Title

Semantic Adversarial Deep Learning.

Permalink

https://escholarship.org/uc/item/1cr980mc

Authors

Seshia, Sanjit A
Jha, Somesh
Dreossi, Tommaso

Publication Date

2020

DOI

10.1109/MDAT.2020.2968274

Peer reviewed

Keynote

Semantic Adversarial Deep Learning

Sanjit A. Seshia
University of California at Berkeley

Somesh Jha
University of Wisconsin—Madison

Tommaso Dreossi
Amazon Search

Editor's note:
Adversarial examples have emerged as a key threat for machine-learning-based systems, especially the ones that employ deep neural networks. Unlike a large body of research in this area, this Keynote article accounts for the semantic, context, and specifications of the complete system with machine learning components in resource-constrained environments.
—Muhammad Shafique, Technische Universität Wien

Machine learning (ML) algorithms, fueled by massive amounts of data, are increasingly being utilized in several domains, including healthcare, finance, and transportation. Models produced by ML algorithms, especially deep neural networks (DNNs), are being deployed in domains where trustworthiness is a big concern, such as automotive systems [1], finance [2], healthcare [3], and cyber security [4]. Of particular concern is the use of ML (including deep learning) in cyber–physical systems (CPSs) [5], such as autonomous vehicles, where the presence of an adversary can cause serious consequences. However, in designing and deploying these algorithms in critical CPSs, the presence of an active adversary is often ignored.

Adversarial ML (AML) [6] is a field concerned with the analysis of ML algorithms to adversarial attacks, and the use of such analysis in making ML algorithms robust to attacks. It is part of a broader agenda for safe and verified ML-based systems [7]. The major focus has been on test-time adversarial attacks, in which *adversarial examples*, inputs crafted by adding small, often imperceptible, perturbations to existing data, force a trained ML model to misclassify. In this article, we contend that the work on AML, while important and useful, is not enough. In particular, we advocate for the increased use of *semantics* in adversarial analysis and design of ML algorithms. *Semantic adversarial learning* explores a space of semantic modifications to the data, uses system-level semantic specifications in the analysis, utilizes semantic adversarial examples in training, and produces not just output labels but also additional semantic information. Focusing on deep learning, we explore these ideas and provide initial experimental data to support them. Although the focus of much of our article is on DNNs, the idea of semantic adversarial learning is applicable to a broad class of ML systems.

Semantic AML can be particularly relevant for developing robust ML models and ML-based systems in resource-constrained environments. A semantic approach can show that traditional (e.g., pixel-level) adversarial robustness is not necessary when those adversarial inputs do not lead to system-level failures.

Digital Object Identifier 10.1109/MDAT.2020.2968274
Date of publication: 20 January 2020; date of current version: 20 April 2020.

2168-2356/20©2020 IEEE Copublished by the IEEE CEDA, IEEE CASS, IEEE SSCS, and TTTC

By focusing efforts to make the ML model robust to only those adversarial inputs that are semantically meaningful and have system-level implications, a semantic adversarial approach makes more efficient use of resources that can be especially valuable in applications in embedded systems and Internet-of-Things (IoT) devices.

We begin in the "Background" section with some relevant background, and then present our proposal for semantic adversarial learning in the "Semantic adversarial analysis and training" section. Some directions for future work are sketched in the "Conclusion" section. An earlier version of this article appeared at CAV 2018 [8].

Background

Background on ML

We describe some general concepts in ML. We will consider the supervised learning setting. Consider a sample space Z of the form $X \times Y$ and an ordered training set $S = ((x_i, y_i))_{i=1}^{m}$ (x_i is the data and y_i is the corresponding label). Let H be a hypothesis space (e.g., weights corresponding to a logistic-regression model or a neural network model). There is a loss function $\ell: H \times Z \rightarrow \mathbb{R}$ so that given a hypothesis $w \in H$ and a sample $(x, y) \in Z$, we obtain a loss $\ell(w,(x, y))$. We consider the case where we want to minimize the loss over the training set S

$$L_S(w) = \frac{1}{m}\sum_{i=1}^{m} \ell(w,(x_i, y_i)) + \lambda\mathcal{R}(w).$$

In the above equation, $\lambda > 0$, and the term $\mathcal{R}(w)$ is called the *regularizer* and enforces "simplicity" in w. Since S is fixed, we sometimes denote $\ell_i(w) = \ell(w,(x_i, y_i))$ as a function only of w. We wish to find a w that minimizes the loss $L_S(w)$ or, in other words, we wish to solve the following optimization problem:

$$\min_{w\in H} L_S(w).$$

Example: We will consider the example of the logistic regression.

In this case, $X = \mathbb{R}^n$, $Y = \{+1, -1\}$, $H = \mathbb{R}^n$, and the loss function $\ell(w, (x, y))$ is as follows (represents the dot product of two vectors):

$$\log\left(1 + e^{-y(w^T \cdot x)}\right).$$

If we use the L_2 regularizer (i.e., $\mathcal{R}(w) = \|w\|_2$), then $L_S(w)$ becomes

$$\frac{1}{m}\sum_{i=1}^{m}\log\left(1 + e^{-y_i(w^T \cdot x_i)}\right) + \lambda\|w\|_2.$$

Classifiers: We focus on ML models that are *classifiers*, which are functions from $\mathbb{R}^n$ to C, where $\mathbb{R}$ denotes the set of reals and C is the set of class labels. To emphasize that a classifier depends on a hypothesis $w \in H$, which is the output of the learning algorithm described earlier, we will write it as F_w (if w is clear from the context, we will sometimes simply write F). For example, after training in the case of the logistic regression, we obtain a function from $\mathbb{R}^n$ to $\{-1, +1\}$. Vectors will be denoted in boldface, and the rth component of a vector $\mathbf{x}$ is denoted by $\mathbf{x}[r]$.

Throughout this article, we refer to the function $s(F_w)$ as the *softmax layer* corresponding to the classifier F_w. In the case of the logistic regression, $s(F_w)(\mathbf{x})$ is the following tuple (the first element is the probability of -1 and the second one is the probability of $+1$):

$$\left\langle \frac{1}{1 + e^{w^T \cdot \mathbf{x}}}, \frac{1}{1 + e^{-w^T \cdot \mathbf{x}}} \right\rangle.$$

Formally, let $c = |C|$ and F_w be a classifier. We let $s(F_w)$ be the function that maps $\mathbb{R}^n$ to $\mathbb{R}_+^c$ such that $\|s(F_w)(\mathbf{x})\|_1 = 1$ for any $\mathbf{x}$ [i.e., $s(F_w)$ computes a probability vector]. We denote $s(F_w)(\mathbf{x})[l]$ to be the probability of $s(F_w)(\mathbf{x})$ at label l. Recall that the softmax function from $\mathbb{R}^k$ to a probability distribution over $\{1, \ldots, k\} = [k]$ such that the probability of $j \in [k]$ for a vector $\mathbf{x} \in \mathbb{R}^k$ is

$$\frac{e^{\mathbf{x}[j]}}{\sum_{r=1}^{k} e^{\mathbf{x}[r]}}.$$

Some classifiers $F_w(\mathrm{x})$ are of the form argmax_l $s(F_w)(\mathbf{x})[l]$ (i.e., the classifier F_w outputs the label with the maximum probability according to the "softmax layer"). For example, in several DNN architectures, the last layer is the *softmax* layer. We are assuming that the reader is familiar with the basics of DNNs. For readers not familiar with DNNs, we can refer to the excellent book by Goodfellow et al. [9].

Background on logic

Temporal logics are commonly used for specifying desired and undesired properties of systems. For CPSs, it is common to use temporal logics that can specify properties of real-valued signals over real time, such as signal temporal logic (STL) [10] or metric temporal logic (MTL) [11].

A *signal* is a function $s : D \rightarrow S$, with $D \subseteq \mathbb{R}_{\geq 0}$ being an interval, and either $S \subseteq \mathbb{B}$ or $S \subseteq \mathbb{R}$, where $\mathbb{B} = \{\top, \bot\}$ and $\mathbb{R}$ is the set of reals. Signals defined on $\mathbb{B}$ are called *Booleans*, whereas those on $\mathbb{R}$ are said *real valued*. A *trace* $w = \{s_1, \ldots, s_n\}$ is a finite set of signals defined over the same interval D. We use variable x_i to denote the value of a real-valued signal at a particular time instant.

Let $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a finite set of predicates $\sigma_i : \mathbb{R}^n \rightarrow \mathbb{B}$, with $\sigma_i \equiv p_i(x_1, \ldots, x_n) \lhd 0$, $\lhd \in \{<, \leq\}$, and $p_i : \mathbb{R}^n \rightarrow \mathbb{R}$ a function in the variables $x_1, \ldots, x_n$. An STL formula is defined as follows:

$$\varphi := \sigma | \neg\varphi | \varphi \wedge \varphi | \varphi \, \mathrm{U}_I \, \varphi \tag{1}$$

where $\sigma \in \Sigma$ is a predicate and $I \subset \mathbb{R}_{\geq 0}$ is a closed nonsingular interval. Other common temporal operators can be defined as syntactic abbreviations in the usual way, like for instance $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\mathrm{F}_I \, \varphi := \top \, \mathrm{U}_I \, \varphi$, or $\mathrm{G}_I \, \varphi := \neg\mathrm{F}_I \, \neg\varphi$. Given $t \in \mathbb{R}_{\geq 0}$, a shifted interval I is defined as $t + I = \{t + t' \mid t' \in I\}$. Let w be a trace, $t \in \mathbb{R}_{\geq 0}$, and φ be an STL formula. The *qualitative (Boolean) semantics* of φ is inductively defined as follows:

$$\begin{aligned}
w, t &\models \sigma \ \text{ iff } \sigma(w(t)) \text{ is true} \\
w, t &\models \neg\varphi \ \text{ iff } w, t \not\models \varphi \\
w, t &\models \varphi_1 \wedge \varphi_2 \ \text{ iff } w, t \models \varphi_1 \text{ and } w, t \models \varphi_2 \\
w, t &\models \varphi_1 \mathrm{U}_I \varphi_2 \ \text{ iff } \exists t' \in t + I \text{ s.t. } w, t' \models \varphi_2 \text{ and} \\
&\qquad \forall t'' \in [t, t'], w, t'' \models \varphi_1.
\end{aligned} \tag{2}$$

A trace w satisfies a formula φ if and only if $w, 0 \models \varphi$, in short $w \models \varphi$. STL also admits a quantitative or robust semantics, which we omit for brevity. This provides quantitative information on the formula, telling how strongly the specification is satisfied or violated for a given trace.

Adversarial robustness

The field of AML has grown rapidly in recent years, and a full survey is beyond the scope of this article; we refer the reader to other papers on this topic [6], [8]. Instead, in this section, we present a general formulation of adversarial robustness [12] to test-time attacks that captures all the formulations in the literature that we are aware of.

In such adversarial attacks, the adversary starts with a given example $x \in X$ and perturbs it so as to produce "wrong" output. Formally, let $\tilde{X} \subseteq X$ be a set of allowed perturbed inputs, $\mu : X \times X \rightarrow \mathbb{R}_{\geq 0}$ be a quantitative function (such as a distance, risk, or divergence function), $D : (X \times X) \times \mathbb{R} \rightarrow \mathbb{B}$ be a constraint defined over μ, $A : X \times X \times \mathbb{R} \rightarrow \mathbb{B}$ be a target behavior constraint, and $\alpha, \beta \in \mathbb{R}$ be the parameters. Then, the problem of finding a set of inputs that falsifies the ML model can be cast as a decision problem as follows.

Definition 1: Given $x \in \mathrm{X}$, find $x^* \in \mathrm{X}$ such that the following constraints hold:

1) admissibility constraint: $x^* \in \tilde{X}$;
2) distance constraint: $D(\mu(x, x^*), \alpha)$;
3) target behavior constraint: $A(x, x^*, \beta)$.

The admissibility constraint (1) ensures that the adversarial input x^* belongs to the space of admissible perturbed inputs. The distance constraint (2) constrains x^* to be no more distant from x than α. Finally, the target behavior constraint (3) captures the target behavior of the adversary as a predicate $A(x, x^*, \beta)$ which is true iff the adversary changes the behavior of the ML model by at least β modifying x to x^*. If the three constraints hold, then we say that the ML model has failed for input x. We note that this is a so-called "local" robustness property for a specific input x, as opposed to other notions of "global" robustness to changes to a population of inputs [8], [13].

Typically, the problem of finding an adversarial example x^* for a model f at a given input $x \in X$, as formulated above, is formulated as an optimization problem in one of two ways.

- *Minimizing perturbation*: Find the closest x^* that alters f's prediction. This can be encoded in constraint (2) as $\mu(x, x^*) \leq \alpha$.
- *Maximizing the loss*: Find x^* that maximizes false classification. This can be encoded in the constraint (3) as $L(f(x), f(x^*)) \geq \beta$.

Definition 2: The optimization version of Definition 1 is to find an input x^* such that either $x^* = \mathrm{argmin}_{x^* \in \mathrm{X}} \, \alpha$ or $x^* = \mathrm{argmax}_{x^* \in \mathrm{X}} \, \beta$, subject to the constraints in Definition 1.

We refer the reader to [12] for a description of how the variants of adversarial robustness published in the literature can all be captured by the definitions above.

Semantic adversarial analysis and training

A central tenet of this article is that the analysis of DNNs (and ML components, in general) must be more *semantic*. In particular, we advocate for the increased use of semantics in several aspects of adversarial analysis and training, including the following.

- *Semantic modification space*: Recall that the goal of adversarial attacks is to modify an input vector $\mathbf{x}$ with an adversarial modification δ so as to achieve a target misclassification. Such modifications typically do not incorporate the application-level semantics or the context within which the neural network is deployed. We argue that it is essential to incorporate more application-level, contextual semantics into the modification space. Such *semantic modifications* correspond to modifications that may arise more naturally within the context of the target application. For example, for a DNN used for perception in an autonomous vehicle, the semantic space would be the 3-D scene around the vehicle, including the location and characteristics of vehicles and other agents. We view this approach not as ignoring arbitrary modifications (which are indeed worth considering with a security mind set), but as prioritizing the design and analysis of DNNs toward semantic adversarial modifications. The "Compositional falsification" section discusses this point in more detail.

- *System-level specifications*: The goal of much of the work in adversarial attacks has been to generate misclassifications. However, not all misclassifications are made equal. We contend that it is important to find misclassifications that lead to violations of desired properties of the system within which the DNN is used. Therefore, one must identify such *system-level specifications* and devise analysis methods to verify whether an erroneous behavior of the DNN component can lead to the violation of a system-level specification. System-level counterexamples can be valuable aids to repair and re-design ML models. See the "Compositional falsification" section for a more detailed discussion of this point.
- *Semantic loss functions for training*: Most ML models are trained with the main goal of reducing misclassifications as measured by a suitably crafted loss function. We contend that it is also important to train the model to avoid undesirable behaviors at the system level. For this, we advocate using methods for *semantic training*, where a *semantic loss function* is used that incorporates semantic properties including system-level specifications and confidence levels in the training process. The "Semantic training" section explores a few ideas along these lines.
- *Semantic data set augmentation*: An important way to (re)design ML models is to augment the data set with carefully generated or selected data so as to improve the dependability of the model without losing much accuracy on the original data set. We advocate for the use of data generated via semantic adversarial analysis for such augmentation. In particular, *counterexample-guided data augmentation*, in which counterexamples generated via semantic adversarial analysis are utilized for training and testing,shows a great deal of promise in improving ML models. We present some ideas and results in the"Semantic training"section.

Compositional falsification

We discuss the problem of performing the system-level analysis of a deep learning component, using recent work by Dreossi et al. [14], [15] to illustrate the main points. The material in this section is mainly based on [16].

We begin with some basic notation. Let S denote the model of the full system under verification, E denote a model of its environment, and Φ denote the specification to be verified. C is an ML model (e.g., DNN) that is part of S. Let $\mathbf{x}$ be an input to C. We assume that Φ is a trace property—a set of behaviors of the closed system obtained by composing S with E, denoted as $S\|E$. The goal of falsification is to find one or more counterexamples showing how the composite system $S\|E$ violates Φ. In this context, the semantic analysis of C is about finding a modification δ from a space of semantic modifications Δ such that C, on $\mathbf{x} + \delta$, produces a misclassification that causes $S\|E$ to violate Φ.

1) Example problem: As an illustrative example, consider a simple model of an automatic emergency braking system (AEBS) that attempts to detect objects in front of a vehicle and actuate the brakes when needed to avert a collision. Figure 1 shows the AEBS as a system composed of a controller (automatic braking), a plant (vehicle subsystem under control, including

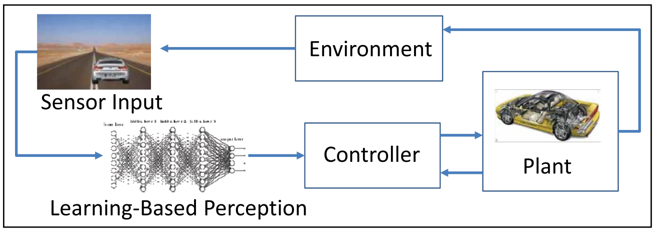


Figure 1. AEBS in closed loop. An image classifier based on DNNs is used to perceive objects in the ego vehicle's frame of view.

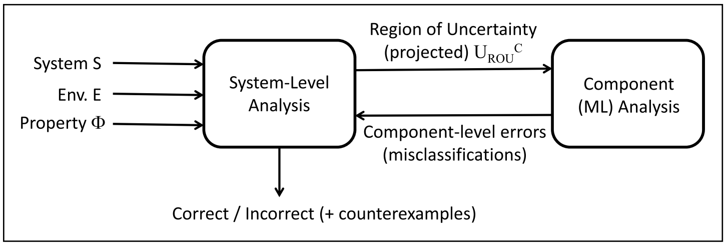

System S
Env. E
Property Φ
System-Level Analysis
Region of Uncertainty (projected) U_{ROU}^{C}
Component (ML) Analysis
Component-level errors (misclassifications)
Correct / Incorrect (+ counterexamples)

Figure 2. Compositional verification approach. A system-level verifier cooperates with a component-level analysis procedure (e.g., adversarial analysis of a ML component to find misclassifications).

transmission), and an advanced sensor (camera along with an obstacle detector based on deep learning). The AEBS, when combined with the vehicle's environment, forms a closed-loop control system. The controller regulates the acceleration and braking of the plant using the velocity of the subject (ego) vehicle and the distance between it and an obstacle. The sensor used to detect the obstacle includes a camera along with an image classifier based on DNNs. In general, this sensor can provide noisy measurements due to incorrect image classifications, which in turn can affect the correctness of the overall system.

Suppose we want to verify whether the distance between the ego vehicle and a preceding obstacle is always larger than 2 m. In STL, this requirement Φ can be written as $\mathrm{G}_{0,T}(\|\mathbf{x}_{\mathrm{ego}} - \mathbf{x}_{\mathrm{obs}}\|_2 \geq 2)$. Such verification requires the exploration of a very large input space comprising of the control inputs (e.g., acceleration and braking pedal angles) and the ML component's feature space (e.g., all the possible pictures observable by the camera). The latter space is particularly large—for example, note that the feature space of RGB images of dimension 1000 × 600 pixels (for an image classifier) contains $256^{1000 \times 600 \times 3}$ elements.

This case study has been implemented in MATLAB /Simulink[1] in two versions that use two different convolutional neural networks (CNNs): the Caffe [17] version of AlexNet [18] and the Inception-v3 model created with Tensorflow [19], both trained on the ImageNet database [20]. Further details about this example can be obtained from [14].

2) Approach: A key idea in our approach is to have a *system-level verifier* that abstracts away the component C while verifying Φ on the resulting abstraction. This system-level verifier communicates with a component-level analyzer that searches for semantic modifications δ to the input $\mathbf{x}$ of C that could lead to violations of the system-level specification Φ. Figure 2 illustrates this approach.

We formalize this approach while trying to emphasize the intuition. Let T denote the set of all possible traces of the composition of the system with its environment, $S \| E$. Given a specification Φ, let T_Φ denote the set of traces in T satisfying Φ. Let U_Φ denote the projection of these traces onto the state and interface variables of the environment E. U_Φ is termed as the *validity domain* of Φ, i.e., the set of environment behaviors for which Φ is satisfied. Similarly, the complement set $U_{\neg\Phi}$ is the set of environment behaviors for which Φ is violated.

Our approach works as follows:

1) The system-level verifier initially performs two analyses with two extreme abstractions of the ML component. First, it performs an *optimistic* analysis, wherein the ML component is assumed to be a "perfect classifier," i.e., all feature vectors are correctly classified. In situations where ML is used for perception/sensing, this abstraction assumes perfect perception/sensing. Using this abstraction, we compute the validity domain for this abstract model of the system, denoted as U_Φ^+. Next, it performs a *pessimistic* analysis where the ML component is abstracted by a "completely-wrong classifier," i.e., all feature vectors are misclassified. Denote the resulting validity domain as U_Φ^-. It is expected that $U_\Phi^+ \supseteq U_\Phi^-$.

   Abstraction permits the system-level verifier to operate on a lower-dimensional search space and identify a region in this space that may be affected by the malfunctioning of component C—a so-called "region of uncertainty" (ROU). This region, U_{ROU}^C is computed as $U_\Phi^+ \setminus U_\Phi^-$.

   In other words, it comprises all environment behaviors that could lead to a system-level failure when component C malfunctions. This region U_{ROU}^C, projected onto the inputs of C, is communicated to the ML analyzer. (Concretely, in the context of our example of the "Example problem" section, this corresponds to finding a subspace of images that corresponds to U_{ROU}^C.)
2) The component-level analyzer, also termed as an ML analyzer, performs a detailed analysis of

[1] https://github.com/dreossi/analyzeNN

the projected ROU U^C_{ROU}. A key aspect of the ML analyzer is to explore the *semantic modification space* efficiently. Several options are available for such an analysis, including the various adversarial analysis techniques surveyed earlier (applied to the semantic space), as well as systematic sampling methods [14]. Even though a component-level formal specification may not be available, each of these adversarial analyses has an implicit notion of "misclassification." We will refer to these as *component-level errors*. The working of the ML analyzer from [14] is shown in Figure 3.

3) When the component-level (ML) analyzer finds component-level errors (e.g., those that trigger misclassifications of inputs whose labels are easily inferred), it communicates that information back to the system-level verifier, which checks whether the ML misclassification can lead to a violation of the system-level property Φ. If yes, we have found a system-level counterexample. If no, component-level errors are found, and the system-level verification can prove the absence of counterexamples, then it can conclude that Φ is satisfied. Otherwise, if the ML misclassification cannot be extended to a system-level counterexample, the ROU is updated and the revised ROU passed back to the component-level analyzer.

The communication between the system-level verifier and the component-level (ML) analyzer thus continues until we either prove or disprove Φ, or we run out of resources.

3) Sample results: We have applied the above approach to the problem of *compositional falsification* of CPSs with ML components [14]. For this class of CPS, including those with highly nonlinear dynamics and even black-box components, simulation-based falsification of temporal logic properties is an approach that has proven effective

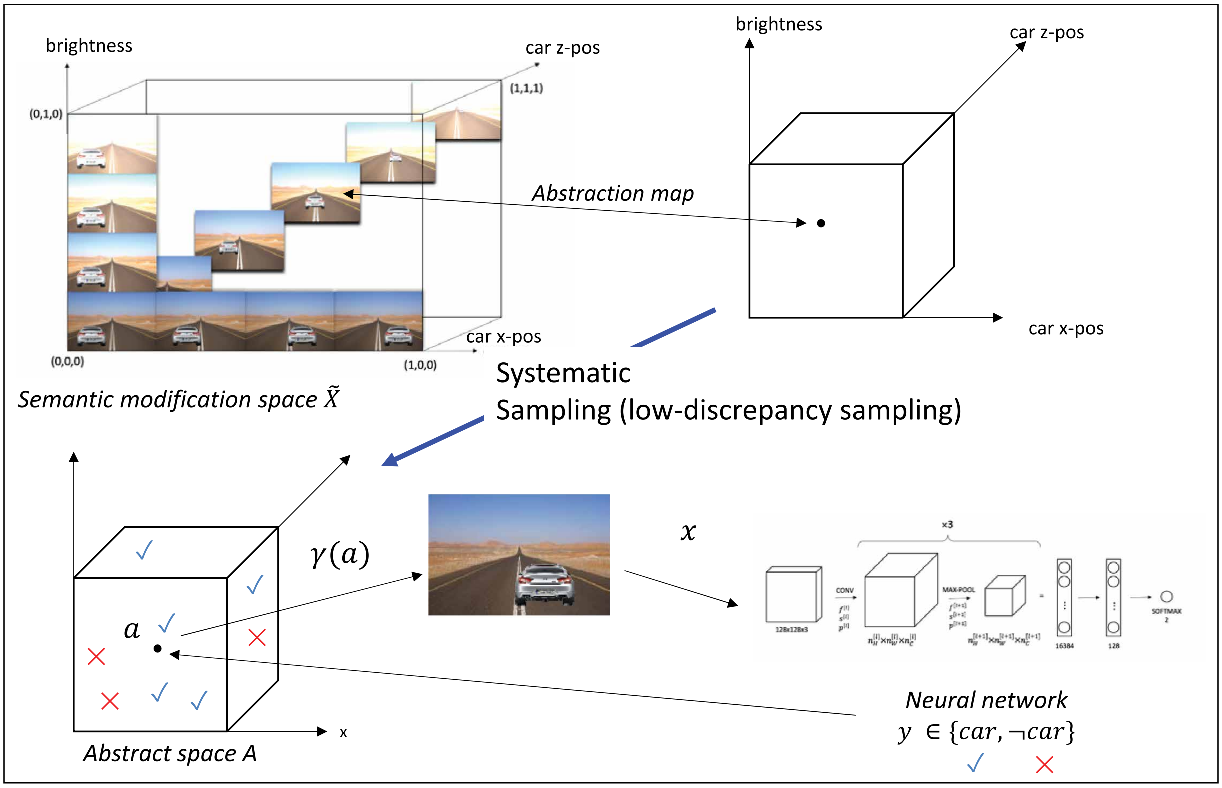


Figure 3. ML analyzer: searching the semantic modification space. A concrete semantic modification space (top left) is mapped into a discrete abstract space. Systematic sampling, using low-discrepancy methods, yields points in the abstract space. These points are concretized and the NN is evaluated on them to ascertain if they are correctly or wrongly classified. The misclassifications are fed back for system-level analysis.

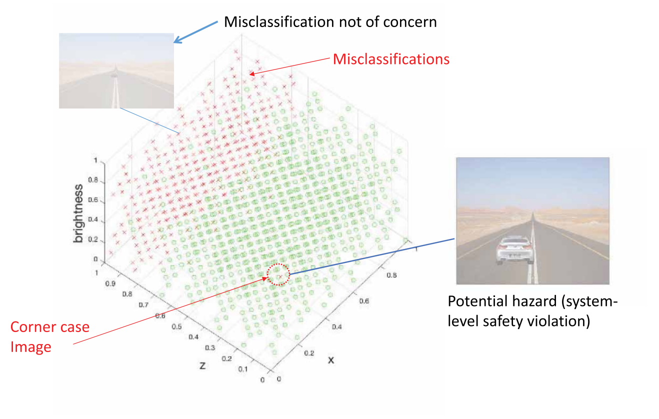


Figure 4. Misclassified images for Inception-v3 neural network (trained on ImageNet with TensorFlow). Red crosses are misclassified images and green circles are correctly classified. Our system-level analysis finds a corner-case image that could lead to a system-level safety violation.

in industrial practice [21], [22]. We present here a sample of results on the AEBS example from [14], referring the reader to more detailed descriptions in the other papers on the topic [14], [15].

In Figure 4, we show one result of our analysis for the Inception-v3 DNN. This figure shows both correctly classified and misclassified images on a range of synthesized images where: 1) the environment vehicle is moved away from or toward the ego vehicle (along the *z*-axis); 2) it is moved sideways along the road (along the *x*-axis); or 3) the brightness of the image is modified. These modifications constitute the three axes of the figure. Our approach finds misclassifications that do not lead to system-level property violations and also misclassifications that do lead to such violations. For example, Figure 4 shows two misclassified images: one with an environment vehicle that is too far away to be a safety hazard, and another image showing an environment vehicle driving slightly on the wrong side of the road, which is close enough to potentially cause a violation of the system-level safety property (of maintaining a safe distance from the ego vehicle).

For further details about this and other results with our approach, we refer the reader to [14] and [15].

Semantic training

In this section, we discuss two ideas for *semantic training* and *retraining* of DNNs. We first discuss the use of *hinge loss* as a way of incorporating confidence levels into the training process. Next, we discuss how system-level counterexamples and associated misclassifications can be used in the retraining process to both improve the accuracy of ML models and to gain more assurance in the overall system containing the ML component. A more detailed study of using misclassifications (ML component-level counterexamples) to improve the accuracy of the neural network, termed *counterexample-guided data augmentation*, is presented in [23].

1) Experimental setup: As in the preceding section, we consider an AEBS using a DNN-based object detector. However, in these experiments, we use an AEBS deployed within Udacity's self-driving car simulator, as reported in our previous work [15].[2] We modified the Udacity simulator to focus exclusively on braking. In our case studies, the car follows some predefined way points, while accelerating and braking are controlled by the AEBS connected to a CNN. In particular, whenever the CNN detects an obstacle in the images provided by the onboard camera, the AEBS triggers a braking action that slows the vehicle down and avoids the collision against the obstacle.

We designed and implemented a CNN to predict the presence of a cow on the road. Given an image taken by the onboard camera, the CNN classifies the picture in either "cow" or "not cow" category. The CNN architecture is shown in Figure 5. It consists of eight layers: the first six are alternations of convolutions and max-pools with rectified linear unit (ReLU) activations, the last two are a fully connected layer and a softmax that outputs the network prediction (confidence level for each label).

We generated a data set of 1,000 road images with and without cows. We split the data set into 80% training and 20% validation data. Our model was implemented and trained using the Tensorflow library with crossentropy cost function and the Adam algorithm optimizer (learning rate 10^{-4}). The model reached 95% accuracy on the test set. Finally, the resulting CNN is connected to the Unity simulator via the Socket.IO protocol.[3] Figure 6 depicts a

[2] Udacity's self-driving car simulator: https://github.com/udacity/self-driving-car-sim
[3] Socket.IO protocol: https://github.com/socketio

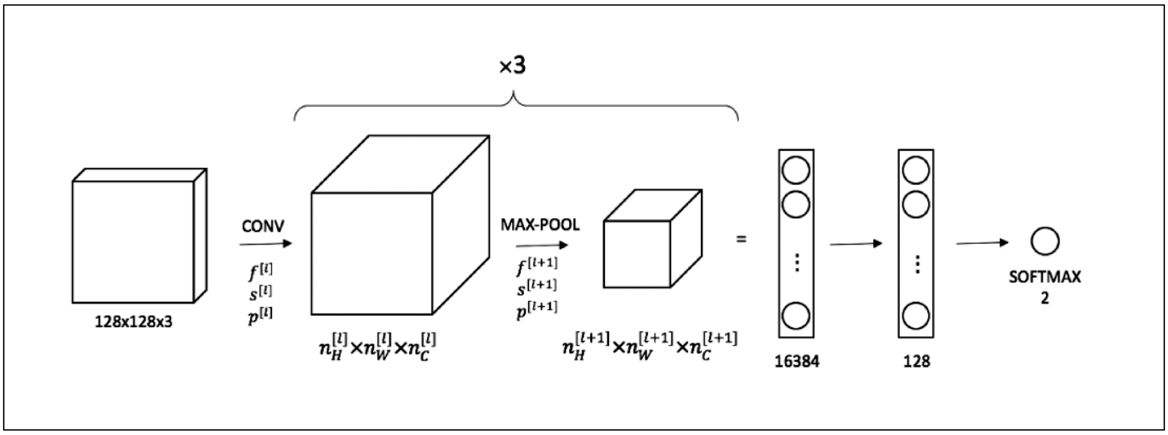


Figure 5. CNN architecture.

screenshot of the simulator with the AEBS in action in proximity of a cow.

2) Hinge loss: In this section, we investigate the relationship between multiclass hinge loss functions and adversarial examples. *Hinge loss* is defined as follows:

$$l(\hat{y}) = \max(0, k + \max_{i \neq l}(\hat{y}_i) - \hat{y}_l) \qquad (3)$$

where (x, y) is a training sample, $\hat{y} = F(x)$ is a prediction, and l is the *ground truth* label of x. For this section, the output $\hat{y}$ is a numerical value that indicates the *confidence level* of the network for each class. For example, $\hat{y}$ can be the output of a softmax layer as described in the "Background" section.

Consider what happens when we vary k. Suppose there is an $i \neq l$ s.t. $\hat{y}_i > \hat{y}_l$. Pick the largest such i, call it i^*. For $k = 0$, we will incur a loss of $\hat{y}_i^* - \hat{y}_l$ for the example (x, y). However, as we make k more negative, we increase the tolerance for "misclassifications" produced by the DNN F. Specifically, we incur no penalty for a misclassification as long as the associated confidence level deviates from that of the ground truth label by no more than $|k|$. The larger the absolute value of k, the greater the tolerance. Intuitively, this biases the training process toward avoiding "high confidence misclassifications."

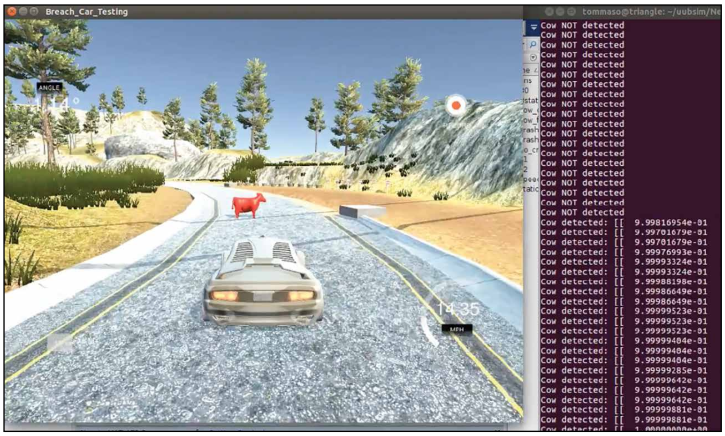
Table 1. Hinge loss with different *k* values.

	$T_{original}$		$T_{countex}$	
k	acc	log-loss	acc	log-loss
0	0.69	0.68	0.11	0.70
- 0.01	0.77	0.69	0.00	0.70
-0.05	0.52	0.70	0.67	0.69
-0.1	0.50	0.70	0.89	0.68
-0.25	0.51	0.70	0.77	0.68

In this experiment, we investigate the role of k and explore different parameter values. At training time, we want to minimize the mean hinge loss across all training samples. We trained the CNN described above with different values of k and evaluated its precision on both the original test set and a set of counterexamples generated for the original model, i.e., the network trained with crossentropy loss.

Table 1 reports accuracy and log loss for different values of k on both original and counterexamples test sets (T_{original} and T_{countex}, respectively).

Table 1 shows interesting results. We note that a negative k increases the accuracy of the model on counterexamples. In other words, biasing the training process by penalizing high-confidence misclassifications improves accuracy on counterexamples! However, the price to pay is a reduction of accuracy on the original test set. This is still a



```
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow NOT detected
Cow detected: [[  9.99816954e-01
Cow detected: [[  9.99701679e-01
Cow detected: [[  9.99701679e-01
Cow detected: [[  9.99976993e-01
Cow detected: [[  9.99993324e-01
Cow detected: [[  9.99993324e-01
Cow detected: [[  9.99988198e-01
Cow detected: [[  9.99986649e-01
Cow detected: [[  9.99986649e-01
Cow detected: [[  9.99999523e-01
Cow detected: [[  9.99999523e-01
Cow detected: [[  9.99999523e-01
Cow detected: [[  9.99999404e-01
Cow detected: [[  9.99999404e-01
Cow detected: [[  9.99999404e-01
Cow detected: [[  9.99999285e-01
Cow detected: [[  9.99999642e-01
Cow detected: [[  9.99999642e-01
Cow detected: [[  9.99999642e-01
Cow detected: [[  9.99999881e-01
Cow detected: [[  9.99999881e-01
Cow detected: [[  1.00000000e+00
```

Figure 6. Udacity simulator with a CNN-based AEBS in action.

preliminary result and further experimentation and analysis is necessary.

3) System-level counterexamples: *Counterexample-guided data augmentation* [23] is a technique for augmenting an existing data set with carefully chosen semantic adversarial examples that produce incorrect output at the ML component level. In this work, we show that a combination of semantic modifications and analysis of the generated counterexamples can improve the accuracy of a state-of-the-art DNN for object detection and classification in autonomous vehicles by around 10% over the original accuracy and in comparison with other state-of-the-art augmentation methods. The question we consider in this article is: Can we use such an approach at the system level to completely eliminate counterexamples?

By using the composition falsification framework presented in the "Compositional falsification" section, we identify orientations, displacements on the x-axis, and color of an obstacle that leads to a collision of the vehicle with the obstacle. Figure 7 depicts configurations of the obstacle that lead to specification violations, and hence, to collisions.

In an experiment, we augment the original training set with the elements of T_{countex}, i.e., images of the original test set T_{original} that are misclassified by the original model (see the "Hinge loss" section).

We trained the model with both crossentropy and hinge loss for 20 epochs. Both models achieve a high accuracy on the validation set (≈92%). However, when plugged into the AEBS, neither of these models prevents the vehicle from colliding against the obstacle with an adversarial configuration. This seems to indicate that simply retraining with some semantic (system-level) counterexamples generated by analyzing the system containing the ML model may not be sufficient to eliminate all semantic counterexamples.

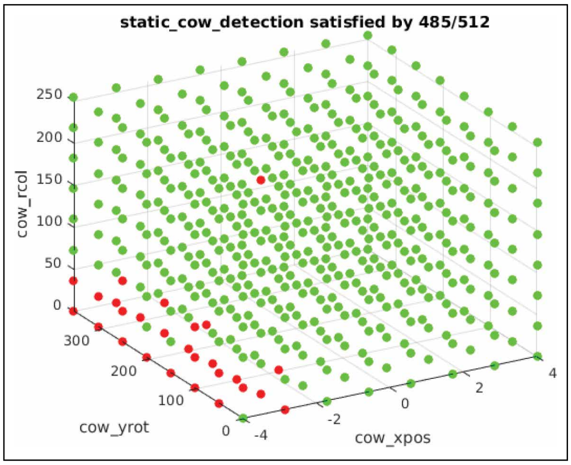


Figure 7. Semantic counterexamples: obstacle configurations leading to property violations (in red).

Interestingly, though, it appears that in both cases the impact of the vehicle with the obstacle happens at a slower speed than the one with the original model. In other words, the AEBS system starts detecting the obstacle earlier than the original model, and therefore starts braking earlier as well. This means that despite the specification violations, the counterexample retraining procedure seems to help with limiting the damage in case of a collision. Coupled with a run-time assurance framework [24], semantic retraining could help mitigate the impact of misclassifications on the system-level behavior.

This article introduced the idea of *semantic adversarial machine (deep) learning*, where adversarial analysis and training of ML models are performed using the semantics and context of the overall system within which the ML models are utilized. We identified several ideas for integrating semantics into adversarial learning, including using a semantic modification space, system-level formal specifications, and semantic training using counterexamples and more semantic loss functions. Initial results not only show the promise of these ideas, but also indicate that much remains to be done. We outline below some of the interesting directions for further research; see [7] for more details.

Programmatic modeling of the semantic feature space: High-dimensional semantic feature spaces require more structured representations. A promising approach is to design domain-specific programming languages to represent the semantic feature space in a way that is easy to understand, modify, and use to guide semantic adversarial learning. In particular, probabilistic programming languages such as Scenic [25] provide this capability while also permitting a way to represent distributional assumptions and enable tasks such as data generation, inference, and verification.

Efficient algorithms to search semantic space: In addition to devising suitable representations of the semantic space, we need efficient algorithms to search the resulting high-dimensional space. The VerifAI toolkit [26] is an initial step to develop such algorithmic methods for the design and analysis of artificial intelligence (AI)/ML-based systems. Another promising direction is the combination of "standard" AML methods with differentiable renderers/simulators [27].

Formal specification for ML and deep learning: An important direction is to develop formalisms to capture properties of the ML model and the ML-based system that enable semantic adversarial analysis. Although some initial progress has been made in this regard [13], [12], much more remains to be done.

Exploring tradeoffs between semantic robustness and resource-efficient implementation: As discussed in the "Introduction" section, a semantic approach can help make an ML model robust in a resource-efficient manner. In fact, evidence from past work on error-resilient system design [28] suggests that using semantic/system-level specifications can enable targeting scarce resources to exactly those components that need to be made robust. This offers a fruitful direction for further research on robust ML implementations.

In summary, the field of semantic adversarial learning promises to be a rich domain for research at the intersection of ML, formal methods, design automation, programming languages, and related areas. ■

Acknowledgments

We thank the anonymous reviewers for their feedback. The work of Sanjit A. Seshia and Tommaso Dreossi was supported in part by NSF under Grant 1545126 (VeHICaL), Grant 1646208, and Grant 1837132; in part by the DARPA BRASS Program under Agreement FA8750-16-C0043; in part by the DARPA Assured Autonomy Program; in part by the iCyPhy Center; and in part by Berkeley Deep Drive. The contributions of Tommaso Dreossi to this article were made while he was affiliated with the University of California at Berkeley (UC Berkeley).

References

[1] NVIDIA, "NVIDIA tegra drive PX: Self-driving car computer," 2015. Accessed: Nov. 2019. [Online]. Available: http://www.nvidia.com/object/drive-px.html

[2] E. Knorr, "How PayPal beats the bad guys with machine learning," 2015. Accessed: Nov. 2019. [Online]. Available: http://www.infoworld.com/article/2907877/machine-learning/howpaypal-reduces-fraud-with-machine-learning.html

[3] B. Alipanahi et al., "Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning," *Nature Biotechnol.*, vol. 33, no. 8, pp. 831–838, 2015.

[4] G. E. Dahl et al., "Large-scale malware classification using random projections and neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP)*, 2013, pp. 3422–3426.

[5] E. A. Lee and S. A. Seshia, *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*, 2nd ed. Cambridge, MA: MIT Press, 2016.

[6] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Commun. ACM*, vol. 61, no. 7, pp. 56–66, 2018.

[7] S. A. Seshia, D. Sadigh, and S. S. Sastry, "Towards verified artificial intelligence," Jul. 2016, *arXiv:606.08514*.

[8] T. Dreossi, S. Jha, and S. A. Seshia, "Semantic adversarial deep learning," in *Proc. 30th Int. Conf. Comput.-Aided Verification (CAV)*, 2018, pp. 3–26.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. Accessed: Nov. 2019. [Online]. Available: http://www.deeplearningbook.org

[10] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. Formal Modeling Anal. Timed Syst. (FORMATS)*, 2004, pp. 152–166. Accessed: Nov. 2019. [Online]. Available: https://doi.org/10.1007/978-3-540-30206-3_12

[11] R. Koymans, "Specifying real-time properties with metric temporal logic," *Real-Time Syst.*, vol. 2, no. 4, pp. 255–299, 1990.

[12] T. Dreossi et al., "A formalization of robustness for deep neural networks," in *Proc. AAAI Spring Symp. Workshop Verification Neural Netw. (VNN)*, Mar. 2019. Accessed: Nov. 2019. [Online]. Available: https://arxiv.org/abs/1903.10033

[13] S. A. Seshia et al., "Formal specification for deep neural networks," in *Proc. Int. Symp. Automated Technol. Verification Anal. (ATVA)*, Oct. 2018, pp. 20–34.

[14] T. Dreossi, A. Donze, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," in *Proc. NASA Formal Methods Conf. (NFM)*, May 2017, pp. 357–372.

[15] T. Dreossi, A. Donze, and S. A. Seshia, "Compositional falsification of cyber-physical systems with machine learning components," *J. Autom. Reason.*, vol. 63, no. 4, pp. 1031–1053, 2019.

[16] S. A. Seshia, "Compositional verification without compositional specification for learning-based systems," EECS Dept., Univ. California, Berkeley, Tech. Rep. UCB/EECS-2017-164, Nov. 2017. Accessed: Nov. 2019. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-164.html

[17] Y. Jia et al., “Caffe: Convolutional architecture for fast feature embedding,” in *Proc. ACM Multimedia Conf. (ACMMM)*, 2014, pp. 675–678.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in *Proc. Advances Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[19] M. Abadi et al., “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Accessed: Nov. 2019. [Online]. Available: http://tensorflow.org/

[20] “Imagenet.” Accessed: Nov. 2019. [Online]. Available: http://image-net.org/

[21] X. Jin et al., “Mining requirements from closed-loop control models,” *IEEE Trans. Comput.-Aided Design Circuits Syst.*, vol. 34, no. 11, pp. 1704–1717, 2015.

[22] T. Yamaguchi et al., “Combining requirement mining, software model checking, and simulation-based verification for industrial automotive systems,” in *Proc. IEEE Int. Conf. Formal Methods Comput.-Aided Design (FMCAD)*, Oct. 2016, pp. 201–204.

[23] T. Dreossi et al., “Counterexample-guided data augmentation,” in *Proc. Int. Joint Conf. Artifi. Intell. (IJCAI)*, Jul. 2018, pp. 2071–2078.

[24] A. Desai et al., “SOTER: A Runtime Assurance Framework for Programming Safe Robotics Systems,” in *Proc. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 138–150.

[25] D. J. Fremont et al., “Scenic: A language for scenario specification and scene generation,” in *Proc. 40th Annu. ACM SIGPLAN Conf. Program. Lang. Design Implementation (PLDI)*, Jun. 2019, pp. 63–78.

[26] T. Dreossi et al., “VERIFAI: A toolkit for the formal design and analysis of artificial intelligence-based systems,” in *Proc. 31st Int. Conf. Comput.-Aided Verification (CAV)*, Jul. 2019, pp. 432–442.

[27] L. Jain et al., “Generating semantic adversarial examples with differentiable rendering,” *CoRR*, 2019, *arXiv:1910.00727.*

[28] S. A. Seshia, W. Li, and S. Mitra, “Verification-guided soft error resilience,” in *Proc. Conf. Design Autom. Test Eur. (DATE)*, ACM Press, Apr. 2007, pp. 1442–1447.

Sanjit A. Seshia is currently a Professor at the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA. His research interests are in formal methods for dependable and secure computing, with a current focus on the areas of cyber–physical systems, computer security, machine learning, and robotics. He is a Fellow of the IEEE.

Somesh Jha is currently a Lubar Professor at the Computer Sciences Department, University of Wisconsin—Madison, Madison, WI. His work focuses on analysis of security protocols, survivability analysis, intrusion detection, formal methods for security, and analyzing malicious code. Recently, he has also worked on privacy-preserving protocols and adversarial machine learning (ML). He is a Fellow of the ACM and IEEE.

Tommaso Dreossi is currently an Applied Scientist at Amazon Search, Palo Alto, CA. His research interests include formal verification of cyber-physical systems, computer vision, and natural language processing. Dreossi has a PhD in computer science from the University of Grenoble, Grenoble, France, and the University of Udine, Udine, Italy (2016).

■ Direct questions and comments about this article to Sanjit A. Seshia, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720-1770 USA; sseshia@eecs.berkeley.edu.