Enhancing Robustness of Deep Neural Networks Against Adversarial Malware Samples: Principles, Framework, and AICS'2019 Challenge

Deqiang Li [1,4], Qianmu Li [1], Yanfang Ye [2], Shouhuai Xu [3]

[1] School of Computer Science and Engineering, Nanjing University of Science and Technology
[2] Department of Computer Science and Electrical Engineering, West Virginia University
[3] Department of Computer Science, University of Texas at San Antonio
[4] School of Computing and Information Sciences, Florida International University
lideqiang@njust.edu.cn, qianmu@njust.edu.cn, yanfang.ye@mail.wvu.edu, shouhuai.xu@utsa.edu

Abstract

Malware continues to be a major cyber threat, despite the tremendous effort that has been made to combat them. The number of malware in the wild steadily increases over time, meaning that we must resort to automated defense techniques. This naturally calls for machine learning based malware detection. However, machine learning is known to be vulnerable to adversarial evasion attacks that manipulate a small number of features to make classifiers wrongly recognize a malware sample as a benign one. The state-of-the-art is that there are no effective countermeasures against these attacks. Inspired by the AICS'2019 Challenge, we systematize a number of principles for enhancing the robustness of neural networks against adversarial malware evasion attacks. Some of these principles have been scattered in the literature, but others are proposed in this paper for the first time. Under the guidance of these principles, we propose a framework and an accompanying training algorithm, which are then applied to the AICS'2019 challenge. Our experimental results have been submitted to the challenge organizer for evaluation.

1 Introduction

Malware remains a big threat to cyber security despite communities' tremendous countermeasure efforts. For example, Symantec (Symantec 2018) reports seeing 355,419,881 new malware variants in year 2015, 357,019,453 in year 2016, and 669,974,865 in year 2017. Worse yet, there is an increasing number of malware variants that attempted to undermine anti-virus tools and indeed evaded many malware detection systems (CISCO 2018).

In order to cope with the increasingly severe situation, we have to resort to machine learning techniques for automating the detection of malware in the wild (Ye et al. 2017). However, machine learning based techniques are vulnerable to adversarial evasion attacks, by which an adaptive attacker perturbs or manipulates malware samples into adversarial samples that would be detected as benign rather than malicious (see, for example, (B. Biggio and et al. 2013; Al-Dujaili et al. 2018; Hou et al. 2018; Chen, Ye, and Bourlai 2017)).[1] The state-of-the-art is that there are many attacks, but the problem of effective defense is largely open. This is indeed the context in which the AICS'2019 malware classification challenge is proposed.

Our Contributions

In this paper, we make the following contributions. First, we propose, to the best of our knowledge, the first systematic framework that aims to enhance the robustness of malware classifiers against adversarial evasion attacks. The framework is designed under the guidance of a set of principles, some of which are known but scattered in the literature (e.g., using an ensemble of classifiers), but others are explicitly proposed for the first time, such as the following. We propose using the capability of the optimal white-box attack to bound the capability of any ℓ_p ($p \geq 1$) norm based gray-box attack from above, and propose using semantics-preserving representation learning for malware classification. Both the principles and framework should be seen as a starting point and systematically refined in the future.

Second, we apply the framework to address the AICS 2019 malware classification challenge. Among the 3,133 testing samples in 5 classes, our classifier predicts 2,245 samples in class '0', 461 samples in class '1', 162 samples in class '2', 176 samples in '3', and 89 samples in class '4'. Since we do not know the ground truth of the testing set, which has yet to be announced by the challenge organizer, we cannot tell the effectiveness of the framework at the time for writing the present paper. Instead, we have submitted our classification result to the challenge organizer.

Related Work

Since the present paper focuses on defense against adversarial malware classification, we review existing studies in this topic by emphasizing two complementary approaches: *input prepossessing* and *adversarial training*.

Input Prepossessing Input prepossessing transforms the input to a different representation with the aim to reduce the degree of perturbation to the original input. For example, Random Feature Nullification (RFN) randomly nullifies features in the training and testing phases (Wang et al. 2017);

[1] The term "adversarial example" is often used in the literature. We instead propose using the term "adversarial sample" because it is arguably more natural in the context of malware detection, for which we already get used to terms like "benign sample" and "malicious sample". Corresponding to these two kinds of samples, an adversarial sample is a malicious sample that would be misclassified as a benign one.

hash transformation leverages a locality-preservation property to reduce the degree of perturbation to the original input (Li et al. 2018); DroidEye (Chen et al. 2018) quantizes binary feature representation via count featurization.

Our framework uses binarization to reduce the degree of perturbation, which is inspired by the idea of feature squeezing in the context of image processing (Xu, Evans, and Qi 2017). This effectively reduces the perturbation space because there are now only two kinds of perturbations: flipping from '1' to '0' or flipping '0' to '1'. This means that we effectively consider the number of perturbations but not the 'scale' of perturbation (i.e., reducing the sensitivity of classifiers to small degrees of perturbation).

Adversarial Training Adversarial training augments the training data with adversarial samples to improve the robustness of classifiers. This idea has been independently proposed in different application settings, including (Goodfellow, Shlens, and Szegedy ; Kurakin, Goodfellow, and Bengio 2016b) and (Xu et al. 2014; Xu et al. 2013). In particular, it has been proposed to consider adversarial training with the optimal attack, which in a sense corresponds to the worst-case scenario and therefore could lead to classifiers that are robust against the non-optimal attacks (Al-Dujaili et al. 2018). The challenge is of course to find the optimal attack. In our framework, we use this approach to regularize our model and seek the optimal attack via the gradient descent method.

2 Review on Adversarial Evasion Attacks against Malware Classification

Basic Idea

Consider a classifier $f : \mathcal{X} \to \mathcal{Y}$ that takes an unperturbed malware instance $\mathbf{x} \in \mathcal{X}$ as input and correctly outputs its label $y \in \mathcal{Y}$. Given $\mathbf{x}$ that is to be classified, the adversarial evasion attack problem is to manipulate or perturb $\mathbf{x}$ to an adversarial malware sample $\mathbf{x}'$ such that the malicious functionality of $\mathbf{x}$ is preserved while satisfying:

$$f(\mathbf{x}') \neq f(\mathbf{x}), \tag{1}$$

$$\text{s.t.} \quad \mathbf{x}_{\text{lb}} \leq \mathbf{x}' \leq \mathbf{x}_{\text{ub}} \tag{2}$$

$$||\mathbf{x}' - \mathbf{x}|| \leq \epsilon \tag{3}$$

where $\mathbf{x}_{\text{lb}}$ ($\mathbf{x}_{\text{ub}}$) is the element-wise lower (upper) bound of feature vectors, the box-constraint Eq.(2) means that the manipulation or perturbation cannot violate the constraints imposed by the feature definitions and $\mathbf{u} \leq \mathbf{v}$ means that each element of $\mathbf{u}$ is no greater than the corresponding element in $\mathbf{v}$, and $||\cdot||$ refers to a metrics of interest (e.g., ℓ_p norm for some $p \in \{0, 2, \infty\}$) and Eq.(3) says that perturbations may be bounded by a given ϵ in the norm. The perturbation vector is $\delta_{\mathbf{x}} = \mathbf{x}' - \mathbf{x}$.

In the present paper, we focus on classifiers f that are learned as neural networks, $\mathbf{F} : \mathcal{X} \to \mathbb{R}^o$, which output (softmax) the probability mass function over o classes or labels. Since the constraint given by Eq.(1) is hard to formulate, researchers proposed considering two scenarios: in the case of *non-targeted* attacks, maximize the cost of classifying the $\mathbf{x}'$ as y

$$\max_{\mathbf{x}'} \; L(\mathbf{F}(\mathbf{x}'), y); \tag{4}$$

in the case of *targeted* attacks, minimize $L(\mathbf{F}(\mathbf{x}'), y_t)$, where y_t ($y_t \neq y$) is a target label given by the attacker.

Threat Model

As elaborated below, a threat model against malware classifiers is specified by *what the attacker knows*, *what the attacker can do*, and *how the attacker wages the attack*.

What the attacker knows There are three kinds of models from this perspective. A *black-box* attacker knows nothing about classifier f except what is implied by f's responses to the attacker's queries. A *white-box* attacker knows all kinds of information about f, including its model parameters. A *gray-box* attacker knows an amount of information about f that resides in between the preceding two extremes. For example, the attacker may know the training set or feature definitions.

What the attacker can do In evasion attacks, the attacker only can manipulate the testing data, while obeying some constraints. One constrain is to preserve the malicious functionality of a malware. Although the attacker can manipulate a malware sample by inserting, deleting, and replacing features (Dang, Huang, and Chang 2017; Anderson et al. 2017), a simplifying assumption is to consider insertion only (i.e., flipping a feature value from '0' to '1' (B. Biggio and et al. 2013; Nedim rndic 2014; Grosse et al. 2017; Wang et al. 2017; Rosenberg et al. 2017; Chen, Hou, and Ye 2017; Al-Dujaili et al. 2018). The other constraint is to maintain the relation between features. Using the ACIS'2019 malware classification challenge as an example, we note that n-gram (uni-gram, bi-gram, and tri-gram) features reflect sequences of Windows system API calls. This means that when the attacker inserts an API call into a malware sample, several features related to this API call will need to be changed according to the definition of n-gram features.

How the attacker wages the attack Researchers generate adversarial malware samples using various machine learning techniques such as genetic algorithms, reinforcement learning, generative networks, feed-forward neural networks, decision trees, and Support Vector Machine (SVM) (Xu, Qi, and Evans 2016; Anderson et al. 2017; Hu and Tan 2017; B. Biggio and et al. 2013; Xu et al. 2014; Nedim rndic 2014; Carlini and Wagner 2017). In order to generate adversarial malware samples effectively and efficiently, attacks often leverage the gradients with respect to inputs of neural network (Goodfellow, Shlens, and Szegedy ; Papernot et al. 2016).

3 Framework

Guiding Principles

The design of the framework is guided by a number of principles. These principles are geared towards neural network classifiers, which are chosen as our focus because deep

learning techniques are increasingly employed in malware defense, but their vulnerability to adversarial evasion attacks has yet to be tackled (Raff et al. 2017).

Principle 1: Knowing the enemy This principle says that we should strive to extract useful information about the data as much as we can. This kind of information will offer insights into designing countermeasures. For example, we can ask questions of the following kinds:

- Is the training set imbalanced? If the training set is not balanced, various methods need to be considered for alleviating the imbalance issue. For example, the widely-used oversampling is to expand the samples of the "minority" classes via random and repetitive sampling (Buda, Maki, and Mazurowski 2018).
- Are there sufficiently many samples? This issue is important especially when there are a large number of features and when neural networks are considered. One widely-used method is data augmentation, which generates new samples by making slight modifications on original samples (Lemley, Bazrafkan, and Corcoran 2017).
- Are there simple indicators of adversarial samples? If there are simple indicators of adversarial samples, we can possibly design tailored classifiers for them.

Principle 2: Bridging the gap between countermeasures against gray-box attacks and countermeasures against white-box attacks In gray-box attacks, the attacker knows some information about the feature set and therefore can train a surrogate classifier $\hat{f} : \mathcal{X} \to \mathcal{Y}$ from a training set (where the realization of $\hat{f}$ is a neural network $\hat{\mathbf{F}}$) and leverage the transferability from $\hat{f}$ to f to generate adversarial samples. Consider an input $\mathbf{x}$ for which a gray-box attacker generates perturbations using

$$\hat{\delta}_{\mathbf{x}} \in \max_{||\hat{\delta}_{\mathbf{x}}|| \leq \epsilon} L(\hat{\mathbf{F}}(\mathbf{x} + \hat{\delta}_{\mathbf{x}}), y),$$

the change to the loss of f incurred by $\hat{\delta}_{\mathbf{x}}$ is

$$\begin{aligned} |\Delta L| &= \left| L(\mathbf{F}(\mathbf{x} + \hat{\delta}_{\mathbf{x}}), y) - L(\mathbf{F}(\mathbf{x}), y) \right| \\ &= \left| \int_0^{\hat{\delta}_{\mathbf{x}}} \nabla L(\mathbf{F}(\mathbf{x} + \delta), y) d\delta \right| \\ &= \left| \int_0^1 \nabla L(\mathbf{F}(\mathbf{x} + t\hat{\delta}_{\mathbf{x}}), y)^\top \hat{\delta}_{\mathbf{x}} dt \right| \\ &\leq \epsilon \sup_{||\delta|| \leq \epsilon} \left\| \nabla L(\mathbf{F}(\mathbf{x} + \delta)) \right\|_* , \end{aligned}$$

where "$||\cdot||_*$" means the dual norm of $||\cdot||$. The preceding observation indicates that corresponding to the same (and potentially large) perturbation upper bound ϵ, the loss incurred by gray-box attacks is upper bounded by the loss incurred by white-box attacks. This suggests us to focus on the robustness of classifier f against the optimal white-box attack because it accommodates the worst-case scenario. It is worth mentioning that our observation, which applies to an arbitrary perturbation upper bound ϵ, enhances an earlier insight that holds for a small perturbation upper bound ϵ (Demontis et al. 2018).

Principle 3: Using ensemble of classifiers rather than a single one (i.e., not putting all eggs in one basket) This is suggested by the observation that no single classifier may be effective against all kinds of evasions. Worse yet, we may not know the kinds of evasions the attacker uses to generate adversarial samples. Therefore, we propose using *ensemble* learning to enhance the robustness of neural networks based malware classifiers.

An ensemble can be constructed by many methods (e.g., bagging, boosting or using multiple classifiers). The generalization error of an ensemble decreases significantly with the ensemble size when the base classifiers are effective and mutual independent (Hoeffding 1963). For example, *random subspace* (Ho 1998) is seemingly particularly suitable for formulating malware classifier ensembles because the dimension of malware feature vectors is often very high, which indicates a high vulnerability of malware classifiers to adversarial samples (Simon-Gabriel et al. 2018).

Since the output of a neural network (with softmax) is the probability mass function over the classes in question, the final prediction result is produced according to these probabilities. Formally, an ensemble $f_{en} : \mathcal{X} \to \mathcal{Y}$ contains a set of neural network classifiers $\{f_i\}_{i=1}^{l}$, namely $f_{en} = \{f_i : \mathcal{X} \to \mathcal{Y};\ (1 \leq i \leq l)\}$. Given a testing sample $\mathbf{x}$, each classifier f_i defines a conditional probability on predicting $y : P(y|\mathbf{x}, f_i)$. We treat the base classifier equally, and the voting method is

$$P(y|\mathbf{x}) = \frac{1}{l} \sum_{i=1}^{l} P(y|\mathbf{x}, f_i), \tag{5}$$

where $P(y|\mathbf{x})$ is the probability that the ensemble predicts $\mathbf{x}$ as the label y.

Principle 4: Using input transformation to reduce the degree of perturbation caused by evasion manipulation In the context of malware classification, input transformation techniques, such as adversarial feature selection (Zhang et al. 2016) and random feature nullification (Wang et al. 2017), can reduce the degree of perturbation in adversarial samples so as to improve the robustness of classifiers. In typical applications, the defender does not know what kinds of evasion manipulations are used by the attacker to generate adversarial samples, including the number of features that are manipulated by the attacker. Therefore, we propose considering a spectrum of evasion manipulations, from manipulating a few features (measured by, for example, the ℓ_0 norm) to manipulating a large number of features (but the magnitude of these manipulations may be small and therefore measured for example by the ℓ_∞ norm). Moreover, we may give higher weights to the transformation techniques that can simultaneously reduce the degrees of perturbations in terms of the ℓ_∞ norm, ℓ_0 norm, or ℓ_2 norm. This suggests us to propose using the *binarization* technique: When the feature value of the ith feature, denoted by x_i, is smaller than a threshold Θ_i, we binarize x_i to 0; otherwise, we binarize x_i to 1. The threshold Θ_i may be set as the median value of the ith feature. This input transformation reduces the perturbation space to two kinds: flipping '0' to '1' and flipping '1' to '0'.

Principle 5: Using adversarial training to "inject" immunity into classifiers Adversarial training (also known as proactive training (Xu et al. 2014)) incorporates some adversarial samples into the training set. Various kinds of *heuristic* training strategies have been proposed (see, e.g., (Grosse et al. 2017; Szegedy et al. 2013; Xu et al. 2014; Goodfellow, Shlens, and Szegedy ; Kurakin, Goodfellow, and Bengio 2016a). However, these strategies typically deal with some specific evasion methods and therefore are not known to effective against other evasion methods. Inspired by robust optimization, (Madry et al. 2017) propose solving the saddle point problem so as to improve the robustness of neural networks against a wide range of adversarial samples. This strategy has been adapted to the context of malware classification in (Al-Dujaili et al. 2018).

The preceding discussion suggests us to train neural networks that can accommodate the optimal attack and the unperturbed examples on distribution $\mathcal{D}$:

$$\min_{\theta} \mathbb{E}_{(\mathbf{x},y)\in\mathcal{D}} \left[\max_{\delta_{\mathbf{x}}\leq\epsilon} L(\mathbf{F}(\mathbf{x}+\delta_{\mathbf{x}}), y) + L(\mathbf{F}(\mathbf{x}), y) \right], \tag{6}$$

where $L(\cdot,\cdot)$ is the cross-entropy and θ denotes the parameters of neural network $\mathbf{F}$ (as the realization of f_i). A key issue is how to deal with the optimal attack. (Madry et al. 2017) showed that the projected gradient descent (PGD) can solve the inner maximum problem effectively for calculating the first-order adversarial samples. Inspired by this insight, we leverage gradient descent on the negative cross-entropy to generate adversarial samples. In order to avoid local minima, we can randomly repeat several times for each point and pick the point minimizing the negative cross-entropy as the initial point.

Principle 6: Using semantics-preserving representations Adversarial malware samples must assure that a manipulated sample is still a malware (i.e., preserving the malicious functionality of the original malware sample). This suggests us to strive to learn neural network models that are sensitive to malware semantics, but not the perturbations because the latter must preserve the malicious functionality of the original malware. Specifically, we propose using *denoising autoencoder* to learn semantics-preserving representations because they can make neural network less sensitive to perturbations. A denoising autoencoder $ae = d \circ e$ unifies two components: an encoder $e : \mathcal{X} \to \mathcal{H}$ that maps an input $M(\mathbf{x})$ to a latent representation $\mathbf{r} \in \mathcal{H}$ and a decoder $d : \mathcal{H} \to \mathcal{X}$ that reconstructs $\mathbf{x}$ from $\mathbf{r}$, where the $\mathcal{H}$ is the latent representation space and M refers to some operations applied to $\mathbf{x}$ (e.g., adding Gaussian noises to $\mathbf{x}$). (Vincent et al. 2010) showed that the lower bound of the *mutual information* between $\mathbf{x}$ and $\mathbf{r}$ is maximized when the reconstruction error is minimized. In the case of Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and reconstruction loss

$$\mathbb{E}_{\epsilon\sim N(0,\sigma^2)} \|ae(\mathbf{x}+\epsilon) - \mathbf{x})\|_2^2, \tag{7}$$

(Alain and Bengio 2014) showed that the optimal $ae^*(\mathbf{x})$ is

$$ae^*(\mathbf{x}) = \frac{\mathbb{E}_{\epsilon}\left[p(\mathbf{x}-\epsilon)(\mathbf{x}-\epsilon)\right]}{\mathbb{E}_{\epsilon}\left[p(\mathbf{x}-\epsilon)\right]}, \tag{8}$$

where $p(\cdot)$ is the probability density function. Eq.(8) says that representations of a well-trained denoising autoencoder are insensitive to $\mathbf{x}$ because of the weighted average from the neighbourhood of $\mathbf{x}$, which is reminiscent of the *attention* mechanism (Luong, Pham, and Manning 2015).

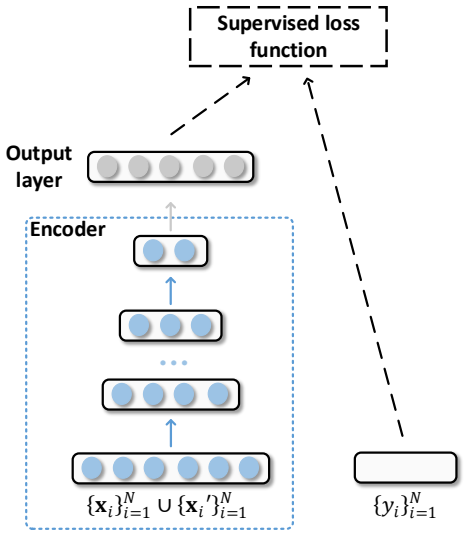

Figure 1: Illustration of neural network for classification. Blue dashed box contains the learned encoder, which and the output layer comprise the learned neural network classifier. The model parameters are tuned to minimize the supervised loss function.

Figure 1 depicts how the learned encoder is leveraged for classification, where the encoder structure is the same as described in (Vincent et al. 2010). Two examples of noise are:

- *Salt-and-pepper noise*: A fraction α of the elements of original sample $\mathbf{x}$ are randomly selected, and then set their values as their respective minimum or maximum (i.e., effectively flipping '1' to '0' or flipping '0' to '1').
- *Perturbation* $\delta_{\mathbf{x}}$: A perturbation $\delta_{\mathbf{x}}$ is added to $\mathbf{x}$ such that classifier f_i misclassifies adversarial sample $\mathbf{x}' = \mathbf{x} + \delta_{\mathbf{x}}$.

Note that when an input transformation technique mentioned above is used together with a denoising autoencoder, the former should be applied first and the latter is applied to the transformed input.

Given a mini-batch of N training samples $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$, the empirical risk of a denoising autoencoder is

$$L_{ae} = \frac{1}{N}\sum_{i=1}^{N} \left[\|ae(M(\mathbf{x}_i)) - \mathbf{x}_i\|_2^2 + \|ae(\mathbf{x}_i') - \mathbf{x}_i\|_2^2 \right], \tag{9}$$

where $M(\cdot)$ denotes adding salt-and-pepper noise on $\mathbf{x}$ and $\mathbf{x}_i'$ is the adversarial sample generated via adversarial training.

Turning principles into a framework

The principles discussed above guide us to propose a framework for adversarial malware classification, which is high-

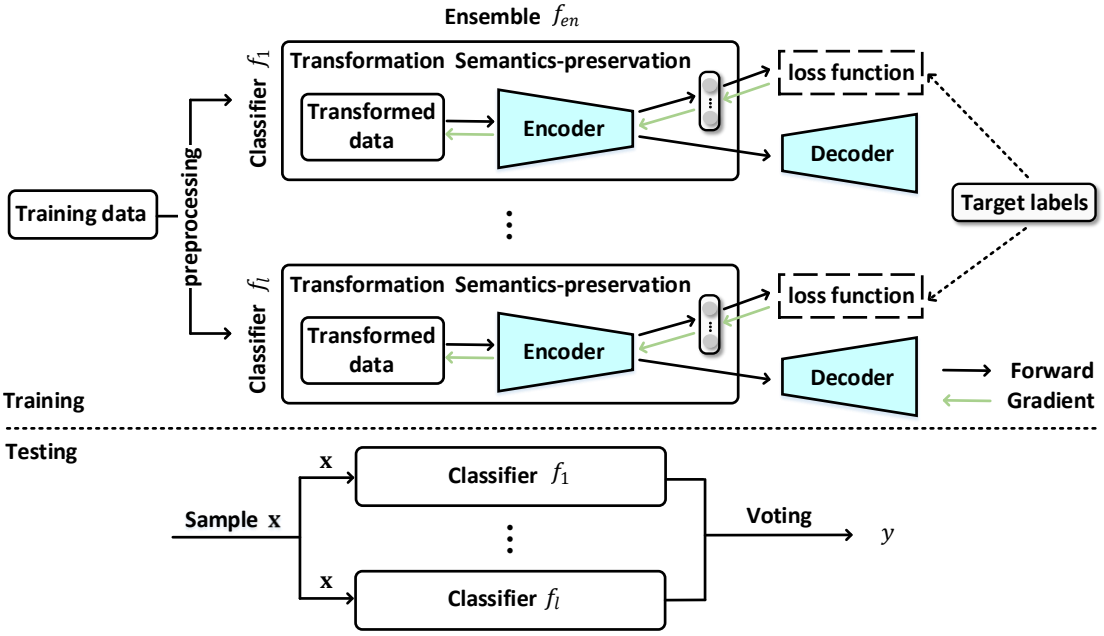

Figure 2: Overview of the proposed malware classification framework. In the training phase, an ensemble of l neural network classifiers are trained, with each classifier hardened by three countermeasures (i.e., input transformation, semantics-preserving, and adversarial training using gradient descent on the transformed data). In the testing phase, the label of a sample is determined according to the result of voting by the l classifiers.

lighted in Figure 2 and elaborated below. Specifically, we need to examine whether or not the input has some issues (e.g., imbalanced data) that need to be coped with via an appropriate prepossessing (according to Principle 1). We propose using an ensemble f_{en} of classifiers $\{f_i\}_{i=1}^{l}$ (according to Principle 3), which are trained from random subspace of the original feature space. Each classifier f_i is hardened by three countermeasures: input transformation via binarization (according to Principle 4); adversarial training models on the optimal attacks using gradient descent (green arrows in Figure 2, according to Principle 2 and Principle 5); semantics-preservation is achieved via an encoder and a decoder (according to Principle 6). In order to attain adversarial training and at the same time semantics-preservation, we learn classifier f_i via block coordinate descent to optimize different components of the model.

Putting the pieces together, we obtain Algorithm 1 for training individual classifiers. The training procedure consists of the following steps. (i) Given a training set (X, Y), we randomly select a ratio Λ of sub-features to the feature set, and then transform X into $\overline{X}$ via the binarization technique discussed above. (ii) We sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$ from $(\overline{X}, Y)$, and calculate the adversarial samples $\mathbf{x}'_i$ for $\mathbf{x}_i \in \{\mathbf{x}_i\}_{i=1}^{N}$ according to Lines 5-11 in Algorithm 1). (iii) We pass the $\{M(\mathbf{x}_i)\}_{i=1}^{N}$ and $\{\mathbf{x}'_i\}_{i=1}^{N}$ through the denoising autoencoder to compute the reconstruction loss with respect to the target $\{\mathbf{x}_i\}_{i=1}^{N}$ via Eq.(9), and update the parameters of the denoising autoencoder. (iv) We pass the $\{\mathbf{x}_i\}_{i=1}^{N}$ and $\{\mathbf{x}'_i\}_{i=1}^{N}$ together through the neural networks to compute the classification error with respect to

Algorithm 1: Training classifier f_i

Input: Training set (X, Y), maximum training epoch N_{epoch}, mini-batch size N, and the number of repeat times K.

1 Cope with issues like imbalanced input;
2 Select a ratio Λ of sub-features to the feature set;
3 Transform input X to $\overline{X}$ via binarization;
4 **for** $epoch = 0$ to N_{epoch} **do**
5 Sample a mini-batch $\{\mathbf{x}_i, y_i\}_{i=1}^{N}$ from the $(\overline{X}, Y)$;
6 **for** $repeat = 0$ to K **do**
7 Apply slight salt-and-pepper noise to $\{\mathbf{x}_i\}_{i=1}^{N}$;
8 Calculate perturbations $\{\delta_{\mathbf{x}_i}^{repeat}\}_{i=1}^{N}$ for manipulating sample $\{\mathbf{x}'_i\}_{i=1}^{N}$;
9 Project $\{\mathbf{x}'_i\}_{i=1}^{N}$ into the binary space;
10 **end**
11 Select the best perturbation $\delta_{\mathbf{x}_i}$ from $\delta_{\mathbf{x}_i}^{repeat}$ where $0 \leq repeat < K$ for $\mathbf{x}_i$ $(1 \leq i \leq N)$ so as to minimize the negative cross-entropy;
12 Calculate the reconstruction loss via Eq.(9);
13 Backpropagate the loss and update the denoising autoencoder parameters;
14 Calculate the adversarial training loss via Eq.(6);
15 Backpropagate the loss and update classifier parameters;
16 **end**

the ground truth label $\{y_i\}_{i=1}^N$ via Eq.(6), and update the parameters of the classifier via backpropagation. Note that Steps (ii)-(iv) are performed in a loop. The output of the training algorithm is a neural network classifier.

4 Experiment: Applying the Framework to the AICS'2019 Challenge

The AICS'2019 Challenge

The challenge is in the context of adversarial malware classification (i.e., labeling the class to which a malware sample belongs or *multiclass classification*), namely constructing evasion-resistant, machine learning based malware classifiers. The dataset, including both the training set and the testing set, consists of Windows malware samples (or instances), each of which belongs to exactly one of the following five classes: *Virus*, *Worm*, *Trojan*, *Packed malware*, and *AdWare*.

For each sample, the features are collected by the challenge organizer via dynamic analysis, including the Windows API calls and further processed unigram, bigram, and trigram API calls. The feature names (e.g., API calls) and the class labels are "obfuscated" by the challenge organizer as integers, while noting the obfuscation preserves the mapping between the features and the integers representation of them. For example, three API calls are represented by three unique integers, say 101, 102, and 103; then, a trigram API call "101;102;103" means a sequence of API calls 101, 102, and 103. In total there are 106,428 features.

The testing set consists of adversarial samples and non-adversarial samples (i.e., unperturbed malware samples). Adversarial samples are generated by a variety of perturbation methods, which are not known to the participating teams. However, the ground truth labels of the testing samples are not given to the participating teams. This means that the participating teams cannot calculate the accuracy of their detectors by themselves. Instead, they need to submit their classification results (i.e., labels on the samples in the testing set) to the challenge organizer, who will calculate the classification accuracy of each participating team.

Basic Analysis

As discussed in Principle 1 of the framework, our basic analysis aims to identify some basic characteristics that should be taken into consideration when adapting Algorithm 1 to this specific case study.

Is the training set imbalanced? The training set consists of 12,536 instances, and the testing set consists of 3,133 instances. The training set contains 8,678 instances in class '0', 1,883 instances in class '1', 771 instances in class '2', 692 instances in class '3', and 512 instances in class '4'. Figure 3 plots the histogram of the instances in the training set according to the given labels of malware class (i.e., the five malware classes that have been obfuscated as integers '0', '1', '2', '3', '4'). We can calculate the maximum ratio between the number of instances in different classes is 16.95, indicating that the training set is highly imbalanced.

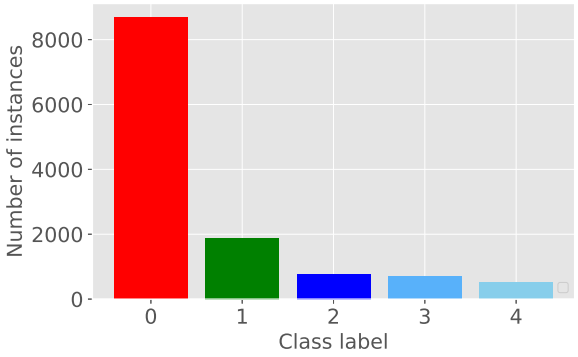

Figure 3: Histogram of the training set in malware classes.

In order to cope with the imbalance in the training set, we use the *Oversampling* method to replicate randomly selected samples from a class with a small number of samples. The replication process ends until the number of samples is comparable to that of the largest class (i.e., the class with the largest number of samples), where "comparable" is measured by a predefined ratio. In order to see the effect of this ratio, we use a 5-fold cross validation on the training set to investigate the impact of this ratio. The classifier consists of neural networks with two fully-connected layers (each layer having 160 neurons with the ReLU activation function), which are optimized via Adam (Kingma and Ba 2014) with epochs 50, mini-batch size 128, learning rate 0.001. The model is selected when achieving the best Macro F1 score on the validation set.

Table 1: Accuracy (%) and Macro F1 score (%) are reported with a 95% confidence interval with respect to the ratio parameter (%), where '—' means learning a classifier using the original training dataset.

Ratio (%)	Accuracy (%)	Macro F1 (%)
—	93.20±1.04	85.52±1.12
30	92.86±0.75	85.47±1.04
40	92.38±1.00	84.87±1.07
50	92.21±0.60	84.87±1.00
60	92.48±1.12	84.62±1.01

Table 1 shows that the Macro F1 score decreases as the oversampling ratio of minority classes increases. In order to make each mini-batch of training samples contain samples from all classes, which would be critical in muticlass classification, our experience suggests us to select the 30% ratio.

Are there sufficiently many samples? Machine learning, especially deep learning, models need to be trained with a "large" number of samples, where "large" is relative to the number of features. In the challenge dataset, the training set contains 12,536 samples while noting that there are 106,428 features, which may lead to overfitting. To cope with this, we note that adversarial training is a data augmentation method (Goodfellow, Shlens, and Szegedy), which can be leveraged to regularize the resulting classifiers.

Are there simple indicators of adversarial samples? In the first testing set published by the challenge organizer, we see negative values for some features. These negative values would indicate that they are adversarial samples. In the revised testing set provided by the challenge organizer, there are no negative feature values, meaning that there are no simple ways to tell whether a sample is adversarial or not. In spite of this, we can speculate the count of perturbed features by comparing the number of nonzero entries corresponding to the samples in the training set to their counterparts in the testing set. It is important because the attack success rate increases with the number of perturbed features.

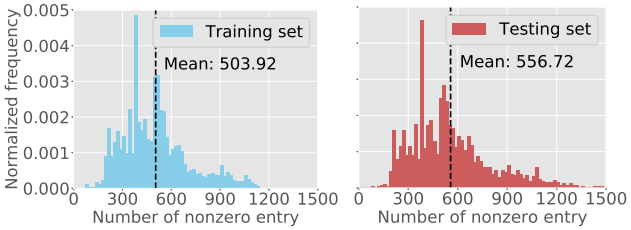

Figure 4: Histogram of the normalized frequency of the number of nonzero entries corresponding to samples in the training set and their counterparts in the testing set. The dashed line represents the mean value.

Figure 4 shows the normalized frequency of the number of nonzero entries corresponding to the samples in the training set and their counterparts in the testing set. We observe that their normalized frequencies are similar except that some testing samples have more nonzero entries (> 1200). Their mean values are much smaller than the input dimension $(106,428)$, suggesting that the average number of perturbed features may be small.

Classification Result

For adversarial training, the gradient descent with respect to the transformed input iterates 55 times via Adam optimizer (Kingma and Ba 2014) with learning rate 0.01. The perturbed input is projected into the range $[0,1]$ and rounded into binary space (i.e., binarization as discussed in the framework). Since we do not have access to the malware samples, we cannot tell whether a feature perturbation preserves the malware functionality or not. We train 10 neural network based classifiers to formulate an ensemble, including 6 classifiers using the input transformation, adversarial training, and semantics-preservation techniques discussed in the framework, and the other 4 classifiers using the input transformation and adversarial training techniques because some samples may be perturbed without preserving the malicious functionality in the training. The ratio for random subspace method is set as $\Lambda = 0.5$. Each classifier has two fully-connected hidden layers (each layer having neurons 160), uses the ELU activation function, and is optimized by Adam with epochs 100, mini-batch size 128, and learning rate 0.001. The classification result has been submitted to AICS 2019 organizer for evaluation.

5 Conclusion

We have systematized six principles for enhancing the robustness of neural network classifiers against adversarial evasion attacks in the setting of malware classification. These principles guided us to design a framework, which leads to a concrete training algorithm. We applied the training algorithm to the AICS'2019 challenge, and submitted the classification result to the challenge organizer for evaluating the effectiveness of our framework.

The problem of adversarial malware detection has not received the due amount of attention. We hope this paper will inspire more research into this important problem. Future research problems are abundant, such as: extending and refining the principles against evasion attacks, seeking principles to enhance robustness of adversarial malware detection against poisoning attacks, designing more systematic frameworks and more robust techniques against adversarial malware.

Acknowledgments. We thank the AICS'2019 challenge organizers for preparing the challenge. DL is supported by the China Scholarship Council under Grant 201706840123. YY and SX are supported in part by NSF Grant SaTC-1814825. YY is additionally supported in part by NSF Grants CNS-1618629, CNS-1814825 and OAC-1839909, NIJ 2018-75-CX-0032, WV HEPC Grant (HEPC.dsr.18.5), and WVU RSA Grant (R-844). SX is additionally supported in part by ARO Grant W911NF-17-1-0566. The views and opinions of the author(s) presented in the paper do not reflect, in any sense, those of the funding agencies.

References

[Al-Dujaili et al. 2018] Al-Dujaili, A.; Huang, A.; Hemberg, E.; and OReilly, U.-M. 2018. Adversarial deep learning for robust detection of binary encoded malware. In *2018 IEEE Security and Privacy Workshops (SPW)*, 76–82. IEEE.

[Alain and Bengio 2014] Alain, G., and Bengio, Y. 2014. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research* 15(1):3563–3593.

[Anderson et al. 2017] Anderson, H. S.; Kharkar, A.; Filar, B.; and Roth, P. 2017. Evading machine learning malware detection. *Black Hat*.

[B. Biggio and et al. 2013] B. Biggio, I. C., and et al., D. M. 2013. Evasion attacks against machine learning at test time. In *Machine Learning and Knowledge Discovery in Databases: European Conference*, 387–402. Springer.

[Buda, Maki, and Mazurowski 2018] Buda, M.; Maki, A.; and Mazurowski, M. A. 2018. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks* 106:249–259.

[Carlini and Wagner 2017] Carlini, N., and Wagner, D. 2017. Towards evaluating the robustness of neural networks. In *2017 38th IEEE Symposium on Security and Privacy (SP)*, 39–57. IEEE.

[Chen et al. 2018] Chen, L.; Hou, S.; Ye, Y.; and Xu, S. 2018. Droideye: Fortifying security of learning-based classifier

against adversarial android malware attacks. In *FOSINT-SI'2018*, 253–262.

[Chen, Hou, and Ye 2017] Chen, L.; Hou, S.; and Ye, Y. 2017. Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks. In *ACSAC*, 362–372. USA: ACM.

[Chen, Ye, and Bourlai 2017] Chen, L.; Ye, Y.; and Bourlai, T. 2017. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In *EISIC'2017*, 99–106.

[CISCO 2018] CISCO. 2018. Cisio @ONLINE.

[Dang, Huang, and Chang 2017] Dang, H.; Huang, Y.; and Chang, E.-C. 2017. Evading classifiers by morphing in the dark. In *CCS*, 119–133. ACM.

[Demontis et al. 2018] Demontis, A.; Melis, M.; Pintor, M.; Jagielski, M.; Biggio, B.; Oprea, A.; Nita-Rotaru, C.; and Roli, F. 2018. On the intriguing connections of regularization, input gradients and transferability of evasion and poisoning attacks. *arXiv preprint arXiv:1809.02861*.

[Goodfellow, Shlens, and Szegedy] Goodfellow, I. J.; Shlens, J.; and Szegedy, C. Explaining and harnessing adversarial examples (2014). *arXiv preprint arXiv:1412.6572*.

[Grosse et al. 2017] Grosse, K.; Papernot, N.; Manoharan, P.; Backes, M.; and McDaniel, P. 2017. Adversarial examples for malware detection. In *European Symposium on Research in Computer Security*, 62–79. Springer.

[Ho 1998] Ho, T. K. 1998. The random subspace method for constructing decision forests. *IEEE Transactions on PAMI* 20(8):832–844.

[Hoeffding 1963] Hoeffding, W. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58(301):13–30.

[Hou et al. 2018] Hou, S.; Ye, Y.; Song, Y.; and Abdulhayoglu, M. 2018. Make evasion harder: An intelligent android malware detection system. In *Proceedings of the Twenty-Seventh IJCAI*, 5279–5283.

[Hu and Tan 2017] Hu, W., and Tan, Y. 2017. Generating adversarial malware examples for black-box attacks based on gan.

[Kingma and Ba 2014] Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

[Kurakin, Goodfellow, and Bengio 2016a] Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016a. Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

[Kurakin, Goodfellow, and Bengio 2016b] Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016b. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.

[Lemley, Bazrafkan, and Corcoran 2017] Lemley, J.; Bazrafkan, S.; and Corcoran, P. 2017. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access* 5:5858–5869.

[Li et al. 2018] Li, D.; Baral, R.; Li, T.; Wang, H.; Li, Q.; and Xu, S. 2018. Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples. *arXiv preprint arXiv:1809.06498*.

[Luong, Pham, and Manning 2015] Luong, T.; Pham, H.; and Manning, C. D. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421.

[Madry et al. 2017] Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2017. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

[Nedim rndic 2014] Nedim rndic, P. L. 2014. Practical evasion of a learning-based classifier: A case study. In *Security and Privacy (SP), 2014 IEEE Symposium on*, 197–211. IEEE.

[Papernot et al. 2016] Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2016. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, 372–387. IEEE.

[Raff et al. 2017] Raff, E.; Barker, J.; Sylvester, J.; Brandon, R.; Catanzaro, B.; and Nicholas, C. 2017. Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435*.

[Rosenberg et al. 2017] Rosenberg, I.; Shabtai, A.; Rokach, L.; and Elovici, Y. 2017. Generic black-box end-to-end attack against rnns and other calls based malware classifiers. *arXiv preprint*.

[Simon-Gabriel et al. 2018] Simon-Gabriel, C.-J.; Ollivier, Y.; Schölkopf, B.; Bottou, L.; and Lopez-Paz, D. 2018. Adversarial vulnerability of neural networks increases with input dimension. *arXiv preprint arXiv:1802.01421*.

[Symantec 2018] Symantec. 2018. Symantec @ONLINE.

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

[Vincent et al. 2010] Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; and Manzagol, P.-A. 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research* 11(Dec):3371–3408.

[Wang et al. 2017] Wang, Q.; Guo, W.; Zhang, K.; and et al. 2017. Adversary resistant deep neural networks with an application to malware detection. In *Proceedings of the 23rd KDD*, 1145–1153. ACM.

[Xu et al. 2013] Xu, L.; Zhan, Z.; Xu, S.; and Ye, K. 2013. Cross-layer detection of malicious websites. In *Third ACM Conference on Data and Application Security and Privacy (CODASPY'13)*, 141–152.

[Xu et al. 2014] Xu, L.; Zhan, Z.; Xu, S.; and Ye, K. 2014. An evasion and counter-evasion study in malicious websites detection. In *CNS, 2014 IEEE Conference on*, 265–273. IEEE.

[Xu, Evans, and Qi 2017] Xu, W.; Evans, D.; and Qi, Y. 2017. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv preprint:1704.01155*.

[Xu, Qi, and Evans 2016] Xu, W.; Qi, Y.; and Evans, D. 2016. Automatically evading classifiers: A case study on pdf malware classifiers. In *NDSS*.

[Ye et al. 2017] Ye, Y.; Li, T.; Adjeroh, D. A.; and Iyengar, S. S. 2017. A survey on malware detection using data mining techniques. *ACM Comput. Surv.* 50(3):41:1–41:40.

[Zhang et al. 2016] Zhang, F.; Chan, P. P.; Biggio, B.; Yeung, D. S.; and Roli, F. 2016. Adversarial feature selection against evasion attacks. *IEEE transactions on cybernetics* 46(3):766–777.