PEACEPACT: Prioritizing Examples to Accelerate Perturbation-Based Adversary Generation for DNN Classification Testing

Zijie Li*†, Long Zhang*†, Jun Yan‡, Jian Zhang*†, Zhenyu Zhang*§, and T. H. Tse¶
*State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100 190, China
†University of Chinese Academy of Sciences, Beijing 100 039, China
‡Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100 190, China
¶Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong
§Email: zhangzy@ios.ac.cn

***Abstract*—Deep neural networks (DNNs) have been widely used in classification tasks. Studies have shown that DNNs may be fooled by artificial examples known as adversaries. A common technique for testing the robustness of a classification is to apply perturbations (such as random noise) to existing examples and try many of them iteratively, but it is very tedious and time-consuming. In this paper, we propose a technique to select adversaries more effectively. We study the *vulnerability* of examples by exploiting their class distinguishability. In this way, we can evaluate the probability of generating adversaries from each example, and prioritize all the examples accordingly. We have conducted an empirical study using a classic DNN model on four common datasets. The results reveal that the vulnerability of examples has a strong relationship with distinguishability. The effectiveness of our technique is demonstrated through 98.90 to 99.68% improvements in the F-measure.**

I. Introduction

Deep learning has shown promising results in speech recognition [1], image recognition [2], and natural language processing [3].

Deep neural network (DNN for short)is an artificial neural network system with multiple layers between the input layer and the output layer. However, most DNNs heavily depend on the quality of examples since trained DNNs are actually an effective data representation of a given set of examples. Such a trained DNN model may fail to classify a new example correctly in practice. It is important to know whether a DNN model is reliable. As a result, studies about the vulnerability of DNN models become popular. Many of them tried to expose potential errors in a DNN model.

An *adversary* (also known as an *adversarial example*) is an example that the given DNN models cannot classify correctly.

Some previous studies focused on generating adversaries based on existing examples. Goodfellow et al. [4] proposed a framework to estimate generative models via an adversarial process, and many variants have been proposed. They are called generative adversarial networks (GANs), which form a popular way to generate adversaries.

Researchers found that they can perturb existing examples to generate adversaries to test the DNN models and improve their robustness. Tabacof and Eduardo [5] generated adversaries for shallow and deep network classifiers on MNIST [6], [7] and ImageNet [8] datasets and probed the pixel space of adversaries by using perturbation of various distribution and intensity.

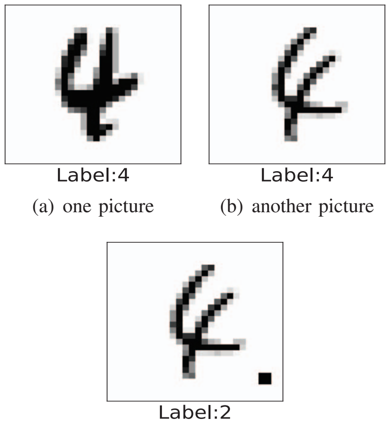
However, adding too much perturbation can make the resultant example either meaningless or easily detected by a defense mechanism [9]. For example, Akhtar et al. [9] proposed a defense framework appending a pre-input layer against the adversarial attacks. It is useful to study the minimal amount of perturbation to achieve DNN attacks. The ad hoc practice of adding random noise or trying many brute-force examples may not be sufficient.

Nevertheless, adversaries are seldom easy to find. In practice, existing work [10], [4], [11] randomly selected examples from the example set, perturbed them, and used the generated adversaries to attack a DNN model or improve its robustness. It often needs a long process of perturbation trials before a success in obtaining an effective adversary. Many of them ignored the underlying difference of examples in responding to perturbation. We realize that instead of perturbing a long list of examples, it may be more effective if examples sensitive to perturbation can be picked in advance.

In this paper, we will propose a prioritization technique named PEACEPACT (for **P**rioritizing **E**xamples to **A**c**CE**lerate **P**erturbation-Based **A**dversary generation for DNN **C**lassification **T**esting) to tackle the problem. Instead of perturbing every example from a candidate set, our technique finds examples that are of high vulnerability to perturbations. PEACEPACT is a black-box technique, and it works without knowing the structure of the machine learning model.

First, PEACEPACT collects the probability vectors of original examples in a given DNN model. Here, a probability vector indicates the probabilities an example being attributed to different classes. PEACEPACT next uses the distance of the probability vectors to compute the vulnerability values of original examples. Such vulnerability captures the probability of example class changing after perturbation. Finally, PEACEPACT ranks the original examples according

978-1-7281-8913-0/20/$31.00 ©2020 IEEE
DOI 10.1109/QRS51102.2020.00059



(a) one picture (b) another picture

(c) an adversary generated from (b)

Fig. 1. Two Pictures and One Adversary

to their vulnerability values. The examples having higher priorities are expected to be more vulnerable to perturbations and thus easier to generate adversaries.

A test on dataset MNIST [7] to train a LeNet-5 model shows that different examples have dissimilar vulnerability levels in response to perturbations, and the vulnerability is related to the class distinguishability of each example. Such an observation matches the basic assumption of our technique. To validate the effectiveness of PEACEPACT, we use four common datasets to build up a DNN model. The experiment results show that the more sensitive an example is to perturbations, the higher a rank is given by PEACEPACT. It shows that our technique is effective in finding vulnerable examples.

The main contribution of our work is fourfold.

- First, we conducted an empirical study to confirm that the vulnerability levels of different examples vary from one another. As a result, it is necessary to find promising examples.
- Second, our study also unveiled the relationship between vulnerability and class distinguishability of examples.
- Third, we proposed a prioritization technique that can effectively find the examples having higher vulnerability to perturbations.
- Fourth, we conducted an empirical research to validate the effectiveness of our technique and evaluate its robustness.

The rest of the paper is organized as follows. Section II uses the observations on examples from a common data set to motivate our work. Section III presents our technique and elaborates on it in detail. Section IV evaluates our technique with experiments. We introduce the related work in Section V. Finally, we conclude our work in Section VI.

II. Motivation

A. A Typical DNN Problem

MNIST [7] is a large database of handwritten digits that are commonly used for training various image processing systems. We used MNIST to train LeNet-5 model, randomly chose two pictures from the data set, and showed them in Figure 1 (a) and (b). LeNet-5 outputs the *probability vectors* of the two pictures as follows.

$$\langle\ 0,\ 0,\ 0,\ 0,\ 0.95,\ 0.01,\ 0,\ 0,\ 0.01,\ 0.03\ \rangle$$
$$\langle\ 0,\ 0,\ 0.18,\ 0,\ 0.81,\ 0,\ 0,\ 0,\ 0.01,\ 0\ \rangle$$

Let us take the first vector to illustrate. It shows the classification result of the DNN model with respect to the first image. The DNN model deems that the probability of the picture containing a number 4 is 0.95 (i.e., 95%), while the probabilities it containing a number 5 and 8 are both 0.01 (i.e., 1%). The only difference is that the DNN has less confidence (with probability 0.81) to recognize the second picture than the first one (with probability 0.95).

B. Adversarial Attacks to the Above DNN Model

Fuzz testing [12], [13], [14] is an automated software testing technique that involves providing invalid, unexpected, or random data as inputs to a computer program. It is extensively used to test DNNs. We use fuzz testing to generate adversaries. For each of the two pictures, we randomly choose a 2×2 pixel region and reverse the value of each pixel in the region. We repeat such a step to generate 50 new pictures for each picture. Among the 50 new examples generated for the first picture, there are five adversaries; while eight adversaries are generated for the second picture.

We have the observation that the effective adversaries with respect to picture (a) (i.e., Figure 1 (a)) are less than those of picture (b) (i.e., Figure 1 (b)). We repeated ten times to check the difference among the numbers of effective adversaries with respect to picture (a) and picture (b), and always had the same observations. Moreover, in the ten repeated tests, the effective adversaries with respect to picture (b) are mostly recognized as a digit number 2 (shown in Figure 1 (c)).

C. The Inspiration

We attribute such observations to the difference in the probability vectors of the two pictures.

First, we revisit the probability vectors of the two pictures and find that DNN has more confidence that picture (a) is a digit number 4 (95% probability), and less confidence that picture (b) is a digit number 4 (81% probability). That explains why it is more difficult to generate adversaries from picture (a). Our basic understanding is that the more confidently DNN recognizes an example, the more difficult it is to generate adversaries from it.

Second, we focus on the probability vector of the second picture. Though DNN has high confidence that picture (b) is a digit number 4 (81% probability), it still has nonnegligible confidence that it is a digit number 2 (18% probability). That explains why most of the adversaries generated from picture (b) are mislabeled as a digit number 2. Our basic understanding is that the less different are the two highest probabilities with respect to an example, the easier it will be to generate adversaries from it. If we take the difference between the two highest probabilities as a metric to measure the robustness of the DNN classification in recognizing pictures (a) and (b), we obtain $0.95 - 0.03 = 0.92$ and $0.81 - 0.18 = 0.63$. Apparently, we want to choose picture (b) to generate adversaries since it seems more vulnerable to noise.

The observation inspires us that examples are sensitive to noise, and their vulnerabilities may bear a relationship with their probability vectors. Our basic idea is to (1) collect the probability vectors of examples, (2) use a distance metric to assess class distinguishability of examples, (3) reference it to evaluate the vulnerability of examples, and (4) rank the examples accordingly with the aim of selecting most promising candidates for adversary generation.

In next section, we will elaborate on our model to generalize the basic idea.

III. Our Technique

In this section, we present a novel technique to address the efficiency of perturbation-based adversary generation against DNNs.

We have motivated that examples are not equally effective in generating adversaries. Different from the conventional RAND technique that applies random examples without distinguishing one from another, our technique can pick out promising examples that have higher chances of producing adversaries, before the actual generation process begins.

A. Problem Settings

We focus on the problem of adversary generation, which is based on the DNNs with the capability of n-class classification. We describe the problem in the scenario where we have a training set S for a DNN model C. Here, all the examples in S are used to train C. We use $s_i \in S$ to express an input to C, and use p_i to represent the probability vector produced by C for each given s_i. This is expressed as:

$$p_i = C(s_i) \tag{1}$$

Here, p_i is a vector $\langle p_i^1,\ p_i^2,\ \ldots,\ p_i^n \rangle$. Each element of p_i represents the probability that s_i is of class 1, 2, ..., or n.

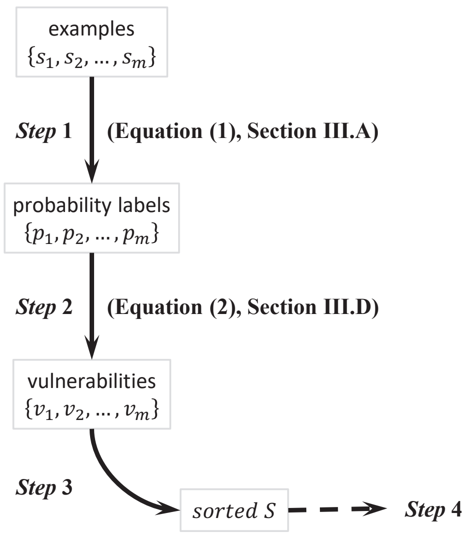
In practice, such a probability vector can be obtained by using a SoftMax function, which, as a tradition of machine learning, takes as input a vector of real numbers, and normalizes it into a probability distribution consisting of probabilities proportional to the exponents of the input numbers. The *class* of example s_i is usually determined as:

$$class(s_i) = \underset{j}{\text{argmax}} \left\{ p_i^j \right\}.$$

where "argmax" denotes the index of the largest element in the vector.

Suppose that we are using S as a candidate set and plan to perturb each example $s_i \in S$ with the aim of generating adversaries.

In practice, an adversarial attack technique is often used for such a purpose, which accepts any s_i as input and modifies it by adding perturbations. If the class of the modified s_i is different from $class(s_i)$, we deem that an adversary is found; otherwise, the process continues to try another input.



examples
$\{s_1, s_2, \ldots, s_m\}$
Step 1
(Equation (1), Section III.A)
probability labels
$\{p_1, p_2, \ldots, p_m\}$
Step 2
(Equation (2), Section III.D)
vulnerabilities
$\{v_1, v_2, \ldots, v_m\}$
Step 3
sorted S
Step 4

Fig. 2. The workflow of our PEACEPACT method

Existing studies focus on developing effective attack techniques that try examples without differentiating them. Such a generation process accepts a list of examples, working on each example sequentially. We realize that the choice of candidate examples plays an important role in the success of attacking. Instead of randomly trying all the examples by brute force, it would be more efficient if we can manage to select those examples that have high chances to generate adversaries from. For the purpose of comparison, we let the input of the adversary generation process be the set of examples in a random order in the conventional manner, and aim at finding a ranked list that helps find adversaries more quickly.

In next section, we will propose a novel technique that prioritizes examples to select promising candidates for adversary generation.

B. Our PEACEPACT *Technique*

We design our technique to compute the vulnerability for each example and rank the examples accordingly.

First, PEACEPACT collects information about the examples by inputting each example to the DNN model and getting its probability vector, which determines the class of that example. PEACEPACT then assesses each probability vector to evaluate the vulnerability of the corresponding example to attack, which estimates the possibility of generating adversaries by introducing perturbation to the example. The higher the value of vulnerability, the more easily it is deemed to generate an adversary from the corresponding example. After that, all examples are sorted according to their vulnerability values to produce a ranked list such that the examples with higher vulnerabilities have higher ranks.

To start the process of our technique, we assume that a trained model C and a set of candidate examples S are ready. Figure 2 illustrates the following process, each step of which is shown using an arrow.

Step 1: Input each examples $s_i \in S$ to C to collect the probability vector p_i;
Step 2: Use a metric to evaluate the vulnerability v_i for each probability vector p_i;
Step 3: Sort s_i in descending order of v_i to obtain a ranked list, based on the results of last step.

In Figure 2, we also use a dashed arrow to show a follow-up *Step* 4 that uses the ranked list of s_i to generate adversaries. However, it is beyond the scope of the model development. We next elaborate on each step of PEACEPACT.

C. Step 1 – Data collection

PEACEPACT aims at finding the promising examples. Without loss of generality, we solely collect the basic information of the DNN, and use that as input to our technique.

Since PEACEPACT is to evaluate each example s_i of the DNN model C, we assume that the probability vectors of s_i are also available. As introduced in Section III-A, p_i is further expressed as $\langle p_i^1,\ p_i^2,\ \ldots,\ p_i^n\rangle$. Here, p_i^j is the probability of s_i being of class j.

Note that for any $j \in [1, n]$, we have $p_i^j \geq 0$ and $\sum p_i^j = 1$. It means that any s_i belongs to one and only one class.

D. Step 2 – Vulnerability

PEACEPACT evaluates the vulnerability of an example by perturbing it and assessing how easy to generate an adversary from it.

Let N stands for the perturbation added to s_i by an adversarial attack technique, and $t_i = s_i \oplus N$ to express the new example generated. We use the following probability to capture the *vulnerability* of s_i to perturbation N.

$$v_i = \text{Prob}\Big(class\,(s_i) \neq class\,(t_i)\Big) \tag{2}$$

It captures the probability that s_i is of a different class after perturbing. "Prob" means the probability.

We next use $q_i = class(t_i)$ to express the probability vector of t_i according to Equation (1). The probability in Equation (2) can be further expressed as $\text{Prob}\left(\underset{j}{\text{argmax}}\left\{p_i^j\right\} \neq \underset{j}{\text{argmax}}\left\{q_i^j\right\}\right)$. However, due to the existence of the quantifier argmax, it is not possible to directly assess the above probability. Without any wild guess to the effect of N on *class* s_i, we based on the following intuitive assumption to develop our model.

$S1$: The less distinguishable are the probabilities p_i^1, p_i^2, …, and p_i^n, the more probably will the class inferred from the example s_i vary after perturbing.

It is interpreted as follows. For an example s_i having little distinguishable probabilities $p_i^1, p_i^2, \ldots, p_i^n$, it means that all the probabilities are close to one another. Because adding perturbation N may have unpredictable impact on each probability value, it is very likely the class of t_i may vary from that of s_i. On the contrary, for an example s_i having huge distinguishable probabilities, the highest probability is relatively greater than the other probabilities. It is not easy to be caught up and very likely still the maximal one after perturbing. As a result, the class of s_i may not change.

We reinterpret $S1$ as $v_i \propto \Delta(p_i)$. Here, $\Delta(p_i)$ is the *class distinguishability distance*, which calculates the distance among p_i^1, p_i^2, …, and p_i^n. We will simply refer to $\Delta(p_i)$ as *distance* when there is no ambiguity.

In next section, we propose the distance metric we used to measure the class distinguishability, and evaluate the class distinguishability with respect to an example.

E. Step 3 – Distance metrics

A distance metric Δ evaluates the vulnerability v_i of an example s_i, based on the difference among the probabilities p_i with respect to s_i.

Given the probability vector $\langle p_i^1,\ p_i^2,\ \ldots,\ p_i^n\rangle$, we define the distance metric PD in Equation (3).Suppose there are n values in the given probability vector, PD is the sum of the probability differences with relevant weights.

$$\texttt{PD} : \Delta(p_i) = \sum_{j=1}^{n-1} \frac{{p'}_i^j - {p'}_i^{j+1}}{j} \tag{3}$$

$$\tag{4}$$

Here, $\langle {p'}_i^1,\ {p'}_i^2,\ \ldots,\ {p'}_i^n\rangle$ is the list obtained by sorting $\langle p_i^1$, $p_i^2, \ldots, p_i^n\rangle$ in descending order. The average of ${p'}_i^1$, ${p'}_i^2$, …, and ${p'}_i^n$ is denoted as $\overline{p'}_i$.

F. Step 4 – Sorting the examples

With distance metric PD, we thus evaluate the class distinguishability of each example s_i using its probability vector p_i. The less the distance $\Delta(p_i)$, the less distinguishable the probabilities $p_i^1, p_i^2, \ldots, p_i^n$ is, and the more vulnerable s_i is to noise.

We then sort the examples in ascending order of their distance values. In such a way, we avoid directly measuring the vulnerabilities of s_i on N, and manage to prioritize the examples susceptible to noise among all examples. In the resultant ranked list, the examples having high chance to generate adversaries are given high ranks hopefully.

IV. Evaluation

This section gives the experiment to verify our technique. We first describe the experiment design. Then, we introduce the metrics used to evaluate our technique. Finally, we present the results of the experiment.

A. Data sets

Our experiment is based on four common data sets, namely MNIST [7], fashion-MNIST [18] (abbreviated as f-MNIST), CIFAR-10 [19], and CIFAR-100 [19].

MNIST and f-MNIST are database of handwritten digis. CIFAR-10 and CIFAR-100 are color images. More detailed information about the data sets is listed in Table I.

TABLE I
DATA SETS

	Data sets			
	MNIST	f-MNIST	CIFAR-10	CIFAR-100
# of samples	70 000	70 000	60 000	60 000
Size of image	28×28	28×28	32×32	32×32
# of categories	10	10	10	100

TABLE II
MODEL ARCHITECTURES

	DNN models Net-1
Convolution + ReLU	$5 \times 5 \times 64$
Max pooling	3×3
Convolution + ReLU	$5 \times 5 \times 64$
Max pooling	3×3
Convolution + ReLU	$5 \times 5 \times 64$
Max pooling	3×3
Fully connected + ReLU	512
Fully connected + ReLU	512
Fully connected + ReLU	192

1) Generation of C: In our experiment, we use the entire training sets of each dataset to train DNN models. In particular, we train a classic network model with the four data sets. It is named Net-1 in the experiment, whose model architecture is listed out in Table II. For Net-1, there are three groups of convolution layers, each activated with the Rectifier Linear Unit (ReLU) function. The size of the convolution layers is $5 \times 5 \times 64$. There is a 3×3 max pooling layer following each convolution layer. After the convolution layers, we use three full connection layers with dimensions 512, 512, and 192, respectively.

The hyper-parameters used for the four data sets (without differentiating DNN models) are listed out in Table III. They are the recommended settings and have been used in existing previous studies. There are 512 train samples in a train batch. For MNIST and f-MNIST, we train the model in the learning rate of 0.1 for 20 epochs. The train process of CIFAR-10 and CIFAR-100 is exhibited similarly in table III.

B. Relationship between vulnerability and probability of examples

This test is to understand the correlation between probability label of examples and their vulnerabilities. We apply perturbation on each example, and use a metrics Eff to evaluates its vulnerability to perturbations.

TABLE III
TRAINING PARAMETERS

	Data sets			
	MNIST	f-MNIST	CIFAR-10	CIFAR-100
Learning rate	0.1	0.1	0.01	0.01
Batch size	512	512	512	512
Epochs	20	20	60	60

Similar to Su et al. [12], we also encode the perturbation into an optimized array, albeit using a different mechanism. In the experiment, each perturbation holds x-y coordinates and RGB value of the perturbation, but modifies four (2×2) pixels. The perturbation is generated by using the random function `numpy.random.random` of programming language Python.

Calculating the ratio of effective adversaries among generated examples is the straightforward way to evaluate the vulnerability of examples. The higher the ratio of effective adversaries generated, the higher the example is vulnerable to perturbations. For example, given an example s_i, a DNN C, and a perturbation technique, M new examples are generated from s_i. Among the M new examples, M_e of them are effective adversaries that can fool the given C. The *ratio of effectiveness* is calculated as:

$$\text{Eff}(s_i) = \frac{M_e}{M}.$$

In this test, M is set to be 100 throughout the experiment.

After calculating $\text{Eff}(s_i)$ for each example s_i, we calculate a distance $\Delta(p_i)$ for s_i by using the distance metrics PD, and compute the correlation between $\text{Eff}(s_i)$ and $\Delta(p_i)$. If a higher correlation can be found, the distance metric will be deemed more reliable to predict the vulnerability of examples.

In our experiment, there are four image datasets and a DNN model. We conduct an exponential fitting on the relationship between the distance $\Delta(p_i)$ and the value of Eff. The fitted results are also listed out in Table IV, and V. Take the result of Net-1 on MNIST as example. A function $f(x) = 0.49e^{-90.37x}$ gives the best fitting. We further conduct a correlation coefficient analysis on the fitting and obtain a strong correlation of 0.97.

Finally, the conclusion is as follows:

conclusion: Most examples show sensitivity to perturbation. The vulnerability shows a monotone non-linear inverse relationship with the class distinguishability distance.

C. Effectiveness comparison with random

This test is designed to answer whether PEACEPACT is effective in finding the promising examples in terms of generating effective adversaries. It compares the effectiveness of our PEACEPACT technique, which prioritizes examples, with that of the peer RAND technique, which selects examples in a random manner.

We follow [15] to adopt F-measure, which calculates the expected number of generations to obtain an effective adversary. For example, given a list of example candidates L, a DNN C, and a perturbation technique, the generation process will (1) perturb the first example to generate M examples; (2) if none of the M examples is an adversary, the process moves to the next example in the list and redo (1); (3) otherwise, if any of the M examples is an adversary, the generation process stops. Suppose the process stops when operating the M_0-th

TABLE IV
FITTED FUNCTIONS BETWEEN DISTANCE AND EFFECTIVENESS

	Net-1
MNIST	$f(x) = 0.49 \cdot e^{-90.37x}$
f-MNIST	$f(x) = 0.47 \cdot e^{-13.27x}$
Cifar-10	$f(x) = 0.42 \cdot e^{-20.76x}$
Cifar-100	$f(x) = 0.43 \cdot e^{-8.52x}$

TABLE V
CORRELATION BETWEEN DATA POINTS AND FITTING FUNCTION

	Net-1
MNIST	0.97
f-MNIST	0.98
Cifar-10	0.97
Cifar-100	0.97

example, i.e., at least one adversary is generated from it, the *F-measure* is calculated as:

$$\text{F-measure}(L) = M_0.$$

The lower the F-measure value, the higher quality the ranked list of example is of.

In this test, the effectiveness of PEACEPACT and RAND in revealing adversaries for a DNN is evaluated using the F-measure. First, we trained the DNN model on four image datasets. We randomly select 1000 examples to initialize a set of example candidates S_C. We then shuffle S_C to obtain a list of example candidates L_R in the RAND manner, and let PEACEPACT prioritize examples in S_C to obtain a ranked list of example candidates L_P. We thus calculate the F-measure values for L_R and L_P separately. Note that for each given examples, we use the attack strategy in above test to generate $M = 100$ adversaries, and average the results of ten individual tests to avoid experiment bias.

Though we expect a lower F-measure value for PEACEPACT than that of RAND, we are also interested to contrast the effectiveness of RAND and PEACEPACT. We introduce the Inc metrics to measure the effective improvement in terms of F-measure, from RAND to PEACEPACT. The effectiveness improvement (Inc in short) is calculated as follows,

$$\text{Inc} = \frac{\text{F-measure}(L_R) - \text{F-measure}(L_P)}{\text{F-measure}(L_R)} \times 100\%$$

A positive Inc value denotes that PEACEPACT is effective in finding vulnerable examples by prioritizing them first. A negative Inc value denotes that PEACEPACT has a negative effectiveness by doing so. A zero-valued Inc denotes that PEACEPACT has no difference from the random manner.

The experiment results of PEACEPACT and RAND in terms of F-measure and Inc are shown in Table VI.

Table VI lists out the effectiveness of PEACEPACT and RAND on each data set and each model. Let us take the first cell to illustrate. It shows that the results of PEACEPACT and RAND on the Net-1 model, and the MNIST data set

TABLE VI
CONTRASTING PEACEPACT AND RAND IN F-MEASURE

		Net-1
MNIST	PEACEPACT	9.99
	RAND	908.52
	Inc	98.90%
f-MNIST	PEACEPACT	1.83
	RAND	488.45
	Inc	99.63%
Cifar-10	PEACEPACT	1.94
	RAND	609.31
	Inc	99.68%
Cifar-100	PEACEPACT	1.54
	RAND	287.74
	Inc	99.46%

are 7.13 and 144.82, respectively. It means that, on average, 7.13 and 144.82 examples are needed to be evaluated before an effective adversary is generated, by PEACEPACT and RAND, respectively. We further compute the effectiveness improvement from RAND to PEACEPACT, which is calculated as $\text{Inc} = \frac{144.82 - 7.13}{144.82} \times 100\% = 95.07\%$.

We further check the Inc values with respect to all tests and list them out in Table VI. Our observation is that they range from 98.90 to 99.68%, which shows that PEACEPACT is more effective than the random manner in finding vulnerable examples. Finally, the conclusion as follows:

conclusion: Our technique is effective to select examples vulnerable to permutations. It empirically shows 98.90 to 99.68% improvements over the standard random manner.

D. Threats to Validity

The choice of attack strategy, the DNN model, and data sets could be a threat to validity. Different attack strategies may use dissimilar perturbations to generate adversaries. Dissimilar perturbations, such as randomly modifying 4 pixels, may lead to different observations. We conduct our experiment on representative image datasets with CNN network. Different datasets and architectures of models, i.e. speech recognition and RNN models, may have dissimilar result.

As for the evaluation metrics, we use F-measure [15] to evaluate our method. It denotes how many test cases are executed before successfully attacking a system. In order to judge the effectiveness of distance metrics, an alternative metric is the weighted average percentage of fault detected (APFD) [16], [17] in test case prioritization.

V. Related Work

A. Adversary Generation and Defense

Machine learning models are often vulnerable to adversarial manipulation of their input intended to cause incorrect classification [20]. For example, Moosavi-Dezfooli et al. [21] showed the existence of "universal noise", fooling a network classifier on most images.

In order to assist the test of machine learning models, people design different techniques to generate adversaries. Apart from GAN [4] mentioned in Section I, there has been much work on generating adversaries. Tian et al. [22] proposed DeepTest, which can generate realistic synthetic images by applying image transformations, scale, shear, and rotation on original images. Zhang et al. [23] proposed an unsupervised framework to generate semantic-equivalent adversaries, which are used to test the consistency of autonomous driving systems across different scenes.

Most of these adversarial attacks rarely consider which examples are more likely to survive. Our technique can effectively prioritize the examples to accelerate the process of adversary generation. It is able to work with the generation techniques mentioned above.

The techniques to prevent models from being attacked by adversaries have been studied.

While defenses that cause obfuscated gradients appear to survive iterative optimization based attacks, Athalye et al. [24] found that defenses relying on this effect may be circumvented. They developed attack strategies to overcome it. Tramèr et. al. [25] found that adversarial training remains vulnerable to black-box attacks and introduced "Ensemble Adversarial Training", which augments training data with perturbations transferred from other models. Their techniques yielded models with strong robustness to black-box attacks. Akhtar et al. [9] proposed a defense framework against adversarial attacks generated using universal noise. Bhagoji et al. [26] proposed to compress input data using Principal Component Analysis for adversarial robustness. Mekala et al. [27] applied metamorphic test to detect adversarial attacks. Distillation was introduced by Hinton et al. [28] as a training procedure to transfer knowledge of a more complex network to a smaller network. Papernot et al. [29] exploited the notion of "distillation" [28] to make DNNs robust against adversarial attacks.

Since our technique is designed to accelerate adversary generation, we are also interested in the integrating with adversary defenses.

B. Test of Machine Learning Models

Apart from generating adversaries, there have been other strategies of test machine learning models.

Pei et al. [30] proposed the concept of neuron coverage and the first white-box framework DeepXplore for systematically testing real-world DL systems to generate test inputs for a deep learning system. Ma et al. [31] adopted mutation test in deep learning system. Both the train data and the model are injected faults, and the quality of test data are evaluated by the extent to which the injected bugs are detected. Both the work focuses on white-box test, in which the information of the structure of neural network is necessary. We do not compare the effectiveness of our work against theirs, becasue we are having a black-box technique.

Wu et al. [32] used noise metamorphic relation pattern in the verification and validation of machine translation and named entity recognition systems.

C. Prioritization in Regression Testing and Machine Learning

Test case prioritization (TCP) [33], [34] has long been popular in regression testing with a view to avoiding the duplication of time and effort. Jiang et al. [35] examined 16 state-of-the-art TCP techniques in statistical fault localization, and revealed that strategy and time-cost contribute more to effectiveness than the coverage granularity does.

Prioritization techniques have been utilized in the test of machine learning methods. Ma et al. [36] proposed a set of multi-granularity criteria for testing in the DL system, which could be used in prioritizing tests based on coverage. Byun et al. [37] prioritized input data based on the analysis of the DNN's sentiment, which includes confidence, uncertainty, and surprise.

The experiment demonstrated that it is useful to reveal the weakness of model by identifying suspicious inputs. Jiang et al. [38] prioritized test cases with high loss to accelerate training of deep learning model.

Jacob et al. [39] utilized greedy algorithm to select representative subset of data.

Our technique is different from theirs because we prioritize examples dedicated for adversary generation.

VI. Conclusion

DNNs have been widely used in natural language processing, computer vision, and image recognition. Related work has shown that they can be fooled by artificial examples and many attack models have been proposed to generate such adversaries. A popular trend to test and detect adversary generation is to try many existing examples by applying perturbations to them. In practice, such a process can be improved if vulnerable examples could be given high priority with the aim of accelerating the generation of adversaries. However, inadequate studies are carried out in this direction.

In this paper, we studied the correlation between the vulnerability and the class distinguishability of examples. We proposed a prioritization technique for DNNs that evaluates the vulnerability of examples by referencing their class distinguishability, and prioritize the examples accordingly. We conducted a controlled experiment on a classic model over four common datasets. The experimental result confirmed that examples are sensitive to perturbations and the vulnerability is related to their class distinguishability. The experiment also validated the correctness of our technique, and showed that applying the ranked list of examples generated from our technique can save 98.90 to 99.68% of the generation cost, compared to the conventional random manner.

One future work is to study the quality of the adversaries generated and the influence of other distance metrics. Future work also includes a thorough study on other datasets and other deep learning models.

REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[2] J. Schmidhuber, "Deep learning in Neural Networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[3] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th International Conference on Machine Learning (ICML "08)*. ACM, 2008, pp. 160–167.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems (NIPS "14)*, 2014, pp. 2672–2680.

[5] P. Tabacof and E. Valle "Exploring the space of adversarial images," in *International Joint Conference on Neural Network (IJCNN "16)*, 2016, pp. 426–433.

[6] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[7] Y. LeCun, "The MNIST database of handwritten digits," *http://yann.lecun.com/exdb/mnist/*, 1998.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR "09)*. IEEE, 2009, pp. 248–255.

[9] N. Akhtar, J. Liu, and A. Mian, "Defense against universal adversarial perturbations," *arXiv preprint arXiv:1711.05929*, 2017.

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

[11] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 IEEE Symposium on Security and Privacy (SP "17)*. IEEE, 2017, pp. 39–57.

[12] J. Su, D. V. Vargas, and S. Kouichi, "One pixel attack for fooling deep neural networks," *arXiv preprint arXiv:1710.08864*, 2017.

[13] A. Odena and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," *arXiv preprint arXiv:1807.10875*, 2018.

[14] D. She, K. Pei, D. Epstein, J. Yang, B. Ray, and S. Jana, "Neuzz: Efficient fuzzing with neural program learning," *arXiv preprint arXiv:1807.05620*, 2018.

[15] T. Y. Chen, H. Leung, and I. Mak, "Adaptive random testing," in *Proceedings of the 9th Asian Computing Science Conference (ASIAN "04)*. Springer, 2004, pp. 320–329.

[16] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Test case prioritization: An empirical study," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM "99)*. IEEE, 1999, pp. 179–188.

[17] S. Elbaum, A. Malishevsky, and G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization," in *Proceedings of the 23rd International Conference on Software Engineering (ICSE "01)*. IEEE Computer Society, 2001, pp. 329–338.

[18] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[19] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Citeseer*, 2009.

[20] N. Dalvi, P. Domingos, S. Sanghai, D. Verma *et al.*, "Adversarial classification," in *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD "04)*. ACM, 2004, pp. 99–108.

[21] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," *arXiv preprint arXiv:1708.08559*, 2017.

[22] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering (ICSE "18)*. ACM, 2018, pp. 303–314.

[23] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE "18)*, ACM, 2018,pp.132-142.

[24] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples," *arXiv preprint arXiv:1802.00420*, 2018.

[25] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[26] A. N. Bhagoji, D. Cullina, C. Sitawarin, and P. Mittal, "Enhancing robustness of machine learning systems via data transformations," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS "18)*. IEEE, 2018, pp. 1–5.

[27] R.R. Mekala, G.E. Magnusson, A. Porter, M. Lindvall, and M. Diep, "Metamorphic detection of adversarial examples in deep learning models with affine transformations," in *Proceedings of the IEEE/ACM 4th International Workshop on Metamorphic Testing (MET "19), in conjunction with ICSE)*. 2019, pp. 55–62.

[28] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[29] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *2016 IEEE Symposium on Security and Privacy (SP "16)*. IEEE, 2016, pp. 582–597.

[30] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles (SOSP "17)*. ACM, 2017, pp. 1–18.

[31] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao *et al.*, "DeepMutation: Mutation testing of deep learning systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE "18)*. IEEE, 2018, pp. 100–111.

[32] C. Wu, L. Sun, and Z.Q. Zhou, "The impact of a dot: Case studies of a noise metamorphic relation pattern," in *Proceedings of the IEEE/ACM 4th International Workshop on Metamorphic Testing(MET "19), in conjuction with ICSE*. IEEE, 2019, pp. 17–23.

[33] S. G. Elbaum, A. G. Malishevsky, and G. Rothermel,"Test case prioritization: A family of empirical studies," in *IEEE Transactions on Software Engineering*, vol. 28, no. 2, pp. 159–182, 2002.

[34] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Transactions on Software Engineering*, vol. 27, no. 10, pp. 929–948, 2001.

[35] B. Jiang, Z. Zhang, W. K. Chan, T. H. Tse, and T. Y. Chen, "How well does test case prioritization integrate with statistical fault localization?" *Information and Software Technology*, vol. 54, no. 7, pp. 739–758, 2012.

[36] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "DeepGauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE "18)*. ACM, 2018, pp. 120–131.

[37] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Input prioritization for testing neural networks," in *2019 IEEE International Conference On Artificial Intelligence Testing (AITest "19)*. IEEE, 2019, pp. 63–70.

[38] A.H. Jiang, H. Angela, et al. "Accelerating Deep Learning by Focusing on the Biggest Losers." *arXiv preprint arXiv:1910.00762*, 2019.

[39] S. Jacob, J. Bilmes, and W.S. Noble."apricot: Submodular selection for data summarization in Python." *arXiv preprint arXiv:1906.03543*, 2019.