Query-Efficient and Imperceptible Attacks on Multivariate Time Series DNN Models

Haiying Shen, Fan Yao, Tanmoy Sen, Gustavo Moreira, and Ankur Sarker
University of Virginia, Charlottesville, VA 22904
{hs6ms, fy4bc, ts5xm, gm2qb, as4mz}@virginia.edu

***Abstract*—Many black-box adversarial attacks for deep neural network (DNN) models for two-dimensional image datasets have been proposed. Though there are many pervasive and computing application scenarios that need multivariate time-series data as DNN inputs, little research has been devoted to black-box adversarial attacks on multivariate time-series DNN models, which have higher requirements on efficiency and imperceptibility. To meet the requirements, in this paper, we propose three different black-box adversarial attacks based on an existing attack: a self-adaptive step technique to improve the query-efficiency and success rate; an L_0 distance-based attack to improve imperceptibility; and an input coordinate importance oriented attack based on multiplicative weight update (MWU), which exploits the time-series structure and improves query-efficiency and imperceptibility. These proposed attacks are able to make trade-offs between successful rate, L_2/L_0 distance, and the number of queries tailored for a particular targeted attack task. We conducted extensive experiments using ten different combinations of DNN models and datasets to test the effectiveness of the proposed black-box adversarial attacks for multivariate time-series based DNNs. The proposed attacks achieve up to 51% higher success rates with 25% fewer queries, and 84% fewer perturbation amounts over the existing black-box adversarial attacks.**

I. Introduction

Multivariate time-series based Deep Neural Network (DNN) models are widely applied in many pervasive computing applications. For example, multivariate time-series based recurrent neural networks (RNNs) have achieved high performance in accurately classifying driving states and maneuvers for autonomous vehicles [1] in pervasive vehicle-to-vehicle wireless networks, and for air pollution evaluation, surveillance, human activity recognition, sleep quality monitoring, and pedestrian detection by pervasive wireless sensor networks [2], [3].

Recent studies [4], [5] show that a well-trained DNN can be easily fooled to output an incorrect class by a well-designed input sample (i.e., consisting of multiple input signals), called an *adversarial input* [6]. Adversarial attacks generate significant threats to many applications. For example, consider a scenario where an autonomous vehicle uses a DNN model to predict nearby vehicles' driving maneuvers based on their time-series driving signals. If the DNN fails to classify driving maneuvers correctly, potentially disastrous consequences may occur, ranging from a large-scale interruption of transportation services (e.g., clogged roads) to even a life-threatening accident [7]. There are three different adversarial attack scenarios: white-box [4], [8]–[10], grey-box [5], [11], [12], and black-box attacks [13]–[16].

The black-box attacks are more realistic [14], [16]. In a black-box attack, an attacker finds an adversarial input by solving an optimization problem aiming to find the minimum *perturbation*. In this process, a perturbation is generated to be added to a given input and the generated adversarial input is applied to the DNN model to query whether the attack is successful repeatedly until it is successful or the number of queries reaches a certain threshold. The attack is successful when the predicted class label is the target class label (targeted attack) or any class labels except the original class label (untargeted attack). In Zeroth-Order Optimization Black-Box Attack (ZOO) [14], each time, the perturbation is generated by adding a newly generated gradient (calculated based on step size h and other factors by the standard gradient descent algorithm) to the previous perturbation under the constraint that the L_2 distance is less than a predefined threshold. The L_2 distance is calculated by $\|\boldsymbol{\delta}\|_2 = \sqrt{\sum_{i=1}^{n} \delta_i^2}$, where $\boldsymbol{\delta} = [\delta_1, \delta_2, ..., \delta_n]$ is the vector of the perturbation, δ_i is the i-th coordinator, and n is the total number of coordinates in the input. Each gradient calculation is performed for one coordinate, so there are many gradient calculations and the attacker needs to send many queries to the black-box DNN model (using newly generated adversarial input) for the attack success checking. *Success rate* equals the number of successful attacks over the total number of queried attacks.

In the pervasive computing applications that need multivariate time-series data as DNN inputs [2], [3], [17], the input is a $m \times n$ dataset formed by m multiple time-series and each time-series is for one signal or variate (e.g., gyroscope, accelerator, magnetometer) for n time stamps. The black-box adversarial attacks on the multivariate time-series DNNs impose more formidable challenges on achieving query-efficiency and imperceptibility [18]. First, the existing black-box adversarial attacks (on image datasets) already require many queries to generate an adversarial input. Directly applying these attacks on the multivariate time-series datasets will make the query-efficiency issue even more severe. This is because a multivariate time-series data has higher dimensions than

ISBN 978-3-903176-48-5©2022 IFIP

two-dimensional image data and it requires more gradient calculations. Second, the multivariate time-series dataset offers more variate correlations to be analyzed to detect adversarial attacks. To tackle the challenges, we propose three different black-box adversarial targeted attacks for multivariate time-series DNN models as listed below.

- ZOO [14] uses a fixed value (i.e., 0.0001) as the step size h of gradient estimation. Based on our analysis, a small value of step size h often leads to a near zero-gradient estimation, thus reaching the query number limit and a low success rate. To address this issue, we propose an adaptive strategy to adjust the step size h when there is a zero-gradient estimation in the attack generating process. We name this method as Ada-ZOO adversarial attack.
- A sparser perturbation (i.e., more zero coordinate values) is more imperceptible to humans. Therefore, the L_0 distance (i.e., the number of non-zero coordinates), denoted by $\|\boldsymbol{\delta}\|_0$, is a more appropriate metric than the L_2 distance. However, the L_0 distance metric is non-differentiable and thus not suitable for the standard gradient descent algorithm since a function should be differentiable to estimate its standard gradient descent. To solve this issue, we propose the ZOO-L_0 black-box adversarial attack to find L_0 perturbation starting from a successful L_2 perturbation. Basically, it repeats zeroing out each coordinate with the lowest perturbation in the L_2 perturbation and finding a new L_2 perturbation using the Ada-ZOO algorithm.
- Since the gradient estimation with respect to each input coordinate is expensive in the black-box setting, we propose a ZOO-MWU black-box attack that selectively updates partial coordinates that are more important to the target class label by using the Multiplicative Weight Update (MWU) method [19]. Explicitly, we assign an initial weight (i.e., 1) to each input coordinate and the weights are updated in the standard gradient descent algorithm process based on their contributions to a successful attack. The standard gradient descent algorithm then chooses the coordinates with higher weights to add perturbations.

In summary, the major contributions of this paper are:

- **Ada-ZOO attack.** We analyzed the impacts of static small step size h in the existing ZOO attack and propose a novel Ada-ZOO black-box adversarial attack, which adaptively selects the step size h during the attack generation procedure to increase the success rate.
- **ZOO-L_0 attack.** We propose a novel ZOO-L_0 black-box adversarial attack, which minimizes the number of input coordinates altered to increase the imperceptibility.
- **ZOO-MWU attack.** We propose a novel ZOO-MWU black-box adversarial attack that calculates the weights of different coordinates and only modifies the high-weight coordinates to create perturbation to improve query-efficiency and imperceptibility. We propose two ZOO-MWU attack approaches for the case of a single attack and for the case of a batch of attacks.
- **Experimental evaluations.** We conducted extensive experimental studies using different combinations of four real-world datasets and three DNN models. From the evaluation, we observe that the proposed attacks are able to achieve over up to 51% higher success rates with 25% fewer queries, and 84% fewer perturbation amounts over the existing black-box adversarial attacks.

The rest of the paper is organized as follows. Section II discusses existing literature. Section III presents the necessary overview of black-box attacks and ZOO for understanding our approaches. Then, Section IV presents the three proposed attacks. Section V evaluates the proposed attacks through extensive experimental studies. Finally, Section VI concludes this paper with future work.

II. Related Work

Based on the capability of the adversary, we can divide existing adversarial attacks into three major categories: white-box attacks [4], [8]–[10], black-box attacks [13]–[16], and gray-box attacks [5], [11], [12], which are somewhat in-between the white-box attacks and black-box attacks, where an attacker has specific knowledge of the inputs and outputs with the DNN model architecture.

For example, for the white-box scenarios, in [8], different time-series features are scored to find out the globally vulnerable features to be altered, where the architecture of the DNN model is known, and the gradients of the loss function for each feature are used to measure the scores. Croce *et al.* [20] proposed a white-box iterative optimization-based attack that constructs targeted audio adversarial examples on automatic speech recognition.

For the grey-box scenarios, in [11], [12], the authors analyzed the activation map of different neurons toward different class labels to design the adversary. Karim *et al.* [5] proposed a grey-box adversarial attack on time-series data, where a transfer-learning aided adversarial transformation network is used to generate the adversarial inputs. The adversarial transformation network (as a student model), along with an optimization problem, help to transform a given input into an adversarial input targeting a teacher model in minimum time.

Chen *et al.* [14] first introduced the zeroth-gradient optimization-based black-box attack, which estimates the gradient estimation of the DNN model to generate the adversarial inputs. To generate a black-box adversarial input, Tu *et al.* [16] proposed AutoZoom that uses the random full gradient estimator method to estimate gradients of all coordinates at once and uses a decoder DNN model to reduce perturbation search space by transforming a small perturbation into a large perturbation to be added to the input. Cheng *et al.* [21] also introduced a boundary black-box adversarial attack where the classification boundaries are first identified using the zeroth-gradient optimization method to generate the adversarial inputs. Brunner *et al.* [15] proposed a black-box adversarial attack framework that identifies three different prior biases (i.e., low-frequent patterns, regional masks, and surrogate gradients) applicable to certain areas of an image to generate the query-efficient adversarial inputs. Chen *et al.* [22] proposed a black-box adversarial attack for speaker recognition, named FAKEBOB, to craft adversarial samples. Specifically, they formulated the adversarial sample generation as an optimization problem, incorporated with the confidence of adversarial samples and maximal distortion to balance between the strength and imperceptibility of adversarial voices. However, little research has been devoted to black-box attacks on multivariate time series DNN models, which impost more formidable challenges as explained previously. For example, the random full gradient estimator method is not sufficiently efficient for higher dimensional data as indicated in [23].

III. Background

The existing black-box adversarial attacks aim to generate an adversarial input, which has the least distance from the original input in terms of L_2 distance metric (in order to avoid being detected), while maintaining a high attack success rate [14]. Formally, given an input $\vec{x}_0$, and the adversarial input $\vec{x}$, the objective function is:

$$\min_{\vec{x}} \ \left(\|\vec{x}-\vec{x}_0\|^2 + cf(\vec{x},t_0)\right), \tag{1}$$

where $c > 0$ is a regularization parameter, $f(\vec{x},t_0)$ measures the level of unsuccessful adversarial attacks, which is defined as follows:

$$f(\vec{x},t_0) = \max([F(\vec{x})]_{t_0} - \max_{i\neq t_0}[F(\vec{x})]_i, -\tau), \tag{2}$$

where t_0 is the ground truth label for $\vec{x}_0$; $F(\vec{x})$ is the model output represented by a vector of probabilities for each class; $[F(\vec{x})]_i$ denotes the i-th coordinate in vector $F(\vec{x})$; $[F(\vec{x})]_{t_0}$ denotes the probability of ground truth label t_0 and τ is the threshold for successful attack. If $f(\vec{x},t_0) \geq 0$, then the attack is unsuccessful since the predicted label is still t_0. Our goal is to assign the maximum probability value on target class label rather than on t_0 by the DNN model (i.e., targeted attack). If $f(\vec{x},t_0) \geq -\tau$, even though the predicted label is different from the true label, the attack is still considered unsuccessful since the probability difference between the ground truth label and predicted label does not reach the threshold τ. The objective in Equation (1) is the weighted sum of the L_2 distance and the level of unsuccessful attacks.

In black-box attacks, the gradient information of the DNN model is unknown and the conventional gradient estimation methods (e.g., gradient descent) cannot be applied to solve Equation (1). As a result, existing approaches utilize non-gradient optimization techniques (e.g., the finite difference method, zeroth optimization, and random gradient estimation) to solve Equation (1). As an example, ZOO [14] uses the finite difference method [24] to estimate the gradient for the i-th input coordinate as follows:

$$\hat{\vec{g}}_i = \frac{\partial f(\vec{x},t_0)}{\partial x_i} = \frac{f(\vec{x}+h\vec{e}_i) - f(\vec{x}-h\vec{e}_i)}{2h}, \tag{3}$$

where x_i is the i-th coordinate of $\vec{x}$; h is a very small number (e.g., 0.0001 as in [14]); $\vec{e}_i$ is an all-zero vector except an one at the i-th coordinate.

ZOO uses ADAM based zeroth-order optimizer to iteratively update each coordinate gradually in order to reduce the number of queries. ADAM's update rule for gradient estimation helps to reduce the number of queries comparing to other stochastic coordinate descent algorithms. Specifically, at each iteration, it randomly selects a coordinate i to update, and computes δ_i^* based on $\hat{\vec{g}}_i$ using its update rule, where δ_i^* is the best update in perturbation vector $\boldsymbol{\delta}$ satisfying the objective function (Equation (1)) with previous gradients. If the predicted class label is the target class label, Equation (2) is satisfied, and $\delta_i^*\vec{e}_i$ is under the constraint that the L_2 distance (i.e., $\sqrt{\sum_{i=1}^{n}(\delta_i^*\vec{e}_i)^2}$) is less than a predefined threshold, the attack is successful. Then, $(\vec{x}+\delta_i^*\vec{e}_i)$ becomes the final adversarial input. That is, in $\vec{x}$, $x_i = x_i + \delta_i^*$. The perturbation vector $\boldsymbol{\delta}$'s i-th element is added by δ_i^* to be the final perturbation vector.

To get δ_i^*, the update rule of ADAM optimizer is used, where δ_i^* is generated at each time from the newly generated gradient estimation. More specifically, $\hat{\vec{g}}_i$ is first computed via Equation (3). Then, given hyperparameters β_1, β_2, η, and ϵ, it computes [14]:

$$T_i = T_i + 1, \tag{4}$$

$$M_i = \beta_1 M_i + (1-\beta_1)\hat{\vec{g}}_i, \tag{5}$$

$$v_i = \beta_2 v_i + (1-\beta_2)\hat{\vec{g}}_i, \tag{6}$$

$$\hat{M}_i = M_i/(1-\beta_1^{T_i}), \tag{7}$$

$$\hat{v}_i = v_i/(1-\beta_2^{T_i}), \tag{8}$$

$$\delta^* = -\eta\frac{\hat{M}_i}{\sqrt{\hat{v}_i}+\epsilon} \tag{9}$$

where T_i is the total number of iterations for coordinate i, $\hat{M}_i$ and $\hat{v}_i$ are the running averages of the first and second moment estimates for the gradients, and η is the learning rate. The values of M_i, v_i, and T_i are set to zero at the beginning of the stochastic coordinate descent. The above procedure is continued until $\vec{x}$ is converged or a predefined max iteration L has been reached. In ZOO, each gradient calculation (in ADAM optimizer) is for an image coordinate of two-dimensional image data. If we extend it to m-variate time-series data, there will be many gradient calculations, which significantly increases the queries and hence reduces query-efficiency.

IV. System Design

A. *Ada-ZOO: Self-adaptive Step based Attack*

In the existing ZOO attack, the estimation of the gradient at each iteration step is given in Equation (3) with an error bound $O(h^2)$. ZOO [14] uses a fixed value of step size $h = 0.0001$ for the gradient estimation. In order to show the effect of h on the attack success rate, we conducted an experiment. First, let's introduce the datasets and DNN models we used in this paper.

TABLE I
DIFFERENT COMBINATIONS OF DATASETS-MODELS.

Dataset and model pair	**Label**
(Arabic_voice, MLSTM-FCN)	1
(Arabic_voice, MALSTM-FCN)	2
(EEG, MLSTM-FCN)	3
(EEG, MALSTM)	4
(HAR, MLSTM-FCN)	5
(HAR, MALSTM-FCN)	6
(Arabic_voice, FCN)	7
(EEG, FCN)	8
(HAR, FCN)	9
(Driving data, MALSTM-FCN)	10

TABLE II
TOTAL NUMBER OF ATTACKS IN EACH DATASET.

Datasets	Dataset percentage for training	Dataset percentage for testing	Total number of attacks
Arabic_voice	75%	25%	11000
EEG	50%	50%	126
HAR	71%	29%	14735
Driving dataset	70%	30%	4758

Datasets. We used four different multivariate time-series datasets: arabic_voice [25], electroencephalography (EEG) [26], human activity recognition (HAR) [26], and driving datasets [17]. All the datasets are multivariate time-series and we can define a time-series dataset as a tensor of shape (s, n, m), where s is the number of data samples in the dataset, n is the maximum number of time stamps in a sample and m is the number of variates per time stamp.

We used three different DNN models for multivariate time-series datasets: *Multivariate Long Short Term Memory Fully Convolutional Network (MLSTM-FCN) [5]*, MALSTM-FCN model [5], and *Fully Connected Network (FCN) [27]*. Table I shows the different combinations of (dataset, DNN model) pairs and their labels. To find the optimal solution, we used the ADAM based zeroth-order optimizer with the hyperparameter settings as in [14]. Also, we set $\tau = 0$ as in [14] unless otherwise specified. We generate one adversarial attack for each input data sample.

Figure 1 shows the success rates of different h values for the ten (dataset, model) combinations used in this experiment. The x-axis represents different h values (i.e., 0.0001, 2, 4, 6, 8, and 10) and y-axis represents success rates corresponding to h values (using Equation(3)). For each dataset, we used each sample from the testing dataset to generate a targeted attack using a fixed value of h. We took each of the output classes (except the ground truth label) as a targeted attack. We used the same dataset percentages as in [5], [27] for training and testing the DNN models. The percentages and the number of attacks are shown in Table II. Finally, we calculate the overall success rate using the total number of successful attacks over all attacks attempted from the testing dataset. We can see that the success rates usually increase when h values are higher. There is a sudden change of success rate and it does not change drastically afterwards.

From the evaluation, we observe that a small value of step size h leads to a low success rate because $\hat{\vec{g}}_i$ can be zero in many cases due to the uncouthness nature of the ReLu activation function in the DNN model [28]. That is, because of a certain h value, the output of the ReLu activation function may not be smooth, which results in a zero-value estimation for the gradient.

To address this issue, we propose an adaptive strategy called Ada-ZOO to dynamically update h during the process to generate an adversarial input. The pseudo-code of Ada-ZOO is shown in Algorithm 1. Due to the unsmooth nature of the ReLu activation function, the gradient estimation can be zero. A large enough value of h can ensure a non-zero estimation for the gradient. Therefore, we assign a large initial value $h = h_0$ at the beginning of each iteration (step 1). Then, it gradually updates the value of h (steps 2–19). At each iteration, the algorithm calculates the new gradient $\hat{\vec{g}}_i$ (step 3). Next, the algorithm checks if there is a zero-value estimation of gradient $\hat{\vec{g}}_i$ and the value of h is greater than 0.0001 (step 4). If yes, it updates the step size h (steps 5–8). For example, we can set the h values to 6 initially and then gradually update it in the sequence of (6, 4, 2, 0.0001). Next, the algorithm calculates the current perturbation of the i-th coordinate δ_i^* (step 10), and then updates the adversarial input $\vec{x}$ using the new perturbation $\delta_i^* \vec{e}_i$ (step 11). The algorithm then checks if the attack is successful using the new adversarial input $\vec{x}$ (steps 13–18). If so, the algorithm terminates (step 14). Otherwise, it increases the iteration count (step 17) and repeats the above procedures until the attack is successful for a given input sample or it reaches the maximum iteration count.

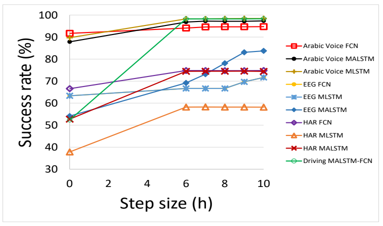

Fig. 1. Success rates for different h values.

B. *ZOO-L_0: Additional L_0 Distance based Attack*

Black-box adversarial attacks mainly focus on searching adversarial perturbations under the constraints of L_2 distance. In the L_2 based black-box adversarial attacks, the upper bound of L_2 for the perturbation (i.e., $\|\boldsymbol{\delta}\|_2$) is directly used as an optimization constraint.

For any neural network models handling time-series data as input, to be more imperceptible, an black-box attack should have a sparse structure in the perturbation, where only a few input coordinates are modified in order to generate a more imperceptible adversarial input. This is because of higher data dimension (than image datasets), the multivariate time-series datasets offer more variate correlations to be analyzed. Therefore, we additionally use the L_0 distance for adversarial perturbation that can measure the sparsity with the goal to achieve lower L_0 distance or higher sparsity for the sake of imperceptibility. However, the L_0 distance metric is non-differentiable and

Algorithm 1: Pseudocode for Ada-ZOO.

```
Input: $(\vec{x}, y)$: (input, target label) pair,
       Model: target black-box model,
       $h_0$: Initial step size,
       MAX_ITE: maximum number of iterations.
1  $h = h_0$
2  while $iters$ < $MAX\_ITE$ do
3      Calculate $\hat{g}_i$ using Equation (3)
4      if $|\hat{g}_i|$ == $0$ and $h > 0.0001$ then
5          $h = h - 2$
6          if $h \leq 0$ then
7              $h$ = 0.0001
8          end
9      end
10     ADAM optimizer outputs $\delta_i^*$
11     $\vec{x} = \vec{x} + \delta_i^* \vec{e}_i$
12     $success$ = $Model(\vec{x}$,MAX_ITE)
13     if $success$ then
14         break
15     end
16     else
17         $iters$ =$iters$ + 1
18     end
19 end
```

therefore is not suitable for the standard gradient descent algorithms, which require to learning the gradient at each step of the optimization procedures.

Though an existing black-box attack [29] uses an auxiliary parameter to realize the L_0 distance attack, it caused the drastic reduction of attack success rate [30]. There are also some existing white-box attacks focusing on L_0 sparse adversaries such as Jacobian Saliency Map Attack (JSMA) [31] and CW-L_0 [9]. Basically, the gradient is calculated based on the known loss function and the internal layers of the DNN model. Therefore, this approach is not suitable for black-box adversarial attacks in which the internal layers of the DNN model are unknown. These works motivate us to design the L_0 based black-box adversarial attack for time-series data.

In order to create sparse perturbation measured by L_0 distance, we propose ZOO-L_0, an iterative algorithm based on Ada-ZOO. ZOO-L_0 finds L_0 adversarial perturbation starting from a successful L_2 perturbation. Here, we use previously found L_2 perturbations in Ada-ZOO (as in Section IV-A). The pseudo-code of the ZOO-L_0 procedure is shown in Algorithm 2. At the beginning, we use Ada-ZOO to find a successful L_2 adversarial perturbation $\boldsymbol{\delta}$ for a given sample (step 1). We calculate the current $\|\boldsymbol{\delta}\|_2$ and use it as the L_2 distance threshold (denoted by $C\|\boldsymbol{\delta}_0\|_2$ and C is a constant) for finding a new perturbation. Afterwards, at each iteration, we identify a coordinate i with the lowest perturbation δ_i in the L_2 adversarial perturbation $\boldsymbol{\delta}$, which does not have much effect on the classifier output (step 5). Then, we assign that coordinate to zero (step 6). Next, to minimize the objective value calculated by Equation (1) as much as possible, we use the Ada-ZOO optimizer to update the remaining non-zero coordinates of $\boldsymbol{\delta}$ while ensuring that the updated $\boldsymbol{\delta}$ is successful and its $\|\boldsymbol{\delta}\|_2 \leq C\|\boldsymbol{\delta}_0\|_2$ (i.e., L_2 norm of adversarial perturbation $\boldsymbol{\delta}$ does not exceed threshold $C\|\boldsymbol{\delta}_0\|_2$ (steps 8–12). The iteration is repeated until $\boldsymbol{\delta}$ fails to remain as an adversarial perturbation (i.e. there is no way to modify the non-zero coordinates of $\boldsymbol{\delta}$ to generate the targeted attack after the maximum number of iterations is reach) or $\|\boldsymbol{\delta}\|_2$ exceeds the threshold. By this perturbation coordination elimination process, we are able to identify a minimal (i.e., not minimum) subset of coordinates (i.e., satisfying Equation (1)) that can be modified to generate an adversarial input. A minimal subset of coordinates represents higher sparsity of the adversarial perturbation and more imperceptibility.

Algorithm 2: Pseudocode for ZOO-L_0.

```
Input: (X, Y): (input, target label) pair,
       Init_Adv: initial value of adversarial perturbation,
       Model: target black-box model,
       Find_L2_Adv: A base $L_2$ attack algorithm,
       C: $L_2$ explosion constant,
       MAX_ITE: maximum number of iterations.
Output: Adversarial perturbation $\boldsymbol{\delta}$ if successful.
1  $\boldsymbol{\delta}_0$, iters := $Find\_L2\_Adv$(X, Y, Model, $Init\_Adv$)
2  $\boldsymbol{\delta}$ := $\boldsymbol{\delta}_0$
3  Mask := Set of non-zero coordinates in $\boldsymbol{\delta}$
4  while $Mask.size > 1$ and $iters < MAX\_ITE$ do
5      $shrink\_index$ := $\arg\min_i \{\delta_i | \delta_i \neq 0\}$
6      Mask.remove($shrink\_index$)
7      $\boldsymbol{\delta}(shrink\_index)$ = 0
8      $\boldsymbol{\delta}^{new}, iters\_new$ = $Find\_L2\_Adv$(X, Y, Model, $\boldsymbol{\delta}$,
       Mask)
9      if $\boldsymbol{\delta}^{new}$ is successful and $\|\boldsymbol{\delta}\|_2 \leq C\|\boldsymbol{\delta}_0\|_2$ then
10         $\boldsymbol{\delta} = \boldsymbol{\delta}^{new}$
11         iters += $iters\_new$
12     end
13     else
14         return $\boldsymbol{\delta}$, iters
15     end
16 end
17 return $\boldsymbol{\delta}$, iters
```

C. ZOO-MWU: Coordinate Importance Oriented and MWU-based Attack

The benefit of using the ADAM based zeroth-order optimizer (as a stochastic coordinate descent algorithm) is to choose explicitly which coordinates to update. However, estimating gradient for each coordinate in each iteration is expensive in the black-box attack scenario. Therefore, we propose to selectively update certain chosen coordinates which are important to the DNN output. To find these important coordinates, we find the importance of all coordinates with respect to a given target class label, and then choose the most important coordinates.

In the process of creating a black-box attack as presented in Section III, the interaction between the attacker and the target model can be considered as an online learning environment, where the attacker takes actions (i.e., adding random perturbation to a benign input) and then receives feedback (i.e., get output label from the DNN model to see whether the attack is successful or not) from the environment (i.e., DNN model) at each attack attempt.

The attacker is trying his/her best to gradually gather information from the environment to gain a maximum accumulated utility in the whole process. Here, the utility of an attacker is the successful probability to fool the target model. In this case, we can view each input coordinate as a bandit where adding a small amounts of perturbation on an input coordinate can be regarded as a trial to pull this bandit, and then the feedback is collected by observing whether that added perturbation amounts result in a successful attack or not. This process can be considered as a Multiplicative Weights Update (MWU) [19] decision making process. The MWU method is an algorithmic technique most commonly used for decision making. The decision maker needs to iteratively decide on an expert whose advice to follow. The method assigns identical initial weights to the experts, and updates these weights multiplicatively and iteratively according to the feedback of how well an expert performed. Therefore, we can use MWU to guide choosing the coordinates with higher importance to add perturbation. In this MWU, coordinates are experts and the feedback is the importance values of the coordinates. We use the accumulated moving average of previous gradients for coordinate i in each signal, v_i, in ADAM (Equation (6)) to represent the importance weight of coordinate i in the signal. Recall that at each iteration for a selected coordinate i during this process, ADAM calculates v_i, and calculates its perturbation δ_i^*; higher v_i leads to lower δ_i^* and vice versa. Therefore, we can use v_i to represent the importance of coordinate i of the signal and use the MWU method to find the coordinates with higher importance to add perturbation in the signal.

Specifically, to apply the MWU method, we need to assign an initial weight to each input coordinate according to the MWU method and update the weights gradually during the searching process based on their individual contributions to a successful attack. From the searching process, we collect the learned weights updated by the ADAM optimizer to calculate the probability for all the coordinates in each signal as the importance calculation strategy for further attacks. We propose ZOO-MWU for a single attack and for a batch of attacks as follows.

ZOO-MWU for a single attack. At the t-th iteration of the ADAM optimizer, the algorithm calculates weight v_i for coordinate i and $\vec{v}_t = \{v_0, v_1, ...\}$ is a weight distribution over all the coordinates given a target class label. For a single attack, we utilize the accumulated moving average of previous gradients v_i for each coordinate i. An input coordinate with v_i in $\vec{v}_t$ that has a large absolute value means that the coordinate plays a major role in manipulating the output probability of the DNN model for the iteration steps in ADAM optimizer so far. We propose to apply the softmax activation function on v_i to convert it to a probability as follows:

$$p_i = \frac{1}{1+\exp(\alpha v_i)}, \tag{10}$$

where α is a hyperparameter. The benefit of this conversion is that we can use parameter α to control the shape of the probability and also balance the trade-off between exploitation and exploration. Larger α means that we believe the previous v_i tells us a lot about how to find a perturbation and we want to exploit it more; and a smaller α means we do not want to use too much information in v_i and tend to sample i from a nearly uniform distribution. At each iteration, we calculate the probability p_i and choose the coordinate with the highest probability p_i value (i.e., highest-weight coordinate) to calculate the gradient. We name this algorithm as ZOO-MWU for a batch of attacks. Algorithm 3 shows the overall procedures of the ZOO-MWU attack for a single attack. To calculate p_i, the algorithm calculates the coordinate-wise $\vec{v}_t$ and perturbation $\boldsymbol{\delta}$ (steps 4 – 7). At each iteration, it normalizes the result to a probability vector P based on Equation (10) (step 8).

Algorithm 3: ZOO-MWU for a single attack

```
Input: (X, Y): (input, target label),
       $\boldsymbol{\delta_0}$: initial value of adversarial perturbation,
       Model: target black-box model,
       MAX_ITE: maximum number of iterations,
       $\alpha$: parameters for samping.
Output: Adversarial perturbation $\boldsymbol{\delta}$ for a successful attack.
1  $M = 0, v = 0, t = 0, \boldsymbol{\delta} = \boldsymbol{\delta}_0$,
2  $P = \frac{1}{Z}$ ones(size($\boldsymbol{\delta}$))
3  while not successful and $t <$ MAX_ITE do
4      Pick a coordinate $i$ with the highest $p_i$ from $P$
5      Estimate $\hat{g}_i$ using Equation (3)
6      $t = t + 1$
7      ADAM optimizer outputs $\vec{v}_t$ and $\boldsymbol{\delta}$.
8      $P = \frac{1}{1+\exp(\alpha \vec{v}_t)}$ (Equ. (10))
9  end
10 return $\boldsymbol{\delta}$, t
```

ZOO-MWU for a batch of attacks. When attacking a batch of inputs, we can further leverage the accumulated gradient information across different samples. The overall idea is that the adversarial perturbations for different given inputs are likely to be correlated as the adversarial perturbations are generated from the same target DNN model.

Based on this idea, we first apply the ZOO-MWU algorithm (as in Algorithm 3) to generate a successful attack for each individual input sequentially. For each attack, we exploit the successful adversarial perturbations found for the previous attacks. The basic reasoning is quite straightforward: the vulnerability of each coordinate should follow a prior probability for the time-series DNN models as we observed from our data analysis in the above. Therefore, the statistical probability from non-zero entries of the successful perturbations should be a useful indications for further attacks. Specifically, we consider the previously successful adversarial perturbations to calculate the prior perturbation probability and use the prior probability to guide the importance calculation for the future attacks. To do so, we maintain a global weight W over each input coordinate. Each time when we find a successful attack $\boldsymbol{\delta}$

for a new input, we simply update the global weight W by an amount proportional to $|\boldsymbol{\delta}|^\gamma$ (i.e., $\frac{|\boldsymbol{\delta}|^\gamma}{||\boldsymbol{\delta}||_\gamma^\gamma}$), where γ is a hyperparameter, $|\boldsymbol{\delta}|^\gamma = [|\delta_1|^\gamma, ..., |\delta_n|^\gamma]$, and $||\boldsymbol{\delta}||_\gamma^\gamma = \sum_{i=1}^{n} |\delta_i|^\gamma$. Let us consider an example in which $\gamma = 2$ and $\boldsymbol{\delta} = [1, 2, 3; 1, 1, 2]$. Then, $\frac{|\boldsymbol{\delta}|^\gamma}{||\boldsymbol{\delta}||_\gamma^\gamma} = \frac{[1, 4, 9; 1, 1, 4]}{(1+4+9+1+1+4)} = [0.05, 0.2, 0.45; 0.05, 0.05, 0.2]$. Therefore, W actually stores the averaged perturbation for all the successful attacks controlled by γ. If $\gamma = 0$, W is updated by a uniform distribution and we do not use the information in $\boldsymbol{\delta}$ at all; if γ is a very large positive number, W is updated by a tensor like $[0, 0, ..., 0, 1, 0, ..., 0]$. Q is calculated as the probability obtained by normalizing W. For example, if W is a 2×3 matrix $[1, 2, 1; 3, 3, 0]$, then $Q = \frac{W}{(1+2+1+3+3+0)} = [0.1, 0.2, 0.1; 0.3, 0.3, 0]$. In the above procedure, we also maintain a probability vector P_j for each sample or attack j. The probability vector P_j is obtained by the updating rule specified in Algorithm 3. Therefore, the final probability is assigned as the combination of prior probability and global weight, i.e. softmax$(P_j \times Q)$. Here, Q is a prior global probability for choosing a high-weight coordinate and $P_j \times Q$ is the posterior probability for choosing a high-weight coordinate. We use $P_j(i)$ and $Q(i)$ to denote the i-th coordinate of P_j and Q. Therefore, when $P_j(i)$ and $Q(i)$ for coordinate i are both large, the probability of sampling high-weight coordinate i is high.

Algorithm 4: ZOO-MWU for a batch of attacks

```
Input: $(X_i, Y_i)_{i=1}^s$: (input, target label) pairs for $s$ samples
    in a dataset,
      $\boldsymbol{\delta}_0$: initial value of adversarial perturbation,
      Model: target black-box model,
      MAX_ITE: maximum number of iterations,
      $Z$: length of $\boldsymbol{\delta}$        $\alpha, \gamma$: parameters for
  sampling.
Output: Adversarial perturbations $\boldsymbol{\Delta} = \{\boldsymbol{\delta}_i\}_{i=1}^s$ for
    successful attacks.
1  $W = \frac{1}{Z}$ zeros(size($\boldsymbol{\delta}$))
2  for j=1:size(Y) do
3      $\vec{v}_t = 0, t = 0, \boldsymbol{\delta} = \boldsymbol{\delta}_0$,
4      $P_j = \frac{1}{Z}$ ones(size($\boldsymbol{\delta}$))
5      while not successful and $t <$ MAX_ITE do
6          Pick a coordinate $i$ with the highest value from
           probability $P_j \times Q$
7          Estimate $\hat{g}_i$ using Equation (3)
8          $t = t + 1$
9          ADAM optimizer outputs $\vec{v}_t$ and $\boldsymbol{\delta}$
10         $P_j = \frac{1}{1+\exp{(\alpha \vec{v}_t)}}$
11     end
12     if success then
13         $W = W + \frac{|\boldsymbol{\delta}|^\gamma}{||\boldsymbol{\delta}||_\gamma^\gamma}$
14         $Q = norm(W)$
15         $\Delta$.add($\boldsymbol{\delta}$)
16     end
17 end
18 return $\Delta$
```

The pseudo-code of ZOO-MWU for a batch of attacks is presented in Algorithm 4. The output is Δ which contains a set of adversarial perturbations for the attacks. The inner loop is exactly the Algorithm 3 (steps 5 – 16). The only difference is that when choosing a high-weight coordinate i, we use the coordinate-wise product of P_j and Q (instead of P_j) (step 6). The outer loop maintains another discrete weight W (steps 2 – 17), which will only be updated when a successful attack is found (step 13). If so, W is updated by adding the normalized $\boldsymbol{\delta}$. Q is calculated as the probability obtained by normalizing W (step 14). Δ.add($\boldsymbol{\delta}$) means adding the element $\boldsymbol{\delta}$ into the set Δ and $\boldsymbol{\delta}$ is the newly found adversarial perturbation for a successful attack (step 15).

V. PERFORMANCE EVALUATION

A. Experimental Settings

1) Metrics: For each attack algorithm, we evaluate five different metrics in targeted attack settings: average attack success rate, average L_2 percentage distance, average L_0 percentage distance, average number of iterations, and average number of queries for successful attacks.

- The average attack success rate is calculated by: $\frac{1}{N}\sum_{i=1}^{N} \mathcal{S}^{(i)}$, where N is the total number of input samples and $\mathcal{S}$ is the binary indicator value representing if the attack is successful or not.
- The average L_p distance of different attacks is calculated by: $\frac{1}{N}\left(\sum_{i=1}^{N} |\vec{x} - \vec{x}_0|^p\right)^{\frac{1}{p}}$, where p can be any positive number. In our evaluations, we considered L_0 and L_2 distance of different successful attacks.
- The average number of queries of each attack is calculated by: $\frac{1}{N}\sum_{i=1}^{N} \mathcal{Q}^{(i)}$, where $\mathcal{Q}^{(i)}$ represents the number of queries to the DNN model for the input sample i.

For each metric, we measured the improvement by $(\widetilde{x} - x)/x$, where x and $\widetilde{x}$ are the performance measures of the existing attack and proposed attack, respectively.

2) Comparison Methods: We implemented our attacks including Ada-ZOO, ZOO-L_0, and ZOO-MWU. We also implemented a method that merges the three methods and call it ZOO-L_0-MWU. Further, we used the ZOO [14] and AutoZoom [16] as comparison methods.

3) Hyperparameter Setting: Table III shows the hyperparameters used in the attack algorithms.

TABLE III
SHARED HYPERPARAMETERS.

Hyperparameter	Symbol	Value
Unsuccessful attack threshold	τ	0.0
Weight of L2 distance	c	10000
Learning rate	η	0.025
Maximum query threshold		2000
Max iterations	L	500
Init. finite diff. step	h	1×10^{-1}
ADAM parameter 1	β_1	0.9
ADAM parameter 2	β_2	0.999
ADAM parameter 3	ϵ	1×10^{-8}
L_2 explosion constant	C	1.1
α	Weight prob. param. 1	-0.1
γ	Weight prob. param. 2	2.0

B. Experimental Evaluations

1) Attack Success Rate: Figure 2 shows the targeted attack success rates for different methods. Overall the aver-

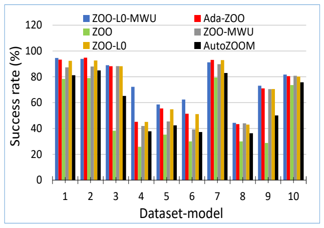

Fig. 2. Success rates.

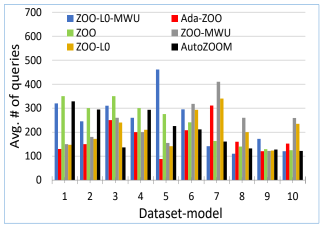

Fig. 3. Average number of queries.

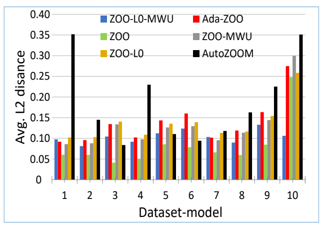

Fig. 4. Average L_2 distance.

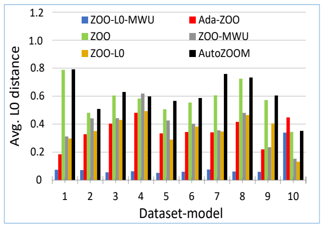

Fig. 5. Average L_0 distance.

age success rates for six different attacks (ZOO, Ada-ZOO, ZOO-L_0, ZOO-MWU, ZOO-L_0-MWU, and AutoZOOM) are 50%, 72%, 71%, 74%, 76%, and 60%, respectively. The improvements of four proposed attacks (Ada-ZOO, ZOO-L_0, ZOO-MWU, and ZOO-L_0-MWU) over the ZOO attack are: 42%, 41%, 43%, and 51%, respectively. The improvements of them over the AutoZOOM attack are: 18.16%, 17.73%, 19.52%, and 26.03%, respectively. Overall, the attack success rates of different attacks follow: ZOO$<$AutoZOOM$<$ Ada-ZOO$\approx$ZOO-$L_0$$<$ ZOO-MWU $<$ ZOO-L_0-MWU.We can see that attack success rate of ZOO is around 75% for first three dataset-model pairs, but it degrades drastically for the last six dataset-model pairs. This means that the success rate may vary for different DNN models or different datasets. AutoZOOM increases ZOO's success rate since AutoZOOM generates the gradient for all coordinates each time and reduces the search space.All of our methods achieve higher success rate than ZOO and AutoZOOM. Due to the adaptive step size h, Ada-ZOO can more quickly find the successful adversarial input. The attack success rate of ZOO-L_0 is similar as Ada-ZOO because ZOO-L_0 additionally reduces the L_0 distance after it uses Ada-ZOO to find a successful attack. ZOO-MWU chooses more influential input coordinates to add perturbation to more efficiently create an adversarial attack. ZOO-L_0-MWU generates the highest success rate by combining all of these proposed methods.

2) The Number of Queries: Figure 3 shows the average number of queries for comparison attacks. Overall, the average number of queries follow: ZOO-L_0-MWU$<$Ada-ZOO$<$ZOO-$L_0$$<$ ZOO-MWU$<$AutoZOOM$<$ZOO. The improvements of four proposed attacks (Ada-ZOO, ZOO-L_0, ZOO-MWU, and ZOO-L_0-MWU) over ZOO are: 25%, 15%, 11%, and 29%, respectively. Their improvements over AutoZOOM attack are 18%, 13%, 11%, and 25%, respectively. ZOO and AutoZOOM require a higher number of queries than all our methods. ZOO has many zero-gradient estimations due to a small step size since it aims to reduce the amount of perturbation in general. AutoZOOM estimates the average gradient for all coordinates at once and then enlarges the perturbation space using the decoder model. Due to the adaptive step size h, Ada-ZOO needs fewer zero-gradient estimations to find a successful adversarial attack, so it reduces the number of queries to generate a successful attack than ZOO. ZOO-L_0 is based on Ada-ZOO, and it needs more queries since it re-runs Ada-ZOO after removing the lowest perturbation in a coordinate. ZOO-MWU focuses on high-weight coordinates to add perturbations to generate successful adversarial attacks with few queries than ZOO and AutoZOOM. ZOO-MWU still uses the small step size as in ZOO, so its numbers of queries are higher than those of ZOO-L_0. ZOO-L_0-MWU requires the lowest numbers of queries among all attacks by incorporating ZOO-MWU and Ada-ZOO.

3) L_2 Distance: Figure 4 shows the average L_2 distance for different comparison attacks. The overall result follows: ZOO$<$ZOO-L_0-MWU$\approx$ZOO-MWU$\approx$ZOO-$L_0$$\approx$Ada-ZOO$<$AutoZOOM. The overall improvements of the four proposed attacks (Ada-ZOO, ZOO-L_0, ZOO-MWU, and ZOO-L_0-MWU) over the existing AutoZOOM attack are: 80%, 80%, 81%, and 84%, respectively. Note that the results vary for different dataset-model pairs due to their different features. The L_2 distance of perturbation added by ZOO is the smallest because this is the goal of ZOO. The L_2 distance of perturbation added by AutoZOOM is dramatically larger than others. This is because that it focuses on the attack success rate by enlarging the perturbation space using the decoder model and also calculates the gradient for all coordinate at a time rather than for one coordinate at each iteration, which are not effective for multivariate time-series data [30].Our methods generate a slightly higher L_2 distance than ZOO due to higher step size in gradient calculation in Ada-ZOO or focusing on historically high-weight coordinates in ZOO-MWU, which help increase success rate. Recall that ZOO-L_0 is based on Ada-ZOO, and further reduce its L_0 distance. Therefore, it does not significantly change the L_2 distance from Ada-ZOO's. Combining the three methods together, ZOO-L_0-MWU does not change the L_2 distance much compared to other methods.

4) L_0 Distance: Figure 5 shows the average L_0 distance for the six different attacks. The result according to the L_0 distance is as follows ZOO-L_0-MWU$<$ZOO-$L_0$$<$ZOO-MWU$<$Ada-ZOO$<ZOO<$AutoZOOM. The improvements of three proposed attacks (Ada-ZOO, ZOO-L_0, ZOO-MWU, and ZOO-L_0-MWU) over ZOO are: 29%, 38%, 33%, and 64%, respectively. Their improvements over AutoZOOM are: 35%, 41%, 39%, and 65%, respectively. We see that AutoZoom generates larger L_0 distance of perturbation than other attacks due to the same reason explained above. Since ZOO-L_0 particularly aims to reduce L_0 distance by removing small perturbations of some coordinates and ZOO-MWU only focuses on high-weight coordinates to add perturbations, they reduce the L_0 distance of ZOO and ZOO-L_0 has higher reduction than ZOO-MWU. Ada-ZOO increases the step size of ZOO in gradient calculation and does not focus on reducing L_0 distance. By combining ZOO-L_0 and ZOO-MWU, ZOO-L_0-MWU achieves the lowest L_0 distance though it also includes Ada-ZOO.

VI. Conclusion

Achieving high query-efficiency and imperceptibility is a challenge in generating black-box adversarial attacks on multivariate time-series DNN models. In this paper, we proposed three different black-box adversarial attacks for the multivariate time-series DNN models. The three proposed attacks are: a self-adaptive technique to improve the success rate; an L_0 distance-based attack to improve imperceptibility; a coordinate importance oriented method based on MWU, which can exploit the time-series structure and improve its query-efficiency and imperceptibility. With these proposed attacks, we found the trade-offs between attack success rate, perturbation amount (i.e., L_2 and L_0 distances), and the number of queries for a given attack task. From our extensive evaluations using ten different combinations of datasets and DNN models, we found that proposed attacks are able to achieve up to 51% higher success rates with 25% fewer queries over the existing black-box attacks. In the future, we will also study and design defense mechanisms to tackle the proposed attacks for multivariate time-series DNN models.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1827674, CCF-1822965, FHWA grant 693JJ31950016, Microsoft Research Faculty Fellowship 8300751, and the Commonwealth Cyber Initiative (CCI), an investment in the advancement of cyber research, innovation and workforce development. For more information about CCI, visit cyberinitiative.org.

References

[1] Yao Qin, Dongjin Song, Haifeng Chen, Wei Cheng, Guofei Jiang, and Garrison Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[2] Wenjun Jiang, Chenglin Miao, Fenglong Ma, Shuochao Yao, Yaqing Wang, Ye Yuan, Hongfei Xue, Chen Song, Xin Ma, and Dimitrios Koutsonikolas. Towards environment independent device free human activity recognition. In *Proc. of MobiCom*, 2018.

[3] Xiangyu Xu, Jiadi Yu, Yingying Chen, Yanmin Zhu, Linghe Kong, and Minglu Li. Breathlistener: Fine-grained breathing monitoring in driving environments utilizing acoustic signals. In *Proc. of MobiSys*, 2019.

[4] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proc. of CVPR*, 2016.

[5] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Samuel Harford. Multivariate lstm-fcns for time series classification. *Neural Networks*, 116:237–245, 2019.

[6] Xiaoyong Yuan, Pan He, Qile Zhu, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *IEEE Trans. Neural Netw Learn Syst*, 2019.

[7] Chawin Sitawarin, Arjun Nitin Bhagoji, Arsalan Mosenia, Mung Chiang, and Prateek Mittal. Darts: Deceiving autonomous cars with toxic signs. *arXiv preprint arXiv:1802.06430*, 2018.

[8] Mengying Sun, Fengyi Tang, Jinfeng Yi, Fei Wang, and Jiayu Zhou. Identify susceptible locations in medical records via adversarial attacks on deep predictive models. In *Proc. of SigKDD*, 2018.

[9] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Proc. of S&P*, 2017.

[10] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Proc. of EuroS&P*, 2016.

[11] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proc. of SOSP*, 2017.

[12] Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

[13] Satya Narayan Shukla, Anit Kumar Sahu, Devin Willmott, and J Zico Kolter. Black-box adversarial attacks with bayesian optimization. *arXiv preprint arXiv:1909.13857*, 2019.

[14] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proc. of AISW*, 2017.

[15] Thomas Brunner, Frederik Diehl, Michael Truong Le, and Alois Knoll. Guessing smart: Biased sampling for efficient black-box adversarial attacks. In *Proc. of ICCV*, 2019.

[16] Chun-Chen Tu, Paishun Ting, Pin-Yu Chen, Sijia Liu, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Shin-Ming Cheng. Autozoom: Autoencoder-based zeroth order optimization method for attacking black-box neural networks. In *Proc. of AAAI*, volume 33, 2019.

[17] Byung Il Kwak, JiYoung Woo, and Huy Kang Kim. Know your master: Driver profiling-based anti-theft method. In *Proc. of PST*, 2016.

[18] Arka Ghosh, Sankha Subhra Mullick, Shounak Dutta, Swagatam Das, Rammohan Mallipeddi, and Asit Kr Das. One sparse perturbation to fool them all, almost always! *arXiv preprint arXiv:2004.13002*, 2020.

[19] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1), 2012.

[20] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *Proc. of SPW*, 2018.

[21] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *Proc. of ICLR*, 2018.

[22] Guangke Chen, Sen Chen, Lingling Fan, Xiaoning Du, Zhe Zhao, Fu Song, and Yang Liu. Who is real bob? adversarial attacks on speaker recognition systems. *Proc. of S&P*, 2021.

[23] Krishnakumar Balasubramanian and Saeed Ghadimi. Zeroth-order (non)-convex stochastic optimization via conditional gradient and gradient updates. In *Proc. of NISP*, 2018.

[24] Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

[25] Nacereddine Hammami and Mouldi Bedda. Improved tree model for arabic speech recognition. In *Proc. of CSIT*, volume 5, 2010.

[26] Arthur Asuncion and David Newman. UCI machine learning repository. https://archive.ics.uci.edu/ml/index.php, 2007.

[27] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *Proc. of IJCNN*, 2017.

[28] Jiale Cao, Yanwei Pang, Xuelong Li, and Jingkun Liang. Randomly translational activation inspired by the input distributions of relu. *Neurocomputing*, 275, 2018.

[29] Pu Zhao, Sijia Liu, Pin-Yu Chen, Nghia Hoang, Kaidi Xu, Bhavya Kailkhura, and Xue Lin. On the design of black-box adversarial examples by leveraging gradient-free optimization and operator splitting method. In *Proc. of ICCV*, 2019.

[30] Binxin Ru, Adam Cobb, Arno Blaas, and Yarin Gal. Bayesopt adversarial attack. In *Proc. of ICLR*, 2019.

[31] Rey Wiyatno and Anqi Xu. Maximal jacobian-based saliency map attack. *arXiv preprint arXiv:1808.07945*, 2018.