Adversary Resistant Deep Neural Networks with an Application to Malware Detection

Qinglong Wang[12*], Wenbo Guo[1], Kaixuan Zhang[1], Alexander G. Ororbia II[1],
Xinyu Xing[1], Xue Liu[2], C. Lee Giles[1]
[1]Pennsylvania State University, [2]McGill University

ABSTRACT

Outside the highly publicized victories in the game of Go, there have been numerous successful applications of deep learning in the fields of information retrieval, computer vision, and speech recognition. In cybersecurity, an increasing number of companies have begun exploring the use of deep learning (DL) in a variety of security tasks with malware detection among the more popular. These companies claim that deep neural networks (DNNs) could help turn the tide in the war against malware infection. However, DNNs are vulnerable to adversarial samples, a shortcoming that plagues most, if not all, statistical and machine learning models. Recent research has demonstrated that those with malicious intent can easily circumvent deep learning-powered malware detection by exploiting this weakness.

To address this problem, previous work developed defense mechanisms that are based on augmenting training data or enhancing model complexity. However, after analyzing DNN susceptibility to adversarial samples, we discover that the current defense mechanisms are limited and, more importantly, cannot provide theoretical guarantees of robustness against adversarial sampled-based attacks. As such, we propose a new adversary resistant technique that obstructs attackers from constructing impactful adversarial samples by randomly nullifying features within data vectors. Our proposed technique is evaluated on a real world dataset with 14,679 malware variants and 17,399 benign programs. We theoretically validate the robustness of our technique, and empirically show that our technique significantly boosts DNN robustness to adversarial samples while maintaining high accuracy in classification. To demonstrate the general applicability of our proposed method, we also conduct experiments using the MNIST and CIFAR-10 datasets, widely used in image recognition research.

ACM Reference format:
Qinglong Wang[121], Wenbo Guo[1], Kaixuan Zhang[1], Alexander G. Ororbia II[1],
Xinyu Xing[1], Xue Liu[2], C. Lee Giles[1]. 2017. Adversary Resistant Deep Neural Networks with an Application to Malware Detection. In *Proceedings of KDD'17, August 13–17, 2017, Halifax, NS, Canada.*, , 9 pages.
DOI: http://dx.doi.org/10.1145/3097983.3098158

[1]The first and second author contribute equally.

KDD'17, August 13–17, 2017, Halifax, NS, Canada.
 ISBN 978-1-4503-4887-4/17/08…$15.00.
DOI: http://dx.doi.org/10.1145/3097983.3098158

1 INTRODUCTION

Malware detection has evolved sigificantly over the past. Approaches have been introduced that range from signature-based solutions that compare an unidentified piece of code to known malware to sandboxing solutions that execute a file within a virtual environment so as to determine whether the file is malicious or not. Unfortunately, these technologies seem to quickly fall behind in the never-ending battle against malware infection. According to a recent report from Symantec Corporation [26], one million malware variants hit the Internet every day and go completely undetected by most of the common cybersecurity technologies in use today.

Substantial progress in neural network research, or deep learning (DL), has provided promising alternatives to the cybersecurity community in the form of automatic feature learning. Recent research has demonstrated that malware detection approaches based on deep neural networks (DNNs) can recognize abstract complex patterns from a large amount of malware samples. This could offer a far better way to detect all types of malware, even in instances of heavy mutation [2, 7, 8, 14, 17–19, 23, 28].

Despite their potential, deep neural architectures, like all other machine learning approaches, are vulnerable to what is known as adversarial samples [3, 6, 25]. This means that these systems can be easily deceived by non-obvious and potentially dangerous manipulations [5, 9, 12, 27]. To be more specific, an adversary can infer the model underlying an application, examine feature/class importance, and identify the features that have greatest significant impact on correct classification. With this knowledge of feature importance, an adversary can, with minimal effort, craft an *adversarial sample* – a synthetic example generated by slightly modifying a real example in order to trick deep learning system into "believing" this modified sample belongs to an incorrect class with high confidence.

This flaw has been widely exploited to fool DNNs trained for image recognition (e.g., [9, 21, 27]). With the broad adoption of DNNs in malware detection, we speculate malware authors will also increasingly seek to exploit this vulnerability to circumvent malware detection. Recent research has already demonstrated that a malware author can leverage feature amplitude inequilibrium to bypass malware detectors powered by DNNs [1, 10].

Past research [9, 22] in developing defense mechanisms relies on strong assumptions, which typically do not hold in many real-world scenarios. Also, these proposed techniques can only be empirically validated and do not provide any theoretical guarantees. This is particularly disconcerting when they are applied to security-critical applications such as malware detection.

Here we propose a new technical approach that can be empirically and theoretically guaranteed to be effective for malware

detection and, more importantly, resistant to adversarial manipulation. To be specific, we introduce random feature nullification to both the training and testing phases of DNN models, making the architectures *non-deterministic*. This non-deterministic nature is primarily useful when attackers attempt to examine feature/class importance or when a DNN model uses input for classification. Even if attackers could infer critical features and construct a reasonable adversarial sample, the stochasticity we introduce into the model's input processing significantly reduces the effectiveness of adversarial samples.

Technically speaking, our random feature nullification approach can also be viewed as stochastically "dropping" or omitting sensory inputs. It can be viewed as a special case of dropout regularization [24], which involves randomly dropping unit activities (along with their connections), especially in the hidden layers, of a standard DNN. However, in normal drop-out, a DNN is treated as a deterministic system at test-time [2] which means that critical features of the DNN model can still be correctly identified and manipulated to synthesize adversarial samples. Our approach is fundamentally different in that we nullify features at both train and test time. In Section 5, we compare our random feature nullification with standard drop-out.

The simple approach proposed is beneficial for several key reasons. First, random feature nullification makes it much more difficult for attackers to exploit the "blind spots" of DNNs. Second, our adversary-resistant DNNs maintain desirable classification performance while requiring only minimal modification to existing underlying architecture. Third, the technique we propose theoretically guarantees the resistance of DL to adversarial samples. Lastly, while this work is primarily motivated by the need to safeguard DNN models used in malware detection, it should be noted that the proposed technique is rather general and can be readily applied to other applications where deep learning proves to be useful, such as image recognition. We demonstrate this applicability using two additional, publicly-available datasets in Section 5.

The rest of the paper is organized as follows. Section 2 provides background on adversarial samples and in Section 3 a survey of relevant work. Section 4 presents our technique and its properties. Experimental results are shown in Section 5, where our technique is compared to other approaches. Finally, Section 6 summarizes our work and points our future directions.

2 BACKGROUND

Even though a well-trained model is capable of recognizing out-of-sample patterns, a deep neural architecture can be easily fooled by introducing perturbations to the input samples that are often indistinguishable to the human eye [27]. These so-called "blind spots", or adversarial samples, exist because the input space of a DNN is too broad to be fully explored [9]. Given this, an adversary can uncover specific data samples in the input space that bypass DNN models. More specifically, work [9] has shown that attackers can find the most powerful blind spots through effective optimization procedures. In multi-class classification tasks, the adversarial samples uncovered through this optimization can cause a DNN model to classify a data point under an incorrect category.

Furthermore, other work [27] shows that for DNN models that share the same design goal, i.e. recognizing the same image set, all of these models approximate a common highly complex, nonlinear function. Therefore, a relatively large fraction of adversarial examples generated from one trained DNN will be misclassified by other DNN models trained on the same original data set but with different hyper-parameters. Given a target DNN, we refer to adversarial samples that are generated from other different DNN models but still maintain their attack efficacy against the target as *cross-model adversarial samples*.

Adversarial samples can be generated by computing the derivative of the cost function with respect to the network's input variables. The gradient of any input sample represents a direction vector in this high-dimensional input space. Along this direction, any small change of this input sample will cause a DNN to generate a completely different prediction result. This particular direction is important since it represents the most effective way to degrade the performance of a DNN. Discovering this particular direction is done by passing the layer gradients from the output layer all the way back to the input layer via back-propagation of errors. The gradient at the input may then be applied to the input samples to craft an adversarial example.

To be more specific, define a cost function $\mathcal{L}(\theta, X, Y)$, where θ, X and Y denotes the parameters of the DNN, the input dataset, and the corresponding labels respectively. In general, adversarial samples are created by adding an *adversarial perturbation* δX to real samples. The *fast gradient sign* method [9] was proposed for calculating adversarial perturbations as follows:

$$\delta X = \phi \cdot sign(\mathcal{J}_{\mathcal{L}}(X)), \tag{1}$$

here δX is calculated by multiplying the sign of the gradients of the real sample X with some coefficient ϕ. $\mathcal{J}_{\mathcal{L}}(X)$ denotes the derivative of the cost function $\mathcal{L}(\cdot)$ with respect to X. ϕ controls the scale of the gradients to be added.

An adversarial perturbation indicates the actual direction vector to be added to the real samples. This vector drives a data point X towards a direction that the cost function $\mathcal{L}(\cdot)$ is most sensitive to. However, it should be noted that δX must be maintained within a small scale. Otherwise adding δX will cause significant distortions to real samples, leaving the manipulation to be easily detected.

3 RELATED WORK

In order to defend against adversarial samples, recent research has mainly focused on two different approaches – data augmentation and model complexity enhancement. In this section, we summarize these techniques and discuss their limitations as follows.

3.1 Data Augmentation

To resolve the issue of "blind spots" (a more informal name given to adversarial samples), many methods which could be considered as sophisticated forms of data augmentation[3] have been proposed (e.g. [9, 11, 20]). In principle, these methods expand the training

[2] In fact, "inverted" drop-out is applied in practice, which requires an extra division of the drop-out probability at training time in order to avoid the need for re-scaling at test-time. This specific implementation is used so that feed-forward inference is directly comparable to that under standard DNNs.

[3] Data augmentation refers to artificially expanding the data-set. In the case of images, this can involve deformations and transformations, such as rotation and scaling, of original samples to create new variants.

set by combining known samples with potential blind spots, the process of which is called adversarial training [9]. Here, we analyze the limitations of data augmentation mechanisms and argue that these limitations also apply to adversarial training methods.

Given the high dimensionality of data distributions that a DNN typically learns from, the input space is generally too broad to be fully explored [9]. This implies that, for each DNN model, there could also be an adversarial space carrying an infinite amount of blind spots. Therefore, data augmentation based approaches must face the challenge of covering these very large spaces. Since adversarial training is a form of data augmentation, such a tactic cannot possibly hope to cover an infinite space.

Adversarial training can be formally described as adding a regularization term known as *DataGrad* to a DNN's training loss function [20]. The regularization penalizes the directions uncovered by adversarial perturbations (introduced in Section 2). Therefore, adversarial training works to improve the worst case performance of a standard DNN. Treating the standard DNN much like a generative model, adversarial samples are produced via back-propagation and mixed into the training set and directly integrated into the model's learning phase. Despite the fact that there exists an infinite amount of adversarial samples, adversarial training has been shown to be effective in defending against those which are powerful and easily crafted. This is largely due to the fact that, in most adversarial training approaches [9, 20], adversarial samples can be generated efficiently for a particular type of DNN. The fast gradient sign method [9] can generate a large pool of adversarial samples quickly while *DataGrad* [20] focuses on dynamically generating them per every parameter update. However, the simplicity and efficiency of generating adversarial samples also makes adversarial training vulnerable when these two properties are exploited to attack the adversarial training method *itself*. Given that there exists an infinite supply of adversarial samples, we would need to repeat an adversarial training procedure each time a new adversarial example is encountered. Let us briefly consider DataGrad [20], which could be viewed as taking advantage of adversarial perturbations to better explore the underlying data manifold. While this leads to improved generalization, it does not offer any guarantees in covering all possible blind-spots. In this work, we do not address this issue by training a DNN model that covers the entire adversarial space. Rather, our design principle is to increase the difficulty for adversaries in finding the adversarial space efficiently.

3.2 Enhancing Model Complexity

DNN models are already complex, with respect to both the nonlinear function that they try to approximate as well as their layered composition of many parameters. However, the underlying architecture is straightforward when it comes to tracing the flow of information forwards and backwards, greatly alleviating the effort in generating adversarial samples. Therefore, several ideas [11, 22] have been proposed to enhance the complexity of DNN models, aiming to improve the tolerance of complex DNN models with respect to adversarial samples generated from simple DNN models.

[22] developed a *defensive distillation* mechanism, which trains a DNN from data samples that are "distilled" from another DNN. By using the knowledge transferred from the other DNN, the learned

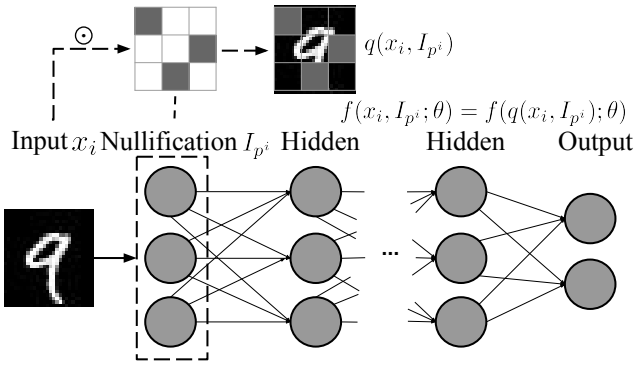

Figure 1: DNN modified with a random feature nullification layer.

DNN classifiers become less sensitive to adversarial samples. Although shown to be effective, this method is still vulnerable. This is because both DNN models used in this defense can be approximated by an adversary via training two other DNN models that share the same functionality and have similar performance. Once the two approximating DNN models are learned, the attacker can generate adversarial samples specific to this distillation-enhanced DNN model. Similar to [22], [11] proposed to stack an auto-encoder together with a standard DNN. It shows that this auto-encoding enhancement increases a DNN's resistance to adversarial samples. However, the authors also admit that this stacked model can also be easily approximated and exploited.

Given the observation and analysis above, going beyond concealing the adversarial space, we argue that an adversary-resistant DNN model also needs to be robust against adversarial samples generated from its best approximation. In light of this argument, this paper presents a new adversary-resistant DNN that not only increases the difficulty in finding its blind spots but also "immunizes" itself against adversarial samples generated from its best approximation.

4 RANDOM FEATURE NULLIFICATION

Figure 1 illustrates how a DNN would be modified with our random feature nullification method. Different from a standard DNN, the method introduces an additional layer between the input and the first hidden layer. This intermediate layer is stochastic, serving as a source of randomness during both training and testing phases. In particular, it randomly nullifies or masks the features within the input. Let us consider image recognition as an example. When a DNN passes an image sample through the layer, it randomly cancels out some pixels within the image and then feeds the partially corrupted image to the first hidden layer. The proportion of pixels nullified is determined from the hyper parameters μ_p and σ_p^2.

Here, in addition to describing feature nullification and how to train a model using it, we will explain why our method offers some theoretical guarantees of resistance to adversarial samples and how it is different from other adversary-resistant techniques.

4.1 Model Description

Given input samples denoted by $X \in \mathbb{R}^{N \times M}$, where N and M denote the number of samples and features, respectively, random feature nullification is simply performing element-wise multiplication of X with $\hat{I_p}$. Here, $\hat{I_p} \in \mathbb{R}^{N \times M}$ is a mask matrix with the same dimensions as X. Note that in performing random nullification, it is inevitable that some feature information, which might be useful for classification, will be lost. To compensate for this, we choose a different nullification rate p^i for each data sample. We hypothesize that this process could potentially lead to a better exploration of the input data's underlying manifold during training, which might result in slightly better classification performance. This is because that although DNN models [24] trained with all neurons preserved can work well on a training set, these models are more likely to produce worse testing results than those trained with randomly selected neurons. Recent work [24] inspired us to further increase the randomness to be added during training by also treating p^i as a random variable. More specifically, in our training algorithm, not only which neurons to be nullified are randomly selected, but also how many neurons to be nullified are randomly determined.

When training a DNN, for each input sample x_i a corresponding I_{p^i} is generated, where I_{p^i} is a binary vector, with each element being either 0 or 1. In I_{p^i}, the total number of zeros, determined by p^i, are randomly distributed. Without loss of generality, here we select two typical random distributions, i.e. the Gaussian distribution for p^i and the uniform distribution for generating I_p. However, it is also possible to adopt other random distributions for these two cases. Formally, we denote the number of zeros in I_{p^i} as $\lceil M \cdot p^i \rceil$, which will be randomly located, where $\lceil \cdot \rceil$ is the ceiling function. p^i is sampled from a Gaussian distribution $\mathrm{N}(\mu_p, \sigma_p^2)$.

From Figure 1, random feature nullification can be viewed as a process in which a specialized layer simply passes nullified input to a standard DNN. As such, the objective function of a DNN with random feature nullification can be defined as follows.

$$\min_{\theta} \sum_{i=1}^{N} \mathcal{L}\Big(f(x_i, I_{p^i}; \theta), y_i\Big). \tag{2}$$

Here, y_i is the label of the input x_i and θ represents the set of model parameters. The random feature nullification process is represented by function $q(x_i, I_{p^i}) = x_i \odot I_{p^i}$, where $\odot$ denotes the *Hadamard-Product* and $f(x_i, I_{p^i}; \theta) = f(q(x_i, I_{p^i}); \theta)$.

During training, Equation (2) can be solved using stochastic gradient descent in a manner similar to that of a standard DNN. The only difference is that for each training sample, the randomly picked I_{p^i} is fixed during forward and backward propagation until the next training sample arrives. This makes it feasible to compute the derivative of $\mathcal{L}\Big(f(x_i, I_{p^i}; \theta), y_i\Big)$ with respect to θ and update θ accordingly. During the testing process, when model parameters are fixed, in order to get stable test results, we use the expectation of the Gaussian distribution $\mathrm{N}(\mu_p, \sigma_p^2)$ as a substitute for the random variable p^i. Specifically, we generate a vector I_p following the same procedure described earlier, but with p equal to μ_p.

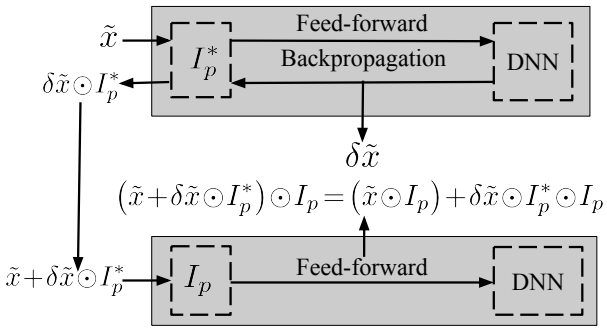

Figure 2: Example of generating an adversarial sample and testing it on a DNN with random feature nullification.

4.2 Analysis: Model Resistance to Adversaries

We now theoretically analyze our model's ability to resist adversarial samples. First, recall (Section 2) that an adversary needs to generate adversarial perturbations in order to craft adversarial samples. According to Equation 1, the adversarial perturbation is generated by computing the derivative of the DNN's cost function with respect to the input samples.

Now let us assume that an adversary uses the same procedure to attack our proposed model. To be specific, the adversary computes the partial derivative of $\mathcal{L}\Big(f(\tilde{x}, I_p; \theta), \tilde{y}\Big)$ with respect to $\tilde{x}$, where $\tilde{x}$ denotes an arbitrary testing sample and $\tilde{y}$ denotes the corresponding label. More formally, the adversary needs to solve the following derivative:

$$\begin{aligned} \mathcal{J}_{\mathcal{L}}(\tilde{x}) &= \frac{\partial \mathcal{L}\Big(f(\tilde{x}, I_p; \theta), \tilde{y}\Big)}{\partial \tilde{x}} \\ &= \mathcal{J}_{\mathcal{L}}(q) \cdot \frac{\partial q(\tilde{x}, I_p)}{\partial \tilde{x}}. \end{aligned} \tag{3}$$

where $\mathcal{J}_{\mathcal{L}}(q) = \partial \mathcal{L}\Big(f(\tilde{x}, I_p; \theta), \tilde{y}\Big) / \partial q(\tilde{x}, I_p)$. Here, as mentioned earlier, I_p is a mask matrix used during testing. Once the derivative above (Equation (3)) is calculated, an adversarial sample can be crafted by adding $\phi \cdot sign(\mathcal{J}_{\mathcal{L}}(\tilde{x}))$ to $\tilde{x}$, following [9].

To resolve Equation (3), both $\mathcal{J}_{\mathcal{L}}(q)$ and $\partial q(\tilde{x}, I_p)/\partial \tilde{x}$ need to be computed. Note that $\mathcal{J}_{\mathcal{L}}(q)$ can be easily solved using back propagation of errors. However, the term $\partial q(\tilde{x}, I_p)/\partial \tilde{x}$ carries random variable I_p. It is this multiplicative random variable I_p itself that prohibits attackers from computing a derivative needed to produce an adversarial perturbation. If I_p is designed to be additive instead of multiplicative, the computation of the derivative will not be affected since $\partial q(\tilde{x}, I_p)/\partial \tilde{x} = \partial(\tilde{x} + I_p)/\partial \tilde{x}$ is equal to an all-one vector $\mathbb{1}$. As a result, the exact adversarial perturbation for $\tilde{x}$ can be easily calculated as $\mathcal{J}_{\mathcal{L}}(q)$. In addition, prior work [11] has demonstrated that, using additive Gaussian noise does not actually improve the robustness of a DNN model against adversarial samples.

Recall that for each sample, the locations of the zeroes within I_p are randomly distributed. It is almost impossible for an adversary to pick up a value for I_p that will match that which was randomly generated. Therefore, for this adversary, the best practice would be to approximate the value of I_p. To allow this adversary to make the best possible approximation, we further assume that the value of p

is known. With this assumption, one can randomly sample I_p and treat it as a best approximation I_p^*. Using this approximation, the adversary then computes the most powerful adversarial perturbation. As shown in the top shaded region of Figure 2, for the black-boxed DNN, we assume the most powerful adversarial perturbation is $\delta\tilde{x}$. Then, the adversarial perturbation for real sample $\tilde{x}$ is $\delta\tilde{x} \odot I_p^*$.

Assume the adversary uses a synthesized adversarial sample $\tilde{x} + \delta\tilde{x} \odot I_p^*$ to attack the system shown in the bottom shaded region of Figure 2. As we can see, the synthesized sample must pass through the the feature nullification layer before passing through the actual DNN. We describe this nullification in the following form.

$$\left(\tilde{x} + \delta\tilde{x} \odot I_p^*\right) \odot I_p = \left(\tilde{x} \odot I_p\right) + \delta\tilde{x} \odot I_p^* \odot I_p. \tag{4}$$

Here, $\tilde{x} \odot I_p$ is a nullified real sample, and $\delta \odot I_p^* \odot I_p$ represents the adversarial perturbation added to it. With $I_p^* \odot I_p$, even though $\delta\tilde{x}$ is the adversarial perturbation that impacts the DNN the most, this high-impact adversarial perturbation is still distorted and no longer represents the most effective perturbation for fooling the DNN. In Section 5, we will provide empirical evidence to further validate this result.

In short, stochasticity, which naturally comes from I_p, is potentially our best defense against adversarial perturbation. It is important to interpret our particular form of drop-out as a form of "security through randomness". Our parametrized feature nullification input layer, does not serve as a form of implicit model ensembling (or Bayesian averaging, which drop-out has been shown to be equivalent to in the case of single hidden-layer networks), especially given that randomness is still introduced at test-time.

4.3 Comparison with Existing Defense Methods

In the following, we thoroughly analyze the limited resistance provided by existing defense techniques introduced in Section 3. According to [13, 20, 24], existing defense techniques can be generalized as training a standard DNN with various regularization terms (or even more generally as the DataGrad=regularized objective). More formally, the general objective is as follows:

$$\min_{\theta} \mathcal{G}(\theta, \tilde{x}, \tilde{y}) = \mathcal{L}(\theta, \tilde{x}, \tilde{y}) + \gamma \cdot \mathcal{R}(\theta, \tilde{x}, \tilde{y}), \tag{5}$$

where $\mathcal{L}(\theta, \tilde{x}, \tilde{y})$ is the training objective for a standard DNN, and $\mathcal{R}(\theta, \tilde{x}, \tilde{y})$ is a regularization term. Here, γ controls the strength of the regularization. By adding regularization, (5) penalizes the direction represented by the adversarial perturbation that is optimal for crafting adversarial samples.

However, existing defense methods that fall under this unifying framework are still vulnerable to adversarial samples problems, as shown below. To craft an adversarial sample from a model trained by solving (5), an adversary can easily produce an adversarial perturbation by computing the derivative with respect to a test sample $\tilde{x}$ as follows:

$$\begin{aligned} \mathcal{J}_{\mathcal{G}}(\tilde{x}) &= \frac{\partial \mathcal{G}(\theta, \tilde{x}, \tilde{y})}{\partial \tilde{x}} \\ &= \frac{\partial\left(\mathcal{L}(\theta, \tilde{x}, \tilde{y}) + \gamma\mathcal{R}(\theta, \tilde{x}, \tilde{y})\right)}{\partial \tilde{x}}. \end{aligned} \tag{6}$$

This indicates that prior studies only construct DNN models that are resistant to adversarial samples that target a standard DNN but do not build resistance to adversarial samples that would be generated to trick these newly "hardened" models. In addition, as we will show in Section 5, the added regularization only imposes a limited penalty to the most effective adversarial perturbation. Hence these methods might still be ineffective against adversarial samples that target standard DNNs, especially if an adversary simply increases the scale factor ϕ when generating adversarial samples.

In other words, according to [9], the space containing both real samples and adversarial samples is too broad to be exhaustively explored. In the end, since adversarial training is a form of data augmentation, it cannot possibly hope to fully solve this problem. While all machine learning methods are susceptible to a broad space of adversarial samples, our proposed method, however, is a model-complexity-based approach that hardly adds any extra parameters, thus leaving the per-iteration run-time relatively untouched.

5 EVALUATION

In this section, we first evaluate our proposed technique and compare it with adversarial training and dropout for a malware classification task using the dataset from [4]. Then we will show that our proposed method can be integrated with existing adversarial training methods and compare the combined approach's performance with both standalone methods – random feature nullification (RFN) and adversarial training, respectively. Finally, we will demonstrate the generality of our proposed method by conducting some experiments in image recognition. In particular, we contrast our method with adversarial training and dropout on the MNIST [16] and CIFAR-10 [15] datasets.

5.1 Datasets & Experimental Design

To comprehensively evaluate our method, we measure classification accuracy as well as model resistance to adversarial samples. In particular, to evaluate and compare the resistance of all three defense techniques, we test the DNN models against adversarial samples generated from the exact models trained either with RFN, adversarial training, and dropout. This means that we created three adversarial sample pools, one for each dataset (i.e., malware dataset, MNIST and CIFAR-10). The evaluation of resistance assumes that adversaries had acquired the full knowledge of each DNN model (i.e. hyper-parameters) and could construct the most effective adversarial samples to the best of their abilities. In this experimental setting, the observed resistance will then reflect a lower bound on model resistance against adversarial samples. For each dataset, we specify how to craft adversarial samples, especially with respect to the malware dataset.

Malware. The malware dataset we experimented with is a collection of window audit logs[4]. The dimensionality of the feature-space for each audit log sample is reduced to 10,000 according to the feature select metric used in the prior work [4]. Each feature indicates the occurrence of either a single event or a sequence of events[5], thus taking on the value of 0 or 1. Here, 0 indicates that the sequence of events did not occur while 1 indicates the opposite. Classification labels are either 1, indicating a malware variant, or

[4]Window audit logs are collected using standard, built-in facilities, composed of two sources–users of a enterprise network as well as sandboxed virtual machine simulation runs using a set of malicious and benign binaries

[5]The number of events in one sequence can be as high as 3.

Examples of Changed Features
WINDOWS_FILE:Execute:[system]\slc.dll,
WINDOWS_FILE:Execute:[system]\cryptsp.dll
WINDOWS_FILE:Execute:[system]\wersvc.dll,
WINDOWS_FILE:Execute:[system]\faultrep.dll
WINDOWS_FILE:Execute:[system]\imm32.dll,
WINDOWS_FILE:Execute:[system]\wer.dll
WINDOWS_FILE:Execute:[system]\ntmarta.dll,
WINDOWS_FILE:Execute:[system]\apphelp.dll
WINDOWS_FILE:Execute:[system]\faultrep.dll,
WINDOWS_FILE:Execute:[system]\imm32.dll

Table 1: Sample of manipulated features in malware dataset where each row feature contains a sequence of two events where the events happened in the same order as displayed.

0, indicating a benign program. The dataset is split into 26,078 training examples, with 14,399 benign software samples and 11,679 malicious software samples, and 6,000 testing samples, with 3,000 benign software samples and 3,000 malicious software samples. The task is to classify whether a given sample is benign or malicious.

Adversarial perturbation for malware samples can be computed according to Equation (1). However, a bit of care must be taken when generating adversarial samples for the malware dataset. Malware samples are usually represented by features that take on discrete and finite values, e.g. records of file system accesses, types of system calls incurred, etc. Therefore, it is more appropriate to use the l_0 distance:

$$||\hat{x} - x||_0 < \varepsilon, \tag{7}$$

where $\hat{x} = x + \delta x$ represents adversarial samples generated from legitimate sample x. Note that on the malware data set, the perturbation scale ϕ in (1) is measured by ϵ.

A similar approach [10] of crafting adversarial samples for malware data is realized by adjusting the Jacobian-based saliency map approach [21] proposed for binary classification case. Furthermore, malware data contains stricter semantics in comparison to image data [10]. In our case, each feature of a malware sample indicates whether or not a potential bit of malware has initiated a certain file system access. Therefore, large-scale manipulations across all features, as is typically done with image data, may break down a malicious program's functionality. To avoid this, we restrict the total number of manipulations that can occur per malware sample to be as small as possible. In this paper's setting, we set this to be 10. Moreover, since removing certain file system calls may also jeopardize a malware's internal logic, we further restrict the manipulation by only allowing the addition of new file system accesses. This equivocates to only positive manipulations, i.e. changing a feature from 0 to 1. Finally, since malware manipulation is done with the intent of fooling a DNN malware classifier, there is no need to modify a benign application such that it is classified as malicious. Therefore, in our experiments we only generate adversarial samples from the malware data points. In Table 1, we show a few examples of features added to a malware sample. These added features only cause the malware to call several dynamically linked library files without damaging the program's malicious intent.

MNIST & CIFAR-10 Data Sets. The MNIST dataset is composed of 70,000 greyscale images (of 28×28, or 784, pixels) of handwritten digits, ranging from 0 to 9. The dataset is split into a training set of 60,000 samples and a test set of 10,000 samples.

The CIFAR-10 dataset consists of 60,000 images, divided into 10 classes. The training split contains 50,000 samples while the test split contains 10,000 samples. Since the samples of CIFAR-10 dataset are color images, each image is made up of 32×32 pixels where each pixel is represented by three color channels (i.e., RGB).

For the MNIST and CIFAR-10 datasets, we generate adversarial samples by simply adding the adversarial perturbation δx, introduced in Section 2), directly to the original image (since feature values are continuous/real-valued). The degree of manipulation is controlled by selecting different ϕ, as in Equation (1).

Expectation of nullification rates (%)	Malware	
	Accuracy (%)	Resistance (%)
10	95.22	36.46
20	94.67	36.76
30	93.92	38.56
40	95.20	45.19
50	93.18	51.43
60	93.77	49.03
70	93.10	53.96
80	93.08	62.30
90	90.88	64.86

Table 2: Classification accuracy vs. model resistance with various feature nullification rates on a malware dataset. Note that the nullification rate hyper-parameter p is simply an expectation (see detail in Section 4), while the other hyper parameter σ is set to be 0.05.

5.2 Malware Classification Results

Sensitivity to Nullification Rate We first implement a group of experiments to quantify the effect that nullification rates have on model classification accuracy as well as model resistance. More specifically, we allow the nullification rate to range from 10% to 90% with 10% increments, both at training and testing time. By comparing each experiment result, we may then select the optimal nullification rate. We then integrate our defense mechanism with adversarial training and compare it against all aforementioned methods.

Measures of classification accuracy and model resistance, corresponding to different nullification rates, are shown in Table 2. As observed in Table 2, the classification accuracy of trained models decreases when the nullification rate is increased except when nullification rate is at 40% or 60%. These two rates may roughly imply the proportion of noise contained within the original dataset. The average classification accuracy is 93.66% while the highest achieved is 95.22, when nullification rate is 10%. This shows us that classification performance is more negatively impacted as more important features are discarded. Note that the accuracy remains at a surprisingly high value even when the nullification rate reaches 90%. This aligns with the fact that the malware data is quite sparse.

On the contrary, as shown in Table 2, model resistance shows the opposite trend. Maximum resistance against adversarial samples is reached at a 90% nullification rate. Clearly, with such a high nullification rate, more carefully manipulated features are discarded. The different trends for both classification accuracy and resistance

Defense Methods	Malware	
	Accuracy (%)	Resistance (%)
Standard	93.99	30.00
Dropout	93.16	13.96
Adv. Training	92.68	26.07
RFN	93.08	62.30
RFN & Adv. Training	94.81	68.77

Table 3: Classification accuracy vs. model resistance of different learning methods on the malware dataset. Dropout rates are 50% and feature nullification rates are 80%. 'Adv Training' means "adversarial training." Note that 'Standard' means standard deep neural architecture without any regularization.

demonstrate well the trade-off between achieving one of the two key goals (i.e., accuracy and robustness). By examining Table 2, we adopt 80% as our feature nullification rate expectation for experiments that follow, as the trained model with this nullification rate maintains the best balance between resistance and accuracy.

Comparative Results Next, we implement five distinct DNN models by training them with different learning techniques as specified in Table 3. We present the architecture of these DNN models as well as the corresponding hyper-parameters in Table 7, 6, 8 and 9. With certain perturbations added to the data samples, Table 3 first shows that the standard DNN model exhibits poor resistance when classifying adversarial samples. Surprisingly, as shown for dropout and adversarial training, these two methods yield even worse resistance compared to the standard DNN. This strengthens our previous analysis in Section 4. Although these mechanisms have been shown to provide certain resistance to already seen adversarial samples and so-called 'cross-model' adversarial samples [6], they are even more vulnerable to more specifically crafted adversarial samples. These results are also consistent with those reported in previous work [11]. This implies that the regularization involved in adversarial training and dropout offer poor general resistance to adversarial examples.

In comparison, RFN provides a significantly better resistance against adversarial samples, as is shown in Table 3. The model resistance afforded by our method improves more than 100% (relative error) when comparing with standard DNN. Recall that RFN can also be viewed as a preprocessing approach for the successive DNN. As such, it can be combined with other existing defense mechanisms. It is expected that such a combination would further improve model robustness. In order to verify this, we next combine RNF with adversarial training and compare the hybrid approach to both standalone RFN and adversarial training.

Table 3 specifies the classification accuracy and model resistance of the hybrid technique. We observe that the combined technique does indeed provide better resistance when compared to standalone RFN. This may due to the fact that RFN and adversarial training penalize adversarial samples in two different manners, and an ensemble of the two favorably amplifies the model resistance that each technique induces. From Table 3, we also notice that both

[6] Adversarial samples that are crafted from a different DNN that is built to approximate some standard targeted DNN.

Expectation of nullification rates	MNIST		CIFAR-10	
	Accuracy	Resistance $\phi = 0.15$	Accuracy	Resistance $\phi = 0.15$
10%	98.17%	70.39%	80.01%	55.87%
20%	98.09%	73.55%	77.62%	59.55%
30%	97.89%	78.31%	75.95%	61.63%
40%	97.53%	81.49%	74.49%	65.59%
50%	96.78%	83.68%	74.02%	67.85%

Table 4: Classification accuracy vs. model resistance with various feature nullification rates on MNIST and CIFAR-10. Hyper parameter σ is also set to be 0.05 in this evaluation.

standalone RFN and aforementioned combined approach do slightly but noticeably reduce classification accuracy. However, the combination of RFN and adversarial training results in near-negligible degradation. This indicates that RFN, either standalone or when combined with adversarial training, provides much better resistance either adversarial training and dropout on the malware dataset.

5.3 Image Recognition Results

In the following experiments, we examine the generality of our proposed method by applying it to the MNIST and CIFAR-10 image recognition tasks. For MNIST, we build a standard feed-forward fully connected DNN, while for CIFAR-10, we build a convolutional neural network (CNN). Similar to the experiments implemented on malware dataset, we also implement two groups of experiments, one for determining the optimal p on each dataset, and another for comparing between different defense technologies.

As is shown in Table 4, the trend of accuracy and resistance are consistent with that found in the malware experiments. It should be noted that, since the malware samples are highly sparse in the feature space, we test our method with nullification rate varies in a wide range from 10% to 50%. However, the image data sets are far less sparse. In Table 4, maximum resistance against adversarial image samples is reached at 50% nullification rate. With respect to classification accuracy, our proposed method demonstrates roughly similar performance at various nullification rates. Based on this result, we adopted 50% as our feature nullification rate in the experiments to follow.

In Table 5, we show measures of classification accuracy and model resistance of all aforementioned approaches on the MNIST and CIFAR-10 datasets. Much as in the malware experiments, we further evaluate our RFN method combined with adversarial training on both datasets. In Table 5, we also measure the resistance of these DNN models against various coefficients ϕ.

As is shown in Table 5, adversarial samples generated from a standard DNN are capable of lowering the accuracy of the standard DNN to as low as 0.01% on MNIST and 10.68% on CIFAR-10. In contrast, all of the investigated defense mechanisms yield improved resistance, with, again, models trained with RFN reaching the best level of resistance. In addition, the combination of RFN with adversarial training achieves the best resistance of 91.28 on MNIST and 74.12% on CIFAR-10. Though different than in the case of malware classification, both dropout and adversarial training alone do provide somewhat improved resistance on both datasets. This indicates

Learning Technology	MNIST				CIFAR-10			
	Accuracy	Resistance			Accuracy	Resistance		
		$\phi = 0.15$	$\phi = 0.25$	$\phi = 0.35$		$\phi = 0.15$	$\phi = 0.25$	$\phi = 0.35$
Standard	98.43	8.19	0.56	0.01	73.59	19.48	13.51	10.68
Dropout	98.61	19.51	3.86	0.96	81.07	17.43	16.59	16.40
Adv Training	97.46	67.68	28.37	7.62	80.62	33.97	19.76	13.73
RFN	96.78	83.69	71.44	60.69	74.02	67.85	51.89	41.29
Adv Training & RFN	96.11	91.28	84.92	78.18	74.12	71.03	55.49	49.84

Table 5: Classification accuracy vs. model resistance with different learning methods, under different ϕ, for both MNIST and CIFAR-10. In this table, dropout rates and feature nullification rates are set 50% for both datasets.

Learning Technologies	Hyper Parameters						
	DNN Structure	Activation	Optimizer	Learning Rate	Dropout Rate	Batch Size	Epoch
Standard DNN	784-784-784-784-10	Relu	SGD	0.1	×	100	25
Dropout	784-784-784-784-10	Relu	SGD	0.1	0.5	100	25
Adv. Training	784-784-784-784-10	Relu	SGD	0.01	0.5	100	70
RFN	784-784-784-784-10	Relu	SGD	0.1	0.25	100	25
RFN & Adv. Training	784-784-784-784-10	Relu	SGD	0.01	0.25	100	70

Table 6: The hyper parameters of MNIST models. Note that standard DNN stands for DNN trained without any regularization.

Learning Technology	Hyper Parameters						
	DNN Structure	Activation	Optimizer	Learning Rate	Dropout Rate	Batch Size	Epoch
Standard DNN	5000-1000-100-2	Relu	Adam	0.001	×	500	20
Dropout	5000-1000-100-2	Relu	Adam	0.001	0.5	500	20
Adv. Training	5000-1000-100-2	Relu	SGD	0.01	0.5	500	40
RFN	5000-1000-100-2	Relu	Adam	0.001	0.5	500	15
RFN & Adv. Training	5000-1000-100-2	Relu	SGD	0.01	0.5	500	40

Table 7: The hyper parameters of Malware models.

Learning Technology	Hyper parameters					
	Activation	Optimizer	Learning rate	Dropout rate	Batch Size	Epoch
Standard DNN	Relu	Adam	0.001	×	128	50
Dropout	Relu	Adam	0.001	0.5	128	50
Adv. Training	Relu	SGD	0.01	0.5	128	50
RFN	Relu	Adam	0.001	0.5	128	50
RFN & Adv. Training	Relu	SGD	0.01	0.5	128	50

Table 8: The hyper parameters of CIFAR-10 models, in this experiment we use CNN instead of standard DNN

that the resistance provided by these methods might be highly dependent on the data type (images, in this case). In particular, since adversarial training is designed to handle adversarial samples, it demonstrates much better resistance when compared directly to dropout, though both methods offer model regularization.

As for classification accuracy, dropout achieves the highest accuracy on both datasets. For MNIST dataset, both RFN and adversarial training, as well as their combination, do trade some classification accuracy for better resistance. However, for the CIFAR-10 dataset, these methods demonstrate slightly improvement for accuracy. This is due to the fact that the CIFAR-10 task is much more complex than that of MNIST, hence the regularization provided by all of these methods leads to improved generalization. In general, despite the minor accuracy degradation caused by using RFN or the hybrid method, the significant improvement over resistance in both datasets demonstrates that our proposed method is quite promising for classification tasks when resistance to adversarial samples is important. Finally, our method is agnostic to the choice of the DNN architecture, given that we evaluate RFN with both feed-forward fully connected DNNs and CNNs (as evidenced in Table 5).

6 CONCLUSION

Here we proposed a simple method for constructing deep neural network models that are robust to adversarial samples. Our design is based on a thorough analysis of a neural model's vulnerability to adversarial perturbation as well as the limitations of previously proposed defenses. Using our proposed Random Feature Nullification, we have shown that it is impossible for an attacker to craft a specifically designed adversarial sample that can force a DNN to misclassify its inputs. This implies that our proposed technology does not suffer, as previous methods do, from attacks that rely on generating model-specific adversarial samples.

We apply our method to a malware dataset and empirically demonstrated that we significantly improve model resistance with only negligible sacrifice of accuracy, compared to other defense

Layer type	Learning Technology				
	Standard DNN	Dropout	Adv. Training	RFN	RFN & Adv. Training
Convolutional	64 filter(3×3)	64 filter(3×3)	64 filter(3×3)	64 filter(3×3)	64 filter(3×3)
Convolutional	64 filter(3×3)	64 filter(3×3)	64 filter(3×3)	64 filter(3×3)	64 filter(3×3)
Max pooling	2×2	2×2	2×2	2×2	2×2
Convolutional	72 filter(3×3)	72 filter(3×3)	128 filter(3×3)	128 filter(3×3)	128 filter(3×3)
Convolutional	72 filter(3×3)	72 filter(3×3)	128 filter(3×3)	128 filter(3×3)	128 filter(3×3)
Max pooling	2×2	2×2	2×2	2×2	2×2
Fully Connect	512 units	512 units	256 units	256 units	256 units
Fully Connect	256 units	256 units	256 units	256 units	256 units
Softmax	10 units	10 units	10 units	10 units	10 units

Table 9: The network structure of CIFAR-10 models.

mechanisms. Cross-data generality was also demonstrated through experiments on image recognition. Future work will entail investigating the performance of our method to an even wider variety of applications.

7 ACKNOWLEDGEMENTS

We gratefully acknowledge partial support from the National Science Foundation.

REFERENCES

[1] Hyrum Anderson, Jonathan Woodbridge, and Bobby Filar. 2016. DeepDGA: Adversarially-Tuned Domain Generation and Detection. *arXiv:1610.01969 [cs.CR]* (2016).
[2] Matt Wolff Andrew Davis. 2015. *Deep Learning on Disassembly.* https://www.blackhat.com/docs/us-15/materials/us-15-Davis-Deep-Learning-On-Disassembly.pdf.
[3] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. 2010. The Security of Machine Learning. *Mach. Learn.* 81, 2 (Nov. 2010), 121–148.
[4] Konstantin Berlin, David Slater, and Joshua Saxe. 2015. Malicious behavior detection using windows audit logs. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security.* ACM, 35–44.
[5] Ran Bi. 2015. *Deep Learning can be easily fooled.* http://www.kdnuggets.com/2015/01/deep-learning-can-be-easily-fooled.html.
[6] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Srndic, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. 2013. Evasion Attacks against Machine Learning at Test Time.. In *ECML/PKDD (3).*
[7] BIZETY 2016. *Deep Learning Neural Nets Are Effective Against AI Malware.* BIZETY. https://www.bizety.com/2016/02/05/deep-learning-neural-nets-are-effective-against-ai-malware/.
[8] George Dahl, Jack W. Stokes, Li Deng, and Dong Yu. 2013. Large-Scale Malware Classification Using Random Projections and Neural Networks. In *Proceedings IEEE Conference on Acoustics, Speech, and Signal Processing.*
[9] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2014. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572* (2014).
[10] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2016. Adversarial Perturbations Against Deep Neural Networks for Malware Classification. *arXiv preprint arXiv:1606.04435* (2016).
[11] Shixiang Gu and Luca Rigazio. 2014. Towards deep neural network architectures robust to adversarial examples. *arXiv:1412.5068 [cs]* (2014).
[12] Mike James. 2014. *The Flaw Lurking In Every Deep Neural Net.* http://www.i-programmer.info/news/105-artificial-intelligence/7352-the-flaw-lurking-in-every-deep-neural-net.html.
[13] D.K. Kang, J. Zhang, A. Silvescu, and V. Honavar. 2005. Multinomial event model based abstraction for sequence and text classification. *Abstraction, Reformulation and Approximation* (2005), 901–901.
[14] Will Knight. 2015. *Antivirus That Mimics the Brain Could Catch More Malware.* https://www.technologyreview.com/s/542971/antivirus-that-mimics-the-brain-could-catch-more-malware/.
[15] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).
[16] Yann LeCun, Corinna Cortes, and Christopher JC Burges. 1998. The MNIST database of handwritten digits. (1998).
[17] Cade Metz. 2015. *Baidu, the Chinese Google, Is Teaching AI to Spot Malware.* https://www.wired.com/2015/11/baidu-the-chinese-google-is-teaching-ai-to-spot-malware/.
[18] MIT Technology Review 2016. *Machine-Learning Algorithm Combs the Darknet for Zero Day Exploits, and Finds Them.* MIT Technology Review.
[19] Linda Musthaler. 2016. *How to use deep learning AI to detect and prevent malware and APTs in real-time.*
[20] Alexander G. Ororbia II, C. Lee Giles, and Daniel Kifer. 2016. Unifying Adversarial Training Algorithms with Flexible Deep Data Gradient Regularization. *arXiv:1601.07213 [cs]* (2016).
[21] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. 2016. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 372–387.
[22] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. 2015. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1511.04508* (2015).
[23] Joshua Saxe and Konstantin Berlin. 2015. Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. *CoRR* (2015).
[24] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.
[25] Nedim Srndic and Pavel Laskov. 2014. Practical Evasion of a Learning-Based Classifier: A Case Study. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy.*
[26] Symantec 2016. *Internet Security Threat Report.* Symantec. https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.
[27] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations.*
[28] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-Sec: Deep Learning in Android Malware Detection. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM '14).*