Differential Training: A Generic Framework to Reduce Label Noises for Android Malware Detection

Jiayun Xu
Singapore Management University
jyxu.2015@phdis.smu.edu.sg

Yingjiu Li
University of Oregon
yingjiul@uoregon.edu

Robert H. Deng
Singapore Management University
robertdeng@smu.edu.sg

***Abstract*—A common problem in machine learning-based malware detection is that training data may contain noisy labels and it is challenging to make the training data noise-free at a large scale. To address this problem, we propose a generic framework to reduce the noise level of training data for the training of any machine learning-based Android malware detection. Our framework makes use of all intermediate states of two identical deep learning classification models during their training with a given noisy training dataset and generate a noise-detection feature vector for each input sample. Our framework then applies a set of outlier detection algorithms on all noise-detection feature vectors to reduce the noise level of the given training data before feeding it to any machine learning based Android malware detection approach. In our experiments with three different Android malware detection approaches, our framework can detect significant portions of wrong labels in different training datasets at different noise ratios, and improve the performance of Android malware detection approaches.**

I. Introduction

Machine learning-based Android malware detection has been a major research focus in recent years. Both model training and evaluation rely on a set of sample apps and their associated labels (i.e., benignware and malware). The sample apps and their labels can be either collected from malware detection websites such as VirusTotal [7] or manually examined and labeled by malware detection experts [50], [51], [10].

However, the current approach to labelling sample apps is not perfect due to a couple of reasons. First, the labels provided by malware detection websites are not always reliable [23]. To verify this, we randomly chose 50,000 APPs on VirusTotal, and downloaded their scanning reports twice in 2016.7 and 2018.7, respectively. Among them, over 10% of the samples (5310/50000) are given different labels in the two reports. On the other hand, manually labelling is often costly and time-consuming, and it is difficult to scale up to massive datasets.

The label noises in app datasets may distort Android malware detection in two main aspects.

- According to F. A. Breve. et al. [11], the noises in sample labels worsen the performance of malware detection models trained with them, making them less effective in real-world cases.
- The noisy labels used for model testing and verification misjudge the real performances of existing malware detection solutions. In the case study sections of various research papers on malware detection (e.g., [48], [33], [50]), many false positive/negative cases are reported in fact due to mislabeled samples.

Network and Distributed Systems Security (NDSS) Symposium 2021
21-25 February 2021, Virtual
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24126
www.ndss-symposium.org

The noisy label problem is intrinsic in malware detection and challenging to deal with. The situation is worse due to ever-growing sizes of datasets that are used in machine-learning based malware detection. It remains challenging to work with noisy datasets, so as to improve both training of malware detection models and their evaluations. However, this problem has not been rigorously addressed, especially in the malware detection community.

Standard benchmark datasets have been built and widely used in certain other machine-learning fields such as image processing and natural language processing, where the quality of data labels can be verified by average human users through user studies or crowdsourcing. However, it is highly challenging for domain experts in malware detection (not to mention average human users) to ensure the correctness of all data labels in a massive dataset because the techniques for composing malware are highly complicated and constantly e-volving. This is one of the reasons that no universal benchmark dataset has been built for Android malware detection, making it difficult for the comparison across many malware detection approaches as they were evaluated on different datasets of unknown qualities.

Towards addressing the problem of malware detection with noisy datasets, we propose Differential Training, a novel noisy label detection framework for machine learning based Android malware detection. We make a meaningful assumption that the whole set of apps is noisy (i.e., no individual apps' labels are known 100% correct), but a majority of sample apps are correctly labeled. Differential Training can improve the performance of any machine learning based Android malware detection approaches by reducing label noises in their datasets.

In particular, Differential Training makes use of the intermediate states of deep learning classification models during training for noisy label detection. According to Schein, etc. in [36], the intermediate states of a classification model, represented by variances of sample losses, can be used as an effective measurement on the samples' uncertainty so as to help identify those that are not predicted properly within the current model. In other research (e.g., [37], [22]), such samples are paid extra attention in training so as to accelerate the learning of models.

A fundamental assumption in the previous research mentioned above is that all samples' labels are correct. Therefore, the mismatching between desired labels and predicted labels in training is attributed to immature model training. In the case of noisy labels being present, the wrong labels also contribute to the mismatching between desired labels and predicted labels during model training.

Differential Training relies on a new heuristic, which we call *differential training heuristic*, to reduce label noises in a given set of sample apps. The heuristic differentiates between correctly labeled samples and wrongly labeled samples according to their loss values in training two deep learning classification models of the same model architecture, where one model is trained with the entire dataset, and the other is trained with a randomly down-sampled subset of the given dataset. A sample's label is considered to be "wrong" and thus revised/flipped if its loss values appear to be outliers in comparison to other samples' loss values.

This heuristic is based on an observation that correctly labeled samples tend to behave consistently in training the two classification models, while the wrongly labeled samples tend to behave differently, and thus can be detected and revised. Differential Training applies this heuristic iteratively until a convergency condition is satisfied. After this, any machine learning based malware detection approach can be trained with the set of all sample apps and their revised labels. Rigorous experiments on various datasets and malware detection approaches show that differential training is clearly effective in noisy reduction and performance improvement for machine learning based malware detection.

The main contributions of this paper are summarized below:

- We develop a new generic framework, Differential Training, to reduce label noises for large-scale Android malware detection. Differential Training employs a novel approach to detecting noisy labels in multiple iterations according to the intermediate states of two deep learning classification models of identical architecture, one of which is trained on the whole training set of apps, and the other is trained on a randomly down-sampled set of apps. A new heuristic is proposed to distinguish between wrongly-labeled apps and correctly-labeled apps based on an outlier detection on their loss values, which are taken from the intermediate states of the two classification models.
- Differential Training enjoys high practicality because it is generic, automated, and independent to correctly-labeled datasets. Differential Training is generic as it can work with any machine learning based malware detection approach for reducing label noises of its dataset and improving its training and performance evaluation. Differential Training is fully automated in the label noise reduction process which requires neither domain knowledge nor manual inspection. In addition, Differential Training can operate on noisily-labeled datasets only. It does not rely on any extra datasets whose labels are all correct like other noise-tolerance classification approaches such as MentorNet [22] and distilled-based learning model [26].
- The effectiveness of Differential Training is evaluated with three different Android malware detection approaches, including SDAC [47], Drebin [10], and DeepRefiner [49], as well as three different datasets, whose sizes are 69k, 129k, and 110k, respectively. Applying to these datasets, Differential Training can reduce the size of wrongly-labeled samples to 12.6%, 17.4%, and 35.3% of its original size, respectively. Consequently, Differential Training improve the performance of each malware detection approach considerably after noisy reduction is conducted to their training datasets where the noise level is about 10%. In terms of F-score measured with ground-truth data, SDAC is improved from 89.04% to 97.19% (upper bound 97.71%), Drebin from 73.20% to 84.40% (upper bound 93.34%), and DeepRefiner from 91.37% to 93.41% (upper bound 93.59%). The improved performance is relatively close to their upper bound 97.71% for SDAC, 93.34% for Drebin, and 93.59% for DeepRefiner which are trained with all correctly-labeled datasets. A similar trend is also observed at various noise levels.
- Differential Training also outperforms the state-of-the-art noise-tolerant classification solutions, Co-Teaching and Decoupling, which are designed for training robust deep neural networks with noisy labels [20], [27]. Differential Training detects significantly more wrongly-labeled samples than both Co-Teaching and Decoupling.

II. Preliminaries

A. Machine Learning Based Android Malware Detection

We aim to reduce label noises for machine learning based Android malware detection that relies on a binary classification model to predict the label, which is either benign or malicious, for each given Android app. A machine learning based Android malware detection model is trained by a set of labeled Android apps (i.e., training set) in two main steps, where the first step transforms each Android app into a numerical feature vector, and the second step trains the model classifier using the apps' numeric feature vectors and their corresponding labels. After training, the model's performance can be evaluated using a set of labeled apps (i.e., testing set), based on the differences between their predicted labels and given labels.

B. Training Noise Detection Models

In the process of noisy label detection, Differential Training keeps training two identical deep learning classification models, which we call noise detection models. Each of these noise detection model is trained to classify any given sample app to be either malware or benignware. The training of each noise detection model consists of multiple epochs. In each epoch, each sample and its associated label are taken from a training dataset and fed into the model through two successive phases: forward propagation and backward propagation.

In the forward propagation phase, the feature vector of a given sample is taken as input to the noise detection model. A loss function is used to calculate a loss value for the sample according to the input vector and the parameters the noise detection model. Then, a predicted label is generated for the sample and compared to the given label of the sample.

In the back propagation phase, the gradient of the loss function with respect to each parameter in the noise detection model is calculated. Each parameter is then updated according to the gradient and the loss value in an optimal manner so as to minimize the average loss value in the next epoch Since the model parameters are adjusted in the whole training process, the average loss value for the samples in the whole training dataset is optimized to decrease from the first epoch to the last. The number of epochs is determined by a convergency condition under which the average loss value in the last epoch is considered to be good enough.

C. Underlying Assumption

The underlying assumption made by Differential Training is that the majority of sample apps in the dataset are correctly labeled; however, it is unknown whether the label of any specific sample is correct or not. Note that it is meaningful to assume that more than 50% of the sample apps are correctly labeled since if it is not the case (i.e., the quality of dataset is even lower than random labelling), a flipping of each and every label would make this assumption valid.

Differential Training does not rely on any set of individual apps whose labels are 100% correct, which is different from other noise-tolerance classification approaches such as MentorNet [22] and distilled-based learning model [26]. It requires no manually checking on any sample apps in Differential Training, which is fully automatic in reducing label noises for Android malware detection.

III. Differential Training Heuristic

Differential Training trains two deep learning classification models of identical architecture iteratively for noisy label detection. We call these two models *noise detection models*, which can be any deep learning classification models to classify apps to be either benign or malicious according to the apps' feature vectors. The whole process of Differential Training consists of multiple iterations. In each iteration two different models are trained where the first model is trained with the whole set of available training apps (and their labels), while the other is trained with a randomly down-sampled subset of the whole set. For convenience, we refer to the whole set of apps as *WS*, and the down-sampled subset as *DS*. We also refer to the first noise detection model as *WS model*, and the other as *DS model*.

In each iteration, Differential Training relies on a new heuristic, named *differential training heuristic* to reduce label noises in DS. The heuristic states that the training behaviors of correctly labeled samples across the two models are statistically different from those of the wrongly labeled samples across the two models, where the training behavior of a sample across the two models is described by the concatenation of the loss values produced for the sample in all epochs of the two models. Given the assumption that a majority of sample apps are correctly labeled, the wrongly labeled apps in DS can be detected statistically using an outlier detection on the training behaviors of all apps in DS.

Experimental observation. The heuristic is enlightened by our observation in the following experiments. When we observe a single model, either WS model or DS model, the differences in training behaviors between correctly-labeled samples and wrongly-labeled samples are not too significant; however, when we combine training behaviors across the two models, the differences between the two types of samples become more apparent.

In the experiments of showing our observation, Differential Training is applied to a set of sample apps that are randomly collected from a public Android app sharing project [8]. To guarantee the correctness of the samples' labels, we further check their scanning results from VirusTotal, and remove all (which is equivalent to around 10%) of the samples whose scanning results ever changed since August 2016. We use 50,000 samples to build the WS set, and choose 10% of them randomly as "noises" whose labels are manually flipped.

The architecture of the WS model and the DS model is chosen to be Multi-Layer Perceptron consisting of an input layer, two hidden layers, and a softmax layer, where the first hidden layer consists of 500 nodes, and the second layer consists of 1000 nodes. The training of the WS model and the DS model implements the learning rate decay and the early stopping API, and sets all hyper parameters to their default as provided in TensorFlow [5].

First, we choose 3,000 samples randomly from WS to form DS. Then we use WS to train the WS model, and use DS to train the DS model. For each sample in DS, we use α_w and ω_w to record its loss value in the first epoch and the last epoch, respectively, during the training of the WS model; and further use α_d and ω_d to denote its loss value in the first epoch and the last epoch, respectively, during the training of the DS model.

Figure 1 illustrates the distributions and their fitting curves of correctly-labeled samples and wrongly-labeled samples w.r.t. three variables, including $(\alpha_w/\omega_w)/(\alpha_d/\omega_d)$ in Figure 1 (a), (α_w/ω_w) in Figure 1 (b), and (α_d/ω_d) in Figure 1 (c). The bars in the figure indicate the number of samples at certain value of the corresponding variable, while the curves illustrate the kernel density estimation of the corresponding variable, which is a non-parametric estimation of the variable's probability density function.

Figure 1 shows that the differences between the correctly-labeled samples and the wrongly-labeled samples are more apparent in terms of their distributions measured from the loss values in training both the WS model and the DS model as shown in Figure 1 (a), than in training the WS model alone as shown in Figure 1 (b), and in training the DS model alone as shown in Figure 1 (c).

The distribution differences shown in Figure 1 between the correctly-labeled samples and the wrongly-labeled samples can be measured in Wasserstein distance [6], which quantifies the minimum "cost" of turning one distribution to another. A larger Wasserstein distance represents more significant difference between two distributions. Table I measures the Wasserstein distances between the distributions that are given in Figure 1. It

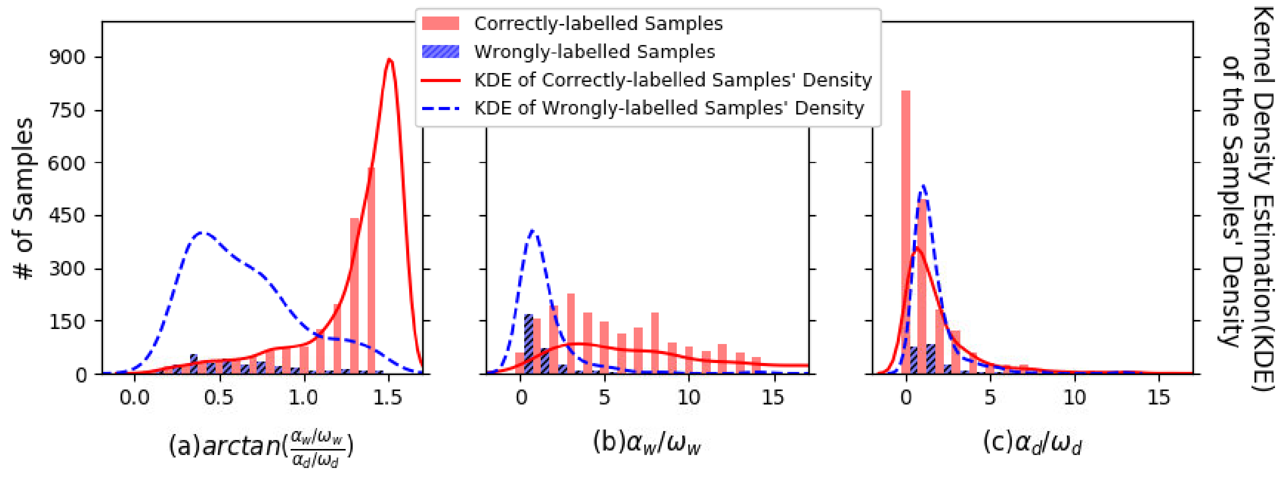


Fig. 1: Distributions of Correctly-Labeled Samples and Wrongly-Labeled Samples

TABLE I: Wasserstein Distance between Distributions of Correctly-Labeled and Wrongly-Labeled Samples

Model(s) Used	Wasserstein Distance
WS model	0.00295
DS model	0.01047
Both WS model and DS model	0.04387

TABLE II: Wasserstein Distance between Distributions of Correctly-Labeled and Wrongly-Labeled Samples with Different DS

Size of DS	Wasserstein Distance
3000	0.04387
6000	0.03401
9000	0.03334
12000	0.01965
15000	0.02703

suggests to distinguish between correctly-labeled samples and wrongly-labeled samples according to the loss values collected in training both the WS model and the DS model.

The size of DS is an important factor in differentiating the distributions between correctly-labeled samples and wrongly-labeled samples. Figure 2 and Table II show that using smaller DS yields greater difference; it is thus desirable to choose DS as small as possible in Differential Training.

On the other hand, DS should be large enough to converge the training of the DS model [15]. Therefore, we have two criteria for choosing the size of DS: (1) DS should be as small as possible to distinguish between correctly-labeled samples and wrongly-labeled samples, and (2) DS should be large enough for converging the training of the DS model. In our experiments, we use a grid search to choose the size of DS based on these two criteria.

Explanation on the differences of loss values. The differential training heuristic is also enlightened by the following theorem proved by S. Arora et al. in a recent research [9] on the differences of loss values between correctly-labeled samples and randomly-labeled samples during model training:

In a two-layer MLP model using ReLU activation and trained by gradient descent, when there are infinite nodes in hidden layers and the model is fully trained, the following equation holds:

$$\|\mathbf{y}-\mathbf{u}(k)\|_2 = \sqrt{\sum_{i=1}^{n}(1-\eta\lambda_i)^{2k}(\mathbf{v}_i^{\intercal}\mathbf{y})^2} \pm \varepsilon$$

where $\mathbf{y} = (y_1, y_2 \dots y_n)$ denotes all the labels of the n samples, $\mathbf{u}(k)$ denotes all the n predictions in the k_{th} epoch, and thus $\|\mathbf{y}-\mathbf{u}(k)\|_2$ refers to the $L2$-norm distance between the predicted labels and the true labels. Moreover, η refers to the learning rate. $\mathbf{v}_i$ refers to the orthonormal eigenvector of sample i and λ_i refers to its corresponding eigenvalue decomposed from the gram matrix H of the model, while the gram matrix H is decided by the two-layer ReLU model in the k_{th} epoch as defined in [46], [43], [16]. ε is a very small value that can be ignored.

The equation shows that under the ideal condition, the component of $(\|\mathbf{y}-\mathbf{u}(k)\|_2)^2$ for sample i in epochs 1 to k is a geometric sequence which starts at $(\mathbf{v}_i^{\intercal}\mathbf{y})^2$ and decreases at ratio $(1-\eta\lambda_i)^2$.

Furthermore, in section 4 of paper [9], it is proven that the samples with true labels have better alignment with larger eigenvalues than the samples with random labels (or wrong labels). In each epoch of model training, the square of $L2$-norm distance $(\|\mathbf{y}-\mathbf{u}(k)\|_2)^2$ thus demonstrates larger decreasing ratios $(1-\eta\lambda_i)^2$ for correctly-labeled samples than for wrongly-labeled samples.

Since the square of $L2$-norm distance $(\|\mathbf{y}-\mathbf{u}(k)\|_2)^2$ is exactly the same as the $L2$ loss function between the actual

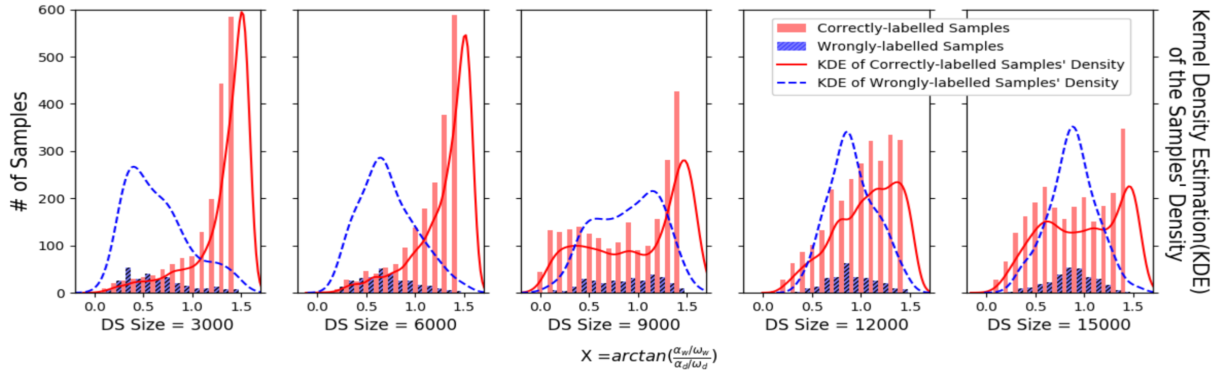
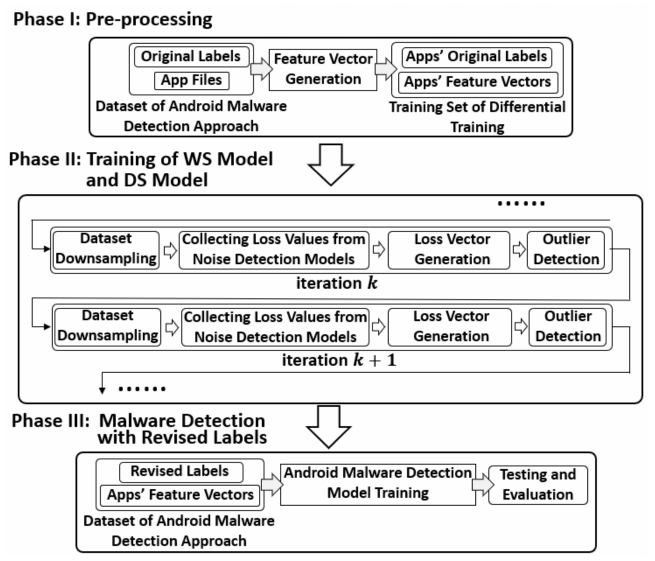


Fig. 2: Distributions of Correctly-Labeled Samples and Wrongly-Labeled Samples with Different Sizes of DS



Phase I: Pre-processing
Original Labels
App Files
Dataset of Android Malware Detection Approach
Feature Vector Generation
Apps' Original Labels
Apps' Feature Vectors
Training Set of Differential Training
Phase II: Training of WS Model and DS Model
Dataset Downsampling
Collecting Loss Values from Noise Detection Models
Loss Vector Generation
Outlier Detection
iteration k
iteration $k+1$
Phase III: Malware Detection with Revised Labels
Revised Labels
Apps' Feature Vectors
Dataset of Android Malware Detection Approach
Android Malware Detection Model Training
Testing and Evaluation

Fig. 3: Structure of Differential Training

labels $\mathbf{y}$ and the predicted labels $\mathbf{u}(k)$, this theorem implies that during the training of DS/WS model, the decreasing rates of the loss values of correctly-labeled samples are larger than (and thus different from) those of wrongly-labeled samples in each epoch.

IV. Differential Training Framework

Differential Training processes a noisy dataset in three phases: "pre-processing," "noisy label detection," and "malware detection with revised labels." The structure of Differential Training is shown in Figure 3.

A. Phase I: Pre-processing

In the first phase, a machine learning based malware detection approach is selected, and the raw app files from the dataset of the approach are transformed into numeric feature vectors through a "Feature Vector Generation" module which should be specified by the malware detection approach. The output of the phase I is the whole training set WS which consists of the transformed feature vectors and their associated labels.

B. Phase II: Noisy Label Detection

The second phase "Noisy Label Detection" of Differential Training consists of multiple iterations, in each of which the noises in the training set are reduced until a stopping criterion is met. Each iteration is illustrated in Figure 4, which consists of four steps, including "Dataset Downsampling", "Training of WS and DS Models", "Loss Vector Generation" and "Outlier Detection".

1) Dataset Downsampling: In the first step "dataset down-sampling", Differential Training randomly downsamples the whole training set WS to a smaller dataset, named "down-sampled set," or DS for short. The size of DS is selected according to the two criteria described in the differential training heuristic.

2) Training of WS and DS Models: After DS is generated, two noise detection models, "WS model" and "DS model," are trained on WS and DS datasets, respectively. The two noise detection models can be any deep learning classification models having the same network architecture for classifying apps to be either malicious or benign according to their feature vectors. Depending on the selection of app features, various deep learning classification models (e.g., Multi-Layer Perceptron, Recurrent Neural Network, and Convolutional Neural Network) may be selected to be noise detection models. Differential Training uses the noise detection models to extract the loss values for each input app during training; any deep learning classification model can be used as a noise detection model as long as it outputs a loss value for each input app in each epoch during its training process.

In our experiments conducted in this paper, we choose the noise detection models to be Multi-Layer Perceptron (MLP) that consists of two hidden layers, where the first hidden layer consists of 500 nodes and the second hidden layer 1000 nodes, and followed by a softmax layer as output. We use the TensorFlow toolkit [5] to train the two models, where all the parameters are set to their default values in the toolkit.

3) Loss Vector Generation: In the third step, the loss values of each app in DS are collected from the two noise detection

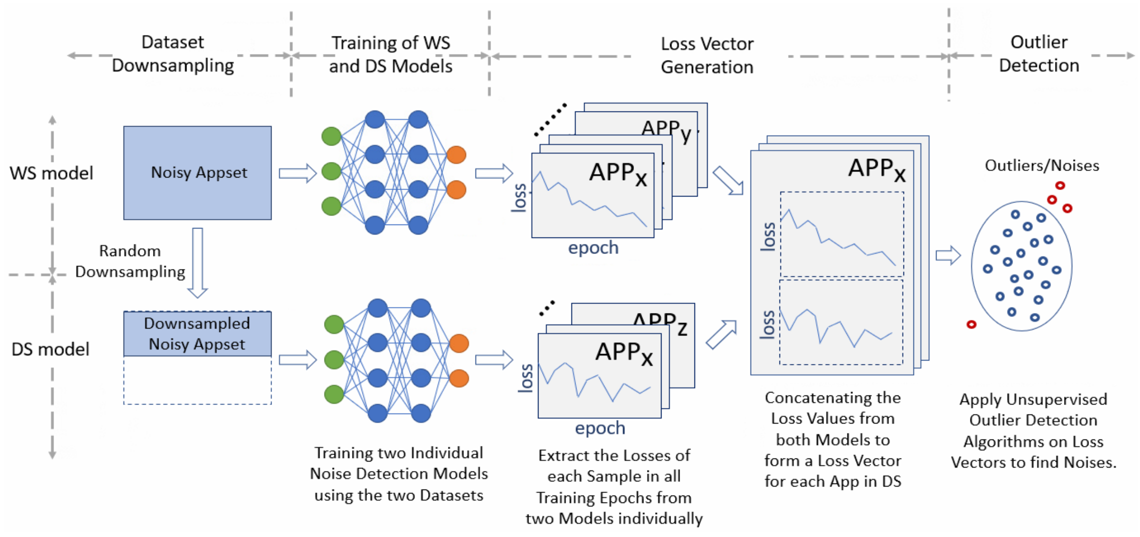

Dataset Downsampling
Training of WS and DS Models
Loss Vector Generation
Outlier Detection
WS model
Noisy Appset
Random Downsampling
DS model
Downsampled Noisy Appset
APPy
APPx
loss
epoch
APPz
Outliers/Noises
Training two Individual Noise Detection Models using the two Datasets
Extract the Losses of each Sample in all Training Epochs from two Models individually
Concatenating the Loss Values from both Models to form a Loss Vector for each App in DS
Apply Unsupervised Outlier Detection Algorithms on Loss Vectors to find Noises.

Fig. 4: Structure of a Single Iteration in Noisy Label Detection

models during their trainings. The loss values collected from each model are arranged into a sequence in the order of training epochs. Then the two sequences are concatenated to form a "loss vector" for each app in DS.

4) Outlier Detection: In this step, a set of outlier detection algorithms are applied to the loss vectors of all the apps in DS. For each app whose loss vector is detected as an outlier, its label is considered to be "wrong", and thus flipped with a probability. Several points on the outlier detection are clarified below:

- Most outlier detection algorithms require a "containment rate" parameter as their input. This parameter works as a threshold in identifying outliers. In Differential training, this parameter is set to the current ratio of wrongly-labeled samples in WS. This noise ratio is estimated using the method proposed by Goldberger [19]. In particular, the noise ratio is estimated to be $(1 - a_{WS})$, where a_{WS} is the accuracy of the WS model in 5-fold cross-validation on WS.
- To avoid any bias of a single outlier detection algorithm, we use 13 different outlier detection algorithms and apply a majority voting to get the final result of outliers. Table III shows the outlier detection algorithms used in Differential Training, where the first 12 algorithms are taken from a public toolkit named "PyOD" [3], while the last algorithm "EllipticEnvelope" is taken from the toolkit "sklearn" [4].
- Another parameter named dropout ratio is introduced in this step. After each outlier is detected according to the majority voting, the label of the corresponding sample is revised/flipped with a probability equal to the dropout ratio. The dropout ratio is used to reduce the impact caused by any accidental error from either outlier detection or noise rate estimation. The use of this dropout ratio is inspired by the random dropout mechanism in neural networks training [40], and the ratio is set to 0.5 in our experiments.

TABLE III: List of Outlier Detection Algorithms used in Differential Training

Angle-based Outlier Detector (ABOD)
Auto Encoder
Clustering Based Local Outlier Factor (CBLOF)
Histogram-based Outlier Detection (HBOS)
IsolationForest Outlier Detector (I-forest)
k-Nearest Neighbors Detector (kNN)
Local Outlier Factor (LOF)
Outlier Detection with Minimum Covariance Determinant (MCD)
Single-Objective Generative Adversarial Active Learning (So-gaal)
One-class SVM detector
Stochastic Outlier Selection (SOS)
Principal Component Analysis Outlier Detector (PCA)
EllipticEnvelope

5) Stopping Criterion: To stop the iterations in training the WS model and the DS model, we use a stop criterion that is similar to the early stopping adopted in neural network training. The iterations stop once the fluctuation of the estimated noise ratios in the last several iterations turns to be smaller than a certain threshold. In experiments, we enforce the stopping criterion through the API $earlystop\_callback()$ from the TensorFlow toolkit, where all parameters are set to their default values.

C. Phase III: Malware Detection with Revised Labels

Once the iterations stop, the apps and their associated labels in the whole set WS are ready for Android malware detection. The original Android malware detection approach can be trained using these apps' feature vectors (which were extracted in phase I) and their labels (which were revised in phase II). In the experiments below, we use ground-truth data to evaluate the performance of Differential Training with three different Android malware detection approaches, including SDAC [47], Drebin [10], and DeepRefiner [49].

We measure the performances of Differential Training using the following metrics: (i) The number and the percentage of wrong labels in the training set being reduced by Differential Training. (ii) The F-scores of the malware detection approach when it is applied to the noisy training set, the noise-reduced training set processed by Differential Training, and the "ground-truth" training set. The differences between these F-scores[1] show that how much improvement in the performance of the malware detection approach is made due to Differential Training, and how close is the improved performance to the upper bound.

In evaluating Differential Training with a malware detection approach, we partition its "ground-truth" dataset into two parts: 80% of them are used as the training set, and the other 20% are used as testing/validation set. Given a noise ratio $0 \leq r_{noise} \leq 0.5$, we randomly select each app in the training set with probability $r_{noise} * 100\%$, and flip the labels of the selected apps to generate a noisily-labeled dataset. The default value of the noise ratio is set to 10%, which is similar to the ratio which we observed from VirusTotal during a period of three years. After Differential Training revises the labels in the training set, we refer it as the processed dataset. While we train a malware detection approach using either ground-truth dataset, noisy dataset, or processed dataset, we always evaluate its performance using a "ground-truth" testing dataset.

Without confusion, we also call the framework Differential Training by excluding phase III if the objective is to detect or reduce noisy labels in a dataset without testing the malware detection approach.

V. Differential Training with SDAC

A. Introduction of SDAC

SDAC is an Android malware detection approach that uses "semantic distance based API clusters" to make malware detection robust to the evolution of Android APIs. It measures the contribution of an API to malware detection by its API context, which refers to its neighbourhood APIs in the API graphs derived from training apps. In the training process, all APIs are transformed into numeric vectors according to their API contexts, and the numeric vectors are clustered into 1,000 API clusters. Then, each app is converted to a 1,000-dimensional binary vector depending on whether or not the app includes any API in each API cluster. A supporting vector machine (SVM) is trained using those binary vectors that are derived from all training apps and their associated labels. In the testing phase, if new APIs are employed by any app, their contributions to malware detection can still be measured by associating their API vectors to the closest API clusters. Therefore, an app can always be converted to a 1,000-dimensional binary vector for malware detection in the testing phase even if it employs any new APIs that never appeared in the training phase.

B. SDAC Dataset

The dataset used by SDAC was collected from an open public Android application collection project [8], which consists of 70,811 apps. This dataset was collected in June 2018 after filtering out 10.62% of the apps whose labels changed by VirusTotal between July 2016 and July 2018. For these apps, we further downloaded their scanning results from VirusTotal in June 2019 and removed 878 (i.e., 1.24%) apps whose labels changed between July 2018 and June 2019 to reduce potential wrong labels as much as possible. Finally, the dataset of SDAC we use consists of 69,933 app samples whose labels are relatively stable over three years; we thus consider these stable labels as "ground-truth" as no other dataset of better quality is available so far. In this dataset, 35,437 are labeled benign, and 34,496 are labeled malicious.

[1]A F-score is the harmonic mean between precision and recall, where precision measures the percentage of true malware among the detected malware, while recall measures the percentage of true malware being detected.

TABLE IV: The Evaluation of Differential Training with SDAC

Android Malware Detection Approach		SDAC	
# Samples in the Whole Dataset		69,933	
# Benignware	# Malware	35,437	34,496
# Samples in Noisy Training Set		56,650	
# of Noises added		5,614 (9.91%)	
# detected TP	# detected FP	5,246	343
# of Remained Noises in Processed Dataset		711 (1.26%)	
F-score with Correctly-laballed Dataset		97.71%	
F-score with Noisily-laballed Dataset		89.04%	
F-score with Processed Dataset		97.19%	

C. Performance of Differential Training with SDAC

Table IV summarizes SDAC dataset and SDAC performances, which are measured on the correctly-labeled dataset, the nosily-labeled dataset, and the noise-reduced training set processed by Differential Training. Overall, Differential Training revise 5,246 labels correctly, revise 343 labels wrongly. The percentage of noise labels thus reduces from 9.91% to 1.26% in the nosily-labeled dataset due to the process of Differential Training.

More details are given in Figure 5 which shows the number of labels that are revised correctly by Differential Training (i.e., true positives), and the number of labels that are revised wrongly (i.e., false positives) in each iteration. It also shows the accuracy of the revised labels (i.e., the percentage of the revised labels that are correct) in each iteration. When Differential Training converges, the accuracy of the revised labels reaches close to 99%.

The F-score of SDAC improves from 89.04% to 97.19% after noise reduction on the training set, and this improved F-score is very close to its upper bound 97.71% (see Table IV). These results show that Differential Training can greatly reduce the number of wrong labels in the training set, and improve the performance of Android Malware detection approach due to the use of noise-reduced training set in training.

D. Runtime Performance of Differential Training with SDAC

The total time cost of Differential Training with SDAC in this experiment is about 55.34 hours. In detail, Differential Training performs 58 iterations; each iteration takes about 57.3 mins on average, which includes 50.6 mins spent on training the WS model, and less than 4 mins spent on training the DS model. The rest of time in each iteration is used for performing outlier detection and noise ratio estimation.

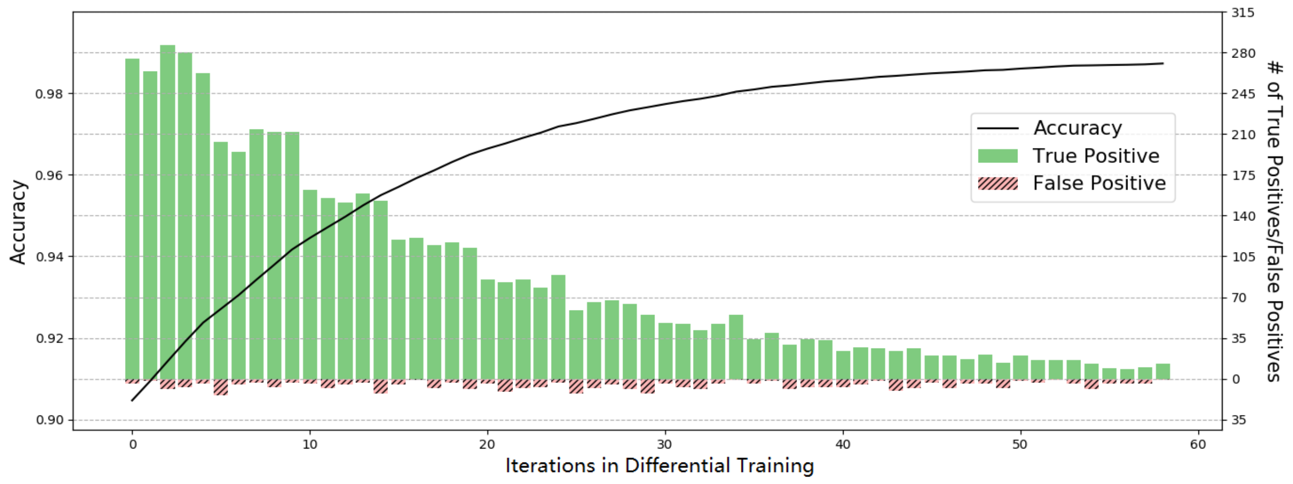


Fig. 5: Noise Reduction on SDAC Dataset

VI. Differential Training with Drebin

A. Introduction of Drebin

Drebin [10] is a lightweight Android malware detection solution published in 2014 based on static analysis. Drebin extracts the following features from each app: hardware components, required missions, App components, filtered intents from Android manifest files, critical API calls, actually used permissions, human-defined suspicious API calls, and network address strings from disassembled codes. Drebin converts each app into a feature vector of 545,433 dimensions. It relies on a linear support vector machine (SVM) classifier for malware detection, and uses its linear weights for identifying the features that make significant contributions to malware detection.

Compared to SDAC that was published recently in 2020, Drebin is more classic which has been cited frequently in malware detection research since 2014. In addition to evaluate the effectiveness of Differential Training on SDAC, we also test it on the classic Drebin with relatively old dataset.

B. Drebin Dataset

Drebin was evaluated on a dataset collected by 2014, which was composed of 5,560 "malware samples" and 123,453 "benign apps." The Drebin dataset has been frequently used in malware research since its publication. We checked the dataset in June 2019 using VirusTotal to guarantee the ground-truth of the dataset. In detail, we found no contradictory labels in the dataset except that some apps were too old to receive any report. Compared to the SDAC dataset where malware takes up 49.3% of all apps, the Drebin dataset is highly imbalanced as malware samples account for 4.3% of all apps. This is another reason that we choose Drebin so that we can test Differential Training on a highly imbalanced dataset.

C. Performance of Differential Training with Drebin

Figure 6 shows the performance of Differential Training in each iteration, where the red bars and green bars are measures of true positives and false positives, respectively. In total, 9,121 noisy labels are detected and revised correctly by Differential Training, and 605 labels are detected and revised mistakenly. The accuracy of Differential Training for label revisions converges close to 98%.

TABLE V: The Evaluation of Differential Training with Drebin

Android Malware Detection Approach		Drebin	
# Samples in the Whole Dataset		129013	
# Benignware	# Malware	123,453	5,560
# Samples in Noisy Training Set		103,210	
# of Noises added		10,009 (9.70%)	
# detected TP	# detected FP	9,121	605
# of Remained Noises in Processed Dataset		1805 (1.75%)	
F-score with Correctly-laballed Dataset		93.34%	
F-score with Noisily-laballed Dataset		73.20%	
F-score with Processed Dataset		84.40%	

Table V shows that Drebin's performance improves from 73.20% to 84.40% in F-score if it is trained on the processed dataset, for which Differential Training reduces the percentage of noise labels from 9.70% to 1.75%. Compared to the original F-score, the improved F-score is closer to its upper bound 93.34% which is achieved by Drebin trained with the correctly-labeled training set.

D. Runtime Performance of Differential Training with Drebin

The total time cost of Differential Training with Drebin in this experiment is about 62.52 hours, which converges in 68 iterations. Each iteration takes 55.2 minutes on average while the training of the WS model takes 52.0 mins and the training of the DS model takes 2.8 mins.

VII. Differential Training with DeepRefiner

A. Introduction of DeepRefiner

DeepRefiner [49] is a Android malware detection approach that connects two deep learning models in sequential. The first model is a Multi-Layer Perceptron model which can detect "most significant" malware samples efficiently based on XML files in app APK packages. The second model is a long short-term memory model which detects "more advanced" malware

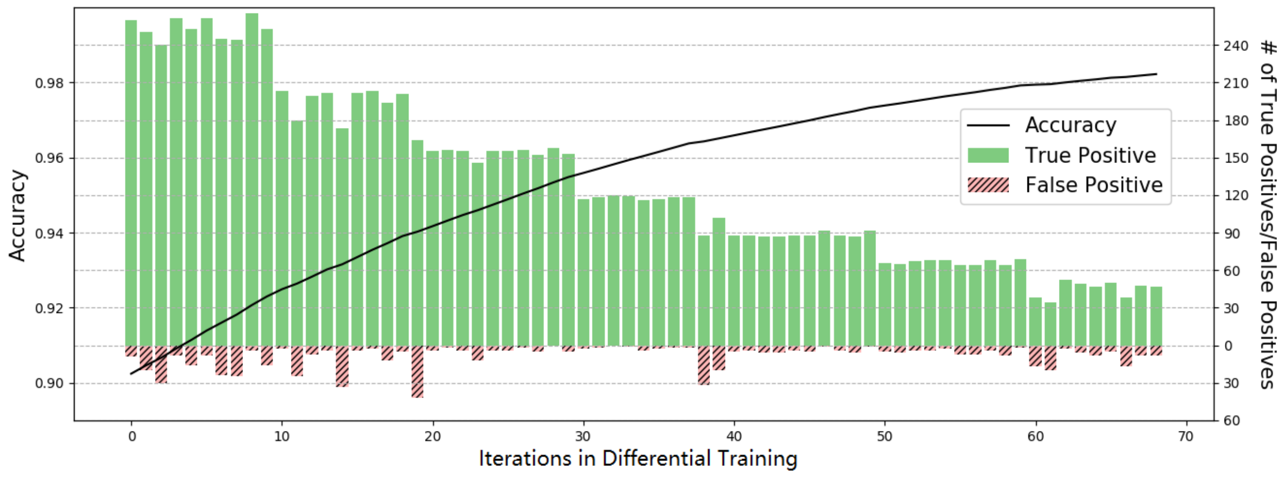


Fig. 6: Noise Reduction on Drebin Dataset

TABLE VI: The Evaluation of Differential Training with DeepRefiner

Android Malware Detection Approach		DeepRefiner	
# Samples in the Whole Dataset		110,440	
# Benignware	# Malware	47,525	62,915
# Samples in Noisy Training Set		88352	
# of Noises added		8,835 (10.00%)	
# detected TP	# detected FP	7,497	2,230
# of Remained Noises in Processed Dataset		3,118 (3.53%)	
F-score with Correctly-laballed Dataset		93.59%	
F-score with Noisily-laballed Dataset		91.37%	
F-score with Processed Dataset		93.41%	

samples from those apps for which the first model cannot provide reliable classification results. The second model relies on checking the semantic structures of Android bytecodes in malware detection.

According to [49], the first model of DeepRefiner can be used alone and it achieves 87.3% accuracy in malware detection on a dataset of 110,440 apps. We choose this model to evaluate how Differential Training performs if the malware detection approach is not extremely accurate but very efficient.

B. DeepRefiner Dataset

The original DeepRefiner dataset consists of a set of benign applications that were collected from Google Play as well as a set of malicious apps that were collected from VirusShare and MassVet in 2015 to 2016. We then downloaded their scanning reports from VirusTotal in June 2019 and removed all the samples with different labels. The remaining dataset contains of 62,915 malicious applications and 47,525 benign applications that were collected in 2016.

C. Performance of Differential Training with DeepRefiner

Figure 7 shows that after being processed by Differential Training, the percentage of samples with correct labels in the DeepRefeiner dataset increases from 90% to 96%, while Table VI further shows that 7,497 wrong labels are revised correctly, while 2,230 correct labels are revised mistakenly.

In total, Differential Training reduces 64.7% of the wrong labels and increases the F-score of DeepRefiner from 91.37% to 93.41%, which is only 0.18% lower than the F-score of DeepRefiner trained with the correctly-labeled dataset.

D. Runtime Performance of Differential Training with DeepRefiner

In the Differential Training with DeepRefiner, the total time cost is about 102.12 hours, which includes 77 iterations. The time cost for each iteration is about 79.6 minutes on average, including 70.0 minutes for training the WS model and 7.9 minutes for training the DS model.

VIII. The Impact of Noise Ratio to Noise Reduction

Differential Training is effective in reducing label noises on three different datasets at noise ratio 10% as shown in the previous sections. In this section, we further investigate the impact of noise ratio to noise reduction. For this purpose, we produce datasets at 5%, 10%, 15%, 20%, 30%, and 45% noise ratios, and apply Differential Training on such datasets with different Android malware detection approaches.

Note that 45% noise ratio is close to the upper bound of noise ratio (i.e., 50%)[2], indicating a data quality that is close to random labelling.

Table VII shows the noise reduction results of Differential Training on SDAC dataset at various noise ratios. In these experiments, the percentage of wrong labels being reduced by Differential Training ranges from 79.73% to 89.04% as the noise ratio changes from 5% to 30%, indicating that the effectiveness of Differential Training is stable in these experiments. When the noise ratio in dataset is set to 45%, which is close to the upper bound (i.e., noise ratio for random labelling), Differential Training reduces 68.09% of wrong

[2]If the noise ratio is greater than 50%, a flipping of each and every label would turn the noise ratio below 50%.

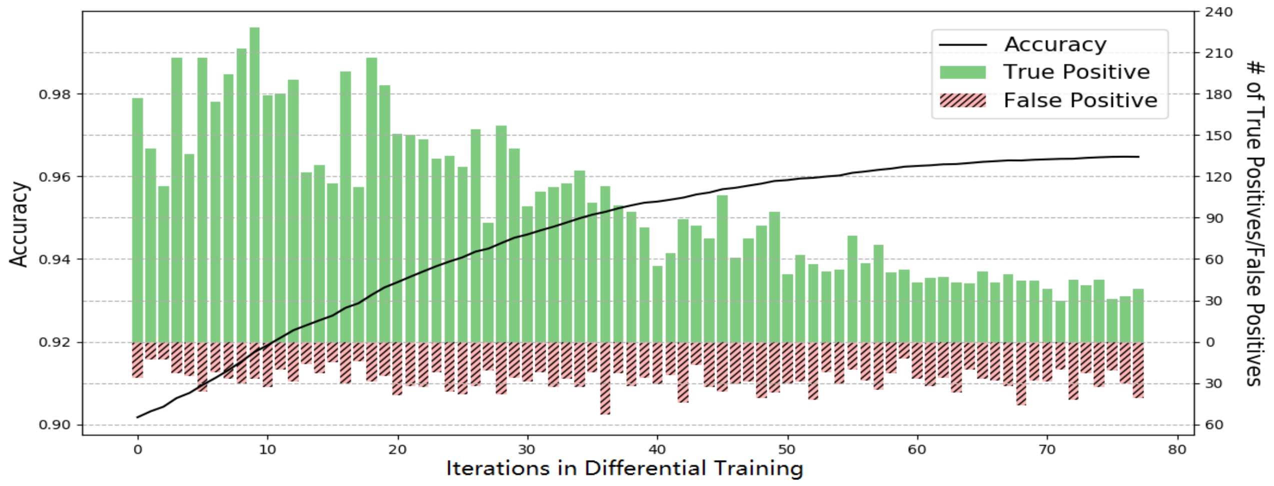


Fig. 7: Noise Reduction on DeepRefiner Dataset

TABLE VII: Noise Reduction on SDAC Dataset at Different Noise Ratios

Noise Ratio	5%	10%	15%	20%	30%	45%
# of Training Set	56,550	56,550	56,550	56,550	56,550	56,550
# of Wrongly-labeled Samples	2,833	5,614	8,497	11,330	16,995	25,493
# of TP Noise Detection Results	2,540	5,246	7,977	10,465	15,762	21,858
# of FP Noise Detection Results	281	343	472	377	655	4,500
% of Noise Reduced	79.73%	87.34%	88.33%	89.04%	88.89%	68.09%
# of Wrongly-labeled Samples Left	574	711	992	1,242	1,888	8,135%

TABLE VIII: Noise Reduction on Drebin Dataset at Different Noise Ratios

Noise Ratio	5%	10%	15%	20%	30%	45%
# of Training Set	103,210	103,210	103,210	103,210	103,210	103,210
# of Wrongly-labeled Samples	5,160	10,321	15,482	20,400	30,936	46,445
# of TP Noise Detection Results	4,658	9,121	12,700	17,870	26,120	44,804
# of FP Noise Detection Results	788	605	878	2,232	1,875	20,247
% of Noise Reduced	75.00%	82.51%	76.36%	76.66%	78.37%	52.87%
# of Wrongly-labeled Samples Left	1,290	1,805	3,660	4,762	6,691	21,888

TABLE IX: Noise Reduction on DeepRefiner Dataset at Different Noise Ratios

Noise Ratio	5%	10%	15%	20%	30%	45%
# of Training Set	88,352	88,352	88,352	88,352	88,352	88,352
# of Wrongly-labeled Samples	4,415	8,835	13,253	17,670	26,502	39,758
# of TP Noise Detection Results	3,985	7,947	12,406	14,737	21,707	23,604
# of FP Noise Detection Results	1,247	2,230	3,677	5,150	8,799	14,795
% of Noise Reduced	62.02%	64.71%	65.86%	54.26%	48.71%	22.15%
# of Wrongly-labeled Samples Left	1,677	3,118	4,524	8,083	13,594	30,949

labels. Though this percentage is lower than the other cases in the experiments, it is still substantial in this extreme case.

Table VIII shows a similar trend for Differential Training's effectiveness on noise reduction working on Drebin dataset at various noise ratios. The percentage of noisy labels being reduced fluctuates from 75.00% to 82.51% if the noise ratio varies between 5% and 30%, and it decreases to 52.87% at noise ratio 45%.

Table IX shows a wider fluctuation margin of noise label detection rate on DeepRefiner dataset, which varies from 48.71% to 65.86% for the noise ratio range of 5% to 30%. This wider fluctuation margin is probably due to the relatively simple feature set selected by DeepRefiner as compared to more comprehensive feature sets used by SDAC and Drebin. Nonetheless, Differential Training can still detect nearly half of wrong labels even if the noise ratio is as high as 30% in the training set. While in the extremely noisy case for 45% of noise ratio, Differential Training reduces 22% of wrong labels. While this result is lower than the other cases as the noise ratio is close to random labelling, the effectiveness of Differential Training is still non-negligible in reducing the noise in the training dataset.

Tables VII, VIII, and IX also show a trend between the detection performance of Differential Training and the noise ratio in the dataset. When the noise ratio ranges from 5% to 30%, the detection accuracy does not change much; while the ratio increases to 45%, a significant decrease in detection accuracy is observed.

Differential Training detects label noises as outliers. When there are almost the same number of outliers as non-outliers, it is difficult for Differential Training to distinguish between outliers and non-outliers, leading to a significant drop in its detection accuracy.

IX. Comparison among Differential Training, Co-teaching, and Decoupling on Noise Reduction

In this section, we compare Differential Training with two state-of-the-art robust learning algorithms, including Co-Teaching [20] and Decoupling [27], both of which are designed for training robust deep neural networks with noisy labels.

Co-Teaching trains two neural networks simultaneously on a noisy dataset, where the two models are of identical architecture but initialized independently at the beginning. Given each mini-batch of the dataset, each network views its small-loss data samples as potentially-clean samples, and provides them to its peer network for updating the parameters in the peer network. In Co-Teaching, the two networks have different learning abilities; they can thus filter each other's different types of error that are introduced by noisy labels in the learning process. After training, the two fully-trained models cooperate to output a predicted label for each sample in the testing phase, where the predicted label is determined by an output weight that is the sum of the output weights of the input sample from the two models. Co-Teaching can be used to detect noisy labels. If the predicated label of an input sample is different from the original label of the input sample, the original label is considered as a noisy label, and thus flipped. The source code of this algorithm is publicly available and provided in [1].

Decoupling also trains two neural networks simultaneously on a noisy dataset, with these two networks being of the same structure but initialized independently. During the model training, each mini-batch of the dataset is fed to both models simultaneously to generate the prediction results. If a sample in the mini-batch is predicted with different labels from the two models, it is regarded as meaningful for the model learning and only the "meaningful" samples in the mini-batch are later used in the backward propagation step to update the parameters in both models. At the end of training, Decoupling randomly chooses one of the two trained models as the produced classifier. Decoupling can be used to detect noisy labels. If an input sample's label that is predicted by the produced classifier is different from its original label, the original label is considered as noisy, and thus flipped for correction. Decoupling's source code is publicly available and provided in [2].

We apply both Differential Training, Co-Teaching, and Decoupling to the noisy versions of all three datasets, including SDAC dataset, Drebin dataset, and DeepRefiner dataset, where the noise ratio is set to 10% as in the default setting. We compare their performances in terms of the percentage of wrongly labels being detected/reduced in the noisy datasets. For fair comparison, the neural networks used in Co-Teaching or Decoupling are chosen to be the same as the ones used in Differential Training, being two-layer MLP networks with 500 nodes in the first layer and 1000 nodes in the second layer.

TABLE X: Comparison between Differential Training (DT), Co-Teaching (CT) and Decoupling (DC)

	SDAC Dataset	Drebin Dataset	DeepRefiner Dataset
% of Wrongly Labels Reduced by DT	87.45%	82.51%	64.71%
% of Wrongly Labels Reduced by CT	76.49%	78.14%	21.72%
% of Wrongly Labels Reduced by DC	68.43%	65.37%	29.80%
# of TP/FP Noises Detected by DT	5,246/343	9,121/605	7,497/2,230
# of TP/FP Noises Detected by CT	5,131/841	8,997/933	3,311/1,392
# of TP/FP Noises Detected by DC	4,305/463	8,107/1,360	4,392/1,606
Detection results (F-score) on Noisy dataset	89.04%	73.20%	91.37%
Detection results (F-score) on dataset processed by DT	97.19%	84.40%	93.41%
Detection results (F-score) on dataset processed by CT	96.01%	77.36%	93.33%
Detection results (F-score) on dataset processed by DC	92.38%	79.80%	92.19%
Runtime Performance of DT (hour)	55.34	62.52	102.12
Runtime Performance of CT (hour)	4.08	20.09	23.93
Runtime Performance of DC (hour)	2.24	21.71	14.15

Table X compares Differential Training with Co-Teaching and Decoupling in terms of noise detection result and runtime performance. The runtime performance is evaluated on a single desktop personal computer without GPU. The PC is equipped with one Intel(R) i5-4590 3.3 GHz CPU and 12 GB physical memory running on the Ubuntu 14.04 (LTS) operating system.

The table shows that while Differential Training takes longer time than Co-teaching and Decoupling, it outperforms these two approaches considerably in all three cases in terms of noise detection accuracy. Specifically, Differential Training produces the most True-Positive (TP) results and the least False-Positive (FP) results in all the cases except for the FP result in the case of DeepRefiner. The malware detection results (F-score) with the datasets processed by Differential Training are also better than those processed by the other two noise detection approaches.

Different strategies are exploited for identifying or processing noise samples. Differential Training identifies a sample as "noise" based on all of its loss values in the whole training process, while Co-Teaching treats a sample to be potentially-clean based on its individual loss value in each mini-batch, and Decoupling identifies a noise sample based on its prediction result that is related to its loss value in the last epoch only. Our comparison shows that the strategy exploited by Differential Training is more reliable in detecting noisy labels than other strategies exploited by Co-Teaching and Decoupling.

X. Discussion

A. Limitation

Time cost of Differential Training. As shown in section IX, Differential Training has a higher time cost than Co-teaching and Decoupling. This is mainly because Differential Training reduces the label noises in a gradual way in multiple iterations, while in each iteration two separate classification models (i.e., WS model and DS model) are trained. In comparison, both Co-teaching and Decoupling rely on two classification models being trained in a single iteration.

Accuracy of noise ratio estimation. The accuracy of the noise ratio estimation used in outlier detection may affect the performance of Differential Training. If the estimated noise ratio is significantly different from the actual ratio, Differential Training may produce more false positives and false negatives in identifying noise labels. While the noise ratio estimation algorithm we adopted enables Differential Training to outperform Co-Teaching and Decoupling in rigorous experiments, it is still possible to further improve the performance of Differential Training by adopting a more accurate noise ratio estimation algorithm in its outlier detection. We leave this as a future work.

B. Generalization on Differential Training

Differential Training is designed to identify and correct wrongly-labeled data samples for Android malware detection. In this paper, we consider Android malware detection as a binary classification problem, where each app is labeled either benign or malicious. While we expect that the idea of Differential Training can be generalized to other fields for *identifying* data samples whose labels are misclassified, it cannot be directly applied to *correcting* wrong labels for multiclass classification. This is because Differential Training simply flips the labels for identified noise samples to correct them. For multiclass classification, additional effort is to be made on how to correct wrong labels.

XI. Related Works

1) Android Malware Detection: Early research on Android malware detection incorporates traditional machine learning approaches such as k nearest neighbors (kNN) [35], [42], [32], support vector machine (SVM) [55], [47] or Decision trees [18], [52] with manually selected features such as system calls [12], [13], permissions [39], embedded strings [10], APIs [42], [31], [53], and communication intents [48]. Later research tends to focus on deep learning algorithms and automatic feature engineering for Android malware detection. Maldozer [24], R2-D2 [21] and DroidDetector [38] are several examples that rely on convolutional neural networks to perform malware detection based on APIs vectors, while some other malware detection approaches make use of recurrent neural networks to process sequential app inputs such as API call sequences [30], [45], [49]. There also exist hybrid approaches (e.g., [25]) that combine multiple types of malware detection models for malware detection. A common assumption taken in the existing malware detection research is that all training apps are correctly-labeled; however, it is difficult, if not possible, to obtain noise-free large-scale training data in practice due to the high complexity of malware detection and the fast evolvement of app development. It is thus important to study the impact of label noises to malware detection and improve the performance of malware detection given noisy training data. Differential Training is complementary to the previous research on malware detection since it provides a generic framework to reduce label noises in training data and can work with any Android malware detection approach for improved performance.

2) The Label Noise Problem in Machine Learning: The label noise problem has been recently addressed in the machine learning literature, where the focus is on how to train classification models that are tolerant to label noises. Various approaches have been developed to alleviate the negative effects of wrongly-labeled samples in model training so as to improve the quality of the finally-trained models.

One approach adjusts the loss calculation in the process of model training according to label noise estimation [29], [54]. Another approach relies on the models of special structure that can reduce the impact caused by label noise in model training [41], [34]. And other approaches aim at training noise-tolerant models for various purposes [17], [28], [44]. All of these approaches perform noise detection according to the final states of input samples in either training phase or testing phase.

According to Schein, et al., the intermediate states of an input sample are useful in measuring the uncertainty between the predicted label and the actual label of the sample in the process of model training [36]. Inspired by this, Chang et al. accelerated the process of model training [14]. However, the intermediate states of input samples during model training have not been utilized to process noisy samples except in Co-Teaching [20], where the individual loss value of each input sample in each mini-batch is examined during model training. Different from all these works, Differential Training detects label noises by examining all loss values for each input sample in the whole training process.

XII. Conclusion

In this paper, we proposed Differential Training as a generic framework to detect and reduce label noises from training data for any machine learning-based Android malware detection. Differential Training is novel due to (i) the use of intermediate states of input samples in the whole training process for noise detection, (ii) the use of downsampled set to maximize the differences between wrongly-labeled samples and correctly-labeled samples, and (iii) the use of outlier detection algorithms for not relying on even a small set of correctly-labeled training samples.

Our experimental results show that Differential Training reduces 87.4%, 82.6% and 64.7% wrong labels in the training sets of SDAC, Drebin and DeepRefiner, respectively in the default setting where the noise ratio is set to 10%. With noise reduced, the F-scores of these malware detection approaches improve from 89.04%, 73.20% and 91.37% to 97.19%, 84.40% and 93.41%, respectively. The improved F-scores are close to their upper bounds (97.71%, 93.34% and 93.59%). Our experiments also show that the performance of Differential Training is consistent for processing datasets at various noise ratios, and it is superior to the state-of-the-art robust learning algorithms Co-Teaching and Decoupling for training robust deep neural networks with noisy labels.

Acknowledgment

Yingjiu Li was supported in part by the Ripple University Blockchain Research Initiative.

References

[1] "Neurips'18: Co-teaching: Robust training of deep neural networks with extremely noisy labels," https://github.com/bhanML/Co-teaching.

[2] "NIPS 2017: Decoupling "when to update" from "how to update"," https://github.com/emalach/UpdateByDisagreement.

[3] "PyOD," https://pyod.readthedocs.io/en/latest/.

[4] "Scikit-learn," https://scikit-learn.org/stable/modules/generated/sklearn.covariance.EllipticEnvelope.html.

[5] "TensorFlow," https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping.

[6] "Wasserstein," https://en.wikipedia.org/wiki/Wasserstein_metric.

[7] "VirusTotal," https://www.virustotal.com/, 2004.

[8] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of Android apps for the research community," in *IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 2016.

[9] S. Arora, S. S. Du, W. Hu, Z. Li, and R. Wang, "Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks," in *International Conference on Machine Learning (ICML)*, 2019.

[10] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *The Network and Distributed System Security Symposium (NDSS)*, 2014.

[11] F. A. Breve, L. Zhao, and M. G. Quiles, "Semi-supervised learning from imperfect data through particle cooperation and competition," in *The 2010 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2010.

[12] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for Android," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM)*, 2011.

[13] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting (a)ndroid malware using sequences of system calls," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile (MobileDeLi)*, 2015.

[14] H.-S. Chang, E. Learned-Miller, and A. McCallum, "Active bias: Training more accurate neural networks by emphasizing high variance samples," in *Advances in Neural Information Processing Systems (AIP-S)*, 2017.

[15] F. Dernoncourt, J. Y. Lee, O. Uzuner, and P. Szolovits, "De-identification of patient notes with recurrent neural networks," *Journal of the American Medical Informatics Association (JAMIA)*, 2017.

[16] S. S. Du, X. Zhai, B. Poczos, and A. Singh, "Gradient descent provably optimizes over-parameterized neural networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[17] B. Frénay and M. Verleysen, "Classification in the presence of label noise: a survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2013.

[18] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of Android malware," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 2018.

[19] J. Goldberger and E. Ben-Reuven, "Training deep neural-networks using a noise adaptation layer," in *5th International Conference on Learning Representations, (ICLR)*, 2017.

[20] B. Han, Q. Yao, X. Yu, G. Niu, M. Xu, W. Hu, I. Tsang, and M. Sugiyama, "Co-teaching: Robust training of deep neural networks with extremely noisy labels," in *Advances in Neural Information Processing Systems (NIPS)*, 2018.

[21] T. Hsien-De Huang and H.-Y. Kao, "R2-D2: Color-inspired convolutional neural network (cnn)-based Android malware detections," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018.

[22] L. Jiang, Z. Zhou, T. Leung, L.-J. Li, and L. Fei-Fei, "MentorNet: Learning data-driven curriculum for very deep neural networks on corrupted labels," in *International Conference on Machine Learning (ICML)*, 2018.

[23] A. Kantchelian, M. C. Tschantz, S. Afroz, B. Miller, V. Shankar, R. Bachwani, A. D. Joseph, and J. D. Tygar, "Better malware ground truth: Techniques for weighting anti-virus vendor labels," in *8th ACM Workshop on Artificial Intelligence and Security (AISec)*. ACM, 2015.

[24] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digital Investigation*, 2018.

[25] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2018.

[26] Y. Li, J. Yang, Y. Song, L. Cao, J. Luo, and L.-J. Li, "Learning from noisy labels with distillation," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2017.

[27] E. Malach and S. Shalev-Shwartz, "Decoupling "when to update" from "how to update"," in *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[28] A. Menon, B. Van Rooyen, C. S. Ong, and B. Williamson, "Learning from corrupted binary labels via class-probability estimation," in *International Conference on Machine Learning (ICML)*, 2015.

[29] N. Natarajan, I. S. Dhillon, P. K. Ravikumar, and A. Tewari, "Learning with noisy labels," in *Advances in neural information processing systems (NIPS)*, 2013.

[30] R. Nix and J. Zhang, "Classification of Android apps and malware using deep neural networks," in *2017 International joint conference on neural networks (IJCNN)*. IEEE, 2017.

[31] L. Onwuzurike, M. Almeida, E. Mariconti, J. Blackburn, G. Stringhini, and E. De Cristofaro, "A family of droids-Android malware detection via behavioral modeling: Static vs dynamic analysis," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2018.

[32] L. Onwuzurike, E. Mariconti, P. Andriotis, E. D. Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models (extended version)," *ACM Transactions on Privacy and Security (TOPS)*, 2019.

[33] X. Pan, X. Wang, Y. Duan, X. Wang, and H. Yin, "DarkHazard: Learning-based, large-scale discovery of hidden sensitive operations in Android apps," in *The Network and Distributed System Security Symposium (NDSS)*, 2017.

[34] G. Patrini, A. Rozza, A. Krishna Menon, R. Nock, and L. Qu, "Making deep neural networks robust to label noise: A loss correction approach," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[35] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2016.

[36] A. I. Schein and L. H. Ungar, "Active learning for logistic regression: an evaluation," *Machine Learning*, 2007.

[37] B. Settles, "Active learning literature survey," Tech. Rep., 2009.

[38] J. Shen, Z. Chen, S. Wang, Y. Zhu, and M. U. Hassan, "DroidDetector: a traffic-based platform to detect Android malware using machine learning," in *Third International Workshop on Pattern Recognition (IWPR)*. International Society for Optics and Photonics, 2018.

[39] A. Skovoroda and D. Gamayunov, "Automated static analysis and classification of Android malware using permission and API calls models," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, 2017.

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research (JMLR)*, 2014.

[41] S. Sukhbaatar, J. B. Estrach, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," in *International Conference on Learning Representations (ICLR)*, 2015.

[42] M. Sun, X. Li, J. C. Lui, R. T. Ma, and Z. Liang, "Monet: a user-oriented behavior-based malware variants detection system for Android," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2016.

[43] R. Tsuchida, F. Roosta, and M. Gallagher, "Invariance of weight distributions in rectified MLPs," in *International Conference on Machine Learning (PMLR)*, 2018.

[44] Y. Wang, W. Liu, X. Ma, J. Bailey, H. Zha, L. Song, and S.-T. Xia, "Iterative learning with open-set noisy labels," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[45] X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimedia Tools and Applications*, 2019.

[46] B. Xie, Y. Liang, and L. Song, "Diverse neural network learns true target functions," in *Artificial Intelligence and Statistics (PMLR)*, 2017.

[47] J. Xu, Y. Li, R. Deng, and K. Xu, "SDAC: A Slow-aging solution for Android malware detection using semantic distance based API clustering," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2020.

[48] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on Android," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2016.

[49] K. Xu, Y. Li, R. H. Deng, and K. Chen, "Deeprefiner: Multi-layer Android malware detection system applying deep neural networks," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018.

[50] L.-K. Yan and H. Yin, "DroidScope: Seamlessly reconstructing the OS and Dalvik semantic views for dynamic Android malware analysis," in *USENIX security symposium*, 2012.

[51] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications," in *European Symposium on Research in Computer Security (ESORICS)*. Springer, 2014.

[52] S. Y. Yerima, S. Sezer, and I. Muttik, "Android malware detection using parallel machine learning classifiers," in *Next Generation Mobile Applications, Services and Technologies (NGMAST)*. IEEE, 2014.

[53] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware Android malware classification using weighted contextual API dependency graphs," in *ACM SIGSAC conference on computer and communications security (CCS)*, 2014.

[54] Z. Zhang and M. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," in *Advances in neural information processing systems (NIPS)*, 2018, pp. 8778–8788.

[55] J. Zhu, Z. Wu, Z. Guan, and Z. Chen, "API sequences based malware detection for Android," in *2015 IEEE 12th Intl Conf on Ubiquitous Intelligence and Computing and 2015 IEEE 12th Intl Conf on Autonomic and Trusted Computing and 2015 IEEE 15th Intl Conf on Scalable Computing and Communications and Its Associated Workshops (UIC-ATC-ScalCom)*. IEEE, 2015.