becomes more computationally cheap to do so, more experiments should be done on the usefulness of a neuron coverage in relationship to robustness and safety. Ideally, we would want to show that increasing neuron coverage means that a larger proportion of the test set is covered. Furthermore, one important question to answer is if higher neuron coverage could lead to more robustness against adversarial attacks, which pose a huge problem to the safety of DNNs as a whole. There has been some research and criticism into the usefulness of neuron coverage as a metric, so investigating how helpful neuron coverage can be in finding good test sets is an area that can be explored further [27]. In addition, if we are able to determine that higher neuron coverage is desirable for safety,

questions about how to train networks to increase their neuron coverage would become more relevant. Finding a way to generate test sets with more neuron coverage would be useful for both more thoroughly testing the logic of the DNNs, and increasing coverage of the input space.

V.2 Conclusion

As the use of deep learning continues to increase throughout many CPSs, it becomes more important for us to ensure that they are safe. Adversarial attacks are currently one major threat to the safety of DNNs. By using reachability and local robustness, we can help prevent adversarial attacks. In addition, neuron coverage could help ensure that all aspects of a DNN's logic are tested for, rather than just focused on what is computed in the test set.

In this thesis, we introduced a mechanism for computing the volume of input sets. We applied that to the star reachability set, which is a polytope-based input set. We then used this volume computation to extend the definition of neuron coverage beyond just discrete inputs to include continuous sets of inputs. We implemented this definition through the NNV software tools. By testing this on a variety of DNNs, including the ACAS Xu dataset, we showed that this method of computing neuron coverage is able to provide accurate results. Overall, we showed that combining neuron coverage with reachable sets is a useful way to evaluate the safety of a DNN.

BIBLIOGRAPHY

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

[2] R. Brandom, "Self-driving cars are headed toward an ai roadblock," *The Verge*, 2018.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[4] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," *arXiv preprint arXiv:1606.08514*, 2016.

[5] D. Wakabayashi, "Self-driving uber car kills pedestrian in arizona, where robots roam," *The New York Times*, vol. 19, 2018.

[6] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[9] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *arXiv preprint arXiv:1903.06758*, 2019.

[10] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson, "Nnv: The neural network verification tool for deep neural networks and learning-enabled cyber-physical systems," *arXiv preprint arXiv:2004.05519*, 2020.

[11] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, 2017.

[12] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[13] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Towards proving the adversarial robustness of deep neural networks," *arXiv preprint arXiv:1709.02802*, 2017.

[14] H.-D. Tran, D. M. Lopez, P. Musau, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson, "Star-based reachability analysis of deep neural networks," in *International Symposium on Formal Methods*, pp. 670–686, Springer, 2019.

[15] H.-D. Tran, S. Bak, W. Xiang, and T. T. Johnson, "Verification of deep convolutional neural networks using imagestars," *arXiv preprint arXiv:2004.05511*, 2020.

[16] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[17] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*, pp. 39–57, IEEE, 2017.

[18] K. Eykholt, I. Evtimov, E. Fernandes, B. Li, A. Rahmati, C. Xiao, A. Prakash, T. Kohno, and D. Song, "Robust physical-world attacks on deep learning visual classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1625–1634, 2018.

[19] T. Dreossi, A. Donzé, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," *Journal of Automated Reasoning*, vol. 63, no. 4, pp. 1031–1053, 2019.

[20] T. Dreossi, D. J. Fremont, S. Ghosh, E. Kim, H. Ravanbakhsh, M. Vazquez-Chanlatte, and S. A. Seshia, "Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *International Conference on Computer Aided Verification*, pp. 432–442, Springer, 2019.

[21] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: a language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 63–78, 2019.

[22] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, pp. 303–314, 2018.

[23] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 132–142, IEEE, 2018.

[24] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson, "Reachable set estimation for neural network control systems: A simulation-guided approach," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[25] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[26] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 120–131, 2018.

[27] F. Y. Harel, "Is neuron coverage a meaningful measure for testing deep neural networks?," *Ann Arbor*, vol. 1001, pp. 48106–1346, 2019.

[28] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Structural test coverage criteria for deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–23, 2019.

[29] Z. Li, X. Ma, C. Xu, and C. Cao, "Structural coverage criteria for neural networks could be misleading," in *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pp. 89–92, IEEE, 2019.

[30] W. Xiang, H.-D. Tran, and T. T. Johnson, "Reachable set computation and safety verification for neural networks with relu activations," *arXiv preprint arXiv:1712.08163*, 2017.

[31] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, IEEE, 2018.

[32] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, pp. 1599–1614, 2018.

[33] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," in *International Conference on Computer Aided Verification*, pp. 97–117, Springer, 2017.

[34] J. Lawrence, "Polytope volume computation," *Mathematics of Computation*, vol. 57, no. 195, pp. 259–271, 1991.

[35] B. Grünbaum, "Polytopes, graphs, and complexes," *Bulletin of the American Mathematical Society*, vol. 76, no. 6, pp. 1131–1201, 1970.

[36] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[37] C. Ge and F. Ma, "A fast and practical method to estimate volumes of convex polytopes," in *International Workshop on Frontiers in Algorithmics*, pp. 52–65, Springer, 2015.

[38] M. Herceg, M. Kvasnica, C. Jones, and M. Morari, "Multi-Parametric Toolbox 3.0," in *Proc. of the European Control Conference*, (Zürich, Switzerland), pp. 502–510, July 17–19 2013. http://control.ee.ethz.ch/~mpt.

[39] B. Barber and H. Huhdanpaa, "Qhull," *The Geometry Center, University of Minnesota, http://www. geom. umn. edu/software/qhull*, 1995.