Semantic Adversarial Deep Learning

Tommaso Dreossi[1], Somesh Jha[2], and Sanjit A. Seshia[1]

University of California at Berkeley, Berkeley[1]
University of Wisconsin, Madison[2]
dreossi@berkeley.edu, jha@cs.wisc.edu, sseshia@berkeley.edu

Abstract. Fueled by massive amounts of data, models produced by machine-learning (ML) algorithms, especially deep neural networks, are being used in diverse domains where trustworthiness is a concern, including automotive systems, finance, health care, natural language processing, and malware detection. Of particular concern is the use of ML algorithms in cyber-physical systems (CPS), such as self-driving cars and aviation, where an adversary can cause serious consequences.
However, existing approaches to generating adversarial examples and devising robust ML algorithms mostly ignore the *semantics* and *context* of the overall system containing the ML component. For example, in an autonomous vehicle using deep learning for perception, not every adversarial example for the neural network might lead to a harmful consequence. Moreover, one may want to prioritize the search for adversarial examples towards those that significantly modify the desired semantics of the overall system. Along the same lines, existing algorithms for constructing robust ML algorithms ignore the specification of the overall system. In this paper, we argue that the semantics and specification of the overall system has a crucial role to play in this line of research. We present preliminary research results that support this claim.

1 Introduction

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 [36], finance [26], health care [2], computer vision [29], speech recognition [18], natural language processing [39], and cyber-security [9,43]. Of particular concern is the use of ML (including deep learning) in *cyber-physical systems* (CPS) [30], where the presence of an adversary can cause serious consequences. For example, much of the technology behind autonomous and driver-less vehicle development is "powered" by machine learning [5,15,4]. DNNs have also been used in airborne collision avoidance systems for unmanned aircraft (ACAS Xu) [23]. However, *in designing and deploying these algorithms in critical cyber-physical systems, the presence of an active adversary is often ignored.*

Adversarial machine learning (AML) 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 the broader agenda for safe and verified ML-based systems [40,42]. In this paper, we first give a brief survey of the field of AML, with a

particular focus on deep learning. We focus mainly on attacks on outputs or models that are produced by ML algorithms that occur *after training* or "external attacks", which are especially relevant to cyber-physical systems (e.g., for a driverless car the ML algorithm used for navigation has been already trained by the manufacturer once the "car is on the road"). These attacks are more realistic and are distinct from other type of attacks on ML models, such as attacks that poison the training data (see the paper [19] for a survey of such attacks). We survey attacks caused by *adversarial examples*, which are inputs crafted by adding small, often imperceptible, perturbations to force a trained ML model to misclassify.

We contend that the work on adversarial ML, 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.

Roadmap. Section 2 provides the relevant background. A brief survey of adversarial analysis is given in Section 3. Our proposal for semantic adversarial learning is given in Section 4.

2 Background

Background on Machine Learning Next we describe some general concepts in machine learning (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). There is a loss function $\ell : H \times Z \mapsto \mathbb{R}$ 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 equation given above, $\lambda > 0$ 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 $L_s(w)$ or we wish to solve the following optimization problem:

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

Example: We will consider the example of 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 ($\cdot$ 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:

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

Stochastic Gradient Descent. *Stochastic Gradient Descent (SGD)* is a popular method for solving optimization tasks (such as the optimization problem $\min_{w\in H} L_S(w)$ we considered before). In a nutshell, SGD performs a series of updates where each update is a gradient descent update with respect to a small set of points sampled from the training set. Specifically, suppose that we perform SGD T times. There are two typical forms of SGD: in the first form, which we call Sample-SGD, we uniformly and randomly sample $i_t \sim [m]$ at time t, and perform a gradient descent based on the i_t-th sample (x_{i_t}, y_{i_t}):

$$w_{t+1} = G_{\ell_t,\eta_t}(w_t) = w_t - \eta_t \ell'_{i_t}(w_t) \tag{1}$$

where w_t is the hypothesis at time t, η_t is a parameter called the *learning rate*, and $\ell'_{i_t}(w_t)$ denotes the derivative of $\ell_{i_t}(w)$ evaluated at w_t. We will denote G_{ℓ_t,η_t} as G_t. In the second form, which we call Perm-SGD, we first perform a random permutation of S, and then apply Equation 1 T times by cycling through S according to the order of the permutation. The process of SGD can be summarized as a diagram:

$$w_0 \xrightarrow{G_1} w_1 \xrightarrow{G_2} \cdots \xrightarrow{G_t} w_t \xrightarrow{G_{t+1}} \cdots \xrightarrow{G_T} w_T$$

Classifiers. The output of the learning algorithm gives us a *classifier*, which is a function from $\Re^n$ to $\mathcal{C}$, where $\Re$ denotes the set of reals and $\mathcal{C}$ 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 logistic regression we obtain a function from $\Re^n$ to $\{-1, +1\}$.

Some classifiers $F_w(\mathbf{x})$ (vectors will be denoted in boldface) are of the form $\arg\max_l s(F_w)(\mathbf{x})[l]$ (i.e., the classifier F_w outputs the label with the maximum probability according to the "softmax layer"). For example, in several deep-neural network (DNN) architectures the last layer is the *softmax* layer. Recall that the softmax function from $\mathbb{R}^k$ to a probability distribution over $\{1, \cdots, k\} = [k]$ such that the probability of $j \in [k]$ for a vector $\mathbf{x} \in \mathbb{R}^k$ is

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

The r-th component of a vector $\mathbf{x}$ is denoted by $\mathbf{x}[r]$. We are assuming that the reader is a familiar with basics of deep-neural networks (DNNs). For readers not familiar with DNNs we can refer to the excellent book by Goodfellow, Bengio, and Courville [16]. Throughout the paper, we refer to the function $s(F_w)$ as the *softmax layer* corresponding to the classifier F_w. In the case of 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 $+1$):

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

Formally, let $c = |\mathcal{C}|$ 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.

Background on Logic Temporal logics are commonly used for specifying desired and undesired properties of systems. For cyber-physical systems, it is common to use temporal logics that can specify properties of real-valued signals over real time, such as *signal temporal logic* (STL) [31] or *metric temporal logic* (MTL) [28].

A *signal* is a function $s : D \to S$, with $D \subseteq \mathbb{R}_{\geq 0}$ 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*, while those on $\mathbb{R}$ are said *real-valued*. A *trace* $w = \{s_1, \ldots, s_n\}$ is a finite set of real-valued signals defined over the same interval D. We use variables 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 $\sigma_i : \mathbb{R}^n \to \mathbb{B}$, with $\sigma_i \equiv p_i(x_1, \ldots, x_n) \lhd 0$, $\lhd \in \{<, \leq\}$, and $p_i : \mathbb{R}^n \to \mathbb{R}$ a function in the variables $x_1, \ldots, x_n$. An STL formula is defined by the following grammar:

$$\varphi := \sigma \,|\, \neg\varphi \,|\, \varphi \wedge \varphi \,|\, \varphi \,\mathtt{U}_I\, \varphi \tag{2}$$

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

Definition 1 (Qualitative semantics). *Let w be a trace, $t \in \mathbb{R}_{\geq 0}$, and φ be an STL formula. The* qualitative semantics *of φ is inductively defined as follows:*

$$\begin{aligned}
w, t &\models \sigma \textit{ iff } \sigma(w(t)) \textit{ is true} \\
w, t &\models \neg\varphi \textit{ iff } w, t \not\models \varphi \\
w, t &\models \varphi_1 \wedge \varphi_2 \textit{ iff } w, t \models \varphi_1 \textit{ and } w, t \models \varphi_2 \\
w, t &\models \varphi_1 \mathtt{U}_I \varphi_2 \textit{ iff } \exists t' \in t + I \textit{ s.t. } w, t' \models \varphi_2 \textit{ and } \forall t'' \in [t, t'], w, t'' \models \varphi_1
\end{aligned} \tag{3}$$

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.

3 Attacks

There are several types of attacks on ML algorithms. For excellent material on various attacks on ML algorithms we refer the reader to [19,3]. For example, in *training time* attacks an adversary wishes to poison a data set so that a "bad" hypothesis is learned by an ML-algorithm. This attack can be modeled as a game between the algorithm ML and an adversary A as follows:

– ML picks an ordered training set $S = ((x_i, y_i))_{i=1}^{m}$.
– A picks an ordered training set $\widehat{S} = ((\hat{x_i}, \hat{y_i}))_{i=1}^{r}$, where r is $\lfloor \epsilon m \rfloor$.
– ML learns on $S \cup \widehat{S}$ by essentially minimizing

$$\min_{w \in H} L_{S \cup \widehat{S}}(w) \ .$$

The attacker wants to maximize the above quantity and thus chooses $\widehat{S}$ such that $\min_{w \in H} L_{S \cup \widehat{S}}(w)$ is maximized. For a recent paper on certified defenses for such attacks we refer the reader to [45]. In *model extraction* attacks an adversary with black-box access to a classifier, but no prior knowledge of the parameters of a ML algorithm or training data, aims to duplicate the functionality of (i.e., steal) the classifier by querying it on well chosen data points. For an example, model-extraction attacks see [46].

In this paper, we consider *test-time attacks*. We assume that the classifier F_w has been trained without any interference from the attacker (i.e. no training time attacks). Roughly speaking, an attacker has an image $\mathbf{x}$ (e.g. an image of stop sign) and wants to craft a perturbation δ so that the label of $\mathbf{x} + \delta$ is what the attacker desires (e.g. yield sign). The next sub-section describes test-time attacks in detail. We will sometimes refer to F_w as simply F, but the hypothesis w is lurking in the background (i.e., whenever we refer to w, it corresponds to the classifier F).

3.1 Test-time Attacks

The adversarial goal is to take any input vector $\mathbf{x} \in \Re^n$ and produce a minimally altered version of $\mathbf{x}$, *adversarial sample* denoted by $\mathbf{x}^\star$, that has the property of being misclassified by a classifier $F : \Re^n \to \mathcal{C}$. Formally speaking, an adversary wishes to solve the following optimization problem:

$$\begin{array}{rr} \min_{\delta \in \Re^n} & \mu(\delta) \\ \text{such that } & F(\mathbf{x} + \delta) \in T \\ & \delta \cdot \mathbf{M} \ = \ 0 \end{array}$$

The various terms in the formulation are μ is a metric on $\Re^n$, $T \subseteq \mathcal{C}$ is a subset of the labels (the reader should think of T as the target labels for the attacker), and $\mathbf{M}$ (called the *mask*) is a n-dimensional $0-1$ vector of size n. The objective function minimizes the metric μ on the perturbation δ. Next we describe various constraints in the formulation.

– $F(\mathbf{x} + \delta) \in T$
The set T constrains the perturbed vector $\mathbf{x} + \delta$ [1] to have the label (according to F) in the set T. For *mis-classification* problems (the label of $\mathbf{x}$ and $\mathbf{x} + \delta$) are different we have $T = \mathcal{C} - \{F(\mathbf{x})\}$. For *targeted mis-classification* we have $T = \{t\}$ (for $t \in \mathcal{C}$), where t is the target that an attacker wants (e.g., the attacker wants t to correspond to a yield sign).

[1] The vectors are added component wise

– $\delta \cdot \mathbf{M} = 0$
The vector M can be considered as a mask (i.e., an attacker can only perturb a dimension i if $M[i] = 0$), i.e., if $M[i] = 1$ then $\delta[i]$ is forced to be 0. Essentially the attacker can only perturb dimension i if the i-th component of M is 0, which means that δ lies in k-dimensional space where k is the number of non-zero entries in Δ. This constraint is important if an attacker wants to target a certain area of the image (e.g., glasses of in a picture of person) to perturb.
– *Convexity*
Notice that even if the metric μ is convex (e.g., μ is the L_2 norm), because of the constraint involving F, the optimization problem is *not convex* (the constraint $\delta \cdot \mathbf{M} = 0$ is convex). In general, solving convex optimization problems is more tractable non-convex optimization [35].

Note that the constraint $\delta \cdot \mathbf{M} = 0$ essentially constrains the vector to be in a lower-dimensional space and does add additional complexity to the optimization problem. Therefore, for the rest of the section we will ignore that constraint and work with the following formulation:

$$\begin{array}{ll} \min_{\delta \in \Re^n} & \mu(\delta) \\ \text{such that} & F(\mathbf{x}+\delta) \in T \end{array}$$

FGSM mis-classification attack - This algorithm is also known as the *fast gradient sign method (FGSM)* [17]. The adversary crafts an adversarial sample $\mathbf{x}^\star = \mathbf{x} + \delta$ for a given legitimate sample $\mathbf{x}$ by computing the following perturbation:

$$\delta = \varepsilon \operatorname{sign}(\nabla_{\mathbf{x}} L_F(\mathbf{x})) \tag{4}$$

The function $L_F(\mathbf{x})$ is a shorthand for $\ell(w, \mathbf{x}, l(\mathbf{x}))$, where w is the hypothesis corresponding to the classifier F, $\mathbf{x}$ is the data point and $l(\mathbf{x})$ is the label of $\mathbf{x}$ (essentially we evaluate the loss function at the hypothesis corresponding to the classifier). The gradient of the function L_F is computed with respect to $\mathbf{x}$ using sample $\mathbf{x}$ and label $y = l(\mathbf{x})$ as inputs. Note that $\nabla_{\mathbf{x}} L_F(\mathbf{x}))$ is an n-dimensional vector and $\operatorname{sign}(\nabla_{\mathbf{x}} L_F(\mathbf{x}))$ is a n-dimensional vector whose i-th element is the sign of the $\nabla_{\mathbf{x}} L_F(\mathbf{x}))[i]$. The value of the *input variation parameter* ε factoring the sign matrix controls the perturbation's amplitude. Increasing its value increases the likelihood of $\mathbf{x}^\star$ being misclassified by the classifier F but on the contrary makes adversarial samples easier to detect by humans. The key idea is that FGSM takes a step *in the direction of the gradient of the loss function* and thus tries to maximize it. Recall that SGD takes a step in the direction that is opposite to the gradient of the loss function because it is trying to minimize the loss function.

JSMA targetted mis-classification attack - This algorithm is suitable for targeted misclassification [38]. We refer to this attack as JSMA throughout the rest of the paper. To craft the perturbation δ, components are sorted by decreasing *adversarial saliency value*. The adversarial saliency value $S(\mathbf{x}, t)[i]$ of component i for an adversarial target class t is defined as:

$$S(\mathbf{x}, t)[i] = \begin{cases} 0 \text{ if } \frac{\partial s(F)[t](\mathbf{x})}{\partial \mathbf{x}[i]} < 0 \text{ or } \sum_{j \neq t} \frac{\partial s(F)[j](\mathbf{x})}{\partial \mathbf{x}[i]} > 0 \\ \frac{\partial s(F)[t](\mathbf{x})}{\partial \mathbf{x}[i]} \left| \sum_{j \neq t} \frac{\partial s(F)[j](\mathbf{x})}{\partial \mathbf{x}[i]} \right| \text{ otherwise} \end{cases} \tag{5}$$

where matrix $J_F = \left[\frac{\partial s(F)[j](\mathbf{x})}{\partial \mathbf{x}[i]}\right]_{ij}$ is the Jacobian matrix for the output of the softmax layer $s(F)(\mathbf{x})$. Since $\sum_{k \in \mathcal{C}} s(F)[k](\mathbf{x}) = 1$, we have the following equation:

$$\frac{\partial s(F)[t](\mathbf{x})}{\partial \mathbf{x}[i]} = -\sum_{j \neq t} \frac{\partial s(F)[j](\mathbf{x})}{\partial \mathbf{x}[i]}$$

The first case corresponds to the scenario if changing the i-th component of $\mathbf{x}$ takes us further away from the target label t. Intuitively, $S(\mathbf{x}, t)[i]$ indicates how likely is changing the i-th component of $\mathbf{x}$ going to "move towards" the target label t. Input components i are added to perturbation δ in order of decreasing adversarial saliency value $S(\mathbf{x}, t)[i]$ until the resulting adversarial sample $\mathbf{x}^\star = \mathbf{x} + \delta$ achieves the target label t. The perturbation introduced for each selected input component can vary. Greater individual variations tend to reduce the number of components perturbed to achieve misclassification.

CW targetted mis-classification attack. The CW-attack [6] is widely believed to be one of the most "powerful" attacks. The reason is that CW cast their problem as an unconstrained optimization problem, and then use state-of-the art solver (i.e. Adam [25]). In other words, they leverage the advances in optimization for the purposes of generating adversarial examples.

In their paper Carlini-Wagner consider a wide variety of formulations, but we present the one that performs best according to their evaluation. The optimization problem corresponding to CW is as follows:

$$\begin{array}{ll} \min_{\delta \in \Re^n} & \mu(\delta) \\ \text{such that} & F(\mathbf{x}+\delta) = t \end{array}$$

CW use an existing solver (Adam [25]) and thus need to make sure that each component of $\mathbf{x} + \delta$ is between 0 and 1 (i.e. valid pixel values). Note that the other methods did not face this issue because they control the "internals" of the algorithm (i.e., CW used a solver in a "black box" manner). We introduce a new vector $\mathbf{w}$ whose i-th component is defined according to the following equation:

$$\delta[i] = \frac{1}{2}(\tanh(\mathbf{w}[i]) + 1) - \mathbf{x}[i]$$

Since $-1 \leq \tanh(\mathbf{w}[i]) \leq 1$, it follows that $0 \leq \mathbf{x}[i] + \delta[i] \leq 1$. In terms of this new variable the optimization problem becomes:

$$\begin{array}{ll} \min_{\mathbf{w} \in \Re^n} & \mu(\frac{1}{2}(\tanh(\mathbf{w}) + 1) - \mathbf{x}) \\ \text{such that} & F(\frac{1}{2}(\tanh(\mathbf{w}) + 1)) = t \end{array}$$

Next they approximate the constraint ($F(\mathbf{x}) = \mathbf{t}$) with the following function:

$$g(\mathbf{x}) = \max\left(\max_{i \neq t} Z(F)(\mathbf{x})[i] - Z(F)(\mathbf{x})[t], -\kappa\right)$$

In the equation given above $Z(F)$ is the input of the DNN to the softmax layer (i.e $s(F)(\mathbf{x}) = \text{softmax}(Z(F)(\mathbf{x}))$) and κ is a confidence parameter (higher κ encourages

the solver to find adversarial examples with higher confidence). The new optimization formulation is as follows:

$$\begin{array}{ll} \min_{\mathbf{w}\in\Re^n} & \mu(\frac{1}{2}(\tanh(\mathbf{w})+1)-\mathbf{x}) \\ \text{such that} & g(\frac{1}{2}(\tanh(\mathbf{w})+1)) \ \leq \ 0 \end{array}$$

Next we incorporate the constraint into the objective function as follows:

$$\min_{\mathbf{w}\in\Re^n} \ \mu(\tfrac{1}{2}(\tanh(\mathbf{w})+1)-\mathbf{x}) \ + \ c\, g(\tfrac{1}{2}(\tanh(\mathbf{w})+1))$$

In the objective given above, the "lagrangian variable" $c > 0$ is a suitably chosen constant (from the optimization literature we know that there exists $c > 0$ such that the optimal solutions of the last two formulations are the same).

3.2 Adversarial Training

Once an attacker finds an adversarial example, then the algorithm can be retrained using this example. Researchers have found that retraining the model with adversarial examples, produces a more robust model. For this section, we will work with attack algorithms that have a target label t (i.e. we are in the targeted mis-classification case, such as JSMA or CW). Let $\mathcal{A}(w, \mathbf{x}, t)$ be the attack algorithm, where its inputs are as follows: $w \in H$ is the current hypothesis, $\mathbf{x}$ is the data point, and $t \in \mathcal{C}$ is the target label. The output of $\mathcal{A}(w, \mathbf{x}, t)$ is a perturbation δ such that $F(\mathbf{x}+\delta) = t$. If the attack algorithm is simply a mis-classification algorithm (e.g. FGSM or Deepfool) we will drop the last parameter t.

An *adversarial training* algorithm $\mathcal{R}_{\mathcal{A}}(w, \mathbf{x}, t)$ is parameterized by an attack algorithm $\mathcal{A}$ and outputs a new hypothesis $w' \in H$. Adversarial training works by taking a datapoint $\mathbf{x}$ and an attack algorithm $\mathcal{A}(w, \mathbf{x}, t)$ as its input and then retraiining the model using a specially designed loss function (essentially one performs a single step of the SGD using the new loss function). The question arises: what loss function to use during the training? Different methods use different loss functions.

Next, we discuss some adversarial training algorithms proposed in the literature. At a high level, an important point is that the more sophisticated an adversarial perturbation algorithm is, harder it is to turn it into adversarial training. The reason is that it is hard to "encode" the adversarial perturbation algorithm as an objective function and optimize it. We will see this below, especially for the virtual adversarial training (VAT) proposed by Miyato et al. [34].

Retraining for FGSM. We discussed the FGSM attack method earlier. In this case $\mathcal{A} = \text{FGSM}$. The loss function used by the retraining algorithm $\mathcal{R}_{\text{FGSM}}(w, \mathbf{x}, t)$ as follows:

$$\ell_{\text{FGSM}}(w, \mathbf{x}_i, y_i) = \ell(w, \mathbf{x}_i, y_i) + \lambda \ell\left(w, \mathbf{x}_i + \text{FGSM}(w, \mathbf{x}_i), y_i\right)$$

Recall that $\text{FGSM}(w, \mathbf{x})$ was defined earlier, and λ is a regularization parameter. The simplicity of $\text{FGSM}(w, \mathbf{x}_i)$ allows taking its gradient, but this objective function requires label y_i because we are reusing the same loss function ℓ used to train the original

model. Further, $\mathrm{FGSM}(w, \mathbf{x}_i)$ may not be very good because it may not produce good adversarial perturbation direction. The retraining algorithm is simply as follows: *take one step in the SGD using the loss function ℓ_{FGSM} at the data point $\mathbf{x}_i$.*

A caveat is needed for taking gradient during the SGD step. At iteration t suppose we have model parameters w_t, and we need to compute gradient of the objective. Note that $\mathrm{FGSM}(w, \mathbf{x})$ depends on w so by chain rule we need to compute $\partial \mathrm{FGSM}(w, \mathbf{x})/\partial w|_{w=w_t}$. However, this gradient is volatile [2], and so instead Goodfellow et al. only compute:

$$\left.\frac{\partial \ell\left(w, \mathbf{x}_i + \mathrm{FGSM}(w_t, \mathbf{x}_i), y_i\right)}{\partial w}\right|_{w=w_t}$$

Essentially they treat $\mathrm{FGSM}(w_t, \mathbf{x}_i)$ as a constant while taking the derivative.

Virtual Adversarial Training (VAT). Miyato et al. [34] observed the drawback of requiring label y_i above. They propose that, instead of reusing ℓ, to use the following for the regularizer,

$$\Delta(r, \mathbf{x}, w) = \mathrm{KL}\left(s(F_w)(\mathbf{x})[y], s(F_w)(\mathbf{x}+r)[y]\right)$$

for some r such that $\|r\| \leq \delta$. As a result, the label y_i is *no longer* required. The question is: what r to use? Miyato et al. [34] propose that in theory we should use the "best" one as

$$\max_{r:\|r\|\leq\delta} \mathrm{KL}\left(s(F_w)(\mathbf{x})[y], s(F_w)(\mathbf{x}+r)[y]\right)$$

In the equation given above KL is the *KullbackLeibler divergence*. Recall that KL divergence of two distributions P and Q over the same finite domain D is given by the following equation:

$$\mathrm{KL}(P, Q) = \sum_{i \in D} P(i) \log\left(\frac{P(i)}{Q(i)}\right)$$

This thus gives rise to the following loss function to use during retraining:

$$\ell_{\mathrm{VAT}}(w, \mathbf{x}_i, y_i) = \ell(w, , \mathbf{x}_i, y_i) + \lambda \max_{r:\|r\|\leq\delta} \Delta(r, \mathbf{x}_i, w)$$

However, one cannot easily take gradient for the regularizer. Hence the authors perform an approximation as follows:

1. Take Taylor expansion of $\Delta(r, \mathbf{x}_i, w)$ at $r = 0$, so $\Delta(r, \mathbf{x}_i, w) = r^T H(\mathbf{x}_i, w)\ r$ where $H(\mathbf{x}_i, w)$ is the Hessian matrix of $\Delta(r, \mathbf{x}_i, w)$ with respect to r at $r = 0$.
2. Thus $\max_{\|r\|\leq\delta} \Delta(r, \mathbf{x}_i, w) = \max_{\|r\|\leq\delta} \left(r^T H(\mathbf{x}_i, w)\ r\right)$. By variational characterization of the symmetric matrix ($H(\mathbf{x}_i, w)$ is symmetric), $r^* = \delta \bar{v}$ where $\bar{v} = \overline{v(\mathbf{x}_i, w)}$ is the unit eigenvector of $H(\mathbf{x}_i, w)$ corresponding to its largest eigenvalue. Note that r^* depends on $\mathbf{x}_i$ and w. Therefore the loss function becomes:

$$\ell_{\mathrm{VAT}}(\theta, \mathbf{x}_i, y_i) = \ell(\theta, \mathbf{x}_i, y_i) + \lambda \Delta(r^*, \mathbf{x}_i, w)$$

[2] In general, second-order derivatives of a classifier corresponding to a DNN vanish at several points because several layers are piece-wise linear.

3. Now suppose in the process of SGD we are at iteration t with model parameters w_t, and we need to compute $\partial \ell_{\text{VAT}}/\partial w|_{w=w_t}$. By chain rule we need to compute $\partial r^*/\partial w|_{w=w_t}$. However the authors find that such gradients are volatile, so they instead fix r^* as a constant at the point θ_t, and compute

$$\left.\frac{\partial \,\text{KL}\left(s(F_w)(\mathbf{x})[y], s(F_w)(\mathbf{x}+r)[y]\right)}{\partial w}\right|_{w=w_t}$$

3.3 Black Box Attacks

Recall that earlier attacks (e.g. FGSM and JSMA) needed white-box access to the classifier F (essentially because these attacks require first order information about the classifier). In this section, we present black-box attacks. In this case, an attacker can *only* ask for the labels $F(\mathbf{x})$ for certain data points. Our presentation is based on [37], but is more general.

Let $\mathcal{A}(w, \mathbf{x}, t)$ be the attack algorithm, where its inputs are: $w \in H$ is the current hypothesis, $\mathbf{x}$ is the data point, and $t \in \mathcal{C}$ is the target label. The output of $\mathcal{A}(w, \mathbf{x}, t)$ is a perturbation δ such that $F(\mathbf{x}+\delta) = t$. If the attack algorithm is simply a misclassification algorithm (e.g. FGSM or Deepfool) we will drop the last parameter t (recall that in this case the attack algorithm returns a δ such that $F(\mathbf{x}+\delta) \neq F(\mathbf{x})$). An *adversarial training* algorithm $\mathcal{R}_{\mathcal{A}}(w, \mathbf{x}, t)$ is parameterized by an attack algorithm $\mathcal{A}$ and outputs a new hypothesis $w' \in H$ (this was discussed in the previous subsection).
Initialization: We pick a substitute classifier G and an initial seed data set S_0 and train G. For simplicity, we will assume that the sample space $Z = X \times Y$ and the hypothesis space H for G is same as that of F (the classifier under attack). However, this is not crucial to the algorithm. We will call G the *substitute classifier* and F the *target classifier*. Let $S = S_0$ be the initial data set, which will be updated as we iterate.
Iteration: Run the attack algorithm $\mathcal{A}(w, \mathbf{x}, t)$ on G and obtain a δ. If $F(\mathbf{x}+\delta) = t$, then **stop** we are done. If $F(\mathbf{x}+\delta) = t'$ but not equal to t, we augment the data set S as follows:

$$S = S \cup (\mathbf{x}+\delta, t')$$

We now retrain G on this new data set, which essentially means running the SGD on the new data point $(\mathbf{x}+\delta, t')$. Notice that we can also use adversarial training $\mathcal{R}_{\mathcal{A}}(w, \mathbf{x}, t)$ to update G (to our knowledge this has been not tried out in the literature).

3.4 Defenses

Defenses with formal guarantees against test-time attacks have proven elusive. For example, Carlini and Wagner [7] have a recent paper that breaks *ten recent defense proposals*. However, defenses that are based on robust-optimization objectives have demonstrated promise [33,27,44]. Several techniques for verifying properties of a DNN (in isolation) have appeared recently (e.g., [24,20,13,14]). Due to space limitations we will not give a detailed account of all these defenses.

4 Semantic Adversarial Analysis and Training

A central tenet of this paper is that the analysis of deep neural networks (and machine learning 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. We view this not as ignoring arbitrary modifications (which are indeed worth considering with a security mind set), but as prioritizing the design and analysis of DNNs towards semantic adversarial modifications. Sec. 4.1 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 machine learning models. See Sec. 4.1 for a more detailed discussion of this point.
- *Semantic (Re-)Training:* Most machine learning 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 system-level specifications, counterexamples, and other artifacts are used to improve the semantic quality of the ML model. Sec. 4.2 explores a few ideas.
- *Confidence-Based Analysis and Decision Making:* Deep neural networks (and other ML models) often produce not just an output label, but also an associated confidence level. We argue that *confidence levels* must be used within the design of ML-based systems. They provide a way of exposing more information from the DNN to the surrounding system that uses its decisions. Such confidence levels can also be useful to prioritize analysis towards cases that are more egregious failures of the DNN. More generally, any *explanations* and *auxiliary information* generated by the DNN that accompany its main output decisions can be valuable aids in their design and analysis.

4.1 Compositional Falsification

We discuss the problem of performing system-level analysis of a deep learning component, using recent work by the authors [10,11] to illustrate the main points. The material in this section is mainly based on [41].

We begin with some basic notation. Let S denote the model of the full system S 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. As in Sec. 3, 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 $S\|E$. The goal of falsification is to find one or more counterexamples showing how the composite system $S\|E$ violates Φ. In this context, *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.*

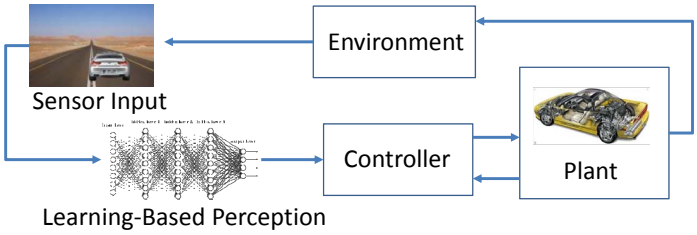

Fig. 1. Automatic Emergency Braking System (AEBS) in closed loop. An image classifier based on deep neural networks is used to perceive objects in the ego vehicle's frame of view.

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 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 meters. In STL, this requirement Φ can be written as $\mathtt{G}_{0,T}(\|\mathbf{x}_{\text{ego}} - \mathbf{x}_{\text{obs}}\|_2 \geq 2)$. 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 machine learning (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 1000×600px (for an image classifier) contains $256^{1000\times600\times3}$ elements.

In the above example, $S||E$ is the closed loop system in Fig. 1 where S comprises the DNN and the controller, and E comprises everything else. C is the DNN used for object detection and classification.

This case study has been implemented in Matlab/Simulink[3] in two versions that use two different Convolutional Neural Networks (CNNs): the Caffe [21] version of AlexNet [29] and the Inception-v3 model created with Tensorflow [32], both trained on the ImageNet database [1]. Further details about this example can be obtained from [10].

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.

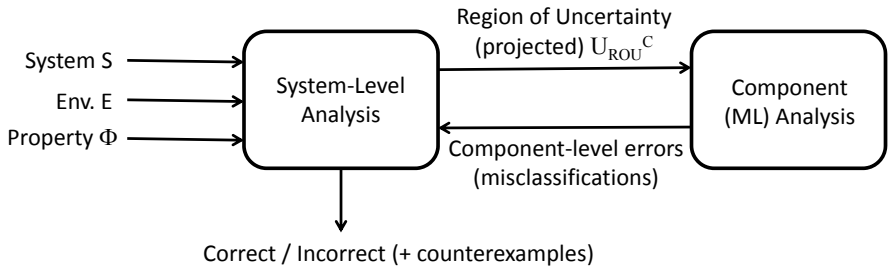

Fig. 2. Compositional Verification Approach. A system-level verifier cooperates with a component-level analysis procedure (e.g., adversarial analysis of a machine learning component to find misclassifications).

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

[3] `https://github.com/dreossi/analyzeNN`

abstraction assumes perfect perception/sensing. Using this abstraction, we compute the validity domain for this abstract model of the system, denoted 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^{C}_{ROU} 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^{C}_{ROU}, projected onto the inputs of C, is communicated to the ML Analyzer. (Concretely, in the context of our example of Sec. 4.1, this corresponds to finding a subspace of images that corresponds to U^{C}_{ROU}.)

2. The Component-level Analyzer, also termed as a Machine Learning (ML) Analyzer, performs a detailed analysis of 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 [10]. 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 [10] is shown in Fig. 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 continues thus, until we either prove/disprove Φ, or we run out of resources.

Sample Results We have applied the above approach to the problem of *compositional falsification* of cyber-physical systems (CPS) with machine learning components [10]. For this class of CPS, including those with highly non-linear dynamics and even black-box components, simulation-based falsification of temporal logic properties is an approach that has proven effective in industrial practice (e.g., [22,47]). We present here a sample of results on the AEBS example from [10], referring the reader to more detailed descriptions in the other papers on the topic [10,11].

In Figure 4 we show one result of our analysis for the Inception-v3 deep neural network. This figure shows both correctly classified and misclassified images on a range of synthesized images where (i) the environment vehicle is moved away from or towards the ego vehicle (along z-axis), (ii) it is moved sideways along the road (along x-axis),

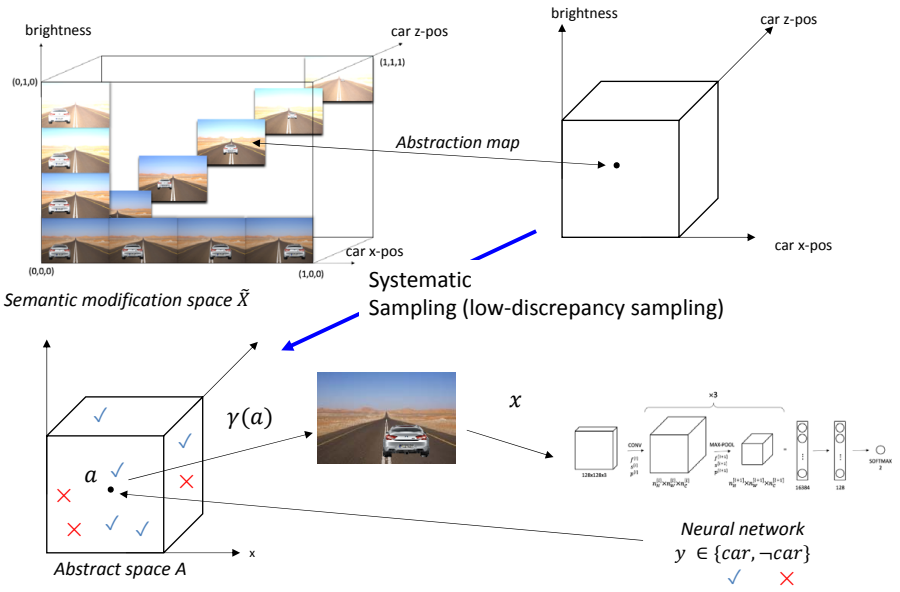

Fig. 3. Machine Learning Analyzer: Searching the Semantic Modification Space. A concrete semantic modification space (top left) is mapped into a discrete abstract space. Systematic sampling, using low-discrepancy methods, yields points in the abstract space. These points are concretized and the NN is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

or (iii) the brightness of the image is modified. These modifications constitute the 3 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, as well as 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 [10,11].

4.2 Semantic Training

In this section we discuss two ideas for *semantic training and retraining* of deep neural networks. 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 also to gain more assurance in the overall system containing the ML component. A more detailed study of using misclassifications (ML

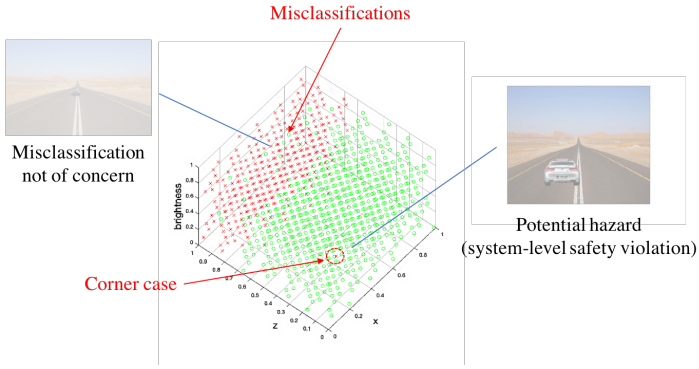

Fig. 4. Misclassified Images for Inception-v3 Neural Network (trained on ImageNet with TensorFlow). Red crosses are misclassified images and green circles are correctly classified. Our system-level analysis finds a corner-case image that could lead to a system-level safety violation.

component-level counterexamples) to improve the accuracy of the neural network is presented in [12]; this approach is termed *counterexample-guided data augmentation*, inspired by counterexample-guided abstraction refinement (CEGAR) [8] and similar paradigms.

Experimental Setup As in the preceding section, we consider an Automatic Emergency Braking System (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 [11].[4] 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 convolutional neural network (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 Fig. 5. It consists of eight layers: the first six are alternations of convolutions and max-pools with 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 1000 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 cross-entropy cost function and the Adam algorithm optimizer (learning rate 10^{-4}). The model reached 95% accuracy on the test

[4] Udacity's self-driving car simulator: `https://github.com/udacity/self-driving-car-sim`

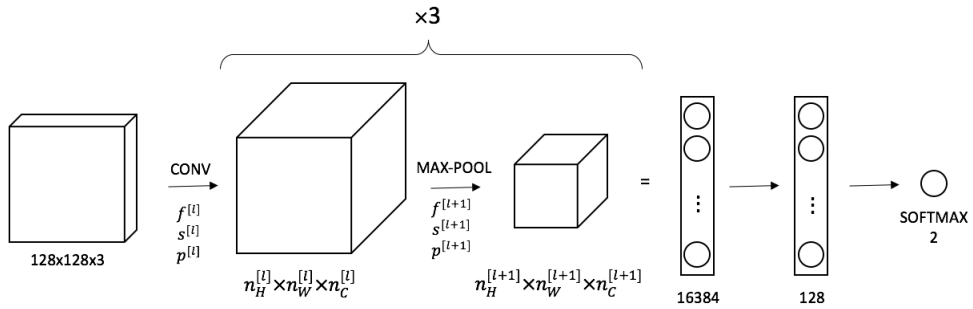

Fig. 5. CNN architecture.

set. Finally, the resulting CNN is connected to the Unity simulator via Socket.IO protocol.[5] Fig. 6 depicts a screenshot of the simulator with the AEBS in action in proximity of a cow.

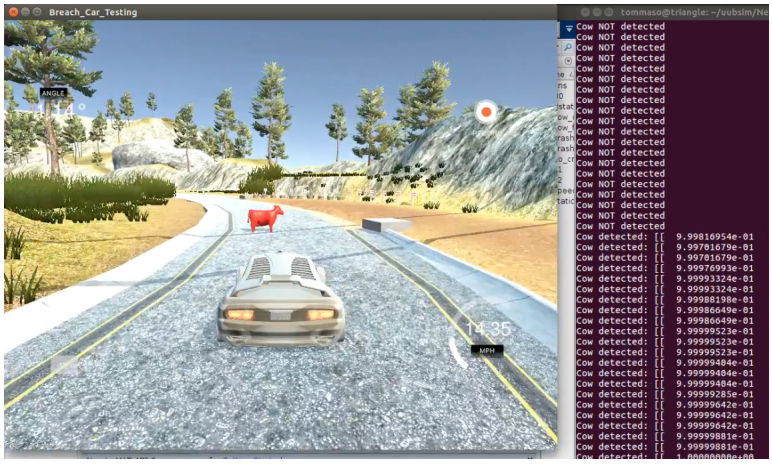

Fig. 6. Udacity simulator with a CNN-based AEBS in action.

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

$$l(\hat{y}) = \max(0, k + \max_{i \neq l}(\hat{y}_i - \hat{y}_l)) \tag{6}$$

[5] Socket.IO protocol: `https://github.com/socketio`

where (x, y) is a training sample, $\hat{y} = F(x)$ is a prediction, and l is the *ground truth* class of x (i.e., y_l is 1). For this section, the output $\hat{y}$ is a numerical value indicating the *confidence level* of the network for each class. For example, $\hat{y}$ can be the output of a softmax layer as described in Sec. 2.

Consider what happens as 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 $|k|$. Larger the absolute value of k, greater the tolerance. Intuitively, this biases the training process towards avoiding "high confidence misclassifications".

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 cross-entropy 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).

	$T_{original}$		$T_{countex}$	
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

Table 1. Hinge loss with different k values.

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 very preliminary result and further experimentation and analysis is necessary.

System-Level Counterexamples By using the composition falsification framework presented in Sec. 4.1, 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 Sec. 4.2).

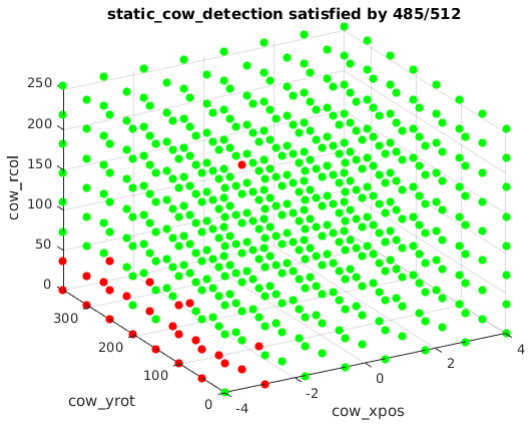

Fig. 7. Semantic counterexamples: obstacle configurations leading to property violations (in red).

We trained the model with both cross-entropy and hinge loss for 20 epochs. Both models achieve a high accuracy on the validation set ($\approx 92\%$). 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.

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 with 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 (see [42]), semantic retraining could help mitigate the impact of misclassifications on the system-level behavior.

5 Conclusion

In this paper, we surveyed the field of adversarial machine learning with a special focus on deep learning and on test-time attacks. We then introduced the idea of *semantic adversarial machine (deep) learning*, where adversarial analysis and training of ML models is 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, training using semantic counterexamples, and utilizing more detailed information about the outputs produced by the ML model, including confidence levels, in the modules that use these outputs to make decisions. Preliminary experiments show

the promise of these ideas, but also indicate that much remains to be done. We believe the field of semantic adversarial learning will be a rich domain for research at the intersection of machine learning, formal methods, and related areas.

Acknowledgments

The first and third author were supported in part by NSF grant 1646208, the DARPA BRASS program under agreement number FA8750-16-C0043, the DARPA Assured Autonomy program, and Berkeley Deep Drive.

References

1. Imagenet. http://image-net.org/.
2. Babak Alipanahi, Andrew Delong, Matthew T Weirauch, and Brendan J Frey. Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning. *Nature biotechnology*, 2015.
3. Marco Barreno, Blaine Nelson, Anthony D Joseph, and JD Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.
4. M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. Technical report, 2016.
5. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. http://arxiv.org/abs/1604.07316.
6. Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017.
7. Nicolas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *ACM Workshop on Artificial Intelligence and Security*, 2017.
8. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *12th International Conference on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.
9. George E Dahl, Jack W Stokes, Li Deng, and Dong Yu. Large-scale malware classification using random projections and neural networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3422–3426. IEEE, 2013.
10. Tommaso Dreossi, Alexandre Donzé, and Sanjit A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. In *NASA Formal Methods Conference (NFM)*, May 2017.
11. Tommaso Dreossi, Alexandre Donzé, and Sanjit A. Seshia. Compositional falsification of cyber-physical systems with machine learning components. *CoRR*, abs/1703.00978, 2017.
12. Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Kurt Keutzer, Alberto Sangiovanni-Vincentelli, and Sanjit A. Seshia. Counterexample-guided data augmentation. In *International Joint Conference on Artificial Intelligence (IJCAI)*, July 2018.
13. Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep neural networks. 2018. to appear.
14. K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli. A Dual Approach to Scalable Verification of Deep Networks. *ArXiv e-prints*, March 2018.
15. Nathan Eddy. Ai, machine learning drive autonomous vehicle development. http://www.informationweek.com/big-data/big-data-analytics/ai-machine-learning-drive-autonomous-vehicle-development/d/d-id/1325906, 2016.
16. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
17. Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Proceedings of the 2015 International Conference on Learning Representations*. Computational and Biological Learning Society, 2015.
18. Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

19. Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.
20. X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *In Proc. 29th International Conference on Computer Aided Verification (CAV)*, 2017.
21. Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *ACM Multimedia Conference, ACMMM*, pages 675–678, 2014.
22. Xiaoqing Jin, Alexandre Donzé, Jyotirmoy Deshmukh, and Sanjit A. Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, 34(11):1704–1717, 2015.
23. K. Julian, J. Lopez, J. Brush, M. Owen, and M. Kochenderfer. Policy compression for aircraft collision avoidance systems. In *Proc. 35th Digital Avionics Systems Conf. (DASC)*, 2016.
24. Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *In Proceedings of the 29-th International Conference on Computer Aided Verification*, 2017.
25. Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. https://arxiv.org/abs/1412.6980, 2017.
26. Eric Knorr. How paypal beats the bad guys with machine learning. http://www.infoworld.com/article/2907877/machine-learning/how-paypal-reduces-fraud-with-machine-learning.html, 2015.
27. J. Zico Kolter and Eric Wong. Provable defenses against adversarial examples via the convex outer adversarial polytope. *CoRR*, abs/1711.00851, 2017.
28. Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.
29. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
30. Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems: A Cyber-Physical Systems Approach*. MIT Press, 2nd edition edition, 2016.
31. Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Modeling and Analysis of Timed Systems, FORMATS*, pages 152–166, 2004.
32. Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
33. Aleksander Mdry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.
34. Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing by virtual adversarial examples. *CoRR*, abs/1507.00677, 2015.
35. Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer, 2006.
36. NVIDIA. Nvidia tegra drive px: Self-driving car computer, 2015.
37. Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z.Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, April 2017.
38. Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy*. arXiv preprint arXiv:1511.07528, 2016.
39. Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.

40. Stuart Russell, Tom Dietterich, Eric Horvitz, Bart Selman, Francesca Rossi, Demis Hassabis, Shane Legg, Mustafa Suleyman, Dileep George, and Scott Phoenix. Letter to the editor: Research priorities for robust and beneficial artif icial intelligence: An open letter. *AI Magazine*, 36(4), 2015.
41. Sanjit A. Seshia. Compositional verification without compositional specification for learning-based systems. Technical Report UCB/EECS-2017-164, EECS Department, University of California, Berkeley, Nov 2017.
42. Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. Towards Verified Artificial Intelligence. *ArXiv e-prints*, July 2016.
43. Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. Recognizing functions in binaries with neural networks. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 611–626, 2015.
44. Aman Sinha, Hongseok Namkoong, and John Duchi. Certifiable distributional robustness with principled adversarial training. In *ICLR*, 2018.
45. Jacob Steinhardt, Pang Wei Koh, and Percy Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.
46. Florian Tramer, Fan Zhang, Ari Juels, Michael Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security*, 2016.
47. Tomoya Yamaguchi, Tomoyuki Kaga, Alexandre Donzé, and Sanjit A. Seshia. Combining requirement mining, software model checking, and simulation-based verification for industrial automotive systems. In *Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD)*, October 2016.