IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received September 24, 2019, accepted October 17, 2019, date of publication October 21, 2019, date of current version October 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2948658

Assessing Optimizer Impact on DNN Model Sensitivity to Adversarial Examples

YIXIANG WANG[1], JIQIANG LIU[1], (Member, IEEE), JELENA MIŠIĆ[2], (Fellow, IEEE), VOJISLAV B. MIŠIĆ[2], (Senior Member, IEEE), SHAOHUA LV[1], AND XIAOLIN CHANG[1], (Member, IEEE)

[1]Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China
[2]Department of Computer Science, Ryerson University, Toronto, ON M5B 2K3, Canada

Corresponding author: Jiqiang Liu (jqliu@bjtu.edu.cn)

This work was supported in part by the Natural Science Foundation of China under Grant 61672092, and in part by the Fundamental Research Funds for the Central Universities of China under Grant 2018JBZ103.

ABSTRACT Deep Neural Networks (DNNs) have been gaining state-of-the-art achievement compared with many traditional Machine Learning (ML) models in diverse fields. However, adversarial examples challenge the further deployment and application of DNNs. Analysis has been carried out for studying the reasons of DNNs' vulnerability to adversarial perturbation and focused on model architecture. No research has been done on investigating the impact of optimization algorithms (namely, optimizers in DNNs) employed in training DNN models on models' sensitivity to adversarial examples. This paper aims to study this impact from an experimental perspective. We analyze the sensitivity of a model not only from the aspect of white-box and black-box attack setups, but also from the aspect of different types of datasets. Four common optimizers, SGD, RMSprop, Adadelta, and Adam, are investigated on structured and unstructured datasets. Extensive experiment results indicate that an optimization algorithm does pose effects on the DNN model sensitivity to adversarial examples. That is, when training models and generating adversarial examples, Adam optimizer can generate better quality adversarial examples for structured datasets, and Adadelta optimizer can generate better quality adversarial examples for unstructured datasets. In addition, the choice of optimizers does not affect the transferability of adversarial examples.

INDEX TERMS Adversarial examples, deep neural network, machine learning, stochastic gradient descent optimization algorithm, transferability.

I. INTRODUCTION

Deep neural networks (DNNs) determine model parameters through a large number of training examples to learn the mapping relationship between inputs and outputs, which later can be applied to predict the output of new inputs. The past years witnessed the amazing development of deep learning (DL) neural networks, in particular convolutional neural networks (CNNs) and recurrent neural networks (RNNs), in various applications of diverse fields, such as object detection and classification in computer vision [1]–[3], natural language processing [4], and time-series data analysis [5]–[7]. The emergence of massive data and the requirements of effective analysis methods for big data in recent years further advance the development of neural networks.

The associate editor coordinating the review of this manuscript and approving it for publication was Benyun Shi.

Recently, Szegedy *et al*. [8] revealed an intriguing property (also called as vulnerability) of neural networks: slightly perturbed inputs which are imperceptive to human beings can easily result in the trained model producing inaccurate outputs. These inputs are called adversarial examples [14], which can make DNN models predict wrong results with high confidence. The aforementioned vulnerability of neural networks caused by adversarial examples reflects the sensitivity of a model's outputs to its inputs. It further indicates that neural network models after being trained are not as generalized as they are expected to be. Therefore, it raises a challenge on various DNN applications. For instance, in automatic-driving, adversaries can cheat those applications through purposefully synthesized disguises and traffic accidences can occur during the error in discerning traffic signals. The application of neural networks in security-critical fields, such as intrusion detection and malware detection [38], is also vulnerable to

adversarial attacks caused by adversarial examples. These phenomena reflect that adversarial examples do pose a potential security threat to DNN models.

One property of adversarial examples is transferability [27]. That is, adversarial examples generated through one model can cause another model misclassification even though the two models have different structures and are trained on different datasets. The transferability of adversarial examples makes black-box attacks possible [23], which means that adversaries don't need to get knowledge of model details regarding architecture and/or training data.

Previous work on adversarial examples included assessing the ability of neural networks in defending against adversarial attacks [9], [10], verifying the influence of adversarial examples in real-world applications [11], [12] or providing defense strategies [13]–[16] in order to strengthen the robustness of DNNs models, and the existing analysis of DNNs' vulnerability towards adversarial examples mainly focused on model architecture [23] such as layer numbers. But there is no study on the impact of adversarial examples from the perspective of DNNs' training phase, especially the optimizer aspect.

This paper aims to investigate the impact of optimization algorithms on models' sensitivity to adversarial examples. The sensitivity of a model is examined under both white-box and black-box attack setups. Four general optimization algorithms, Stochastic Gradient Descent (SGD) [37], RMSprop [36], Adadelta [21] and Adaptive Moment Estimation (Adam) [20], are chosen for evaluation. Both structured and unstructured datasets are adopted to investigate the impact of data types on optimizer effectiveness. By structure dataset, we mean that data (structured data) in it is like the table in the database [41]. By unstructured dataset, we mean that data in it is like audio, video, images etc. [41]. For the unstructured dataset, we use MNIST [35]. For the structured dataset, we use NSL-KDD [18] and DREBIN [19]. To the best of our knowledge, we are the first to investigate the effect of optimizer on model sensitivity to adversarial attacks from an experimental perspective. Extensive experiment results indicate that the optimization algorithm does pose an effect on the DNN model sensitivity to adversarial examples. In addition, experimental results show that the types of input examples affect the performance of an optimizer. The specific contributions are summarized as follows:

- We craft adversarial examples on MNIST, NSL-KDD and DREBIN datasets using four types of deep neural network models, denoted as SGD-model, RMS-model, Adadelta-model, and Adam-model respectively. They have the same structures but different optimizers. Moreover, we investigate the impact of optimizers on the quality of the crafted adversarial examples in terms of metrics described in Section III.D. We find that on structured datasets, the DNN model with Adam optimizer can generate adversarial examples with high convergence rate, low perturbation rate, and high generation rate. But on the unstructured dataset, the optimizer with the same effect is Adadelta optimizer.
- We assess optimizer effect on sensitivity by implementing white-box attacks against SGD-model, RMS-model, Adadelta-model, and Adam-model by using adversarial examples crafted by themselves individually. We find that adversarial examples generated by the DNN model with Adam optimizer can maximize the misclassification on the structured datasets, and on the unstructured dataset, the optimizer with the same effect is Adadelta optimizer.
- We assess optimizer effects on sensitivity by implementing black-box attacks against traditional machine learning classifiers as well as CNN and RNN to evaluate cross-model attack performance of four perturbed examples. These examples are crafted by SGD-model, RMS-model, Adadelta-model, and Adam-model respectively. We find that optimizers have a limited effect on the transferability of adversarial examples.

The rest of the paper is organized as follows. Section II presents background and related work. Section III describes the methodology of establishing experiments. Experiment result discussions are given in Section IV. Section V concludes this paper and presents future work.

II. BACKGROUND AND RELATED WORK

A. BACKGROUND

This section first describes some primary concepts mentioned in this paper and then describes the models used in experiments. We mainly introduce the four common optimizers used to train DNNs. At last, a brief description of Jacobian-based Saliency Map Attack (JSMA) [17] is given.

A machine learning model aims to learn the mapping between its inputs and outputs. In general, given input $\boldsymbol{x}$ composed of n features, a model produces output $\boldsymbol{y}$, which is the vector of m dimensions, representing the probability of $\boldsymbol{x}$ being classified as each category. There are some typical machine learning models to be used in our experiments, including Random Forest [29], Decision Tree [30], and Support Vector Machine (SVM) [31]. Decision Tree is created by continuously maximizing the information gain arising from the selection of a condition as a way of dividing the data in two subsets according to the value of an input feature. Given the training data (X, Y), SVM computes a $(n-1)$-dimensional hyperplane with the largest margin to solve a convex optimization problem. Random Forest is an ensemble learning method for classification and regression. Its operation is by constructing a large number of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

CNN and RNN are based on the architecture of the neural network, which is composed of interconnected neural layers and activation functions for each layer. A neural network learns the mapping function f between inputs and outputs, which is a nonlinear and non-convex function. Neural networks transmit output error to each neuron in hidden

layers through the backward propagation algorithm [34] and update the connection weights of each neuron iteratively using the gradient descent algorithm to reduce error values continuously. $J(\theta)$ is defined as the model error function, where θ denotes the parameters to be updated in the training phase. These are generally the connection weights and biases between the nodes in the network. According to the gradient descent algorithm, the formula for parameters updating is:

$$\theta = \theta - \varepsilon \cdot g$$

where g is the gradient of the cost function with respect to model parameters θ, ε is the learning rate which determines the stride of decrease of θ in the opposite direction of the gradient. There are four advanced gradient descent optimization algorithms considered in this paper as follows:

- Stochastic Gradient Descent. It is by far the most common method for optimizing neural networks, and is one of the most popular algorithms for performing optimization. Many optimization algorithms are optimized and upgraded based on gradient descent.
- RMSprop. It is an unpublished and adaptive learning rate method proposed by Geoff Hinton in his lecture. RMSprop divides the learning rate by an exponentially decaying average of squared gradients as well, which is similar to Adadelta. The core iteration formula is shown below:

$$\theta = \theta - \frac{\varepsilon}{\sqrt{\beta \cdot \mathrm{E}\left[g^2\right]_{t-1} + (1-\beta) \cdot g_t^2 + \epsilon}} g_t$$

  where t denotes the t^{th} epoch, $\mathrm{E}\left[g^2\right]$ is the accumulation of squared gradients. β is the exponential decay rate. ϵ is a small positive number to avoid the divisor of 0.
- Adadelta. It was proposed by Zeiler [21]. The learning rate decreases gradually during training iterations. Unlike previous gradient descent algorithms, this method performs different learning rate ε for each parameter in neural networks. Adadelta performs smaller changes on frequently updated parameters and performs larger changes on infrequently updated parameters. The iteration formula is:

$$\theta = \theta - \frac{\sqrt{\beta \cdot \mathrm{E}\left[\Delta\theta^2\right]_{t-1} + (1-\beta) \cdot \Delta\theta^2 + \epsilon}}{\sqrt{\beta \cdot \mathrm{E}\left[g^2\right]_{t-1} + (1-\beta) \cdot g^2 + \epsilon}} g_t$$

  where $\Delta\theta$ is the change value of the parameter.
- Adam. It was proposed by Kingma *et al*. in 2015 [20] and also calculates the adaptive learning rate for each parameter. It combines the algorithm ideas of momentum and RMSprop. Adam integrates the advantages of the momentum optimization algorithm and the Adagrad [22] optimization algorithm in order to accelerate convergence in the early stage of training, while reducing the learning rate continuously during the training phase. The iteration formula is shown below. Compared with other adaptive learning rate algorithms, the convergence speed is faster and the learning performance is more effective.

$$\theta = \theta - \varepsilon \cdot \frac{\left[\beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t\right] / \left(1-\beta_1^t\right)}{\sqrt{\left[\beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2\right] / \left(1-\beta_2^t\right)} + \epsilon}$$

  where β_1, β_2 are the exponential decay rates, m_i and v_i means the 1st and the 2nd moment vector in the t^{th} epoch.

The reason of selecting these four optimization algorithms is that they are used in a variety of deep learning tasks and the tasks have achieved excellent results based on these optimizers. Therefore, it is meaning to choose these four optimizers. More comprehensive optimization algorithm analysis can refer to [22].

Adversarial examples can be crafted based on the JSMA. The two most important elements of JSMA algorithm are the Jacobian matrix and the saliency map. Jacobian matrix can evaluate the sensitivity of the output to each input of the model. The formula of the Jacobian matrix is:

$$J_F(X) = \frac{\partial F(X)}{\partial X}$$

where F is a mapping function which the deep learning model learns after training. If the input X and the output of F are multi-dimensional, then the $J_F(X)$ is a matrix. The saliency map illustrates which input features adversaries are interested in producing adversarial examples. The formula of the saliency map is:

$$S(X,k) = \begin{cases} 0 & \text{if } J_{ik}(X) < 0 \\ & \text{or } \sum_{j \neq k} J_{ij}(X) > 0 \\ J_{ik}(X) \left| \sum_{j \neq k} J_{ij}(X) \right| & \text{otherwise} \end{cases}$$

where i is the index of the input X. j and k are the index of the output of F. JSMA algorithm first obtains the Jacobian matrix according to the mapping function F, and the saliency map is constructed on the Jacobian matrix. Then the saliency map tells which input dimensions should be perturbed and we modify the selected features slightly. We continue these steps until we reach the misclassification or maximum perturbation.

B. RELATED WORK

Adversarial examples in DNNs were discovered in [8]. They demonstrated a method of reliably detecting these perturbations through a box-constrained optimization method, which depended on internal network states. Another technique was introduced in [14]. Their fast gradient sign method (FGSM) linearizes the cost function of the model around the input to be perturbed and selects the perturbation by differentiating the cost function to the input during the training phase. Based on FGSM and the raw gradient of loss, Rozsa *et al*. [24] introduced a new adversarial example generation method, the hot/cold (HC) approach, which is capable of crafting multiple adversarial examples for each input efficiently. Also, Papernot *et al*. [17] introduced a method to produce perturbations by utilizing the mapping relations between inputs

and outputs of the model. They used the forward derivative to evaluate the sensitivity of the model's output to each input using Jacobian matrix. His follow-up work [25] introduced approaches that crafted adversarial input sequences for RNNs. The attacks described above are white-box attacks. Different from these works, our paper aims to verify that the optimizer poses impact on the quality of adversarial example. One thing to declare is that the purpose of our experiment is not to evaluate the effect of the optimizer on the convergence of the cost function in model training and we did not compare the differences between the two models from the perspective of the accuracy performance in classification. In fact, the work has been done in [22].

Besides researches of white-box attacks, there were studies on the black-box attacks. Gao *et al.* [39] introduced the substitute training and linear augmentation to boost the targeted black-box attacks. Shi *et al*. [40] introduced a new black-box attack using Curls iteration and Whey optimization, which can diversify the trajectory and squeeze the noise. Papernot *et al*. [23] introduced a black-box attack with substitute dataset and proposed that the architecture of DNN had a limited impact on the transferability of adversarial examples, but they fixed the optimizer. In this paper, we choose four optimizers and want to analysis the effect of optimizer on adversarial examples.

Mechanisms for detecting and defending adversarial examples have also been proposed. Breiman *et al*. [29] proposed an approach of detecting adversarial examples by using the statistics of convolutional layer outputs. Papernot *et al*. [13] introduced a defensive mechanism to reduce the effect of adversarial examples on DNNs. Szegedy *et al*. [8] used crafted examples to train models and demonstrated that the overall performance and the adversarial robustness of the trained models can be improved. Tramér *et al*. [15] further introduced ensemble adversarial training, which augments training data with perturbations transferred from other models. Their results show that the technique yield model with strong robustness to black-box attacks. Our paper provides a new perspective from the optimizer for defenders to make the model more robust.

The reason why adversarial examples can affect the model performance was examined in [14], and they focused on the linearity of the model architecture. Our paper focuses on the effect of optimizer algorithm on model robustness, which extends the impact factors in the process of adversarial example generation and provides a new idea to the follow-up researchers.

III. METHODOLOGY

This section presents the methodology for comparing the performance of four optimizers, SGD, RMSprop, Adadelta, and Adam, regarding model robustness against white-box and black-box adversarial attacks. We first introduce the datasets to be used in experiments of Section III.A. Then the white-box attack is described to generate adversarial examples in Section III.B. In Section III.C, black-box attacks against some traditional and deep classifiers are conducted to evaluate the influence of optimizers to transferability of the adversarial examples. Finally, we use evaluation indicators described in Section III.D to quantify the impact of different optimizers on the white-box and the black-box attacks.

TABLE 1. Overview of NSL-KDD, MNIST and DREBIN datasets.

DATASET	TRAIN	TEST	TYPE
KDD	67373	9711	Normal
	45927	7460	Dos
	995	2885	R2L
	52	67	U2R
	11656	2421	Probing
MNIST	60000	10000	0-9
DREBIN	41129	41120	Benign
	1187	1793	Malicious

A. DATA PREPROCESSING

We choose the following three datasets: MNIST handwritten digital dataset, NSL-KDD network-based intrusion detection system dataset and DREBIN android malware detection dataset. The reason of choosing these three datasets is that they are used in different research fields and there are differences in data types among these three selected datasets. MNIST dataset is unstructured dataset. Both NSL-KDD and DREBIN are structured datasets. Table 1 describes the main features of three datasets and detailed descriptions are given below:

- MNIST—MNIST handwritten digit dataset contains 60,000 black and white images with width and height of 28 pixels, of which 50,000 are for training and 10,000 for testing. This dataset is mainly used to train and test the machine learning classification models in image recognition.
- NSL-KDD—NSL-KDD is an improved version of the KDD CUP99 dataset for network intrusion detection that solves a variety of issues in the original dataset. Each connection record in the dataset contains 41 features. The connection types can be divided into 5 categories, of which 4 attack types can be subdivided into 39 attack types.
- DREBIN—DREBIN is a widely used Android malware detection dataset and it contains the Android software from 2010 to 2012, including benign and malicious applications. Also, its advantages lie in apparent features and labels, diversity of software.

We train each model with different optimizers on three different datasets. Before we train models, each dataset needs to be preprocessed. MNIST dataset contains plenty of handwritten images, and the traditional way of dealing with images is feature scaling, which scales all pixel values between -1 and 1. For KDD dataset, the original dataset is not suitable to train neural networks. Thus, we quantify and normalize the record features in the dataset.

After preprocessing the dataset, we get a multi-dimension vector for each training record. For DREBIN dataset, we use the binary indicator vectors to represent the features and normalize the binary indicator vectors to preprocess the DREBIN dataset.

B. WHITE-BOX ATTACK

The model attack itself with self-generated adversarial examples is defined as the white-box attack in this paper. There are various algorithms for crafting adversarial examples. For example, the Fast Gradient Sign Method (FGSM) [14] and the Jacobian-based Saliency Map Attack (JSMA). Both these methods were first proposed in image recognition problem. FGSM generates perturbation by applying the derivative of the cost function to the input. JSMA method is based on the Jacobian matrix of the function the model learned during training. While FGSM perturbs all the components of input, JSMA tries to modify the salient features that contribute to the target class. We consider that adversarial examples should hide itself as much as possible to avoid being detected, especially in malware-detection fields. Therefore, adversarial examples generated on the JSMA algorithm meet our requirements. Also, Researchers applied JSMA to both NSL-KDD (the network-based intrusion detection system dataset) and DREBIN (an android malware detection dataset) in order to verify the suitability for intrusion detection and malware detection tasks in [26]. These discussions motivate the application of JSMA to synthesize adversarial examples in our experiments.

A deep neural network model with three hidden layers is trained on MNIST, NSL-KDD and DREBIN datasets with SGD, RMSprop, Adadelta and Adam optimizers respectively. Then we have four models, SGD-model, RMS-model, Adadelta-model, and Adam-model, for crafting adversarial examples. Then JSMA is used to generate the adversarial examples. The crafting rate and the perturbation rate of crafting adversarial examples against legitimate examples are compared between those models. All the hyperparameters of these models are the same except for the optimizer to be used and learning rate. The reason is that we consider learning rate is part of the optimizer, and different optimizer has different optimal learning rate to help it update the model's parameters quickly. Therefore, we adjust an optimal learning rate for each optimizer so that the model based on this optimizer can converge fastest and reach the best performance. More details about the model we build are that three hidden layers are in turn 256, 512, 128 nodes. ReLU is chosen as the activation function to guarantee the non-linearity of the model. And then dropout method with rate 0.5 is adopted to regularize the model and prevent overfitting. The input dimensions and the output dimensions correspond to each preprocessed dataset. For example, the input dimension and the output dimension of the KDD dataset is 122 and 5.

We train each model via dataset and optimizer, and craft adversarial examples using JSMA. In order to facilitate the comparison of experimental results, we choose the class 'Normal' in KDD, class '0' in MNIST and class 'Benign' in DREBIN as the adversarial target. Concretely, we aim to deceive the model to classify those perturbed samples as 'Normal', '0' and 'Benign' no matter which category the origin inputs should belong to. According to JSMA, in terms of specific input X, we calculate the Jacobian matrix J of the function F learned by the model after training, which is called forward derivative. Then a saliency map is constructed based on the forward derivative. Perturbed features are identified by the saliency map and modified until we achieve our adversarial goal to misclassify X as the adversarial target or reach the upper limits of iterations. In addition, we set epoch to 20 in order to investigate changes of metrics in adversarial example generation in each epoch under different optimizers, so as to get the most suitable optimizer for adversarial example generation.

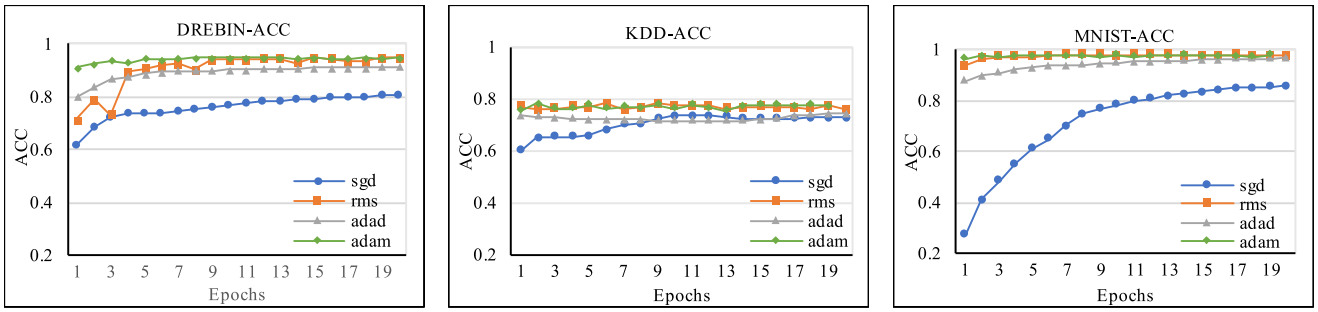
C. BLACK-BOX ATTACK

The black-box attack means that the adversary makes use of transferability of adversarial examples to implement attacks on the target model. We select classifiers such as decision trees, random forests, and linear support vector machine along with CNNs and RNNs to implement the cross-model attack capabilities of the adversarial examples against the traditional machine learning models and deep learning models. The CNN layers we construct is conv16-conv32-full32. Here, convN means a convolutional layer with N filters, and fullN means a fully-connected layer with N nodes. The RNN layers is as follows: LSTM30-LSTM60-full32, where LSTMN means a LSTM layer with N units. According to the thread model taxonomy, an adversary does not know the architecture and specific parameters of the model. Thus, we consider it as a black-box attack.

D. EVALUATION METRICS

As mentioned in Section III.B, we divide all the classes into target classes and non-target classes so it is a binomial classification problem. For binomial classification, we consider confusion matrix and its derived indicators to evaluate the performance of black-box attack. The definition of the confusion matrix is given in Table 2. We only select representative Accuracy, F1-score, and AUC value, the area of ROC curve, to measure the classification results of the black-box attack. The reason is that the accuracy shows the classification results intuitively. F1-score avoids the output of the classifier being biased towards one result, indicating the reliability of Accuracy. ROC curve applies False Positive Rate as the horizontal coordinate, and True Positive Rate as the vertical coordinate to draw the figure. The size of the curve area, denoted as AUC, reflects the performance. The details of calculation indicators are shown in Table 2 and Table 3.

As for the evaluation of white-box attack, we choose Dvalue, crafting rate of generating adversarial examples and the perturbation rate when generating adversarial examples. The indicator of the crafting rate of generating adversarial examples can intuitively reflect whether the adversarial



(a) Accuracy score on DREBIN dataset (b) Accuracy score on KDD dataset (c) Accuracy score on MNIST dataset

FIGURE 1. ACC on datasets and optimizers.

TABLE 2. Confusion matrix.

CONFUSION MATRIX	CONDITION POSITIVE	CONDITION NEGATIVE
PREDICTION POSITIVE	Ture Positive(TP)	False Positive(FP)
PREDICTION NEGATIVE	False Negative(FN)	True Negative(TN)

TABLE 3. Evaluation indicators.

EVALUATION INDICATORS	DESCRIPTION
ACCURACY (ACC)	$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$
PRECISION (FPR)	$FPR = \frac{FP}{FP+TN}$
RECALL	$Recall = \frac{TP}{TP+FN}$
F1-SCORE	$F1-score = \frac{2 \cdot TP}{2TP+FP+FN}$
DVALUE	$Dvalue = ACC(X_{orig}) - ACC(X_{adv})$

examples are successfully generated or not. The accuracy-adversarial score is the accuracy score on the adversarial examples, which demonstrates the quality of adversarial examples generated by JSMA based on the trained model. The perturbation rate is the percent of perturbed features of total features for generating adversarial examples, indicating the degree of adding noise to the original sample when generating adversarial samples. Also, we introduce an advanced evaluation metric called Dvalue shown in Table 3, where X_{orig} denotes original examples and X_{adv} denotes adversarial examples. Dvalue shows the difference between accuracy score on original examples and accuracy-adversarial score on adversarial examples. The value of Dvalue represents the degree of misclassification caused by adversarial examples. Generally speaking, the white-box attack succeeds when the crafting rate of adversarial example generation is high, the perturbation rate is low, and Dvalue is large.

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

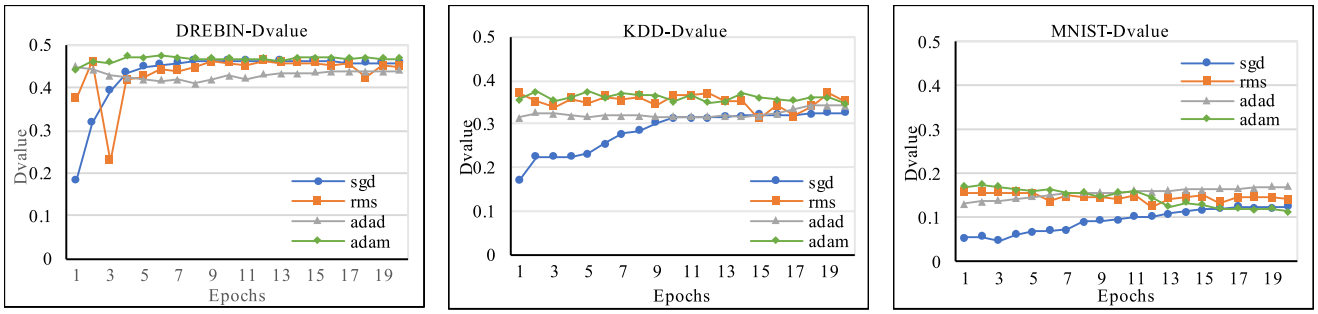
This section first presents the results of the generated adversarial examples. Then results of adversarial example transferability are present. At last, the summary is given after analyzing those experiment results.

A. SENSITIVITY ANALYSIS UNDER WHITE-BOX ATTACK SETTINGS

This section presents the results of generating adversarial examples under four optimizers. SGD-samples, RMS-samples, Adam-samples, and Adadelta-samples denote the adversarial examples generated by SGD-model, RMS-model, Adadelta-model, and Adam-model, respectively.

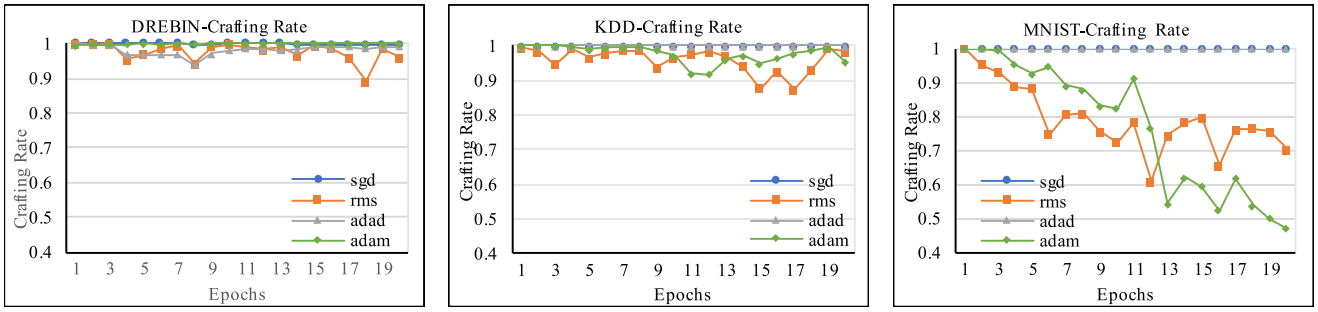
We draw the curves of ACC, Dvalue, the crafting rate and the perturbation rate of adversarial example generation as the number of epochs increased under different datasets and optimizers. The results are shown in Figure 1. Since the curves of Figure 1(b) and Figure 1(c) are similar to those in Figure 1(a), we focus on analyzing Figure 1(a). We observe that the yellow curve is always at the top and the blue curve is always at the bottom of the four curves, which indicates that Adam optimizer is the fastest convergence optimizer and the slowest is SGD optimizer and all the models does not overfit under different datasets, which is in line with our expectation. RMS is closest to the convergence rate of Adam. As for Adadelta optimizer, its convergence rate is slightly inferior to that of Adam and RMS. Therefore, from the perspective of the convergence rate of the DNN model, no matter which dataset, Adam and RMS optimizers perform best, followed by Adadelta, and SGD optimizer is the worst.

Dvalue reflects the aggression of adversarial examples on the model in the white-box attacks. As can be seen in Figure 2, in the structured data (DREBIN and NSL-KDD), Dvalue of Adam-samples is large as a whole considering the fluctuations, and RMS, SGD, and Adadelta are followed by Adam. However, in the unstructured data (MNIST), the trend of Adadelta-samples rises steadily, but the trend of Adam-samples and RMS-samples drops continuously, indicating that only Adadelta-samples maintain and improve the aggression of adversarial examples. So merely considering Dvalue, Adam optimizer is the best choice to generate adversarial



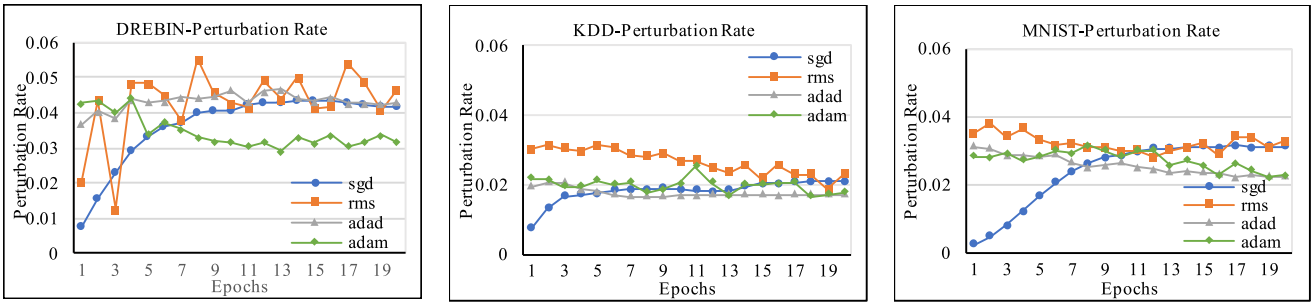
(a) Dvalue on DREBIN dataset (b) Dvalue on KDD dataset (c) Dvalue on MNIST dataset

FIGURE 2. Dvalue on datasets and optimizers.



(a) Crafting rate on DREBIN dataset (b) Crafting rate on KDD dataset (c) Crafting rate on MNIST dataset

FIGURE 3. Crafting rate on datasets and optimizers.



(a) Perturbation rate on DREBIN dataset (b) Perturbation rate on KDD dataset (c) Perturbation rate on MNIST dataset

FIGURE 4. Perturbation rate on datasets and optimizers.

examples on the structured dataset and Adadelta optimizer is the best choice to generate adversarial examples on the unstructured dataset.

In terms of the curves of crafting rate in Figure 3, the crafting rate of Adam and SGD optimizer is stable around 1 in DREBIN dataset. The curves of RMS and Adadelta fluctuate considerably, especially the RMS curve. In KDD dataset, the SGD and Adadelta curves stabilize at 1, and the Adam curve and RMS curve oscillate violently. But Adam is still acceptable because the crafting rate is still over 92%. In MNIST dataset, Adadelta and SGD are stable at 1. But the interesting thing is that the Adam curve and RMS curve show a downward trend as a whole. So only considering the crafting rate, no matter what dataset, the best choice is SGD, and Adam and Adadelta are candidate optimizers. In particular, Adam and Adadelta are more compatible with structured data, and Adadelta is more compatible with non-structured data.

As for the curves of perturbation rate in Figure 4, Adam, Adadelta and RMS curves are falling in fluctuation in three datasets. The perturbation rate curve of SGD optimizer rises slowly on the three datasets. We think the model based on the SGD optimizer does not converge in the previous epochs and learn about potential data connections. Thus, the accuracy is a bit low on the test set and the JSMA algorithm based on the model parameters does not need to add too much noise. With the increase of epochs, the perturbation rates based on the SGD optimizer are basically the same

TABLE 4. Comparison of cross-model attack regarding adversarial examples transferability on DREBIN dataset.

Model	Metric	Original Sample	Adam Sample	Adadelta Sample	RMS Sample	SGD Sample
Decision Tree	ACC	89.05%	69.78%	73.50%	**67.92%**	80.94%
	F1-score	89.08%	69.80%	73.51%	**67.94%**	80.97%
	AUC	89.10%	69.41%	73.21%	**67.52%**	80.82%
Random Forest	ACC	93.91%	**92.03%**	93.78%	93.17%	93.61%
	F1-score	93.95%	**92.10%**	93.81%	93.25%	93.69%
	AUC	93.95%	**92.11%**	93.81%	93.28%	93.73%
Linear SVM	ACC	91.48%	**76.48%**	84.06%	81.57%	91.10%
	F1-score	91.52%	**76.60%**	84.18%	81.62%	91.14%
	AUC	91.54%	**76.23%**	84.15%	81.45%	91.15%
CNN	ACC	91.21%	77.01%	56.73%	**72.60%**	86.21%
	F1-score	91.21%	77.01%	56.73%	**72.60%**	86.21%
	AUC	91.13%	76.63%	55.91%	**72.11%**	86.03%
RNN	ACC	81.63%	81.05%	83.70%	81.19%	**80.04%**
	F1-score	81.63%	81.05%	83.70%	81.19%	**80.04%**
	AUC	81.51%	80.93%	83.63%	81.07%	**79.90%**

as the other optimizers' rates. In DREBIN dataset, Adam adds less noise on average. In KDD and MNIST datasets, Adadelta adds less noise and Adam is slightly inferior to Adadelta. Therefore, when only considering the perturbation rate, Adam or Adadelta can be used as candidate optimizer based on the actual perturbation rate on the structured datasets but Adadelta is the only choice on the unstructured datasets.

In summary, on the structured data, Adam optimizer is a choice as a part of adversarial example generation because of the faster convergence rate, better quality of adversarial example generation, acceptable adversarial example crafting rate and perturbation rate. Adadelta plays the same role in the unstructured data.

B. SENSITIVITY ANALYSIS UNDER BLACK-BOX ATTACK SETTINGS

This section mainly investigates the effectiveness of optimizers under the black-box attacks in terms of adversarial example transferability. We first examine the results with the indicators against cross-model attacks using adversarial examples generated by models with different optimizers and datasets. The results are shown in TABLE 4, TABLE 5 and TABLE 6. Here, the original sample means legitimate test sample, Adam-samples means the adversarial examples generated on the Adam optimizer and so on. We can get an intuitive conclusion: optimizers have limited effects on the transferability of adversarial examples. This means that the influence of the adversarial examples generated on the optimizer on the transferability of some models is apparent, but the effect on others is not apparent. We highlight the minimum value for each column in the following three tables and make a detailed analysis. We start with an analysis of Table 4, showing the cross-model attack comparison results between four types of adversarial examples on DREBIN dataset. Adversarial examples generated on different optimizers have different considerable influence on various machine learning models. For example, RMS-samples has the greatest impact on the Decision Tree model, because the metrics, ACC, F1-score, AUC, are the minimum in the classification results of all types of adversarial examples by Decision Tree. As for the Random Forest model and Linear SVM model, Adam-samples has the greatest influence on the performance of these model. Deep learning models, CNN and RNN, are also affected by the adversarial examples crafted on the different optimizers. Adadelta-samples have the greatest influence on the CNN model, and SGD-samples have the impact on the RNN model.

We compare the results of NSL-KDD dataset and DREBIN dataset since both datasets are structured dataset. Table 5 shows the cross-model attack comparison results between four types of adversarial examples on NSL-KDD dataset. As seen in Table 5, we observe that the performance of adversarial examples generated on different optimizers varies from model to model. And the result is totally different from the results on DREBIN dataset. From Table 5, we can see that for Decision Tree, the most influenced examples generated on the optimizer is Adadelta, which is different from RMS optimizer on DREBIN dataset. As for the Random Forest, the most influential optimizer is RMS, differing from Adam on DREBIN. SGD examples affect the Linear SVM the most, but Adam examples affect the Linear SVM on DREBIN dataset the most. SGD examples also affect two

TABLE 5. Comparison of cross-model attack regarding adversarial examples transferability on NSL-KDD dataset.

Model	Metric	Original Sample	Adam Sample	Adadelta Sample	RMS Sample	SGD Sample
Decision Tree	ACC	73.37%	55.21%	**48.39%**	52.81%	48.47%
	F1-score	75.77%	60.62%	**56.19%**	63.45%	56.53%
	AUC	80.77%	77.90%	**71.25%**	78.16%	72.84%
Random Forest	ACC	74.02%	69.33%	72.18%	69.71%	**65.87%**
	F1-score	75.57%	71.20%	73.99%	**71.09%**	70.49%
	AUC	80.82%	78.41%	79.71%	**77.92%**	79.65%
Linear SVM	ACC	73.22%	52.93%	44.38%	53.88%	**44.10%**
	F1-score	75.09%	57.62%	57.24%	59.61%	**50.39%**
	AUC	77.62%	74.92%	**69.85%**	70.63%	71.72%
CNN	ACC	77.26%	61.51%	59.54%	60.99%	**50.04%**
	F1-score	77.26%	61.51%	59.54%	60.99%	**50.04%**
	AUC	81.20%	79.26%	71.99%	73.92%	**68.82%**
RNN	ACC	66.62%	53.69%	47.08%	56.47%	**46.56%**
	F1-score	66.62%	53.69%	47.08%	56.47%	**46.56%**
	AUC	73.82%	71.61%	70.07%	75.38%	**68.97%**

TABLE 6. Comparison of cross-model attack regarding adversarial examples transferability on MNIST dataset.

Model	Metric	Original Sample	Adam Sample	Adadelta Sample	RMS Sample	SGD Sample
Decision Tree	ACC	80.06%	**66.98%**	69.26%	67.23%	68.44%
	F1-score	87.64%	**68.93%**	77.27%	71.37%	75.58%
	AUC	96.79%	**94.22%**	95.87%	95.57%	95.85%
Random Forest	ACC	90.34%	84.29%	85.19%	84.03%	**83.10%**
	F1-score	94.63%	91.02%	91.74%	90.99%	**90.51%**
	AUC	98.70%	98.69%	98.70%	**98.68%**	98.70%
Linear SVM	ACC	84.42%	72.06%	**71.07%**	73.33%	72.13%
	F1-score	90.36%	82.42%	**81.77%**	83.33%	82.34%
	AUC	98.24%	98.21%	98.13%	98.21%	**98.07%**
CNN	ACC	96.36%	88.32%	89.49%	88.16%	**87.54%**
	F1-score	96.36%	88.32%	89.49%	88.16%	**87.54%**
	AUC	99.16%	98.87%	98.83%	98.96%	**98.61%**
RNN	ACC	37.16%	**32.82%**	34.83%	33.28%	34.37%
	F1-score	37.16%	**32.82%**	34.83%	33.28%	34.37%
	AUC	60.71%	61.45%	60.78%	61.28%	**60.28%**

deep learning models, CNN and RNN, where the evaluation indicator values drop the most. But on DREBIN dataset, the most affected adversarial examples of these two models are Adadalta and SGD.

It can be seen from the results of two datasets, DREBIN and NSL-KDD, adversarial examples generated on different optimizers have excellent transferability in some models, but the transferability is not so prominent in other models. This indicates that adversarial examples generated on the same optimizer may not have the same degree of transferability for all models and this difference are not regular. Therefore, we think that optimizers have a limited impact on the transferability of adversarial examples in terms of structured datasets.

Table 6 shows the cross-model attack comparison results on MNIST dataset. The conclusion is the same with NSL-KDD and DREBIN datasets. As seen in Table 6, each model has a set of adversarial examples with the best

performance of transferability. For instance, Adam examples perform best on Decision Tree and RNN, SGD examples perform best on Random Forest and CNN, and Adadelta examples perform best on Linear SVM. Associated with Table 4 and Table 5, we can conclude that the impact of optimizers on the transferability of adversarial examples is limited.

C. SUMMARY

This section makes a summary through the experiment results and the above analysis of Section IV.A and IV.B as follows:

(1) In terms of white-box attacks, the DNN model with Adam optimizer can generate adversarial examples with high convergence rate, low perturbation rate and high generation rate on structured datasets. But on the unstructured dataset, the optimizer with the same effect is Adadelta optimizer.

(2) In term of black-box attacks, the impact of optimizers on the transferability of adversarial examples is limited.

V. CONCLUSION AND FUTURE WORK

In this paper, we investigate models' sensitivity to adversarial examples with different optimizer algorithms. We construct experiments to compare the four most used optimizers, Adam, Adadelta, RMSprop, and SGD in terms of their model robustness against adversarial examples. Meanwhile, we train the models with different optimizers on structured datasets (NSL-KDD and DREBIN) and unstructured dataset (MNIST) and then compare diverse behaviors of models. Experimental results indicate that (1) for structured datasets, Adam optimizer can generate higher quality adversarial examples when the white-box attack is implemented; (2) for unstructured datasets, Adadelta optimizer can generate higher quality adversarial examples in the white-box attack. We conclude that the gradient descent optimizer has a greater impact on the generation of adversarial examples on neural network models.

This paper uses only one adversarial example attack method and one target class to assess the effect of optimizer on the adversarial examples. In the future work, research on experimental analysis of scenarios of more representative attacks, more complex network architectures and more complicated datasets should be conducted. Moreover, the reason why optimizer can affect the adversarial examples will be explored as well.

REFERENCES

[1] G. Gkioxari, R. B. Girshick, P. Dollár, and K. He, "Detecting and recognizing human-object interactions," in *Proc. IEEE Conf. CVPR*, Jun. 2018, pp. 836–859.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.

[3] Z. Deng, J. Chen, Y. Fu, and G. Mori, "Probabilistic neural programmed networks for scene generation," in *Proc. NIPS*, 2018, pp. 4032–4042.

[4] Y. Adi, N. Zeghidour, R. Collobert, N. Usunier, V. Liptchinsky, and G. Synnaeve, "To reverse the gradient or not: An empirical comparison of adversarial and multi-task learning in speech recognition," in *Proc. IEEE Conf. ICASSP*, May 2019, pp. 3742–3746.

[5] K. Zhang, G. Lv, E. Chen, L. Wu, Q. Liu, and C. L. P. Chen, "Context-aware dual-attention network for natural language inference," in *Proc. PAKDD*, 2019, pp. 185–198.

[6] D. Brooks, O. Schwander, F. Barbaresco, J. Schneider, and M. Cord, "Exploring complex time-series representations for Riemannian machine learning of radar data," in *Proc. ICASSP*, May 2019, pp. 3672–3676.

[7] X. Liu, Z. Deng, and Y. Yang, "Recent progress in semantic image segmentation," *Artif. Intell. Rev.*, vol. 52, no. 2, pp. 1089–1106, Aug. 2019.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proc. ICLR (Poster)*, 2014, pp. 1–10.

[9] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. IEEE Conf. SP*, May 2017, pp. 39–57.

[10] A. Fawzi, O. Fawzi, and P. Frossard, "Analysis of classifiers' robustness to adversarial perturbations," *Mach. Learn.*, vol. 107, no. 3, pp. 481–508, 2017.

[11] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *Proc. ICLR (Workshop)*, 2017, pp. 1–14.

[12] A. Graese, A. Rozsa, and T. E. Boult, "Assessing threat of adversarial examples on deep neural networks," in *Proc. ICMLA*, Dec. 2016, pp. 69–74.

[13] N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Secur. Privacy*, May 2016, pp. 582–597.

[14] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proc. ICLR (Poster)*, 2015, pp. 1–11.

[15] F. Tramèr, A. Kurakin, N. Papernot, D. Boneh, and P. D. McDaniel, "Ensemble adversarial training: Attacks and defenses," in *Proc. ICLR (Poster)*, 2018, pp. 1–20.

[16] M. Cissé, P. Bojanowski, E. Grave, Y. Dauphin, and N. Usunier, "Parseval networks: Improving robustness to adversarial examples," in *Proc. ICML*, 2017, pp. 854–863.

[17] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. EuroS&P*, Mar. 2016, pp. 372–387.

[18] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. CISDA*, 2009, pp. 1–6.

[19] D. Arp, M. L. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in *Proc. NDSS*, vol. 14, 2014, pp. 23–26.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. ICLR (Poster)*, 2015, pp. 1–15.

[21] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980

[22] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*. [Online]. Available: https://arxiv.org/abs/1609.04747

[23] N. Papernot, P. McDaniel, I. J. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proc. ASIA CCS*, New York, NY, USA, 2017, pp. 506–519.

[24] A. Rozsa, E. M. Rudd, and T. E. Boult, "Adversarial diversity and hard positive generation," in *Proc. CVPR Workshop*, Jun. 2016, pp. 410–417.

[25] N. Papernot, P. D. McDaniel, A. Swami, and R. E. Harang, "Crafting adversarial input sequences for recurrent neural networks," in *Proc. MILCOM*, 2016, pp. 49–54.

[26] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, "Adversarial examples for malware detection," in *Proc. ESORICS*, 2017, pp. 62–79.

[27] A. Rozsa, M. Günther, and T. E. Boult, "Are accuracy and robustness correlated," in *Proc. ICMLA*, Dec. 2016, pp. 227–232.

[28] X. Li and F. Li, "Adversarial examples detection in deep networks with convolutional filter statistics," in *Proc. ICCV*, 2017, pp. 5775–5783.

[29] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[30] T. Wang, Z. Li, Y. Yan, and H. Chen, "A survey of fuzzy decision tree classifier methodology," in *Proc. ICFIE*, 2007, pp. 959–968.

[31] N. Cristianini and B. Schölkopf, "Support vector machines and kernel methods: The new generation of learning machines," *AI Mag.*, vol. 23, no. 3, pp. 31–42, 2002.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[33] K. Anders and J. A. Hertz, "A simple weight decay can improve generalization," in *Proc. NIPS*, 1992, pp. 950–957.
[34] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," California Univ. San Diego La Jolla Inst. Cogn. Sci., San Diego, CA, USA, Tech. Rep. ICS-8506, 1985.
[35] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proc. NIPS*, 1989, pp. 396–404.
[36] T. Tijmen and G. E. Hinton, "Lecture 6.5-RmsProp: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural Netw. Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.
[37] C. Darken, J. Chang, and J. Moody, "Learning rate schedules for faster stochastic gradient search," in *Proc. IEEE Workshop Neural Netw. Signal Process. II*, Aug. 1992, pp. 3–12.
[38] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "Droidensemble: Detecting Android malicious applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.
[39] X. Gao, Y. Tan, H. Jiang, Q. Zhang, and X. Kuang, "Boosting targeted black-box attacks via ensemble substitute training and linear augmentation," *Appl. Sci.*, vol. 9, no. 11, p. 2286, 2019. doi: 10.3390/app9112286.
[40] Y. Shi, S. Wang, and Y. Han, "Curls & whey: Boosting black-box adversarial attacks," presented at the IEEE Conf. CVPR, CA, USA, Jun. 2019.
[41] P. Chandarana and M. Vijayalakshmi, "Big data analytics frameworks," in *Proc. IEEE Conf. CSCITA*, Apr. 2014, pp. 430–434.

YIXIANG WANG received the B.S. degree from Beijing Jiaotong University, China, in 2018, where he is currently pursuing the Ph.D. degree with the Beijing Key Laboratory of Security and Privacy in Intelligent Transportation. His research interests include adversarial examples and security in machine learning.

JIQIANG LIU received the B.S. and Ph.D. degrees from Beijing Normal University, in 1994 and 1999, respectively. He is currently a Professor with the School of Computer and Information Technology, Beijing Jiaotong University. He has published more than 80 scientific articles in various journals and international conferences. His main research interests include trusted computing, cryptographic protocols, privacy preserving, and network security..

JELENA MIŠIĆ (M'91–SM'08–F'18) is currently a Professor of computer science with Ryerson University, Toronto, Ontario, Canada. She has published four books, more than 125 articles in archival journals and close to 190 papers at international conferences in the areas of computer networks and security. She is a member of ACM. She serves on editorial boards of the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE INTERNET OF THINGS JOURNAL, the IEEE NETWORK, *Computer Networks*, and *Ad hoc Networks*.

VOJISLAV B. MIŠIĆ (M'92–SM'08) is currently a Professor of computer science with Ryerson University, Toronto, Ontario, Canada. His research interests include performance evaluation of wireless networks and systems, and software engineering. He is a member of ACM. He serves on the editorial boards of the IEEE TRANSACTIONS ON CLOUD COMPUTING, *Ad hoc Networks*, *Peer-to-Peer Networks and Applications*, the *International Journal of Parallel*, and *Emergent and Distributed Systems*..

SHAOHUA LV was born in 1993. He received the B.S. degree from Donghua University, in 2016, and the M.S. degree from Beijing Jiaotong University, in 2019. His research interests include mainly in network intrusion detection and nature language processing.

XIAOLIN CHANG is currently a Professor with the School of Computer and Information Technology, Beijing Jiaotong University. Her current research interests include edge/cloud computing, network security, and security and privacy in machine learning.