Detecting Adversarial Samples of Deep Neural Networks through Mutation Testing ⋆

Jingyi Wang[1], Peixin Zhang[2], Pengfei Yang[3,4]
Jun Sun[1], Dongxia Wang[5], and Xinyu Wang[2]

[1] Singapore University of Technology and Design
[2] Zhejiang University
[3] Chinese Academy of Sciences
[4] University of Chinese Academy of Sciences
[5] University of Oxford

Abstract. It has been shown that deep neural networks (DNN) are intrinsically subject to attacks through adversarial samples, which could be easily generated in different ways. As DNN are increasingly used in safety-critical systems like self-driving cars or face recognition, it is crucial to develop *efficient* techniques for defending against such attacks. Many existing defenses like adversarial training and robust training have been shown to be either ineffective or intractable. An alternative approach is to detect adversarial samples at runtime and reject them to avoid bad decisions. In this work, we first observe that if we randomly sample around a normal and an adversarial sample respectively, there is a significant difference between their *L*abel *C*hange *R*ate (LCR) over the new samples. We further provide a theoretical analysis on such a difference, and design a lightweight detection algorithm based on statistical model checking called *nMutant* (inspired by mutation testing). Our experiments show that LCR is significantly more effective (i.e., close to a perfect classifier) and efficient (such that it can be potentially employed at runtime) than state-of-the-art approaches in detecting adversarial samples generated by recently proposed attacking methods.

Keywords: Deep Neural Networks · Adversarial Attack · Detection · Mutation Testing.

1 Introduction

Deep Neural Networks (DNN) have been extensively used in a wide range of applications in recent years. However, it has been shown that DNN are intrinsically subject to adversarial attacks where even a well-trained DNN can be vulnerable [33]. This is especially the case when DNN are applied to classification tasks [10,23]. Many methods of attacking have been invented to generate such adversarial samples, e.g., Fast Gradient Sign method [10] and its variants [23], Jacobian-based saliency map approach [26], C&W attack [6] and Black-box attack [25]. Besides different kinds of attacks, multiple DNN testing approaches are also found effective to identify adversarial samples [34,28,36].

⋆ Corresponding authors: Xinyu Wang.

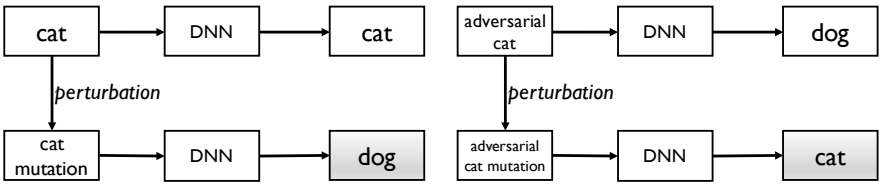


Fig. 1: Label change via perturbation on a normal sample (left) and an adversarial sample (right).

Adversarial samples are created with the intent to trigger errors of the DNN. They are often crafted through careful adversarial perturbation, i.e., manipulating the original sample with minor perturbations so that the DNN model classifies the sample incorrectly. As DNN are increasingly used in safety-critical systems like self-driving cars [4] or face recognition [30], it is crucial to develop *efficient* techniques for defending against such attacks to avoid serious accidents [1,2]. So far, multiple defense strategies have been proposed to improve the robustness of DNN. For instance, adversarial training approaches [12,31,32] take adversarial samples into consideration during model training to make it more challenging for attackers to craft adversarial samples, which however relies on the available adversarial samples and thus are usually limited to defending existing attacks. As a step further, robust training approaches try to consider all the possible perturbations on the original training sample during model training by solving a MinMax optimization problem approximately [31,20], which is computationally expensive and not scalable to large DNN. Worse yet, the above-mentioned defenses provide no guarantee if the DNN is faced with a new unknown attack. Another promising way is to formally verifying DNN to provide guarantee on the robustness of a DNN over adversarial perturbations. Both approaches based on SMT solver [14,35] or abstract interpretation [9] have been able to prove safety properties in some real-world problems. However, these verification approaches are still far from real-time applications. Furthermore, they can usually only prove pointwise safety around a small region and are not able to handle the cases where verification fails .

Due to the inherent difficulty of adversarial defense, we propose a complementary approach to ensure the safe functioning of DNN by detecting adversarial samples at runtime and rejecting them to avoid bad decisions (or raise an alarm for further check). Our approach is lightweight which can be deployed as runtime monitors for real-time safety assurance. The main intuition behind our approach is that adversarial samples are much more sensitive to random perturbations than normal samples (i.e., those which are correctly labeled by the DNN). We further propose a new feature to measure such a sensitivity, i.e., *L*abel *C*hange *R*ate (LCR). That is, the probability of obtaining a different label by imposing random perturbations to an adversarial sample is significantly higher than that of imposing random perturbations to a normal sample. This is illustrated in Fig. 1, i.e., the probability of the scenario illustrated on the right happening is significantly larger than that of the scenario illustrated on the left happening. We then explain this observation through a theoretical analysis of LCR. We confirm our observation and analysis through an empirical study with standard datasets and

recently proposed adversarial perturbation methods. Based on the difference of LCR on normal and adversarial samples, we then propose the method for detecting adversarial samples based on hypothesis testing [17,3]. The basic idea is to measure the LCR of a provided sample and reject the sample if the LCR is above a certain threshold. Through expenrimental evaluation on broadly adopted datasets in the area, we show that our approach is more effective and efficient than several state-of-the-art detection methods against many existing attacking methods (e.g., FGSM [10], C&W [6], JSMA [26], and BlackBox [25]). Furthermore, compared to state-of-the-art detection methods which require the full knowledge of the DNN and a set of adversarial samples (to train a classifier), our approach works in a black-box setting and does not rely on any adversarial samples. It implies that our approach can be potentially applied to a wider range of systems and generalizes to deal with new unknown attacks.

We frame the rest of the paper as follows. We review necessary background and formally define our adversarial sample detection problem in Section 2. Our observation and the theoretical analysis are presented in Section 3. We present our detection algorithm in Section 4 and experiment results in Section 5. Lastly, we discuss related works in Section 6 and conclude in Section 7.

2 Background

We review necessary background to understand this work in this section.

Deep Neural Networks In this work, we focus on DNN for classification tasks which outputs a label given a certain input sample. For instance, a commonly-used Convolutional Neural Network (CNN) for image classification is typically concatenated by convolutional layers with ReLU activation functions, max-pooling layers and fully connected layers. In this work, we assume that we do not have any knowledge of f other than that we can obtain its output (i.e., the label) given a certain sample. We denote the target DNN by $f(X) : X \to C$, where X is the set of input samples and C is the set of output labels. Given a sample x, we denote its true label (ground-truth label obtained by human observer) by c_x. We say that a sample x is a normal sample if $f(x) = c_x$ and is an adversarial sample if $f(x) \neq c_x$. Notice that according to our definition, a sample in the training or testing data which is wrongly-labeled is also an adversarial sample.

Adversarial Attacks Many approaches have been proposed to craft adversarial samples. In the following, we briefly introduce the 4 kinds of state-of-the-art attacking methods used in our experiment. More review of the attacking methods can be found in Section 6.

FGSM: The Fast Gradient Sign Method (FGSM) [10] aims to attack the original input by changing the softmax value of its original label to the largest extent, which could be obtained by its gradient. FGSM could be implemented in a straightforward and efficient way. By simply adding up the sign of gradient (cost function with respect to the input), we could get a potential adversarial counterpart of a normal sample in one step:

$$\hat{x} = x + \epsilon \mathbf{sign}(\nabla \mathbf{J}(\theta, x, c_x))$$

, where **J** is the cost used to train the model and θ is the parameter. Notice that FGSM does not guarantee the adversarial perturbation is minimal.

JSMA: Jacobian-based Saliency Map Attack(JSMA) [27] is a targeted attack method which iteratively changes one most significant pixel during each iteration towards the target label. The selection of the pixel is based on the calculation of a saliency map which characterizes the impact that each pixel imposes on the target label. The algorithm picks the most significant pixel each time and maximizes its value. The process is repeated until it either reaches the target label or the number of pixels modified exceeds a certain threshold. We refer the readers to [27] for details of calculating the saliency map.

C&W: Carlini *et al.* [6] proposed to directly solve an optimization problem which minimizes the perturbation (with certain distance metric) and maximizes the probability of the target class label with objective function formulated as follow:

$$\arg\min \Delta x + c \cdot f(\hat{x}, t)$$

where Δx is the imposed perturbation according to some distance metric, e.g., L_0, L_2, L_∞, $\hat{x} = x + \Delta x$ is the clipped adversarial sample and t is its target label. The idea is to devise a clip function for the adversarial sample such that the value of each pixel dose not exceed the legal range. The clip function and the best loss function according to [6] are shown as follows.

$$\begin{aligned} clip: \hat{x} &= 0.5(tanh(\tilde{x}) + 1) \\ loss: f(\hat{x}, t) &= \max(\max\{G(\hat{x})_c : c \neq t\} - G(\hat{x})_t, 0) \end{aligned}$$

where $G(x)$ denotes the output vector of a model and t is the target class. Readers can refer to [6] for details.

Black-Box: The above mentioned attacks are white-box attacks which require the full knowledge of the DNN model. Black-Box (BB) attack only needs to know the output of a target DNN model given a certain input sample. The idea is to train a substitute model to mimic the behaviors of the targeted model with data augmentation. Then, one of the existing attack algorithms, e.g., FGSM and JSMA, is applied on the substitute model to generate adversarial samples. The key insight is that the adversarial samples transfer between different model architectures [33,10].

Problem definition Our problem is then, given a sample x and the DNN f (without access to the details of f other than obtaining the label $f(x)$), how can we effectively and efficiently determine whether x is a normal sample or an adversarial sample? Can we provide some confidence for the detection result as well? Once we detect an adversarial sample at runtime, we can take different actions like raising an alarm or simply rejecting the sample depending on specific applications to ensure the safe use of the DNN.

3 Mutation Testing Effect

In this section, we present our observation on the difference of a normal sample and an adversarial sample by mutation testing (generating mutations by imposing random

perturbations) w.r.t. *L*abel *C*hange *R*ate (LCR). Given an arbitrary sample x for a DNN, we obtain a set of its mutations $X_m(x)$, each of which $x_m = x + \Delta x$ is obtained by imposing a minor perturbation Δx on x. We remark that there are different metrics like L_1, L_2 or L_∞ norms to measure how close a mutation is from the original input. For instance, the L_∞ norm is the maximum distance between each dimension of x and x_m.

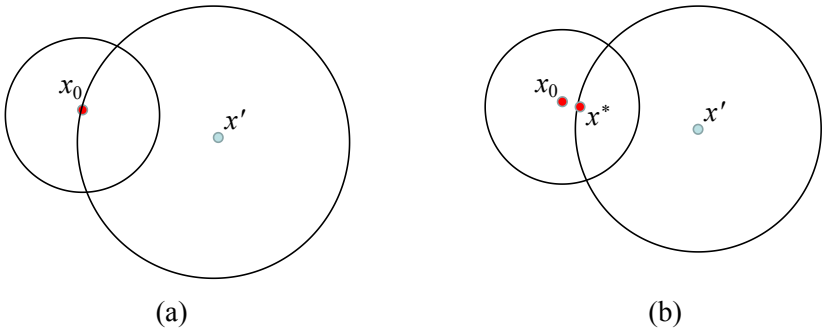
We consider a black-box setting, i.e., for every mutation $x_m \in X_m(x)$, we can only obtain its output label $f(x_m)$ by feeding x_m into the DNN without having access to the details of f. If we have a robust classifier f, for most of the mutations in $X_m(x)$, $f(x_m)$ should be the same as $f(x)$ since we are imposing a small and label-preserving (to humans) perturbation on x. However, a label change might occur as demonstrated in previous works [33,10,16]. An example of label change is illustrated in Figure 1. The left figure takes a normal 'cat' image as input and the DNN correctly classifies it as a 'cat'. After imposing some small perturbation, the DNN labels it as a 'dog'. The right figure however takes an adversarial 'cat' image (which could be either due to adversarial perturbation or training error) as input and the DNN wrongly labels it as a 'dog'. Then, after imposing some small perturbation, the DNN labels it as a 'cat'. We formally define the label change rate (LCR) of a sample x to perturbations as follows:

$$\kappa(x) = \frac{|\{x_m | x_m \in X_m(x) \wedge f(x_m) \neq f(x)\}|}{|X_m(x)|}$$

, where $|S|$ is the number of elements in a set S. Intuitively, $\kappa(x)$ is the percentage of mutations in $X_m(x)$ that have a different label $f(x_m)$ from $f(x)$. To abuse the notations, we use κ_{nor} and κ_{adv} to denote the average LCR of normal samples and that of adversarial samples respectively. *Our observation is that:* $\kappa_{adv} \gg \kappa_{nor}$, which we regard as *mutation testing effect*. The implication is that we can detect an adversarial sample by evaluating whether its LCR is beyond a certain threshold.

A Theoretical Analysis In the following, we aim to provide a theoretical analysis on the above mutation testing effect, according to which, an adversarial sample has a significantly higher LCR than a normal sample. Before delving into the details, an intuitive explanation of the phenomenon is that an adversarial sample is often generated by imposing a perturbation that is as small as possible but still able to cross the decision boundary on a normal sample (otherwise it will be a different or unknown label even to human observers and thus obvious). So the crafted adversarial samples are likely close to the decision boundary. On the contrary, normal samples are likely far from the decision boundary for a well-trained DNN (with good generalization). Therefore, it is more likely to obtain a different label by sampling around an adversarial sample than a normal sample. In the following, we provide a formal analysis for this intuition and a lower bound on the LCR of an adversarial sample.

Let $f : \mathbb{R}^m \to C$ be the classification function of the DNN, and $\|\cdot\| : \mathbb{R}^m \to [0, +\infty)$ a norm on the sample space $\mathbb{R}^m$. We assume that there is a ground-truth classification function $c : \mathbb{R}^m \to C$, and that the model f is well-trained in the sense that the classification boundaries of f and c has the distance close to 0. For a sample $x_0 \in \mathbb{R}^m$, we restrict a sampling region $B(x_0, \delta) := \{x \in \mathbb{R}^m \mid \|x - x_0\| < \delta\}$ and assume a sampling distribution μ. We use $Pr_\mu(f(x) \neq f(x_0))$ to denote the probability that we



(a) (b)

Fig. 2: A lower bound of LCR for an adversarial sample is the measure of the overlapping region on the sampling distribution.

get a sample with a different label from x_0 according to the sampling distribution μ, which is exactly the LCR of x_0.

For the sampling region $B(x_0, \delta)$ and the corresponding sampling distribution μ_δ, we assume the set of sampling distribution $\{\mu_\delta \mid \delta > 0\}$ satisfies the following conditions:

- $\mu_\delta(B(x_0, \delta)) = 1$ for all $\delta > 0$;
- For all $\delta > 0$, μ_δ has the joint density function p_δ satisfying $p_\delta(x) = p_\delta(y)$ for any $x, y \in \mathbb{R}^m$ satisfying $\|x - x_0\| = \|y - x_0\|$.

These assumptions fit the distributions we usually use, like the uniform distribution and the normal distribution. They will also be adopted in the following analysis.

We first consider an extreme case when x_0 is an optimal adversarial example of a normal sample x' in the sense that x_0 is the solution of the following optimization problem:

$$\begin{aligned} &\min \|x - x'\| \\ &\text{s.t. } f(x) \neq c_x. \end{aligned} \tag{1}$$

That is to say, samples in the region $B(x', \|x_0 - x'\|)$ has a different label from x_0. We write $r = \|x_0 - x'\|$. As Fig. 2(a) shows, we have

$$Pr_{\mu_\delta}(f(x) \neq f(x_0)) \geq \int_{\|x-x_0\| \leq \delta \wedge \|x-x'\| \leq r} \mu_\delta(\mathrm{d}x).$$

The following theorem gives an approximation of this lower bound.

Theorem 1. *For any L_p norm $\|\cdot\|$ with $1 < p < +\infty$,*

$$\lim_{\delta \to 0+} \int_{\|x-x_0\| \leq \delta \wedge \|x-x'\| \leq r} \mu_\delta(\mathrm{d}x) = \frac{1}{2}.$$

Proof. Let λ_m be the Lebesgue measure on $\mathbb{R}^m$. First we claim that, for any increasing sequence $r_n \uparrow +\infty$,

$$\lim_{n\to\infty} \frac{\int_{\|x-x_0\|=1\wedge\|x-x'\|\le r_n} \lambda_{m-1}(\mathrm{d}x)}{\int_{\|x-x_0\|=1} \lambda_{m-1}(\mathrm{d}x)} = \frac{1}{2}. \tag{2}$$

Notice the fact that the sequence of sets $\{x \in \mathbb{R}^m \mid \|x-x_0\| = 1 \wedge \|x-x'\| \le r_n\}$ converges to a semi-sphere D of $\partial B(x_0, 1)$ as $n \to \infty$, so

$$\begin{aligned}
\lim_{n\to\infty} \frac{\int_{\|x-x_0\|=1\wedge\|x-x'\|\le r_n} \lambda_{m-1}(\mathrm{d}x)}{\int_{\|x-x_0\|=1} \lambda_{m-1}(\mathrm{d}x)} &= \frac{\lim_{n\to\infty} \int_{\|x-x_0\|=1\wedge\|x-x'\|\le r_n} \lambda_{m-1}(\mathrm{d}x)}{\int_{\|x-x_0\|=1} \lambda_{m-1}(\mathrm{d}x)} \\
&= \frac{\int_D \lambda_{m-1}(\mathrm{d}x)}{\int_{\|x-x_0\|=1} \lambda_{m-1}(\mathrm{d}x)} \\
&= \frac{\frac{1}{2}\int_{\|x-x_0\|=1} \lambda_{m-1}(\mathrm{d}x)}{\int_{\|x-x_0\|=1} \lambda_{m-1}(\mathrm{d}x)} = \frac{1}{2},
\end{aligned}$$

where the second $=$ uses the fact that $\int_{A_n} g\,\mathrm{d}\mu$ converges to $\int_A g\,\mathrm{d}\mu$ given measurable $A_n \to A$ as $n \to \infty$. We can see $\int_{\|x-x_0\|=1\wedge\|x-x'\|\le r} \lambda_{m-1}(\mathrm{d}x)$ is monotone as r increases, so Equ. 2 is equivalent to

$$\lim_{r\to+\infty} \frac{\int_{\|x-x_0\|=1\wedge\|x-x'\|\le r} \lambda_{m-1}(\mathrm{d}x)}{\int_{\|x-x_0\|=1} \lambda_{m-1}(\mathrm{d}x)} = \frac{1}{2}, \tag{3}$$

and

$$\lim_{\delta\to 0+} \frac{\int_{\|x-x_0\|=\delta\wedge\|x-x'\|\le r} \lambda_{m-1}(\mathrm{d}x)}{\int_{\|x-x_0\|=\delta} \lambda_{m-1}(\mathrm{d}x)} = \frac{1}{2}.$$

From the assumptions of μ_δ, we write $q_\delta(t) = p_\delta(x)$ for $\|x\| = t$. Then we consider the set $E = \{x \in \mathbb{R}^m \mid \|x-x_0\| = \delta \wedge \|x-x'\| = r\}$. We draw segments from x_0 to the points in E and they form an $(m-1)$-dimension hyperplane, and we use A_δ denote the region surrounded by this hyperplane and $\{x \in \mathbb{R}^m \mid \|x-x_0\| = \delta \wedge \|x-x'\| \le r\}$. Because $B(x', r)$ is convex, A_δ is a subset of the target integration region $\{x \in \mathbb{R}^m \mid \|x-x_0\| \le \delta \wedge \|x-x'\| \le r\}$, and we have

$$\begin{aligned}
\lim_{\delta\to 0+} \int_{\|x-x_0\|\le\delta\wedge\|x-x'\|\le r} \mu_\delta(\mathrm{d}x) &\ge \lim_{\delta\to 0+} \int_{A_\delta} p_\delta(x)\,\mathrm{d}x = \lim_{\delta\to 0+} \frac{\int_{A_\delta} p_\delta(x)\,\mathrm{d}x}{\int_{B(x,\delta)} p_\delta(x)\,\mathrm{d}x} \\
&= \lim_{\delta\to 0+} \frac{\int_{\|x-x_0\|=\delta\wedge\|x-x'\|\le r} \mathrm{d}\lambda_{m-1} \int_{0\le t\le\delta} q_\delta(t)\,\mathrm{d}t}{\int_{\|x-x_0\|=\delta} \mathrm{d}\lambda_{m-1} \int_{0\le t\le\delta} q_\delta(t)\,\mathrm{d}t} \\
&= \lim_{\delta\to 0+} \frac{\int_{\|x-x_0\|=\delta\wedge\|x-x'\|\le r} \mathrm{d}\lambda_{m-1}}{\int_{\|x-x_0\|=\delta} \mathrm{d}\lambda_{m-1}} = \frac{1}{2}.
\end{aligned} \tag{4}$$

Table 1: The simulation results of the lower bound of the probability $Pr_{\mu_\delta}(f(x) \neq f(x_0))$ with different ϵ and δ. The results are based on 1000000 samples each.

ϵ	δ					
	0.05	0.08	0.1	0.15	0.18	0.2
0.0001	0.2056%	2.6977%	4.9912%	9.0231%	10.1639%	10.4286%
0.00005	6.6341%	14.0125%	16.5768%	18.7768%	18.4909%	17.9706%

Also, the target integration region $\{x \in \mathbb{R}^m \mid \|x - x_0\| \leq \delta \wedge \|x - x'\| \leq r\}$ is always a subset of some semi-ball of $B(x_0, 1)$, so it is easy to see

$$\int_{\|x-x_0\|\leq\delta\wedge\|x-x'\|\leq r} \mu_\delta(\mathrm{d}x) \leq \frac{1}{2}. \tag{5}$$

From Equ. 4 and Equ. 5, we complete the proof. □

Thm. 1 implies that, if the adversarial example x_0 is obtained from the optimization problem 1 and the sampling region is small enough, the lower bound of the probability $Pr_\mu(f(x) \neq f(x_0))$, which is the LCR of an adversarial sample x_0, is almost $1/2$.

In reality, an adversarial example x_0 might not be the exact solution of (1) because some approximation algorithm may be adopted. In this situation, Thm. 1 does not hold anymore. In the following, we assume that the adversarial example x_0 has an ϵ-error to the solution x^* of the problem 1, i.e. $\|x_0 - x^*\| \leq \epsilon$. For simplicity, we use the L_2 norm. Then for $\delta > \epsilon$, Equ. 3 still holds, as Fig. 2(b) shows, which stills provides a lower bound on the LCR of x^*. We illustrate through the following example to get an estimation of this lower bound with different sample radius δ.

Example 1. We use the MNIST dataset as example and the number of dimensions in the input layer is $m = 784$. For the sampling distribution μ_δ, we use the uniform distribution $\mu_\delta = \mathcal{U}(B(x_0, \delta))$. Here we set $\epsilon = \{0.0001, 0.00005\}$ and $r = 0.1$, and Table 1 shows the simulation results of the probability in the RHS of Equ. 3 with different ϵ and δ values. From the results, we could observe that 1) a larger sampling region induces a larger LCR as expected and 2) *a more carefully crafted adversarial sample (with smaller ϵ) has a higher LCR which makes it easier to detect using our approach.*

In contrast to the high LCR of adversarial samples, since most normal samples (with $f(x_0) = c_{x_0}$) are likely far away from the decision boundaries, the probability $Pr_{\mu_\delta}(f(x) \neq f(x_0))$ for a normal sample is very likely to be close to 0. We remark that those normal samples near the classification boundaries may behave like adversarial samples with relatively high LCR. However, such false positive errors are minor (if a DNN is well-trained) and tolerable since it does no harm to report few normal samples as adversarial ones in safety-concerned applications.

Algorithm 1: $DetectAdv(x, f, \kappa_1, \mu, \delta, \alpha, \beta, \sigma)$

```
Let stop = false;
Let c = 0 be the count of mutations that satisfy f(x_m) ≠ f(x);
Let n = 0 be the count of total mutations generated so far;
while !stop do
    Randomly generate a mutation x_m of x from B(x, δ);
    n = n + 1;
    if f(x_m) ≠ f(x) then
        c = c + 1;
        Calculate the SPRT probability ratio as pr;
        if pr ≥ (1−β)/α then
            Accept the hypothesis that κ(x) > μ · κ_1 and report the input as an
              adversarial sample with error bounded by β;
            return;
        if pr ≤ β/(1−α) then
            Accept the hypothesis that κ(x) ≤ μ · κ_1 and report the input as a normal
              sample with error bounded by α;
            return;
```

4 The Detection Algorithm

The implication of the mutation testing effect and its theoretical analysis in previous sections is that we can detect most adversarial samples by identifying those near-boundary samples with high LCR. In this section, we introduce our adversarial sample detection algorithm which is based on the mutation testing effect and inspired by statistical model checking [17]. The basic idea is to determine whether a given input is normal or adversarial by testing whether its LCR is beyond a certain threshold.

Given an input sample x to a DNN f, we determine whether it is a normal sample (i.e., $f(x) = c_x$) or an adversarial sample (i.e., $f(x) \neq c_x$) through mutation testing. Our detection algorithm further reports an error bound on the detection result. Once x is determined to be an adversarial sample, the user may reject the sample and avoid making wrong decisions. In the following, we introduce how our algorithm works.

The algorithm Our detection algorithm is based on hypothesis testing shown in Algorithm 1. It takes a parameter κ_1 which is a threshold of LCR for normal samples, a sampling region δ which is a measure of perturbation to generate mutations from, along with multiple parameters for hypothesis testing. The basic idea is to use acceptance sampling to test the hypothesis that $\kappa(x) > \mu \cdot \kappa_1$ against the hypothesis $\kappa(x) \leq \mu \cdot \kappa_1$ with strength α, β and σ, where μ is a hyper parameter, α, β, and σ are the parameters controlling the strength and indifference region of the test. Recall that $\kappa(x)$ is the LCR of sample x. There are two main methods to decide when the testing process can be stopped, i.e., fixed-size sampling test (FSST) and sequential probability ratio test (SPRT). In our algorithm, we adopt SPRT [3] to dynamically decide when to stop the test (so that it is more likely to stop early).

There are two possible outcomes. If the hypothesis is accepted, it means that the input sample has a higher LCR than a normal sample. Thus, we report that x is an adversarial sample with error bounded by β. If the hypothesis is rejected, we report that the input sample is a normal sample with error bounded by α. If the test does not satisfy a stopping criteria, the algorithm continues to randomly generate a mutation of the provided sample at line 5. The stopping criteria is calculated at line 9. Whenever we observe a label change, pr is calculated as follows.

$$pr = \frac{p_1^c(1-p_1)^{n-c}}{p_0^c(1-p_0)^{n-c}},$$

where $p_1 = \mu \cdot \kappa_1 + \sigma$ and $p_0 = \mu \cdot \kappa_1 - \sigma$. The algorithm stops whenever a hypothesis is accepted either at line 11 or line 14. We remark that SPRT is guaranteed to terminate with probability 1 [3]. On termination of Algorithm 1, we either report the sample as a normal one or an adversarial one with an error bound.

5 Experimental Evaluation

In this section, we present a two-part evaluation of our method. In the first part, we aim to show that most adversarial samples are near and most normal samples are far from the decision boundary by evaluating the difference between κ_{nor} and κ_{adv} (of different kinds of attacks). In the second part, we show the effectiveness and efficiency of LCR and SPRT detection by comparing with state-of-the-art approaches to detect adversarial samples. We also aim to provide some practical guidelines on the parameter selection. All the results and code are available at [29].

Dataset and Models We adopt the MNIST and CIFAR10 dataset, and the set of attacking methods in Cleverhans [24] for our experiments, i.e., FGSM, JSMA, C&W and BB. MNIST and CIFAR10 have 60000/50000 images for training and 10000/10000 images for testing respectively. We obtain a target model for MNIST and CIFAR10 each. The model structures are briefly shown in Table 2 and details can be found in [29]. The models achieve 99.59%/98.99% and 98.84%/81.54% accuracy on the training/testing set for MNIST and CIFAR10 respectively, which both achieve state-of-the-art performance.

Adversarial Samples Generation For each kind of attack, we attempt to generate 5000 adversarial samples according to the parameters shown in Table 2. For FGSM, the parameter controls the scale of the perturbation for the attack. For JSMA, the parameter

Table 2: Experiment settings.

Dataset	#Training	#Testing	Model	Attack parameter/#Adversary			
				FGSM	JSMA	CW	BB
MNIST	60000	10000	LeNet	0.3/5000	0.1/5000	10/5000	0.3/1359
CIFAR10	50000	10000	Deep 12-Layer Convnet [8,19]	0.01/4363	0.1/5000	0.1/4040	0.03/588

Table 3: The confidence interval (99% significance level) of κ_{nor} and κ_{adv} of 1000 images randomly drawn from MNIST and CIFAR10 dataset with 5000 mutations for MNIST and 5000 mutations for CIFAR10 under different attacks.

Dataset	σ	κ_{nor}	κ_{adv}				
			FGSM	JSMA	CW	BB	WL
MNIST	1	0.016±0.004	0.34±0.02	0.45±0.01	0.39±0.01	0.39±0.03	0.33±0.11
	5	0.051±0.004	0.37±0.02	0.47±0.01	0.42±0.01	0.45±0.02	0.37±0.10
	10	0.076±0.005	0.40±0.02	0.49±0.01	0.44±0.01	0.47±0.02	0.40±0.09
CIFAR10	1	0.10±0.02	0.27±0.02	0.50±0.01	0.21±0.01	0.50±0.04	0.38±0.03
	5	0.18±0.02	0.34±0.02	0.55±0.01	0.38±0.01	0.52±0.03	0.41±0.02
	10	0.21±0.01	0.37±0.02	0.57±0.01	0.41±0.01	0.54±0.04	0.43±0.02

controls the maximum distortion of the attack. For C&W, we adopt L_2 attack according to the author's recommendation and set the scale cofficient. For BB, we obtain a substitute model using the 3 Layer DeepNet [24] for MNIST and LeNet for CIFAR10. Since not all attack attempts are successful, in the end we manage to obtain a certain number of adversarial samples for each attack shown in Table 2. Notice that both the models used and the parameters for the attack algorithms are exactly the same as the two baseline methods introduced in the later section for a fair comparison.

5.1 Evaluation on κ_{nor} versus κ_{adv}

In this section, we present the evidence that most adversarial samples are near the decision boundary and most normal samples are away from the decision boundary though evaluating κ_{adv} and κ_{nor} on a set of randomly chosen images. Recall that in our definition, there is no difference between adversarial samples and benign samples which are wrongly labeled by the DNN in the training or testing dataset. We use WL to represent those wrongly-labeled samples later. We randomly select 1000 normal/adversarial samples and vary the sampling region from 1, 5 and 10. For each image x (either normal or adversarial), we randomly generate 5000 mutations to calculate its label change rate $\kappa(x)$. We obtain the confidence interval of κ_{adv} for each attack and κ_{nor} for normal samples by averaging $\kappa(x)$ over all the tested adversarial (using different attacks) and normal images respectively. The detailed results are shown in Table 3. We have the following observations which supports our hypothesis.

Firstly, κ_{nor} is a small value comparable to the training error for a well trained DNN, whereas κ_{adv} is significantly larger than κ_{nor} for all the experimented attacks. This is especially true when we set a small sampling region (e.g., 1) for generating mutations, in which case each of the four evaluated attacks has a κ_{adv} that is 21, 28, 24 and 24 times of κ_{nor} respectively for the MNIST dataset.

Secondly, as we increase the sampling region for generating mutations, both κ_{nor} and κ_{adv} increase (as expected) and the relative distance between κ_{nor} and κ_{adv} reduces. In the following, we measure the relative distance between κ_{nor} and κ_{adv} using their ratio $\kappa_{adv}/\kappa_{nor}$. We can calculate that for all the experimented attacks, a smaller sampling region will result in a larger distance between κ_{adv} and κ_{nor}.

Thirdly, adversarial samples crafted by different attacks have different LCR to perturbations. We can observe that different attack methods have different κ_{adv} values (although all of them are significantly larger than κ_{nor}). As a result, if there is a new attack method, we are unlikely able to know its κ_{adv} value. Thus, it is not a good idea to test against κ_{adv}. However, since 1) κ_{nor} is a relatively stable value for a given DNN and the set of training data, and 2) we know that κ_{adv} is significantly larger than κ_{nor}, we can detect adversarial samples from an unknown attack by testing against κ_{nor}.

5.2 Adversarial Samples Detection

We compare our LCR-based method with two state-of-the-art baselines for detecting adversarial samples. In [8], two joint features, i.e., kernel density estimate (KD) and model uncertainty estimate (MU) are proposed for detecting adversarial samples from artifacts. In [19], local intrinsic dimensionality (LID) is used for the detection and shown to perform better than [8]. We skip the details and interested readers can refer to [8,19]. In the following, we first adopt Receiver Operating Characteristic (ROC) analysis to show LCR is more effective than existing state-of-the-art features. Then, we report the results of our detection algorithm using LCR as the feature for the detection. We also provide some analysis to compare the cost of different approaches as well as some practical guidelines to improve the performance of our detection algorithm.

ROC analysis The ROC curve [7] is a standard method to measure how good a feature is for binary classification. In our context, the ROC curve plots the true positive rate (TPR) against false positive rate (FPR) when using a certain feature (LCR, KD, MU and LID) to distinguish normal and adversarial samples. We further use *Area Under ROC* (AUROC) for a quantitative comparison. The closer AUROC is to 1 (i.e., a perfect feature), the better the feature is. Table 4 shows the AUROC results of LCR compared to state-of-the-art features including KD, MU and LID. The best result among the four features is in bold and the second best result is in italic. We observe that LID is in general better than KD and MU, while LCR achieves the best performance in most cases. For instance, *LCR has the best result in 7 out of 10 cases (among the top-2 results in all cases) and has the best average results.* In many cases like MNIST-WL and MNIST-CW, LCR achieves a perfect or near perfect performance, which is significantly better than the second best feature. In summary, we can confirm that LCR is an effective feature to detect adversarial samples with a high TPR and a low FPR.

Detection We have shown that LCR outperforms state-of-the-art features to detect adversarial samples by ROC analysis. In the following, we evaluate the effectiveness and efficiency of detecting adversarial samples using SPRT with LCR as the feature. The detailed results and a comparison to the baselines are shown in Table 5.

Overall performance On average, our algorithm is able to detect 95.1% (81.2% respectively) adversarial samples generated from the four kinds of attacks for MNIST and (CIFAR10 respectively) dataset with only 26.2 (70.4 respectively) mutations. In Table 5, we provide a complete comparison of our approach with baselines in terms of precision,

Table 4: AUROC results of different features for the detection.

Dataset	CIFAR10				MNIST			
Attack/Feature	KD	MU	LID	LCR	KD	MU	LID	LCR
FGSM	0.5623	0.7783	*0.7924*	**0.8025**	0.8764	0.9598	**0.9877**	*0.9671*
JSMA	0.5627	0.8980	**0.9501**	*0.9450*	0.9076	*0.9893*	0.9727	**0.9919**
CW	0.5682	0.8411	**0.8766**	*0.8647*	0.8919	*0.9828*	0.9682	**0.9970**
BB	0.5428	0.8691	*0.9173*	**0.9417**	0.8561	0.9359	*0.9764*	**0.9930**
WL	0.5603	0.8127	*0.8400*	**0.8477**	0.7596	*0.9932*	0.5283	**1.0000**
Average	0.5593	0.8398	*0.8752*	**0.8803**	0.8583	*0.9722*	0.8866	**0.9898**

recall and detection accuracy with the best results in bold. We could observe that *SPRT with LCR outperforms baseline approaches in terms of almost all metrics.* Note that both state-of-the-art approaches work in a white-box setting which requires the full knowledge of the DNN. Furthermore, they need to train a detector which is shown to be vulnerable to new attacks under a white-box attack setting [5]. On the contrary, LCR works in a black-box setting and is more resilient even under white-box attacks since it only relies on *randomly* mutating the input (a form of randomization which is potentially the most effective approach according to [5]) and obtaining its label.

Effect of μ Intuitively, μ reflects how confident we are that κ_{adv} is significantly larger than κ_{nor}. Fig. 3 shows the effect of μ on the performance of SPRT. We can observe that as we increase μ, we are able to detect adversaries with fewer number of mutations. However, the accuracy of the detection drops slightly. In practice, a smaller μ should be preferred if the system has a higher safety requirement and vice versa.

Wrongly-labeled samples detection We can observe that our algorithm is also able to detect these wrongly-labeled samples similarly to detecting the adversarial samples. This suggests that wrongly-labeled samples are mostly near the decision boundary which are the same as the adversarial samples from a statistical point of view.

Normal samples detection There is a tradeoff between the detection accuracy of normal samples and adversarial samples. From Fig. 3, we could observe that as we increase μ, the accuracy of detecting adversarial samples decreases whereas the accuracy of detecting normal samples increases since we set a larger threshold. This also suggests that there is a minor portion of normal samples which are near the decision boundary.

Table 5: Detection performance compared to baselines.

Metric	MNIST				CIFAR10			
	KD	MU	LID	LCR	KD	MU	LID	LCR
Precision	0.7737	0.9452	0.7395	**0.9728**	0.5588	0.7689	0.8192	**0.834**
Recall	0.8246	**0.9492**	0.7629	0.9317	0.8072	0.7979	0.7758	**0.8101**
Accuracy	0.7906	0.947	0.8502	**0.9508**	0.585	0.7805	0.8026	**0.8121**

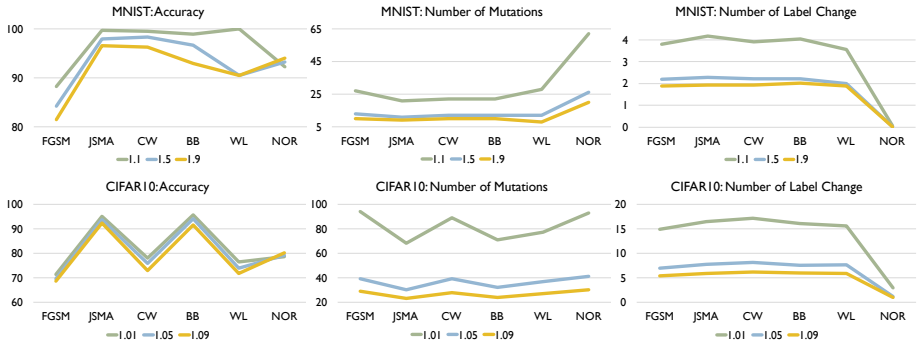


Fig. 3: Effect of μ on the performance of SPRT based on LCR.

Cost analysis The cost of our algorithm mainly consists of two parts, i.e., generating mutated input (denoted by c_g) and performing forward propagation (denoted by c_f) to obtain the label of an input sample by a DNN model. The total cost of detecting an input sample is thus $C = n \cdot (c_g + c_f)$, where n is the number of mutations needed to draw a conclusion for the detection. Given an image x (either from MNIST or CIFAR10), we estimate c_g and c_f by generating 10000 mutated input of x and feed the mutated input to the DNN and calculate the average time cost. Table 6 shows the detailed cost of our approach. We could observe that both obtaining a mutated input (i.e., c_g) and obtaining the label of an the mutated input (i.e., c_f) are reasonably efficient with $c_g = 1.22\ ms$ and $c_f = 0.66\ ms$ for MNIST and $c_g = 2.59\ ms$ and $c_f = 0.85\ ms$ for CIFAR10. Overall, our algorithm is able to detect an adversarial sample of MNIST and CIFAR10 in 49.8 ms and 239.4 ms respectively. One further note is that our algorithm is easy to be paralleled by evaluating multiple mutated inputs at the same time.

Table 6 also shows the cost of the two baseline approaches, which work by extracting the features such as KD, MU and LID, for a single input. We could observe that these baseline features have significantly higher cost than LCR. A close look at the details on extracting the features shows that calculating KD and LID requires us to examine all the samples in the training set to calculate a certain distance, while calculating MU requires us to sample the weights multiple times, get their outputs on the input sample, and compute the uncertainty. In general, the cost of these baseline approaches increases if the training set gets larger and the model gets more accurate, whereas the cost of our approach in the latter case drops because we would need fewer mutations.

Discussions In the following, We discuss several factors which could affect the effectiveness of our approach. The first is model quality. Our method performs better on well-trained models with good generalization. The reason is that fewer normal samples are near the decision boundary for a model of higher quality. The second is the method of adversarial perturbation. Our method is more effective in detect adversarial attacks which generates adversarial samples with smaller perturbations from a normal sample since they are closer to the decision boundary in general. This meas that a more carefully crafted adversarial sample might be easier to detect. The third is sampling region. The optimal sampling region for our detection method might be application-dependent. In reality, users can select a good value by testing over a small set of data.

Table 6: Cost analysis of LCR compared to baselines.

Dataset	KD	MU	LID	LCR	LCR-n
MNIST	2516.8 ms	34.66 ms	2067.7 ms	1.9 ms	26.2
CIFAR10	4886.2 ms	66 ms	4094.7 ms	3.4 ms	70.4

6 Related Works

First of all, our work is devised to detect adversarial samples which can be obtained through many different kinds of adversarial attacks. These attacks can be roughly divided into two categories, i.e., white-box attacks with access to the DNN [10,23,28,6] and black-box attacks without the knowledge of the DNN [25,18]. There are some relevant research in the software engineering community as well. Both white-box and black-box testing strategies have been proposed to identify adversarial samples more efficiently and utilize these samples to improve the original DNN by retraining [28,36,34]. The main problem of these testing approaches is that they provide no guarantees on the robustness against new attacks.

On the defense side, one line of work is to improve the robustness of DNN by adversarial training, which trains models with adversarial samples [10,16,12,32]. A similar line of work is to consider all the perturbations of input samples during training by robust optimization [31,20,22]. These approaches improve the robustness of the DNN to some extent. There are also attempts to formally verify the DNN to provide safety guarantees [13,14,15,35,9]. But they often can only prove pointwise safety around a small region of a given input so far and are in general quite cost-expensive.

Our work falls into detection of adversarial samples. Existing detection algorithms either trains subsidiary models from adversarial samples [21,37] or tests the statistical differences between the training samples and adversarial samples [11,8,19]. Compared to existing detection algorithms, our algorithm does not rely on any specific adversarial samples which generalizes to unknown attacks and is efficient enough to be deployed as runtime monitors in real-time applications.

7 Conclusion

In this work, we propose a new feature, i.e., *L*abel *C*hange *R*ate, to distinguish adversarial samples from normal samples of deep neural networks (DNN). We provide a theoretical analysis on the LCR difference to show the effectiveness of using LCR for the detection. We also design an algorithm for adversarial sample detection based on statistical model checking. Our experiments on broadly adopted datasets show that our algorithm is able to detect adversarial samples with high accuracy and low cost (few mutations needed) compared to state-of-the-art detection methods. Besides, our approach requires no knowledge of the DNN and is not limited to detect a specific kind of attack or a set of available adversarial samples, but can be generalized to a wide range of unknown attacks. It is also efficient enough to be deployed runtime monitors in real-time applications.

References

1. Google's self-driving car caused its first crash. https://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash/, 2018.
2. Self-driving uber kills arizona woman in first fatal crash involving pedestrian. https://www.theguardian.com/technology/2018/mar/19/uber-self-driving-car-kills-woman-arizona-tempe, 2018.
3. A.Wald. *Sequential Analysis*. Wiley, 1947.
4. Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
5. Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM, 2017.
6. Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.
7. Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.
8. Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. Detecting adversarial samples from artifacts. *arXiv preprint arXiv:1703.00410*, 2017.
9. Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.
10. Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.
11. Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280*, 2017.
12. Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.
13. Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.
14. Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.
15. Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Towards proving the adversarial robustness of deep neural networks. *arXiv preprint arXiv:1709.02802*, 2017.
16. Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.
17. Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International conference on runtime verification*, pages 122–135. Springer, 2010.
18. Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.
19. Xingjun Ma, Bo Li, Yisen Wang, Sarah M Erfani, Sudanthi Wijewickrema, Michael E Houle, Grant Schoenebeck, Dawn Song, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *arXiv preprint arXiv:1801.02613*, 2018.

20. Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.
21. Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*, 2017.
22. Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583, 2018.
23. Seyed Mohsen Moosavi Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, number EPFL-CONF-218057, 2016.
24. Ian Goodfellow Reuben Feinman Fartash Faghri Alexander Matyasko Karen Hambardzumyan Yi-Lin Juang Alexey Kurakin Ryan Sheatsley Abhibhav Garg Yen-Chen Lin Nicolas Papernot, Nicholas Carlini. cleverhans v2.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 2017.
25. 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 on Asia Conference on Computer and Communications Security*, pages 506–519. ACM, 2017.
26. Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.
27. Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.
28. Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM, 2017.
29. Jingyi Wang Peixin Zhang. nmutant github repository. `https://github.com/pxzhang94/nMutant`, 2018.
30. Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.
31. Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of neural nets through robust optimization. *arXiv preprint arXiv:1511.05432*, 2015.
32. Aman Sinha, Hongseok Namkoong, and John Duchi. Certifying some distributional robustness with principled adversarial training. 2018.
33. Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.
34. Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. *arXiv preprint arXiv:1708.08559*, 2017.
35. Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Duane Boning, Inderjit S Dhillon, and Luca Daniel. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.
36. Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 408–426. Springer, 2018.
37. Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint arXiv:1704.01155*, 2017.