Learning Adversary-Resistant Deep Neural Networks

Qinglong Wang[1,2], Wenbo Guo[1,3], Kaixuan Zhang[1], Alexander G. Ororbia II[1], Xinyu Xing[1], C. Lee Giles[1], and Xue Liu[2]

[1]*Pennsylvania State University*
[2]*McGill University*
[3]*Shanghai Jiao Tong University*

***Abstract*—Deep neural networks (DNNs) have proven to be quite effective in a vast array of machine learning tasks, with recent examples in cyber security and autonomous vehicles. Despite the superior performance of DNNs in these applications, it has been recently shown that these models are susceptible to a particular type of attack that exploits a fundamental flaw in their design. This attack consists of generating particular synthetic examples referred to as adversarial samples. These samples are constructed by slightly manipulating real data-points in order to "*fool*" the original DNN model, forcing it to mis-classify previously correctly classified samples with high confidence. Addressing this flaw in the model is essential if DNNs are to be used in critical applications such as those in cyber security. Previous work has provided various defense mechanisms by either augmenting the training set or enhancing model complexity. However, after a thorough analysis, we discover that DNNs protected by these defense mechanisms are still susceptible to adversarial samples, indicating that there are no theoretical guarantees of resistance provided by these mechanisms. To the best of our knowledge, we are the first to investigate this issue shared across previous research work and to propose a unifying framework for protecting DNN models by integrating a data transformation module with the DNN. More importantly, we provide a theoretical guarantee for protection under our proposed framework. We evaluate our method and several other existing solutions on both MNIST, CIFAR-10, and a malware dataset, to demonstrate the generality of our proposed method and its potential for handling cyber security applications. The results show that our framework provides better resistance compared to state-of-art solutions while experiencing negligible degradation in accuracy.**

I. Introduction

Beyond highly publicized victories in automatic game-playing as in Go [33], there have been many successful applications of deep neural networks (DNN) in image and speech recognition. Recent explorations and applications include those in medical imaging [2], [38] and self-driving cars [10], [14]. In the domain of cybersecurity, security companies have demonstrated that deep learning could offer a far better way to classify all types of malware [8], [30], [39].

However, though powerful, deep neural architectures, as well as all other machine learning methods, break down in the face of adversarial learning [3], [17], where such systems can be easily deceived by non-obvious and potentially dangerous manipulations [24], [35]. To be more specific, prior work [11] has shown that an attacker could use the same algorithm used to train these models, back-propagation of errors, and a surrogate dataset to construct an auxiliary model that can accurately approximate the target DNN model. Compared to the original model that usually returns categorical classification results (e.g., benign or malicious software), such an auxiliary model could provide the attacker with useful details about a DNN's weaknesses (e.g., a continuous classification score for a malware sample). As such, the attacker can use the auxiliary model to examine class/feature importance and identify the features that have significant impact on the target's classification ability. Armed with this knowledge of feature importance, the attacker can more easily craft an *adversarial sample*, or synthetic example generated by slightly perturbing a real example such that the original DNN model believe it belongs to the wrong class with high confidence. Recently, this flaw inherent in machine learning has been widely exploited to fool DNNs trained for image recognition (e.g., [24]) and malware classification (e.g., [12]).

In the past, research in hardening deep learning relies on the assumption of security through obscurity. As we will discuss in this paper, this assumption typically does not hold in many real world scenarios. In addition, the techniques proposed (e.g., [11], [13], [25], [34]) can only be empirically validated and cannot provide any theoretical guarantees. This is particularly disconcerting when they are applied in security critical applications such as malware detection.

In this work, we relax assumptions made in past research, and propose a framework to facilitate the design of adversary-resistant DNNs. More specifically, we assemble a data transformation module in front of a standard DNN. With this data transformation module, the original input data samples are projected into a new representation before it is passed through the successive DNN. This can be used as a defense since data transformation can potentially conceal the space of adversarial manipulations to a carefully designed hyperspace, making feature/class examination difficult.

Following our proposed framework, we implemented a DNN which takes a non-parametric dimensionality reduction method to transform input data to a new representation. We theoretically prove the DNN developed in this manner can be

The first and second author contributed equally.

adversary-resistant. In addition, we empirically demonstrate this adversary-resistant DNN exhibits high classification accuracy. Last, we compare our adversary-resistant DNN with existing defense mechanisms in the context of image recognition and malware classification. The results show that the proposed adversary-resistant DNN provides strong resistance to adversarial samples while the other defense mechanisms only demonstrate limited effectiveness.

To be best of our knowledge, this work is the first to provide DNNs with theoretically guaranteed resistance to adversarial samples. In addition, our adversary-resistant DNN can maintain desirable classification performance while having minimal modification to existing DNN. Last but not least, our proposed framework are generally applicable to deep learning applications covering a broad research interest.

In summary, this work makes the following contributions.

- We propose a general framework to facilitate the development of adversary resistant DNNs. More specifically, we stack a data transformation module in front a standard DNN which conceals the space of adversarial manipulations in a carefully designed hyperspace.
- Using the framework, we develop an adversary-resistant DNN, and theoretically prove its resistance to adversarial manipulation.
- We empirically evaluate the classification performance of our adversary-resistant DNN in the context of image recognition and malware classification, and compare it with existing defense mechanisms.

The rest of this paper is organized as follows. Section II introduces the background of DNNs and the generation of adversarial samples. Section III discusses a strong assumption of the threat model adopted in previous work and then refines this threat model by relaxing this assumption. Section IV presents our proposed general framework and Section V develops an adversary-resistant DNN under our framework. In Section VI, we evaluate our developed DNN on three datasets. Section VII summarizes existing defense mechanisms. Finally, we conclude this work in Section VIII.

II. Background

In this section, we first briefly introduce the well-established DNN model. Then we describe the adversarial learning problem. Specificaly, we summarize existing ways one could "attack" a DN and then generalize these attacks under a unified framework.

A. Deep Neural Networks

A typical DNN architecture, graphically depicted in Fig. 1, consists of multiple successive layers of processing elements, or so-called "neurons". Each processing layer can be viewed as learning a different, more abstract representation of the original multidimensional input distribution. As a whole, a DNN can be viewed as a highly complex function that is capable of nonlinearly mapping original high-dimensional data points to a lower dimensional space. A DNN contains an input layer, multiple hidden layers, and an output layer. The input layer

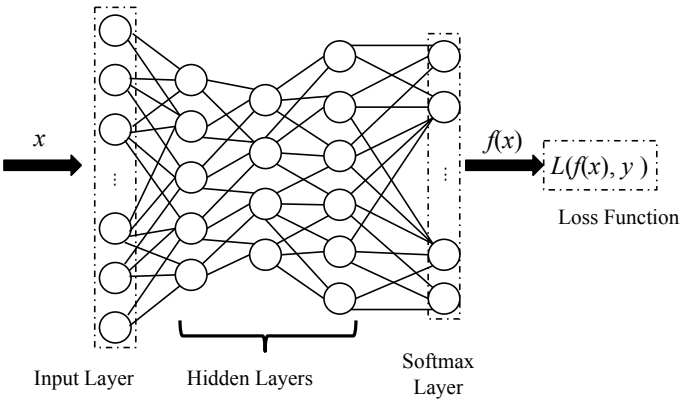


Figure 1: **Neural network architecture:** here we depict a simple neural network with three hidden layers, with interconnections between all neurons in each layer, but no intra-layer connections. This architecture is referred to as a feed-forward neural network [4].

takes in each data sample in the form of a multidimensional vector. Starting from the input, computing the activations of each subsequent layer simply requires, at minimum, a matrix multiplication (where a weight/parameter vector, with length equal to the number of hidden units in the target layer, is assigned to each unit of the layer below) usually followed by summation with a bias vector. This process roughly models the process of a layer of neurons integrating the information received from the layer below (i.e., computing a pre-activation) before applying an elementwise activation function[1]. This integrate-then-fire process is repeated subsequently for each layer until the last layer is reached. The last layer, or output, is generally interpreted as the model's predictions for some given input data, and is often designed to compute a parameterized posterior distribution using the softmax function (also known as multi-class regression or maximum entropy). This bottom-up propagation of information is also referred to as *feed-forward* inference [15].

During the learning phase of the model, the DNN's predictions are evaluated by comparing them with known target labels associated with the training samples (also known as the"ground truth"). Specifically, both predictions and labels are taken as the input to a selected cost function, such as cross-entropy. The DNN's parameters are then optimized with respect to this cost function using the method of steepest gradient descent, minimizing prediction errors on the training set. Parameter gradients are calculated using back-propagation of errors [32]. Since the gradients of the weights represent their influence on the final cost, there have been multiple algorithms developed for finding optimal weights more accurately and efficiently [23].

More formally, given a training set (X, Y):

[1]There are many types of activations to choose from, including the hyperbolic tangent, the logistic sigmoid, or the linear rectified function, etc. [4]

$\{(x_1, y_1), (x_n, y_n), \ldots, (x_n, y_n)\}$, where $x_i \in \mathbb{R}^m$ is a data sample and $y_i \in \mathbb{R}^k$ is data label, where, if categorical, is typically represented through a 1-of-*k* encoding. A standard neural network with a softmax output layer can be represented by the following function:

$$\begin{aligned} f: \mathbb{R}^m &\to \mathbb{R}^k \\ x &\mapsto f(x). \end{aligned} \tag{1}$$

The cost function for training a DNN as a (k-class) classifier is also defined as follows:

$$\begin{aligned} L: \mathbb{R}^m \times \mathbb{R}^k &\to \mathbb{R} \\ f(x) \times y &\mapsto L(f(x), y). \end{aligned} \tag{2}$$

B. Adversarial Sample Problems

Even though a well-trained neural network is capable of recognizing some unseen data samples, such a model has been shown to be easily fooled by introducing perturbations to the input that are often indistinguishable to the human eye [35], as shown in Fig.2. These so-called "blind spots", or adversarial samples, exist because the input space of the DNN is broad [11]. Knowing this, we can uncover specific data samples in the input space that "bypass" DNN models. More specifically, it was shown in [11] that attackers can find the most powerful blind spots using effective optimization procedures. In multi-class classification tasks, such adversarial samples can cause a DNN to classify a data point into a random class other than the correct one (sometimes not even a reasonable alternative).

Adversarial samples are 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 the model's high-dimensional input space. Along this direction, an small change in this input sample can cause the DNN to generate a completely different prediction result. This particular direction is important since it represents the most effective way one might compromise the optimization of the cost function. Discovering this particular direction is realized by transmitting the error gradients from output layer all the way back to the input layer via back-propagation. The gradient with respect to the input may then be applied to the input sample(s) to craft an adversarial example(s).

We now formally present the generation of adversarial samples by first defining the function g for generating any adversarial sample $\hat{x}$:

$$\begin{aligned} g: \mathbb{R}^m \times \mathbb{R}^k &\to \mathbb{R}^m \\ x \times y &\mapsto \hat{x}, \end{aligned} \tag{3}$$

where

$$\begin{aligned} \hat{x} = &\ arg\max_{\hat{x}} L(f(\hat{x}), y) \\ \text{s.t.} &\ \|\hat{x} - x\|_p < \varepsilon, \end{aligned} \tag{4}$$

and $\|\cdot\|_p$ is p-norm:

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}}. \tag{5}$$

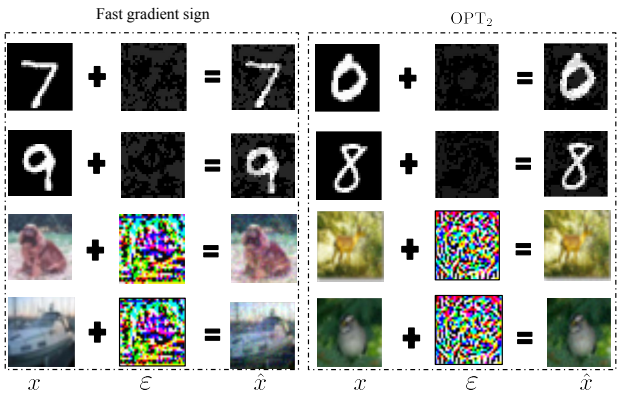


Figure 2: **Adversarial samples:** we demonstrate adversarial examples generated from MNIST dataset [20] and CIFAR-10 dataset [19]. Images on the left panel are adversarial examples generated by fast gradient sign [11]. For each column in the left panel, we use x, ε and $\hat{x}$ to denote original images, generated perturbations and the corresponding adversarial samples. Images on the right panel are generated by solving (4) by L-BFGS and p is set to 2 [21].

Note that g is a one-to-one mapping, meaning that when there exist multiple $\hat{x}$ satisfying (4), only one is chosen. It should be noted that the constraint in (4) ensures that the distortion incurred by manipulation must be maintained at a small scale. Otherwise a significant distortion will leave the manipulation to be easily detected.[2]

Existing attacks [7], [11], [28], [35] consist of different approaches for generating adversarial samples. These approaches, ultimately, can all be described solving following optimization problem, which is a transformation of (4):

$$\min_{\hat{x}} \ c\|x - \hat{x}\|_p - L(f(\hat{x}), y), \tag{6}$$

where c is some weight or coefficient. Here we refer to this type of attack as an *optimal attack*, and, for simplicity, denote $\|x - \hat{x}\|_p$ as the l_p distance.

Solving (6) is done much like solving for the weight coefficients of a standard DNN. Simply put, an attacker can use back-propagation and either gradient descent [11] or L-BFGS [7], [35]. In order to conduct the calculation, one first computes $\partial^n L(f(\hat{x}), y)/\partial \hat{x}^n$ and then use it for manipulating legitimate samples x. Note that here $n = 1$ if we use a first-order optimization method like gradient descent, or $n = 2$ if we utilize a second-order method such as the Newton–Raphson method [6]. Since gradient descent requires an iterative calculation, it can be computationally expensive when the number of adversarial samples to be generated is large. To mitigate this problem, [11] proposed the fast gradient

[2]However, it was explained in [25], that while easily-detectable, applying greater amounts of distortion still yields an easily-recognizeable image for humans. Given this, the classifier should still be able to correctly classify the sample.

sign method to approximate this iterative calculation where one could use a single step of gradient descent, as follows:

$$\hat{x} = x + \epsilon \cdot \text{sign}(\nabla L(f(x), y)), \tag{7}$$

where ϵ is set to control the scale of the distortions applied. Due to its computational efficiency, the fast gradient sign has been widely used in multiple studies [11], [13], [25]. Optimal attacks can be generalized to fast gradient sign attacks by selecting the l_∞ distance. In [35], the authors study an equivalent optimization problem to (6) by setting $p = 2$. Another type of attack, as described in [28], generates adversarial samples using a greedy approach to solve (6) and sets $p = 0$. More specifically, adversarial samples are crafted by iteratively perturbing one coordinate of x at a time. A coordinate is picked by determining by its influence on the final cost.

C. End-to-end Gradient Flow

In order to solve (6), previously introduced attacks must be able to calculate the n-th order [3] derivative of $L(f(\hat{x}), y)$ with respect to $\hat{x}$. The calculation of $\partial^n L(f(\hat{x}), y)/\partial \hat{x}^n$ requires the existence of an *end-to-end gradient flow* between the final objective and the input. More formally, given a DNN F [4] with cost function L, for any pair of input sample x and target label y, if $\partial^n L(F(x), y)/\partial x^n$ can be computed by passing residual error $\partial^n L(F(x), y)/\partial F(x)^n$ backwards to the input through intermediate layers of a (fully-differentiable) DNN, then we say there exists an end-to-end gradient flow. Existing attacks [7], [11], [27], [35] generate adversarial sample $\hat{x}$ by exploiting the existence of an end-to-end gradient flow. As long as the DNN supports end-to-end gradient flow, it will always be susceptible to adversarial perturbation.

III. Threat Model

In this section, we first describe the threat model that previous research has assumed and then discuss the limitations of this particular threat model. In light of this, we develop a new practical threat model.

A. The Black Box Threat Model

Existing defense mechanisms [11], [13], [25], [34] are designed to handle the threat directly imposed on standard DNNs (and other differentiable models, including simple linear ones). Specifically, these defenses only focus on defending adversarial samples generated from standard DNNs. While these mechanisms have yielded promising results, the scope of their provided robustness is restricted to standard DNNs and similar differentiable models. This undesired restriction is the result of a strong assumption shared by all of these defenses– the attacker does not know of the adversarial learning algorithm used to "harden" the target DNN. In other words, these mechanisms themselves are assumed (perhaps inadvertently) to be black boxes to adversaries. In consideration of this, we refer to the threat model in this case as the *black box threat model* and adversarial samples generated under it as *black box adversarial samples*.

B. Limitations

As introduced in Section I, a relatively large fraction of adversarial examples generated from one trained DNN will be misclassified by the other DNNs trained using the same dataset but with different hyper-parameter settings. Therefore, an adversary can easily generate effective adversarial samples to bypass standard DNN classifiers, by exploiting well known training algorithms and training datasets. More importantly, this information disclosure can also be exploited to attack previously proposed defense mechanisms. To make this issue clearer, we contrast the case of neural network adversarial training algorithms against cryptology. In real world scenarios, any secure encryption algorithm can be freely disclosed to the public without suffering from attacks. The black-box assumption underlying the current threat model does not hold in practice. Indeed, given the publicity and the limited number of existing mechanisms, in practice, it is not difficult for an adversary to enumerate the solutions and design new adversarial samples crafted specifically for attacking the current forms of defense. Note that as previously discussed, access to the training algorithm is sufficient to "break" a DNN–an adversary does not need access to the exact form of the model and its defense mechanism. As will be shown later in Section IV, existing defense mechanisms suffer exactly the same threat for standard DNNs. Since the assumption discussed above is too strong to be of practical use in real world scenarios, we relax this assumption and present a *white box threat model* in the following sub-section.

C. The White Box Threat Model

Under the white box threat model, an adversary is assumed to be able to generate adversarial samples specifically crafted for a target DNN. Furthermore, our proposed white box threat model covers two important cases. One is the extreme case where the *exact* information about a target DNN is known to the adversary. This corresponds to the first, innermost region in Fig. 3. The other case is that only the training algorithm, the DNN structure, and the dataset(s) used to build the target DNN can be accessed or inferred by adversaries. This corresponds to the second region in Fig. 3. The latter case is more likely as it is similar to the case of generating cross-model adversarial samples from standard DNNs. Nonetheless, we also consider the first case since it represents the most potent threat to DNNs. We refer to adversarial samples generated under the white box threat model as *white box adversarial samples* [5]. Black box adversarial samples are shown in the third outermost region in Fig. 3.

[3] Note that in this problem, $n = 1$ or 2.

[4] F is different from f, f represents a standard DNN, F refers to any neural network architecture, of which the standard DNN is one example of.

[5] Note that a large fraction of white box adversarial samples are generated under the second case mentioned before. These samples are also fall under the category of cross-model adversarial samples [11].

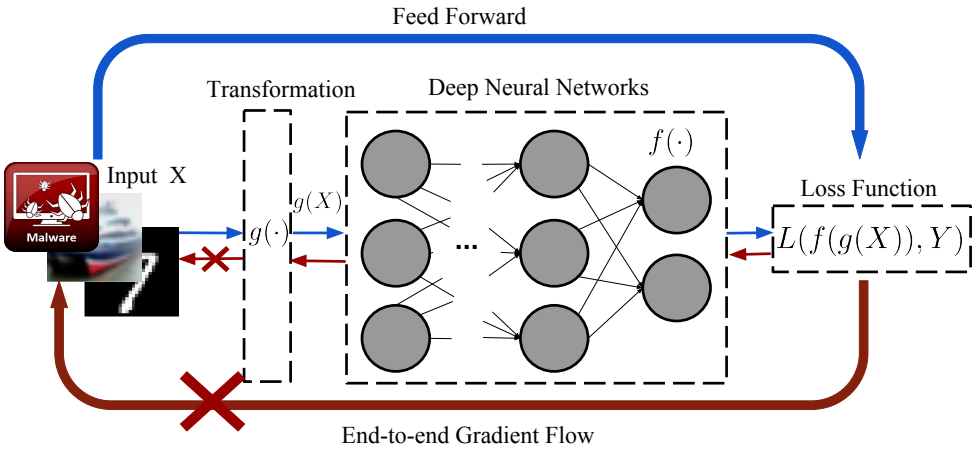


Figure 4: **Adversary-resistant DNN framework:** the proposed framework consists of a data transformation module $g(\cdot)$ and a DNN $f(\cdot)$. In the feed-forward pass, input data is first projected to a certain manifold. Then the transformed data $g(X)$ is used to train a standard DNN classifier. The end-to-end gradient flow (marked by the bold red line) is blocked by the input data transformation module. This module also ensures that recovering original data X from the transformed data $g(X)$ is computationally intractable.

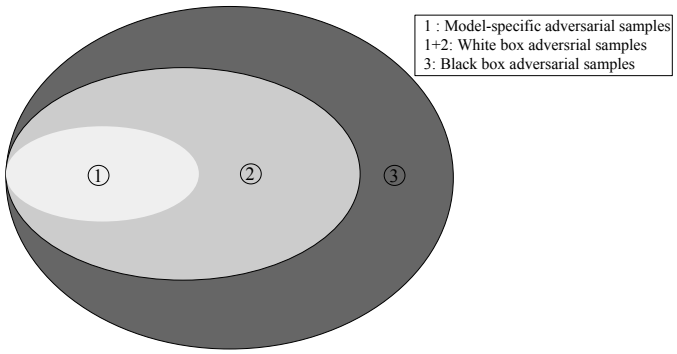


Figure 3: **Relationship between different sets of adversarial samples:** darker colors indicate that the corresponding attack is weaker. Note that there exists overlap between different adversarial samples. Under the threat model of this paper, we focus on white box adversarial samples and black box adversarial samples.

IV. The Adversary-Resistant Framework

In this section, we first specify our design goals and then present our proposed adversary-resistant framework.

A. Design Goals

In light of the previously introduced white box threat model introduced, our design goals for any useful framework are as follows:

1) The designed DNN under the proposed framework provides theoretical guarantees of resistance to white box adversarial samples.
2) The proposed framework has minimal impact on the classification accuracy of the learned DNN model.

The first design goal requires that our proposed framework facilitate the development of truly adversary-resistant DNNs. The second design goal requires that any proposed framework preserve the classification (or predictive) accuracy of DNN models when confronted with legitimate samples.

B. Framework

We propose a framework that satisfies the above design goals by integrating a data transformation module with a standard DNN. In other words, we insert a data transformation module before the input layer of the DNN with the intent of performing a special form of pre-processing of the input.

The proposed framework is graphically depicted in Fig 4. One might notice that the data transformation g is the linch pin in making our proposed framework adversary-resistant. According to the discussion in Section II, we first require that the data transformation module must block the end-to-end gradient flow. This is a direct and ultimately more effective defense to white-box attacks. As long as the end-to-end gradient flow is blocked, existing attack mechanisms [7], [11], [27], [35], as unified in (6), are *guaranteed* to fail.

Although end-to-end gradient flow facilitates the crafting of white box adversarial samples, it should not be considered as the only possible way for generating them. Assuming the end-to-end gradient flow is already blocked, an adversary might still manage to craft adversarial samples using the output of the data transformation module. Since end-to-end gradient flow is blocked at the input layer of the successive DNN, back-propagation can only carry error gradients to the output of the transformation module, or rather, the transformed representation of the data $g(X)$. Given $g(X)$, an adversary could construct an adversarial sample by inverting g using the manipulated output of the data transformation module. Therefore, we further specify another requirement that the

inversion of the DNN's data transformation using the model's output must be computationally intractable. In addressing this, our proposed framework completely prohibits the generation of white box adversarial samples.

Standard DNNs have the often-praised ability to automated learn feature detectors. Our framework is designed with the intent of preserving this ability, as was explained in our second design goal. This means that any data transformation module we utilize must preserve critical information contained in original input distribution.

Satisfying all of the above ensures that our proposed adversary-resistant framework provides resistance to adversarial perturbation while suffering at most only trivial changes in performance. In particular, our proposed framework has several properties that guarantee that a standard DNN is well protected. First, the right kind of data transformation will block gradient flow from reaching the input layer of a DNN. This critically ensures that an adversary cannot generate samples from our framework using a method like the fast gradient sign method (see Section II). Second, any data transformation satisfying the second requirement ensures that adversaries will be unable to map any perturbed transformed data point back to the original input space if inverting the data transformation employed is computationally intractable. Third, the framework is capable of legitimately handling test samples since the standard DNN model is trained on top of a data transformation that preserves crucial information contained in the original input distribution. Fourth, any data transformation method satisfying the above requirements may be used in tandem with a standard DNN model under our unified framework.

V. Adversary-Resistant DNN

One of the design goals described in the previous section called for a data transformation that inverting would be computationally intractable. This property allows for the transformation to effectively block the end-to-end gradient flow. However, it is also important to ensure that the data transformation module used preserves as much of the significant structure in the input data as possible.

In this paper, we chose the our data transformation to be Local Linear Embedding (LLE), a representative non-parametric dimensionality reduction method. As we will show in this section, a non-parametric method like LLE can be proven to block end-to-end gradient flow in a neural architecture. More importantly, it can be theoretically proven that inverting LLE is an NP-hard problem. Last but not least, LLE seeks low-dimensional, neighborhood-preserving map of high-dimensional input samples (see Figure 5), and thus is a method that seems best suited to preserving as much information in the input as possible. We will first describe LLE and then prove that, as a non-parametric dimensionality reduction method, LLE can block end-to-end gradient flow. Furthermore, we will theoretically prove LLE is computationally non-invertible.

A. Local Linear Embedding

LLE is a non-parametric method designed to preserve local properties of high-dimensional input by constructing a graph representation. This graph is built by representing each high-dimensional data point via a linear combination of its nearest neighbors. As a result, the local proprieties of the high-dimensional manifold are encoded in the weights of these combinations for all input data points. The reconstruction weights are computed by solving following optimization problem:

$$\begin{aligned} \min_{W} \quad & \sum_{i} \left\| x_i - \sum_{j} w_{ij} x_j \right\|_2^2 \\ \text{s.t.} \quad & \sum_{j} w_{ij} = 1, \end{aligned} \tag{8}$$

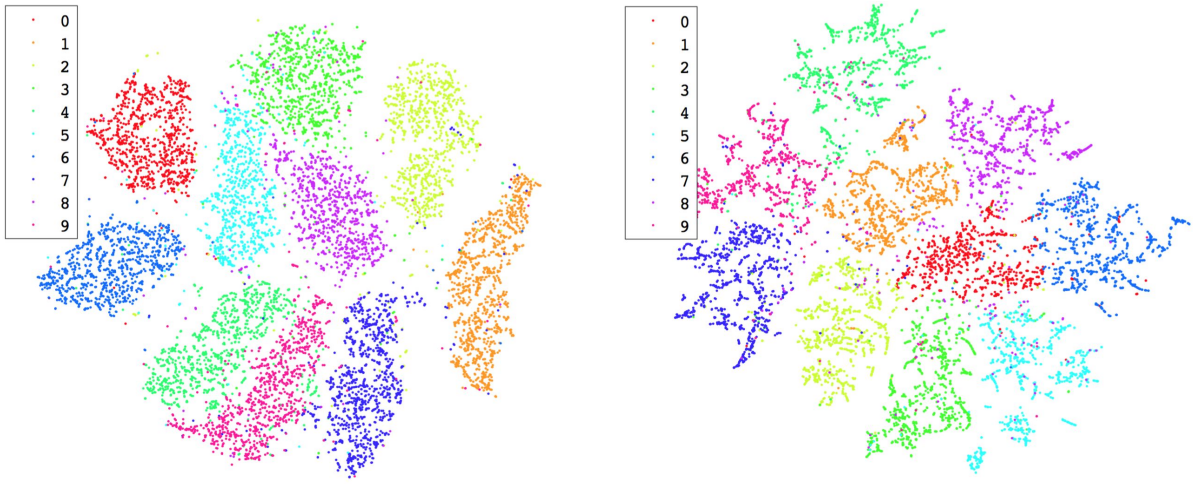
where w_{ij} represents the contribution of $x_j \in \mathbb{R}^m$ to reconstructing x_i. $w_{ij} = 0$ when x_j is not considered as a neighbor of x_i. The total number of neighbors assigned to x_i is a carefully selected hyper-parameter. The neighboring relation between x_j and x_i depends on the value of the l_2 distance between x_j and x_i.

LLE essentially fits a locally linear hyperplane through x_i and its nearest neighbors. This hyperplane is further mapped to some lower-dimensional space and thus preserves the local similarity of high-dimensional data points. Since the low-dimensional data representations preserve local geometry of the hyperplane, the reconstruction weights w_i calculated for reconstructing x_i are also used for reconstructing low-dimensional representations y_i from its neighbors. Furthermore, since it is the local linear proprieties of this hyperplane that are preserved in w_i, it is implied that the reconstruction weight is invariant to rotation and rescaling. As a result, we can obtain the low-dimensional representation y_i by solving the following minimization problem:

$$\begin{aligned} \min_{Y} \quad & \sum_{i} \left\| y_i - \sum_{j} w_{ij} y_j \right\|_2^2 \\ \text{s.t.} \quad & \sum_{i} y_i = 0, \\ & \frac{1}{N} \sum_{i} y_i^{\mathrm{T}} y_i = I. \end{aligned} \tag{9}$$

where $y_i \in \mathbb{R}^{1 \times m_c}$ represents the ith row of data matrix $Y \in \mathbb{R}^{N \times m_c}$. The Rayleitz-Ritz theorem [16] shows that the coordinates of y_i, which minimize (9), are obtained by computing the eigenvectors corresponding to the smallest m_c nonzero eigenvalues of the inner matrix product $(I-W)^{\mathrm{T}}(I-W)$ (for a detailed explication we refer the readers to [16]). Here $I \in \mathbb{R}^{N \times N}$ is the identity matrix, and $W \in \mathbb{R}^{N \times N}$ is the reconstruction weight matrix.

LLE is specifically designed to retain the similarity between pairs of high dimensional samples when they are mapped to lower dimenons [31]. We provide a visualization of the data before and after being processed by LLE in Fig. 5.



(a) Visualization of original MNIST data.

(b) Visualization of LLE mappings of MNIST data.

Figure 5: **Visualization results:** To better verify that LLE does preserve the distribution of original data points in their lower-dimensional projections, we further utilize MNIST images (later used for experiments in Section VI) as examples and visualize the distribution of these examples before and after processed by LLE. More specifically, we employ t-SNE [22] to project the original MNIST images and their LLE mappings onto a 2D space. LLE dimensionality is set to 200 (the choice of output dimensionality size is investigated in Section VI). As shown in Fig. 5a and Fig. 5b, LLE does indeed capture the essence of the distribution of the original MNIST samples by well separating the dissimilar samples and collecting the similar ones (marked by 10 different colors) in its lower-dimensional map. In addition, notie that the visualization shown in Fig. 5b of LLE mappings is more dense in comparison with Fig. 5a. This is due to the fact that LLE tends to cluster points in lower-dimensional mapping space [36]. We also provide the visualizations of two other datasets used in experiments in Section VI in Figure 8.

The visualization result demonstrates that using LLE satisfies our third design requirement, capturing the original data's statistical structure. More importantly, this property helps bound the lower dimensional mapping of black box adversarial samples to a vicinity where it is filled by mappings of original test samples that are highly similar to these black box adversarial samples. As a result, there is a significantly lower chance that an adversarial sample acts as an outlier in the lower dimensional space. In addition, LLE is reported to maintain desirable performance even when the dimension of transformed data is relatively high [36]. Other dimensionality methods like the Sammon Mapping are more specialized, designed more for tasks like visualization [22] where it is required that the dimensionality of the mappings is two or three. However, one would not expect only as few as 2 or 3 dimensions to capture enough about the input distribution, for even in methods like Principal Components Analysis (PCA), it is quite common to select much more than three dimensions in order to build useful predictive models, which themselves are only useful in the regime of high dimensionalities, on top of (for while the components are ordered according to how much variance in the data they explain, it is typical to select the k most informative dimensions).

B. Proof of Blocking the Gradient Flow

Here, we theoretically prove that a non-parametric dimensionality reduction method has the characteristic of blocking an end-to-end gradient flow. Since LLE is a non-parametric dimensionality reduction method itself, this will implicitly show that LLE is capable of blocking end-to-end gradient flow.

Existing dimensionality reduction methods can be categorized as either parametric or non-parametric [36]. Parametric methods utilize fixed a number of parameters to specify a direct mapping from high-dimensional data samples to their low-dimensional projections (or vice versa). This direct mapping is characterized by adopted parameters, which are typically optimized to provide the best mapping performance. This is much alike the functionality provided by a standard DNN. For example, when viewing a standard DNN as a parametrized mapping function, it maps high-dimensional data samples to the final decision space. Recall our introduction of end-to-end gradient flow in Section II. By switching F and L to any parametric dimensionality reduction method (in tandem with a softmax classifier) and its corresponding cost function, it is obvious that the end-to-end gradient flow also exists within the considered parametric method. This parametric nature is a disadvantage for blocking the end-to-end gradient flow. On the contrary, non-parametric methods do not suffer from this issue as there exists no direct mapping between the input and output. Additionally, since most dimensionality reduction techniques are non-parametric according to [36], this also enriches the arsenal for blocking the end-to-end gradient flow.

Although we use $\partial^n L(F(x), y)/\partial x^n$ to denote the gradient of the original input x, it is sufficient to show that any non-parametric transformation method can block the gradient flow under the condition of $n = 1$. This is due to the fact that the

calculation of the gradient with $n \geq 2$ is definitely based on the gradient with $n = 1$. More formally,

$$\begin{aligned}\frac{\partial^n L(F(x), y)}{\partial x^n} =& \partial^{n-1}(\frac{\partial L(F(x), y)}{\partial x})/\partial x^{n-1} \\ & n \geq 2, n \in \mathbb{N}.\end{aligned} \tag{10}$$

Now assuming $n = 1$, our proposed adversary-resistant DNN F can be represented as the composition of two functions f and g, that is,

$$F(x) = (f \circ g)(x) = f(g(x)). \tag{11}$$

Where f represents a standard DNN and g represents a non-parametric transformation module stacked in front of f. Given the residue error at the output end as $\partial L(F(x), y)/\partial F(x)$, we then have the gradient at the input end by the chain rule of derivatives as follows:

$$\begin{aligned}\frac{\partial L(F(x), y)}{\partial x} &= \frac{\partial L(F(x), y)}{\partial F(x)} \frac{\partial F(x)}{\partial x} \\ &= \frac{\partial L(f(g(x)), y)}{\partial f(g(x))} \frac{\partial f(g(x))}{\partial g(x)} \frac{\partial g(x)}{\partial x}\end{aligned} \tag{12}$$

$\partial L(F(x), y)/\partial F(x)$ in (12) refers to the residue error and $\partial f(g(x))/\partial g(x)$ refers to the gradient propagated backwards through the DNN. Both of these terms can be calculated by back-propagation. However, $\partial g(x)/\partial x$ represents the gradient back-propagated through the non-parametric transformation. As discussed above, it is impossible to calculate $\partial g(x)/\partial x$ when there exists no direct mapping characterized by fixed parameters. Therefore, the gradient with respect to the input (before the data transformation) cannot be obtained via this approach.

A rational alternative approach for obtaining $\partial g(x)/\partial x$ is to calculate the gradient by definition. Since the input x is commonly a vector, $\partial g(x)/\partial x$ consists of the partial derivatives of g with respect to all of the components of x. Calculating the partial derivative by definition strongly depends on the calculation of the limit. The limit, however, cannot be solved when the inputs are discrete points, which is always the case for deep architectures. Furthermore, when jointly considering the discrete nature of input samples and their finite number, it is difficult to define the continuity of the mapping with traditional topology. As a result, the differentiability of the mapping cannot be ensured, which makes the computation of the gradient by definition inaccurate, and as a result, unreasonable.

It follows from the above that it is guaranteed that a non-parametric transformation can block the end-to-end gradient flow. Motivated by this result, we employ a classical, non-parametric dimensionality reduction method - Local Linear Embedding (LLE), for building an adversary-resistant DNN architecture.

C. Proof of the Non-invertibility of LLE

In this sub-section, we theoretically prove that reconstructing original high-dimensional data from the low-dimensional representations transformed produced by LLE is computationally intractable. Specifically, we prove that, given the low-dimensional data representation $Y \in \mathbb{R}^{N \times m_c}$ obtained by LLE, reconstructing high-dimensional data $X \in \mathbb{R}^{N \times m}$ from Y is NP-hard.

LLE utilizes the reconstruction weights W in order to preserve the distribution between high-dimensional data points in a lower-dimensional space. W is also essential in reconstructing high-dimensional data. More specifically, we can recover the high-dimensional X by reconstructing the locally linear hyperplane that resides in high-dimensional space from the lower-dimensional Y. Given Y, we can first compute the reconstruction weights for Y and its neighbors. Weight matrix $W \in \mathbb{R}^{N \times N}$ is calculated similarly as shown in (8), except that x_i and x_j are replaced by y_i and y_j. More formally, the reconstruction of X using W and Y is to solve following optimization problem:

$$\min_X \sum_i \left\| x_i - \sum_j w_{ij} x_j \right\|_2^2. \tag{13}$$

Problem (13) essentially defines a optimization problem with quadratic form:

$$\min_X \sum_{i,j} m_{ij}(x_i \cdot x_j), \tag{14}$$

where $m_{ij} = \delta_{ij} - w_{ij} - w_{ji} + \sum_k w_{ki} w_{kj}$, $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. It is easy to see that $M^{\mathrm{T}} = M$. Until now there have been no clear constraints imposed on (14). Therefore, in what follows, we first consider solving (14) by assuming that there are no constraints.

Problem (14) is equal to $R(X) = X^{\mathrm{T}} M X$ if we transform (14) by matrix multiplication. It then follows that the derivative of $R(X)$ with respect to X is:

$$\begin{aligned}\frac{\mathrm{d}R(X)}{\mathrm{d}X} &= X^{\mathrm{T}} M \frac{\mathrm{d}X}{\mathrm{d}X} + X^{\mathrm{T}} M^{\mathrm{T}} \frac{\mathrm{d}X}{\mathrm{d}X} \\ &= 2X^{\mathrm{T}} M.\end{aligned} \tag{15}$$

Equation (15) is obtained under the condition $M^{\mathrm{T}} = M$. The optimal X for minimizing $R(X)$ can be calculated when $\mathrm{d}R(X)/\mathrm{d}X = 0$. This further leads to:

$$2X^{\mathrm{T}} M = 0. \tag{16}$$

Note that the solution of the linear equation (16) depends on the rank(M). If rank(M)=N, (16) has a unique solution, $X = 0$. Otherwise, if rank(M)<N, the optimal solution for (16) cannot be determined without any other constraint. This is obviously not a desirable method when considering the practical case. Therefore, assuming that there are no constraints for (14) is invalid.

It is natural to consider adopting the two constraints in (9). In particular, the first constraint $\sum_i y_i = 0$ can be similarly applied to (14) without affecting the cost. As for the second constraint, $\frac{1}{N} \sum_i y_i^{\mathrm{T}} y_i = I$, it is noted that Y is column-wise orthogonal. This implies that Y represents original data

perfectly without containing redundancy. As a result, this second constraint cannot be applied to (14), as there is no way of mapping a low-dimensional representation to a high-dimensional space while maintaining the property of column-wise orthogonality. Indeed, upon closer examination of X, one may observe that X can be constructed from Y by taking the columns of Y as the basis for the column space of X. This further implies that there exists a correlation between any pair of columns of X. Given the above analysis, it is reasonable to keep the first constraint while removing the second constraint. More formally, we have:

$$\sum_i x_i = 0. \tag{17}$$

Let $A \in \mathbb{R}^{Nm\times 1}$ denote a column vector which is the concatenation of x_i for $i = 1, \ldots, Nm$. Then we have $A = (x_1, x_2, \cdots, x_N)^{\mathrm{T}}$. Let $q \in \mathbb{R}^{1\times Nm}$ denote a row vector in which all components are equal to 1, then we have $q = (1, 1, \cdots, 1)$. We further define two matrices $P, Q \in \mathbb{R}^{Nm\times Nm}$ as follows:

$$\begin{aligned} P &= \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{pmatrix}, \\ Q &= -I, \end{aligned} \tag{18}$$

where $P_{ij} = m_{ij}E$. $E \in \mathbb{R}^{m\times m}$ is a matrix with all its components equal to 1. Since it is trivial to show that $-\|x_i\|_2^2 \leqslant 0$ for all i, then (14) can be transformed into following form:

$$\begin{aligned} &\min \ A^{\mathrm{T}}PA \\ &\text{s.t.} \ \ A^{\mathrm{T}}QA + qA \leq 0. \end{aligned} \tag{19}$$

According to [9], [37], the optimization problem shown in (19) is a quadratic problem with a non-positive semi-definite constraint. As introduced in [9], this kind of problem belongs to a class of NP-hard problems. Recall our proof that LLE, as a non-parametric dimensionality reduction method, can block the end-to-end gradient flow. It suffices to show that an architecture, built from a non-parametric data transformation module, such as LLE, followed by a standard DNN is theoretically guaranteed to be resistant to white box adversarial samples.

VI. Evaluation

As we will describe in Section VII, adversarial training [11] and defensive distillation [29] are the most representative techniques that have been proposed to defend against adversarial samples. Prior studies [25], [29] empirically indicate both techniques are effective in improving the robustness of DNNs under the black box attack scenario. In this work, we evaluate our adversary resistant DNN (LLE-DNN) and compare it against approaches as proposed in [25], [29].

A. Dataset

We evaluate our proposed adversary-resistant models by performing multiple experiments on several widely used datasets, including the MNIST dataset [20], the CIFAR-10 dataset [18], and a malware dataset [5].

MNIST. The MNIST dataset is composed of 70,000 greyscale images (of 28×28, or 784, pixels) of handwritten digits, split into a training set of 60,000 samples and a test set of 10,000 samples. The classification task is simple digit recognition, where the possible digits range from 0 to 9.

CIFAR-10. 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 channels, indicating different scales of RGB colors.

Malware dataset. The malware dataset we experimented with is a collection of window audit logs[6]. The dimensionality of the feature-space for each audit log sample is reduced to 10,000 according to the feature select metric in [5]. Each feature indicates the occurrence of either a single event or a sequence of events[7], 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 the presence of malicious software, or 0, indicating benign software. 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.

B. Experimental Design

To make our evaluation as comprehensive as possible, we conducted our experiments in both white-box and black-box attack scenarios. Specifically, we measure classification accuracy as well as model resistance to adversarial samples. The results of our experiments using the datasets described above can be found in Tables VI, VII, VIII, IX, X.

To evaluate and compare the resistance of all three defense techniques, we test the DNN models against adversarial samples, generated from both the models we trained and the models we approximated. As a result, the samples created in these two settings represent the adversarial samples created in black-box and white-box attack scenarios. This means that we created three adversarial sample pools, one for each dataset (i.e., MNIST, CIFAR-10, the malware dataset).

1) Crafting Adversarial Samples for Image Classification: We generate adversarial images from the testing samples in the MNIST and CIFAR-10 datasets. For both black-box and white-box attack scenarios, the two common approaches include the fast gradient sign (FGS) attack and the optimal (OPT) attack

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

[7]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:[fonts]\times.ttf, WINDOWS_FILE:Execute:[fonts]\times.ttf
WINDOWS_FILE:Execute:[system]\faultrep.dll, WINDOWS_FILE:Execute:[system]\imm32.dll

Table I: **Illustration of manipulated features in malware dataset:** each feature in this table contains a sequence of two events. The two events in each row happened in the same order as displayed.

(as introduced in Section II). In implementing an optimal attack, we selected l_2 and l_∞ distance to represent the dissimilarity between image samples. Notably, l_2 distance measures the standard Euclidean distance between image samples and is a commonly used metric for image recognition applications. In our experiments using the image datasets, we make use of both l_2 and l_∞.

2) Crafting Adversarial Samples for Malware Classification: A bit of care must be taken when generating adversarial samples for the malware dataset. Malware samples are usually represented by features having 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. For example, [12] use l_0 distance to calculate the number of perturbations that occurred among all features in malware examples taken from the DREBIN Android malware dataset. Furthermore, as discussed in [12], malware data contains stricter sentiment compared to image data. 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 on all features, as done with image data, may break down a malware'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. Last but not least, since malware manipulation is done with the intent of fooling a DNN malware classifier, there is no need to modify a benign application so that it is classified as malicious. Therefore, in our experiments we only generate adversarial samples from malware data samples as described in this section. In Table I, we show a few examples of features added to a malware sample. These added features only cause the malware to call several dynamic link library files.

C. DNN Architectures

Here, we specify the architectures of the DNN models we trained for our experiments. In total, we implemented twelve DNNs in following experiments, which fall under–standard DNNs, LLE-DNNs, adversarially trained DNNs, and defensive distillation enhanced DNNs. For each category, we implement three different models for evaluating the classification accuracy and resistance on MNIST, CIFAR-10, and the malware dataset. For details regarding hyper-parameters and further architectural aspects of these implemented DNNs, please refer to the Appendix.

D. Experimental Results

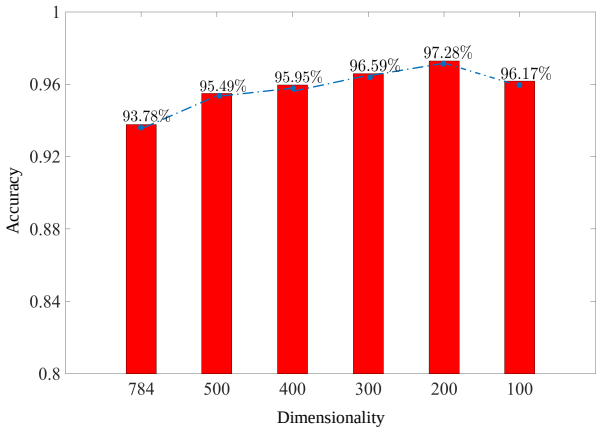
1) Image Classification: On the MNIST dataset and CIFAR-10 dataset, we first measure the accuracy of all aforementioned defense techniques. We then measure their adversarial resistance to both black-box and white-box adversarial samples.

Classification Accuracy. We present in Table II and Fig. 6 the classification accuracy results obtained by all investigated DNNs with respect to the testing sets. In particular, we implement several LLE-DNNs with different settings of the dimensionality of LLE mappings. This illustrates the impact of the LLE mappings' dimensionality on classification accuracy obtained by LLE-DNN.

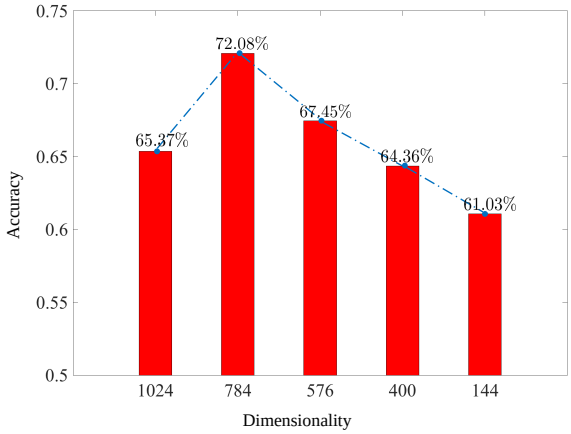
As shown in Table II, the LLE-DNN is quite effective in preserving classification accuracy on both MNIST dataset and CIFAR-10 dataset. For the MNIST dataset, we show the classification accuracy obtained by an LLE-DNN when LLE mapping' dimensionality is allowed to vary from 100 to 784 in Fig. 6a. All of these LLE-DNNs yielded desirable results, with an average classification accuracy of 95.88%. When compared to standard DNN, adversarial training and defensive distillation, as shown in Table II, LLE-DNN shows but negligible degradation in accuracy. In order to better understand these results, bear in mind that other three DNNs all utilize end-to-end training, integrating automated feature extraction into model learning. Therefore, these three DNNs can attribute their better classification accuracy to their feature learning ability. Defensive distillation especially achieves the highest accuracy of 98.77% due to its enhanced model complexity (introduced in Section VII), which further improves its

Defence Mechanism	Dataset	
	MNIST	CIFAR-10
Standard DNN	98.49%	81.08%
Adv. Training	98.13%	79.10%
Distillation	98.77%	82.91%
LLE-DNN	97.28% (LLE-200)	72.08% (LLE-784)

Table II: **Comparison of classification accuracy on image datasets:** for MNIST dataset, we train all investigated DNNs to achieve classification accuracy comparable to the baseline. For CIFAR-10 dataset, the highest classification accuracy obtained by LLE-DNN is comparable to the result reported in [1].



(a) The changes of classification accuracy with varying LLE mappings' dimensionality on MNIST.



(b) The changes of classification accuracy with varying LLE mappings' dimensionality on CIFAR-10.

Figure 6: **Influence of LLE mappings' dimensionality on accuracy:** in these two graphs we demonstrate the variations of classification accuracy on MNIST and CIFAR-10 datasets. On both datasets we train several different LLE-DNNs with different settings of LLE mapping's dimensionality with same training set. The accuracy rates were obtained by testing these LLE-DNNs on the corresponding test set. Note that in Figure 6b, we mapped the different channel of each RGB image to the same dimensionality, and coordinates of along horizontal axis imply the size of each mapped image, e.g. 1024 represents a dimensionality of $32 \times 32 \times 3$ pixels.

learning ability. A similar effect can also be observed in the classification results with CIFAR-10 dataset.

With respect to the CIFAR-10 dataset, we also implement several different LLE-DNNs with varying LLE dimensionalities and show the variations in accuracy in Fig. 6b. Among all these LLE-DNNs, the highest accuracy achieved is 72.08% when LLE mapping dimensionality is set to $28\times28\times3$. Here we also observe insignificant degradation of accuracy in comparison to the accuracy results obtained by other DNNs in Table II. In addition, the degradation shown on CIFAR-10 dataset is more noticeable than that reported on MNIST dataset. This indicates that the difference in the learning ability between LLE-DNNs and other DNNs is dataset-specific. Since CIFAR-10 dataset is less sparse in comparison with MINIST data, end-to-end training can result in further improved generalization when the dataset contains more complex data. As a result, the advantage of having automated feature extraction is amplified on CIFAR-10 dataset.

However, as we will later show with the malware dataset, when the sparsity of the dataset is greater, the LLE-DNN appears to be better at feature learning, achieving the highest classification accuracy. Despite the negligible degradation of classification accuracy caused by LLE-DNN, it is important to show that LLE is capable of retaining most of the critical information inherent in the original data (we visualize this property in the Appendix). Furthermore, recall that it is the end-to-end training that facilitates the design of more model-specific adversarial samples in the white-box attack scenario. In contrast, LLE-DNN prohibits the generation of model-specific adversarial samples and at cost of negligible classification performance degradation. Similar degradation of classification accuracy has also been reported in [1], where the authors employed data transformation to preserve data privacy.

We further explore the impact of LLE mappings' dimensionality on classification accuracy. For the MNIST dataset, as shown in Fig. 6a, the classification accuracy first increases when the dimensionality of the LLE mappings rises from 100 to 200 (since more critical information is being preserved with additional LLE dimensions). However, beyond 200, the accuracy then starts to decrease as the LLE mapping dimensionality continues to increase to 784. While the decrease of accuracy suggests that the original MNIST data contains a good deal of redundant information which can ultimately be treated as noise. This is more obvious when we use LLE to map the original data onto an equally high-dimensional space. In this case, LLE-DNN obtains the worst accuracy–the redundancy is due to the fact that MNIST samples are sparsely represented. Such image samples contain digits only at the central region of images and set the value of many pixels residing in background region to 0. In addition, we observe a similar trend of changing on CIFAR-10 dataset in Fig. 6b. As previously discussed, the CIFAR-10 dataset is less sparse than the MNIST data. Therefore, the classification accuracy keeps increasing when the LLE mappings dimensionality increases from $12\times12\times3$ to $28\times28\times3$, as shown in Table II. This increase vanishes when we use LLE to map the original $32\times32\times3$ dimensional data to an equally high-dimensional space. According to above analysis, we selected the final dimensionality of LLE mappings to be 200 for the MNIST dataset and to be $28\times28\times3$ for CIFAR-10 dataset for the experiments that follow.

Defence Mechanism	MNIST				Defence Mechanism	CIFAR-10			
	Black-Box Adv.		White-Box Adv.			Black-Box Adv.		White-Box Adv.	
	OPT_2	FGD	OPT_2	FGD		OPT_2	FGD	OPT_2	FGD
Adv. Training	94.55%	90.06%	4.44%	33.94%	Adv. Training	69.52%	69.14%	32.75%	50.45%
Distillation	96.22%	87.06%	12.60%	34.43%	Distillation	56.95%	57.04%	35.68%	45.55%
LLE-DNN (LLE-200)	**95.28%**	**96.04%**	-	-	LLE-DNN (LLE-1000)	**63.34%**	**61.58%**	-	-

Table III: **Comparison of resistance on image datasets:** in the black-box attack scenario, all investigated DNNs obtain much lower classification accuracy on CIFAR-10 dataset in comparison with the results obtained on MNIST dataset. On the contrary, these same defense techniques demonstrate stronger resistance to adversarial samples in the white-box attack scenario. This further indicates that the resistance provided by DNNs is also data-dependent.

Resistance to Adversarial Samples. In this experiment, we compare the adversarial resistance of the LLE-DNN with a standard DNN, as well adversarial trained and defensive distillation enhanced DNNs in both black-box and white-box attack scenarios. The results are shown in Table III.

For the MNIST dataset, we first demonstrate the efficacy of adversarial samples generated in the black-box attack scenario. Specifically, we show that black-box adversarial samples generated with a standard DNN can lower the model's accuracy to 6.85% and 6.40% under using OPT_2 attack and FGS attacks respectively. In contrast, all of the investigated defense mechanisms demonstrate strong resistance to these same black-box adversarial samples. In particular, all three defenses demonstrate comparable resistance to black-box adversarial samples generated by OPT_2 attacks. However, when the FGS attack is utilized, the LLE-DNN (with 200 dimensional LLE mappings) obtains the best resistance (correctly classify 96.04% of the FGS adversarial samples), whereas the defensive distillation approach yields the worst resistance of 87.06%. This experiment result is consistent with that reported in [26].

Under the white-box threat model, it is shown in Table III that both adversarial training and defensive distillation suffer from white-box adversarial samples. Their resistance to white-box adversarial samples is significantly worse than those created under the black-box threat model. Despite this defeat, however, it can be observed that defensive distillation provides stronger resistance under the white-box attack scenario when comparing with adversarial training. This may be due to that defensive distillation trains a DNN with additional information of classification probability, which facilitates a better generalization around data samples. In addition, for adversarial training, it can be noticed that it demonstrates much better resistance to FGD attack. This superior performance results from that FGD attack is designed to approximate OPT_∞ attack, thus has inferior attack power. While for LLE-DNN, note that as proved in Section V, our proposed DNN provides theoretically guaranteed resistance to white box adversarial samples. Therefore, we only evaluated our proposed DNN with black box adversarial samples.

For CIFAR-10 dataset, we use the standard DNN previously trained with CIFAR-10 to generate black-box adversarial samples. The generated adversarial samples cause this DNN to obtain the accuracy of 33.76% and 28.72% under OPT_2 attack and FGS attack respectively. When comparing with the standard DNN trained with MNIST dataset, this standard DNN trained with CIFAR-10 dataset demonstrates much better resistance to black-box adversarial samples. This can be explained by highlighting the difference between the architectures of these two standard DNNs. For MNIST dataset, as shown in Appendix, we build a standard feed-forward fully connected DNN, while for CIFAR-10 dataset, we build a convolutional neural network (CNN). Since the convolutional DNN contains max pooling layer within its architecture, the max pooling layer functions as a filter thus is capable of filtering out some perturbations conducted on original CIFAR-10 data. Therefore, CNN is naturally more robust than DNN.

Using the generated black-box adversarial samples, we further evaluate the resistance provided by LLE-DNN, adversarial training and defensive distillation on CIFAR-10 dataset. As shown in Table III, adversarial trained DNN achieves the best resistance to black-box adversarial samples, followed by LLE-DNN in the second place. In particular, adversarial training takes advantages of data augmentation, which leaves the trained DNN be more familiar with adversary-alike testing samples. While LLE-DNN demonstrates limited resistance to black-box adversarial samples on CIFAR-10 dataset. This is because LLE, acting as a filter, only filters out limited amount of perturbations, given that CIFAR-10 data is much less sparse than MNIST. As for defensive distillation, it only achieves the resistance of 56.95% and 57.04% under OPT_2 attack and FGS attack respectively. Both results are different from the result reported in [29]. This is because in our case we generate OPT_2 and FGS adversarial samples using l_2 and l_∞ distance as metrics, which are more appropriate than l_0 distance (used in [29]) for image data. In addition, when conducting manipulations on original data, l_2 and l_∞ distance can incur perturbations at all pixels by a small amount, while l_0 distance causes large amount of perturbations on a few pixels. This indicates that the latter approach put more emphasis on making the manipulation less noticeable. While in our case, we put more emphasis on generating more powerful adversarial samples to evaluate LLE-DNN more comprehesively.

In the white-box attack scenario, the resistance demonstrated by defensive distillation and adversarial training is comparable to that shown by standard DNN under OPT_2 attack. While for FGD attack, former two defenses are more resistant to white-box adversarial samples, attributed to their enhanced model complexity and augmented training set (see Section VII).

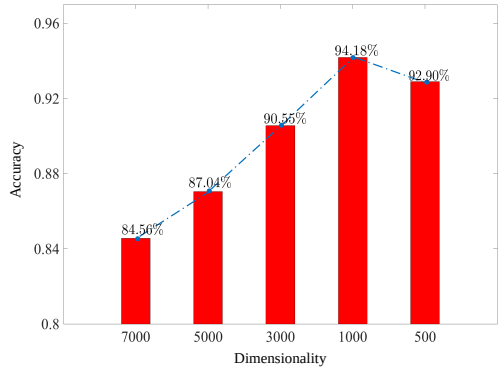


Figure 7: **Influence of LLE mappings' dimensionality on accuracy:** we demonstrate the variations of classification accuracy on malware dataset. We train several different LLE-DNNs with settings of LLE mapping's dimensionality range from 1,000 to 7,000. The accuracy rates were obtained by testing these LLE-DNNs on the corresponding test set.

2) Malware Classification: In this part, we first implement an experiment to evaluate the classification accuracy provided by LLE-DNN and other defense mechanisms when testing with malware testing samples. Then we use a second experiment to compare the resistance of aforementioned DNNs when confronting adversarial samples.

Classification Accuracy. Similar to the case of image classification, we implement several different LLE-DNNs by selecting different dimensionality of LLE mapping. The experiment results of evaluating these different LLE-DNNs, standard DNN and aforementioned defense mechanisms are shown in Fig. 7 and Table IV.

As shown in Fig. 7, adversarial trained and defensive distillation enhanced DNNs get similar results, which are slightly lower than the accuracy achieved by a standard DNN. As for LLE-DNN, it obtains an average classification accuracy of 89.85% among its different implementations. The highest accuracy achieved is 94.18% when the dimensionality of LLE mapping is equal to 1,000. As shown in Table IV, this is also the highest classification accuracy achieved among all investigated DNNs.

Recall that the dimensionality of original malware samples is as high as 10,000, while the experiment results show that the crucial information contained in original data is best preserved in a 1,000-dimensional space. This indicates that malware data is highly sparse than the other two image datasets. As previously discussed, LLE-DNNs can demonstrate better learning ability when confronting more sparse dataset. This explains the superior accuracy achieved by LLE-DNN. In addition, it can be noticed in Fig. 7 that the impact of LLE dimensionality on classification accuracy is similar to that shown on both MNIST dataset and CIFAR-10 dataset. The only difference is that there is more significant accuracy degradation when the dimensionality of LLE mapping increases. This is because the added dimensionality of highly sparse malware data leaves much more noise in LLE mappings, hence making it more challenging for LLE to retain the classification accuracy. With this result, we set LLE mappings' dimensionality to 1,000 for further evaluation of LLE-DNN's provided resistance to adversarial samples in following experiments.

Defence Mechanism	Classification Accuracy
Standard DNN	93.99%
Adv. Training	92.68%
Distillation	92.07%
LLE-DNN (LLE-1000)	**94.18%**

Table IV: **Comparison of classification accuracy on malware dataset:** the LLE-DNN is built by setting LLE mappings' dimensionality to 1,000.

Defence Mechanism	Black Box Adv. (OPT_0)	White Box Adv. (OPT_0)
Adv. Training	92.76%	26.07%
Distillation	29.53%	7.23%
LLE-DNN (LLE-1000)	**91.09%**	-

Table V: **Comparison of resistance:** the experiment results implies that adversarial training can maintain its defensive power against black-box adversarial samples even with rather sparse dataset. On the contrary, the effectiveness of the defense provided by defensive distillation is more data-dependent.

Resistance to Adversarial Samples. In the second experiment, we evaluate the resistance to adversarial samples provided by LLE-DNN and aforementioned defense mechanisms.

We first utilize above mentioned standard DNN to generate adversarial samples from the testing set of malware data. We generate adversarial samples using optimal OPT_0 attack. In this experiment we omit FGS attack, as FGS attack is an approximation of OPT_∞ attack and also inappropriate for malware dataset [12]. Then we test the standard DNN with these adversarial samples and obtain only 30% classification accuracy. This is unacceptable for security critical applications like malware classification tasks. The adversarial samples generated in this approach represent the adversarial samples created in black-box scenario.

As shown in Table V, both LLE-DNN and adversarial training demonstrate strong resistance to black-box adversarial samples. On the contrary, defensive distillation is less effective, given its obtained resistance of 29.53% is even lower than the resistance obtained by the standard DNN. This result is rather surprising given the very competitive resistance obtained by defensive distillation on image datasets. This may be due to that the probabilities assigned to each class (positive or negative) of samples are not as informative as those assigned to MNIST samples and CIFAR-10 samples. As a result, the mechanism proposed by defensive distillation to utilize class probabilities as *soft* labels [12] to enhance model complexity may fail on this malware dataset, as the assigned labels are not as soft as expected. Furthermore, this analysis can be validated by observing the resistance obtained by defensive

distillation in white-box attack scenario. In this case, defensive distillation gets significant lower resistance when comparing with adversarial training. However, it should be noticed that adversarial training also provides rather limited resistance (26.07%) in the white-box attack scenario. This is as expected since the white-box adversarial samples used to test adversarial training are specifically crafted to fool its enhanced DNN.

VII. Related Work

In this section, we classify existing defense mechanisms under one of two categories. In particular, we discuss the strengths of the methods each category encompasses as well as limitations.

A. Existing Defense Mechanisms

Existing solutions fall either under 1) augmenting the training set, or 2) enhancing model complexity.

1) Data Augmentation: To resolve issue of "blind spots" (the more informal name given to adversarial samples), many methods that could be considered as sophisticated forms of data augmentation[8] have been proposed (e.g. [11], [13], [25]). In principle, these methods expand their training set by combining known samples with potential blind spots, the process of which has been called adversarial training [11]. 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 standard DNNs typically learn from, the input space is generally considered infinite [11]. This implies that there could also exist an infinite amount of blind spots that are model-specific, especially to standard DNN models. Therefore, data augmentation based approaches 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 [25]. The regularization penalizes the directions uncovered by adversarial perturbations (introduced in Section II). 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 [11], [25], adversarial samples can be generated efficiently for a particular type of DNN. The fast gradient sign method [11] can generate a large pool of adversarial samples quickly while *DataGrad* [25] 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 in order to attack the adversarial training method *itself*. Given that there exist infinite adversarial samples, we would need to repeat an adversarial training procedure each time a new adversarial example is discovered. Let us briefly consider DataGrad [25], 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. More importantly, adversarial training in general does nothing to block the end-to-end gradient flow, which we argue is the more direct and ultimately more important way in building a strong defense for DNN architectures.

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, when the underlying architecture is straightforward when it comes to facilitating the flow of information forwards and backwards, greatly alleviating the effort in generating adversarial samples. Therefore, several ideas [13], [29] 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.

[29] 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 DNN classifiers become less sensitive to adversarial samples. Although shown to be effective, this method still relies on the end-to-end gradient flow from output to input. 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. [13] proposed stacking an auto-encoder together with a standard DNN, similar to [29]. It was first shown that this auto-encoding enhancement increases DNN resistance to adversarial samples. However, the authors further demonstrate that this stacked model can be simply taken as a new DNN and easily generate new model-specific adversarial samples.

Though the aforementioned approaches, either from data-augmentation and or model complexity perspective, have proven effective in handling samples generated from simple DNN models, they do not handle model-specific adversarial samples designed to attack the methods themselves. When using them for malware detection, since these methods do not provide theoretical guarantees, with a bit of knowledge, malware authors can easily bypass detection. In light of this, this paper offers a unified framework to facilitate the design of adversary-resistant DNNs.

[8]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.

VIII. Conclusion

We proposed a new framework for constructing deep neural network models that are resistant to adversarial samples. The framework is designed based on an analysis of the adversarial learning problem faced by neural network models and the limitations of current solutions.

Following our proposed framework, we developed an adversary-resistant DNN that leverages non-parametric dimensionality reduction methods. More importantly, we theoretically proven that it is impossible for an adversary to craft an adversarial sample to attack a model learned under our framework. In short, our proposed framework provides theoretical guarantees of resistance to adversarial samples.

Empirically, we have demonstrated that two recently proposed defense mechanisms are insufficient when it comes to comprehensively defending against adversarial perturbation. Applying our new framework to two image datasets and one malware dataset, we have shown that our framework effectively defends against adversarial samples. In addition, our framework maintains classification accuracy for out-of-sample data points with negligible degradation. Future work will entail investigating other data transformation methods, e.g. t-SNE [22], to lower the computational complexity for training our framework, and further explore the performance of this framework in a wider variety of applications.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] Y. Bar, I. Diamant, L. Wolf, and H. Greenspan. Deep learning with non-medical training used for chest pathology identification. In *SPIE Medical Imaging*, pages 94140V–94140V. International Society for Optics and Photonics, 2015.

[3] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 16–25. ACM, 2006.

[4] I. G. Y. Bengio and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[5] K. Berlin, D. Slater, and J. Saxe. Malicious behavior detection using windows audit logs. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*, pages 35–44. ACM, 2015.

[6] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[7] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. *arXiv preprint arXiv:1608.04644*, 2016.

[8] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu. Large-scale malware classification using random projections and neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3422–3426. IEEE, 2013.

[9] A. d?Aspremont and S. Boyd. Relaxations and randomized methods for nonconvex qcqps. *EE392o Class Notes, Stanford University*, 2003.

[10] C. Farabet, C. Couprie, L. Najman, and Y. LeCun. Scene parsing with multiscale feature learning, purity trees, and optimal covers. *arXiv preprint arXiv:1202.2160*, 2012.

[11] I. Goodfellow, J. Shlens, and C. Szegedy. Explaining and Harnessing Adversarial Examples. *CoRR*, 2014.

[12] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*, 2016.

[13] S. Gu and L. Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv:1412.5068 [cs]*, 2014.

[14] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 26(2):120–144, 2009.

[15] G. E. Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007.

[16] R. A. Horn and C. R. Johnson. Matrix analysis. corrected reprint of the 1985 original, 1990.

[17] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[18] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[20] Y. LeCun, C. Cortes, and C. J. Burges. The mnist database of handwritten digits, 1998.

[21] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[22] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[23] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011.

[24] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436. IEEE, 2015.

[25] A. G. Ororbia II, C. L. Giles, and D. Kifer. Unifying adversarial training algorithms with flexible deep data gradient regularization. *arXiv:1601.07213 [cs]*, 2016.

[26] N. Papernot and P. McDaniel. On the effectiveness of defensive distillation. *arXiv preprint arXiv:1607.05113*, 2016.

[27] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. *CoRR*, 2015.

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

[29] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *arXiv preprint arXiv:1511.04508*, 2015.

[30] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1916–1920. IEEE, 2015.

[31] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[32] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[33] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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

[35] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

[36] L. van der Maaten, E. O. Postma, and H. J. van den Herik. Dimensionality reduction: A comparative review, 2008.

[37] S. A. Vavasis. *Nonlinear Optimization: Complexity Issues*. Oxford University Press, Inc., New York, NY, USA, 1991.

[38] Y. Xu, T. Mo, Q. Feng, P. Zhong, M. Lai, I. Eric, and C. Chang. Deep learning of feature representation with multiple instance learning for medical image analysis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1626–1630. IEEE, 2014.

[39] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue. Droid-sec: Deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM, 2014.

IX. Appendix

A. DNN Hyper parameters

To augment the experimental setup presented in Section VI, the following tables indicates the hyper parameters that we used for model training.

B. Visualization Results on CIFAR-10 and malware Datasets

We first demonstrate that LLE preserves the distribution of original hign-dimensional data in the lower-dimensional mappings. More specifically, we employ a widely used visualization technique, t-SNE [22] to project original test samples and their corresponding LLE mappings from CIFAR-10 and the malware dataset onto 2D space as shown in Fig. 8. For each panel from top to bottom in Fig. 8, it shows the visualization results on CIFAR-10 and the malware datasets respectively. All visualizations on the left demonstrate the 2D projections generated from original test samples, while the visualizations on the right represent the 2D projections generated from LLE mappings for each dataset. The LLE mappings of CIFAR-10 and malware datasets used for visualization have the dimensionality of 784 and 1,000 respectively (the choices of these different lower dimensions were investigated separately for each dataset in Section VI).

As shown in Fig. 8a and Fig. 8b, LLE preserves the distribution of original CIFAR-10 test data by well separating dissimilar samples and collect similar ones (marked by 10 different colors) in the lower-dimensional mappings. The visualization result shown in Fig. 8b for LLE mapped CIFAR-10 samples are more dense in comparison with Fig. 8a. This is due the fact that LLE tends to cluster points in lower-dimensional mapping space [36]. Similar effects can also be observed comparing Fig. 8c with Fig. 8d. Particularly, as shown in Fig. 8c, t-SNE generates an eye-shaped 2D projection from some of the original malware samples. This eye-shaped 2D projection reflects an interesting distribution that these original samples follow. This interesting distribution is well preserved in LLE mapping results and presents itself in Fig. 8d.

The visualization results well demonstrate that using LLE satisfies the third design requirement previously discussed in Section IV, as LLE retains critical feature information of the original data from all datasets.

Training Algorithms		Hyper Parameters						
		DNN Structure	Activation	Optimizer	Learning Rate	Dropout Rate	Batch Size	Epoch
Standard DNN		784-500-300-10	Sigmoid	Adam	0.001	×	100	70
Adv. Training		784-100-100-100-10	Tanh	SGD	0.1	0.25	100	60
Distillation (T=20)		784-200-50-20-10	Tanh	SGD	0.1	0.25	100	100
LLE	784	784-300-100-10	Relu	Adam	0.001	0.5	100	50
	500	500-300-200-10	Relu	Adam	0.001	0.5	100	50
	400	400-200-100-10	Relu	Adam	0.001	0.5	100	50
	300	300-300-100-10	Relu	Adam	0.001	0.5	100	50
	200	200-200-100-10	Relu	Adam	0.001	0.5	100	50
	100	100-50-10-10	Relu	Adam	0.001	0.5	100	50

Table VI: **The Hyper parameters of MNIST models:** the network structure specifies the number of hidden layers and hidden units in each layer. For example, 784-500-300-10 indicates a DNN with 4 hidden layers, and the numbers of hidden units across each layer are 784, 500, 300 and 10, respectively. The last layer of each model is Softmax. We keep the hyper parameters of Distillation same with [29]

Training Algorithms		Hyper Parameters						
		DNN Structure	Activation	Optimizer	Learning Rate	Dropout Rate	Batch Size	Epoch
Standard DNN		10000-5000-1000-100-2	Relu	RMSprop	0.001	0.1	500	20
Adv. Training		10000-5000-1000-100-2	Relu	RMSprop	0.001	×	500	20
Distillation (T=20)		10000-1000-100-20-2	Relu	RMSprop	0.001	×	100	20
LLE	7000	7000-5000-3000-1000-100-2	Relu	RMSprop	0.001	0.5	500	20
	5000	5000-3000-1000-500-300-2	Relu	RMSprop	0.001	0.5	500	20
	3000	3000-1000-500-100-2	Relu	RMSprop	0.001	0.5	500	20
	1000	1000-500-200-100-2	Relu	RMSprop	0.001	0.5	500	20
	500	500-1000-500-2	Relu	RMSprop	0.001	0.5	500	20

Table VII: **The Hyper parameters of malware models:** the network structure is represented in the same way as VI. During experiment, we uncover that using dropout causes little improvement of performance for distillation and adversarial training.

Training Algorithms		Hyper parameters					
		Activation	Optimizer	Learning rate	Dropout rate	Batch Size	Epoch
Standard DNN		Relu	Adam	0.001	0.5	128	50
Adv. Training		Relu	Adam	0.001	0.5	128	50
Distillation		Relu	SGD	0.01	0.5	128	50
LLE	$32 \times 32 \times 3$	Relu	RMSprop	0.001	0.25	500	50
	$28 \times 28 \times 3$	Relu	RMSprop	0.001	0.25	500	50
	$24 \times 24 \times 3$	Relu	RMSprop	0.001	0.25	500	50
	$18 \times 18 \times 3$	Relu	RMSprop	0.001	0.25	500	50
	$12 \times 12 \times 3$	Relu	RMSprop	0.001	0.25	500	50

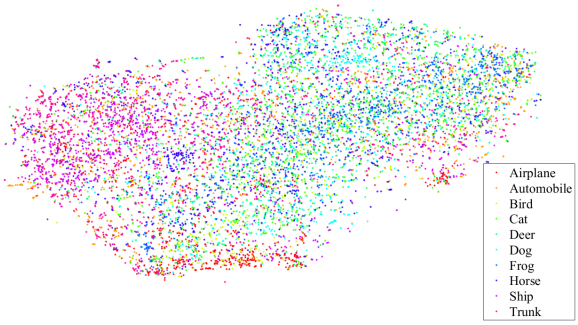
Table VIII: **The Hyper parameters of CIFAR-10 models:** the hyper parameters of distillation are set according to [29]

Layer type	Training Algorithms		
	Standard DNN	Adv. Training	Distillation
Convolutional	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)
Max pooling	2×2	2×2	2×2
Convolutional	72 filter(3×3)	72 filter(3×3)	128 filter(3×3)
Convolutional	72 filter(3×3)	72 filter(3×3)	128 filter(3×3)
Max pooling	2×2	2×2	2×2
Fully Connect	512 units	512 units	256 units
Fully Connect	256 units	256 units	256 units
Softmax	10 units	10 units	10 units

Table IX: **The network structure of CIFAR-10 models (execpt LLE):** the activation fucntion of convolutional layers and fully connect layers are introduced in Table VIII.

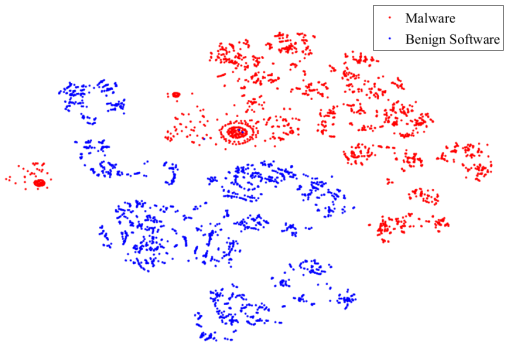
Layer type	LLE Dimensionality				
	$32 \times 32 \times 3$	$28 \times 28 \times 3$	$24 \times 24 \times 3$	$18 \times 18 \times 3$	$12 \times 12 \times 3$
Convolutional	64 filter(20×20)	32 filter(20×20)	32 filter(20×20)	32 filter(10×10)	32 filter(8×8)
Convolutional	64 filter(12×12)	32 filter(8×8)	32 filter(4×4)	32 filter(8×8)	32 filter(4×4)
Max pooling	2×2	2×2	2×2	2×2	2×2
Fully Connect	1024 units	784 units	576 units	324 units	144 units
Fully Connect	512 units	392 units	288 units	162 units	72 units
Fully Connect	256 units	196 units	144 units	81 units	36 units
Softmax	10 units	10 units	10 units	10 units	10 units

Table X: **The Hyper parameters of LLE-DNNs trained CIFAR-10:** LLE-DNNs use less convolutional layers amore fully connect layers than models in Table IX, because their inputs are composed of less margin features than real images.

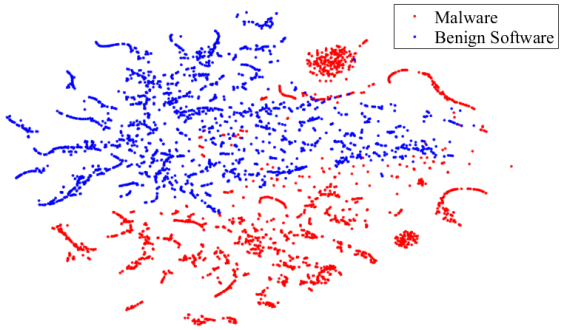


(a) Visualization of original CIFAR-10 test data.

(b) Visualization of CIFAR-10 test data projected by LLE.



(c) Visualization of original malware test data.



(d) Visualization of malware test data projected by LLE.

Figure 8: **Visualization results generated by t-SNE on CIFAR-10 and malware test data**: figures in top and bottom panel are the visualization of CIFAR-10 data and malware data respectively. Figures in left column are visualization of original test data, the other two figures are visualization of original test data processed by LLE. As discussed in the paper, the dimensionality of LLE mappings' for malware and CIFAR-10 are 1000 and $28 \times 28 \times 3$ respectively.