Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A gradient-based approach for adversarial attack on deep learning-based network intrusion detection systems

Hesamodin Mohammadian[a,*], Ali A. Ghorbani[a], Arash Habibi Lashkari[b]

[a] *Canadian Institute for Cybersecurity, University of New Brunswick, Fredericton, New Brunswick, Canada*
[b] *School of Information Technology, York University, Toronto, Ontario, Canada*

A R T I C L E   I N F O

Article history:
Received 5 November 2022
Received in revised form 7 February 2023
Accepted 1 March 2023
Available online 8 March 2023

Keywords:
Network intrusion detection
Network traffic classification
Deep learning
Machine learning
Adversarial attack

A B S T R A C T

Intrusion detection systems are an essential part of any cybersecurity architecture. These systems are critical in defending networks against a variety of security threats. In recent years, deep neural networks have proved their performance and efficiency in various machine learning tasks, including intrusion detection. However, it is shown that deep learning models are highly vulnerable to adversarial attacks. This paper proposes a new approach for performing an adversarial attack against deep learning-based malicious network activity classification. We use the Jacobian Saliency Map to find the best group of features, with different features and perturbation magnitude, to generate adversarial examples. We evaluate our method on three CIC-IDS2017, CIC-IDS2018, and CIC-DDoS2019 datasets. Our experiments show that our proposed method can achieve better performance while using fewer features in adversarial sample generation than other attacks that depend on a higher number of features. Our technique can generate adversarial samples for more than 18% of samples in CIC-IDS2017, 15% of samples in CIC-IDS2018, and 14% of samples in CIC-DDoS2019, using only three features and 0.1 as the perturbation magnitude. We do a deeper analysis of the attack based on its parameters, distance metrics, and the target model performance. Also, an evaluation model with three criteria, including success rates of the best feature sets, average confidence of the adversarial class, and adversarial samples transferability, is used in our analysis.

1. Introduction

Recently, there has been an increase in the usage of machine learning in automated tasks and decision-making problems. The dependence on machine learning model has tremendously grown in critical infrastructure such as medicine, computer security, and autonomous driving [1]. After introducing deep learning, these methods showed excellent performance in machine learning tasks and gained a lot of attention.

In current studies, deep learning showed its potential in security areas such as malware detection and intrusion detection systems (NIDS). A NIDS goal is to detect attacks against a network and distinguish between benign and malicious activities [2]. NIDS is constantly analyzing incoming and outgoing network traffic. There are two major types of NIDS: signature-based or rule-based. These approaches mainly depend on the information of the previous attacks and can only detect existing malicious patterns; therefore, they are weak against daily changing and evolving attack strategies. With the exponential growth of network traffic, these techniques became increasingly insufficient and impractical. Anomaly based detection techniques are more flexible and provide more efficient approaches in high volume data compare to traditional IDS, which makes them attractive for researches [3–5].

Recent studies show adversarial attacks are effective against the deep learning model. These attacks can cause deep learning models to make mistakes either at training or at test time [6]. One of the first attempts to study this phenomenon was reported in [7], where Dalvi et al. evaluated it in spam filtering. They noted that a linear classifier could be easily fooled by small changes in spam emails' content without drastically changing the spam message's readability. This work was the first adversarial example against linear classifiers. The interest in the usage of deep learning increased after Krishevsky et al. [8] demonstrated the success of a Convolutional Neural Network (CNN) [9] on a very large-scale visual recognition task [10]. In [11], Szegedy et al. explained the vulnerability of deep neural networks against adversarial examples, especially in the computer vision domain.

This paper evaluates the adversarial attacks in NIDS and introduces a method for finding the best features to generate adversarial examples in network datasets. Our focus here is on normal network attacks since most of the common datasets, and

* Corresponding author.
E-mail addresses: h.mohammadian@unb.com (H. Mohammadian), ghorbani@unb.ca (A.A. Ghorbani), ahabibil@yorku.ca (A.H. Lashkari).

https://doi.org/10.1016/j.asoc.2023.110173
1568-4946/© 2023 Elsevier B.V. All rights reserved.

deep learning-based NIDS in the literature are working on these types of attacks. We use three well-known CIC-IDS2017, CIC-IDS2018, and CIC-DDoS2019 datasets. Each flow in these datasets has 76 network features, and our goal is to find the best features among these for adversarial attacks. We create different features containing 1, 2, or 3 features and find the best groups for adversarial sample generation. The most important factor in a successful adversarial attack is making changes as small as possible. Our findings reveal that even with changing a small number of features, our proposed method can generate a high number of adversarial examples and can fool the trained deep learning model. Briefly, our contributions could be specified as follows:

- We propose a new approach for performing an adversarial attack on deep learning-based NIDS and finding the most suitable features for manipulation to generate adversarial samples.
- We evaluate our attack method on three famous network traffic datasets and show the effectiveness of our approach.
- We perform extensive analysis on our results from a different point of view with several criteria.

The remainder of this paper is structured as follows: In section two, the related works are reviewed. The background of the presented work such as deep learning and NIDS is discussed in Section Three. In Section four the proposed method is described in details and the experimental results are showed in Section five. Section six contains the analysis and discussion followed by Section seven with the conclusion.

2. Related work

2.1. Adversarial examples

The primary purpose of adversarial machine learning is to fool different machine learning techniques and force them to make wrong decision during test time by crafting inputs which are called adversarial examples. To generate an adversarial example, a small and imperceptible perturbation is carefully added to the original sample to fool the deep learning model. Simultaneously, a human observer can correctly classify these examples [12,13]. A better adversarial attack technique would generate more adversarial samples while keeping the perturbation value low. Increasing the amount of the perturbation will increase the number of successful adversarial samples, but it may also change the generated sample into something completely different.

2.1.1. Adversarial attack threat model

Adversarial attack threat model may be considered based on the attacker's knowledge, capabilities, goals, and used distance metrics.

- **Attacker's Knowledge**. With regard to the attacker's knowledge, the attacker may know all the information about the learned model, such as learning algorithm, model architecture, parameters, hyper-parameters, and training data. This setting is called *White-box*. In contrast, in *Black-box* attacks, the attacker has limited information about the model and only knows the output.
- **Attacker's Goals**. Attackers may have different goals in an adversarial setting. In *Targeted* attacks, the goal is to make the model misclassify the input into a specific class, while in *Untargeted* attacks, the attacker only aims to fool the model and does not care about the output class. The other type of the attack is *Confidence Reduction* or *Reliability* attacks. In this case, the attacker only seeks to reduce the confidence score of the machine learning model without necessarily changing the output class.
- **Distance Metrics**. As we said in the explanation of adversarial examples, these artificially crafted samples should be really similar to the original input. In order to do that, attack algorithms use different types of distance metrics to measure the similarity between an adversarial example, and its original input. The three most common distance metrics are:
  - l_0 distance, which counts the number of perturbed features.
  - l_2 distance, which computes the standard Euclidean distance between original sample and adversarial sample.
  - l_∞ distance, which measures the maximum difference between any of the features of the original sample and adversarial sample.

2.1.2. Adversarial attack algorithms

One the first works that proved there are small perturbations which can force a deep learning classifier into misclassification by adding to the original inputs was in [11]. They proposed the following optimization problem for finding the smallest possible perturbation (r). In the following equation, f is the loss function of the DNN classifier, x is the input image, and l is the target label.

$$min\ \|r\|_2\ \ s.t.\ f(x+r)=l;\ \ x+r\in[0,1]. \tag{1}$$

Since this is a challenging problem, they used a box-constrained L-BFGS to find an approximate solution. They found the required perturbation changing the original image into an adversarial example by solving this problem.

In [12], a faster and simpler method for generating adversarial examples which called Fast Gradient Sign Method (FGSM) is proposed by Goodfellow et al. They used the model gradient's sign direction to calculate the amount of perturbation required to add to the original example. They used the following equation:

$$\eta=\epsilon sign(\nabla_x J(\theta,x,l)), \tag{2}$$

where η is the perturbation, ϵ is the magnitude of the perturbation, and l is the target label. This perturbation can be computed efficiently using backpropagation. This method calculates the perturbation in only one step toward the direction of the gradient of the model. Since this method only uses one step, it may not find the minimum required perturbation to fool the model.

Kurakin et al. [14] proposed a new method based on the FGSM and instead of applying it in one step, they applied the method multiple times with a smaller step size. Also, after each step, they clipped the results to prevent any large changes on each pixel:

$$x_{n+1}=Clip_{x,\epsilon}\left\{x_n+\alpha sign(\nabla_x J(\theta,x,l))\right\}. \tag{3}$$

Their experiments showed that the iterative gradient sign could produce better results than the fast gradient sign.

Papernot et al. [15] designed a new method based on adversarial saliency maps to craft adversarial examples. Their main idea was to craft adversarial examples based on the direct mapping between inputs and outputs of DNNs. They wanted to model the relation between perturbations added to the input and DNN output variations. To understand this relation, they used the forward derivative of the DNN.

$$J_F(X)=\frac{\partial F(X)}{\partial X}=\left[\frac{\partial F_j(X)}{\partial x_i}\right]_{i\in 1..M, j\in 1..N}. \tag{4}$$

The forward derivative here is the Jacobian of the trained model's function regarding the input features. $F(X)$ is the function that maps the input features to an output probability and has been learned during the training phase. x_i is the feature i of input

X and $F_j(X)$ is the output of the neuron j for input X. M is the number of input features, and N is the number of output neurons or classes.

Then they used these computed forward derivatives to create the adversarial saliency maps. They said by using these maps; we can find to which input features we should add perturbations to have our desired changes in output most efficiently.

The purpose of JSMA is to add more minor perturbations to a small subset of features to cause the model to make a mistake compared to FGSM. However, JSMA is much slower than FGSM due to its high computational cost. Other iterative-based methods such as C&W attack [16] and Deepfool [17] are also proposed.

2.2. Adversarial attack in cybersecurity

The focus of the early research endeavors on adversarial attacks were mainly on image domain problems. Still, with the increasing usage of DNN in security problems, the researchers realize that adversarial examples may widely exist in this domain. Grosse et al. [18] have studied adversarial examples in malware detection. They used the adversarial attack based on the Jacobian matrix. They have limited the number changed feature to 20, in order to make sure the modifications does not change the application drastically. They have achieved misclassification rates between 63% to 69%. In [19], the authors did a straightforward experiment on the NSL-KDD dataset. They used FGSM method to generate adversarial examples and showed the possibility of adversarial attack in IDSs. In [20], Rigaki performed adversarial attack against deep learning model used in NIDS, using FGSM and JSMA methods and showed how significantly they can reduced the model's accuracy. Wang did adversarial attack against NSL-KDD dataset using four famous methods including FGSM, JSMA, Deepfool and C&W [21]. In his thorough study he also analyzed the effect of different features in the dataset in the adversarial example generation process. Peng et al. [22], trained four different machine learning based intrusion detection systems with DNN, SVM, Random Forest, and Logistic Regression and studied these models' robustness in adversarial settings. Ibitoye studied the adversarial attacks against deep learning-based intrusion detection in IoT networks [23]. In [24], they showed how to evaluate an anomaly-based NIDS trained on network traffic in the face of adversarial inputs. They explained their attack method based on categorizing network features evaluated three recently proposed NIDS. Hashemi et al., proposed a new technique for anomaly-based NIDS based on denoising autoencoders to increase the robustness of the system against adversarial samples [25]. They trained their model on some part of the input, and in this way, they increased the reconstruction error of the abnormal traffic. Alhajjar et al. in [26], used two evolutionary algorithms, particle swarm optimization (PSO) and genetic algorithm (GA), and generative adversarial networks (GAN) to generate adversarial examples for NSL-KDD and UNSW datasets. Their results showed that the average evasion rate for NSL-KDD was more than 57% and for UNSW was more than 49%. In [27], the authors performed adversarial attack using FGSM, JSMA, C&W, and ENM against the Kitsune network intrusion detection system. Their experiments showed that the Kitsune is more vulnerable to integrity attacks compare to availability attacks.

Adversarial attacks are performed in white-box or black-box settings. In the white-box setting, the adversary has the perfect knowledge of the target model including the model's used technique, architecture, and hyper-parameters. In contrast, in black-box settings, the attacker can only provide an input and receive model's output and does not have any information about the model's characteristics. First, we reviewed the related works in white-box settings and in the following we continue wit the black-box setting. Yang et al. [28] made a black-box attack on the NSL-KDD dataset. They trained the DNN model on the dataset and used three different attacks based on a substitute model, ZOO [29], and GAN [30]. In [31] they proposed a novel black-box attack that generates adversarial examples using spherical local subspaces. They evaluated their attack against seven state-of-the-art anomaly detectors. The authors in [32], evaluated the adversarial attacks against botnet detectors. They randomly changed four selected different features of each network flow to achieve their adversarial goal. They evaluated botnet samples from four famous datasets and considered feature removal techniques as a defense against adversarial attacks. Huang et al. in [33], proposed two black-boxed methods based on genetic algorithm and a packet saliency for generating adversarial examples against LSTM-based DDoS detection models. In [34], a black-boxed adversarial attack was also introduced against NIDS in IoT systems. They create a duplicate local model and use this model for their attack. They rank input features based on their saliency and use iterative FGSM for their attack. They change the selected feature in each step until they fooled the model.

3. Background

3.1. Network intrusion detection systems (NIDS)

NIDS is a mechanism that detects malicious behaviors in a network [35]. The NIDS are placed at the network edge and monitor any incoming or outgoing traffic. If there is any abnormal or malicious behavior, the NIDS raise the alarm. Two main categories of NIDS are signature-based and anomaly-based. In the signature-based NIDS, the malicious traffic is detected by comparing its patterns with previously known attack signatures. If there is a new attack without known signatures, this approach is not going to work. In contrast, anomaly-based NIDS can work against zero-day attacks. Their approach is to extract benign traffics patterns and classify any traffic with different patterns as malicious. Also, they may work against unknown attacks; they suffer from the problem of high false-positive, which is the possibility of classifying legitimate behavior as malicious. One of the most used methods in anomaly-based NIDS is deep learning models because they are capable of learning and extracting usable features from a high volume of data [28].

3.2. Deep neural network (DNN)

DNN is an ML technique that is constructed from layers of neurons. DNN aims to find a non-linear mapping from its input layer feature to its respective output. Each neuron is a simple function followed by an activation function. The neuron role is to compute the weighted sum of its parameter, which are the previous layer's outputs and then generate the final outputs of its layer by using the activation function. There are three different types of layers in a DNN: input layer, hidden layer and output layer. Each DNN consists of one input layer, one output layer and, based on the respective task, several numbers of the hidden layer.

We can consider a DNN model as the following function F:

$$F(\vec{x}) = f_n(\theta_n, f_{n-1}(\theta_{n-1}, \ldots, f_2(\theta_2, f_1(\theta_1, \vec{x})))), \tag{5}$$

where θ_i is the weight vector of layer i, f_i is the activation function used in layer i, and $\vec{x}$ is the input vector.

Using a DNN consists of two phases: training and testing. In the training phase, we feed a large set of input–output pairs $(\vec{x}, \vec{y})$ to the DNN model. Then the model starts to adjust its weight parameters by minimizing the error between the predicted value for the output and the correct output $\vec{y}$. The commonly used technique for adjusting the DNN parameters is the backpropagation

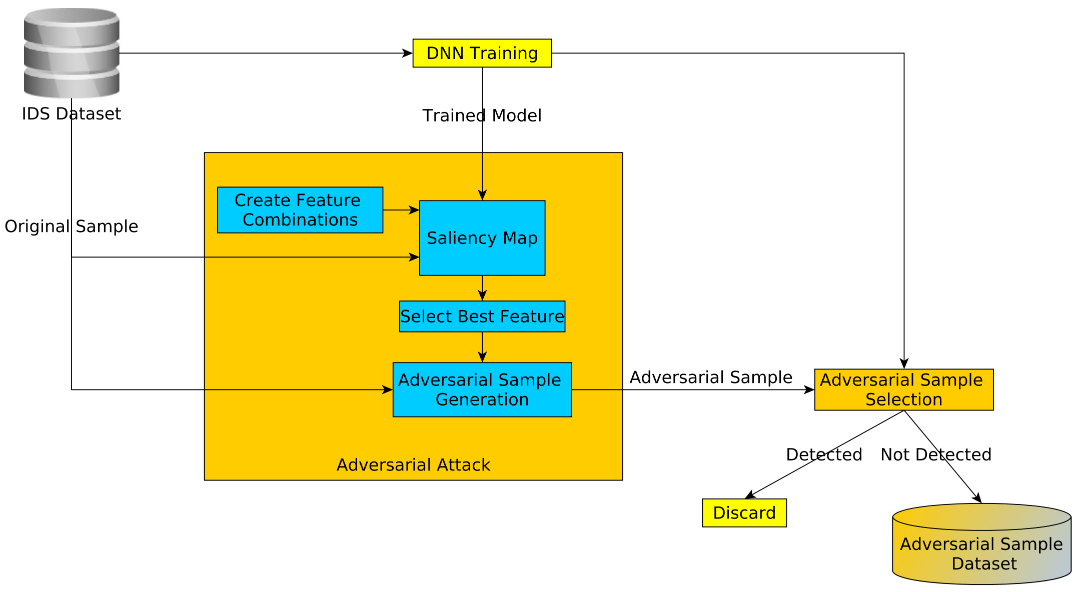

Fig. 1. Overall overview of proposed method.

algorithm with optimization methods such as Stochastic Gradient Descent (SGD) and Adam optimizer [36]. During the test phase, the DNN will predict the output for unseen inputs, using the weight parameters calculated during the training phase.

4. Proposed method

Here we explain the proposed method in details. To do the attack first we have to train a DNN model for classifying the network attacks in the selected datasets. Our attack is in white-box setting, hence the attacker knows the parameter and architecture of the target DNN model.

4.1. Training the DNN target model

First, we train our DNN model for classifying different network attacks. We train a multi-layer perceptron with two hidden layers; each of them contains 256 neurons. During the training, the inputs are 76 features of each network flow, and the outputs are different probability values for each respective attack type and benign flow. We used RelU as our activation function and a Dropout layer with 0.2 probability in both hidden layers.

4.2. Generating adversarial examples

We perform a white-box adversarial attack against our trained deep learning-based network intrusion detection system in this work. Fig. 1 shows an overall overview of our proposed method. In Algorithm 1, the pseudocode of the proposed method is presented.

As shown in Fig. 1, our method consists of different steps, which will be explained sequentially.

4.2.1. Feature combinations

In our method, we would like to find the best combination of features regardless of their nature to perform the adversarial attack. We create different feature combinations using the following equation.

$$C_m^M \Rightarrow f_1, \ldots, f_m, \tag{6}$$

where M is all the features count, m is the number of features in the selected combination, and $f_1, \ldots, f_m$ are the selected features. Also, the number of different feature combinations with m number of features is:

$$C_m^M = \frac{M!}{m!(M-m)!}. \tag{7}$$

Algorithm 1: Generating Adversarial Example

```
Data: X original input, X′ adversarial sample, F trained
      model, η Feature combinations, ϵ perturbation
      magnitude
Result: Adversarial_samples
1  for each $X$ in dataset do
2      Compute forward derivative $J_F(X)$
3      $max \leftarrow 0$
4      for each $\eta_i$ in $\eta$ do
5          $\alpha = \sum_{i=f_1,\ldots,f_n} J_{it}(X)$
6          $\beta = \sum_{f_1,\ldots,f_n} \sum_{j\neq t} J_{ij}(X)$
7          if $\alpha < 0$ and $\beta > 0$ and $-\alpha \times \beta > max$ then
8              $selected\_features \leftarrow \eta_i$
9              $max \leftarrow -\alpha \times \beta$
10         end
11     end
12     $X' = X + selected\_features * \epsilon$
13     if $F(X') \neq F(x)$ then
14         $Adversarial\_samples \leftarrow X'$
15     end
16 end
17 return $Adversarial\_samples$
```

4.2.2. Saliency map

After creating the feature combinations, we would want to rank these combinations and select the best one for performing an adversarial attack. In [15], Papernot et al. extended the saliency map introduced in [37] to create an adversarial saliency map and rank the input features. This ranking shows which input features should be perturbed to have the most effect on model output. The adversarial saliency map is based on the forward derivative of the trained model.

The forward derivative is a tool to help the attacker perform the adversarial attack and cause the model to mistake and misclassify an input. The equation for the forward derivative is:

$$J_F(X) = \frac{\partial F(X)}{\partial X} = \left[\frac{\partial F_j(X)}{\partial x_i}\right]_{i\in 1\ldots M, j\in 1\ldots N}. \tag{8}$$

The saliency map aims to find a feature to change and cause the model to misclassify an input. The model labels an input based on the *argmax* $F_j(X)$. To cause the model to misclassify an input to a target class t, the probability of the t should be increased, and the probability of all other classes should be decreased or stay unchanged until $t = argmax\ F_j(X)$.

Based on the above explanation, the saliency map is generated using the following equations:

$$S(X,t)[i] = \begin{Bmatrix} 0 \text{ if } J_{it}(X) < 0 \text{ or } \sum_{j\neq t} J_{ij}(X) > 0 \\ J_{it}(X)\left|\sum_{j\neq t} J_{ij}(X)\right| \ otherwise \end{Bmatrix}. \tag{9}$$

In the above equation, i is the input feature, X is the input sample, and t is the target class. In the first condition, if the forward derivative of the target class t with respect to feature i is less than zero, which means the probability of the target class is decreasing, or the sum of the forward derivative of all other classes is positive which means the probability of at least one of the other classes is increasing, the saliency map value is zero. In the second line, the forwarding derivative product for the target class and the sum of all other classes is used for the saliency map value.

Everything that has been explained so far about the saliency map is toward performing the targeted adversarial attack on image datasets. But, here, we are mostly interested in fooling NIDS, generate adversarial samples using network features, and do not care about the target class. To achieve this, we make some changes in the saliency map equation. First, we consider the true class as the target class, and instead of increasing this class probability, we try to decrease it. To do so, we change the conditions as follows:

$$S(X,t)[i] = \begin{Bmatrix} 0 \text{ if } J_{it}(X) > 0 \text{ or } \sum_{j\neq t} J_{ij}(X) < 0 \\ |J_{it}(X)|\left(\sum_{j\neq t} J_{ij}(X)\right) \ otherwise \end{Bmatrix}. \tag{10}$$

The only difference between Eqs. (9) and (10), is the direction of the inequalities in the first line condition. Since the goal is to decrease the target class's probability, we set the saliency map to zero when the forward derivative of the target class is positive, or the sum of the forward derivative of all other classes is negative.

After calculating the saliency map, the features with the higher values are the ones that decrease the target class probability or increase or keep unchanged the probability for all the other classes. The above equation calculates the saliency map based on one feature and selects the feature with highest value. In the proposed method, we want to select more than one feature and calculate the saliency map based on the combination of different network features to measure the effect of features with different nature on adversarial samples generation process. In next subsection, we show how the best feature combination is selected.

4.2.3. Best features combination selection

Since we aim to find more than one feature to change during our adversarial attack, the following equation calculates the saliency map values.

$$argmax_{f_1,\ldots,f_n} \left|\sum_{i=f_1,\ldots,f_n} J_{it}(X)\right| \times \left(\sum_{f_1,\ldots,f_n}\sum_{j\neq t} J_{ij}(X)\right). \tag{11}$$

Then, the combination of features that maximizes this equation will be selected to use in the adversarial samples generation step.

Table 1
Records in CIC-IDS2017 dataset.

Attack name	Number of records
Benign	2271320
DDoS	128025
PortScan	158804
Botnet	1956
Infiltration	36
Web Attack-Brute Force	1507
Web Attack-Sql Injection	21
Web Attack-XSS	652
FTP-Patator	7935
SSH-Patator	5897
DoS GoldenEye	10293
DoS Hulk	230124
DoS Slowhttp	5499
Dos Slowloris	5796
Heartbleed	11

Table 2
Records in CSE-CIC-IDS2018 dataset.

Attack name	Number of records
Benign	13390249
Bot	286191
Infiltration	160639
Brute Force-Web	611
Brute Force-XSS	230
SQL Injection	87
FTP-BruteForce	193354
SSH-BruteForce	187589
DoS GoldenEye	41508
DoS Hulk	461912
DoS Slowhttp	139890
Dos Slowloris	10990
DDoS LOIC-HTTP	576191
DDoS LOIC-UDP	1730
DDoS HOIC	686012

Table 3
Records in CIC-DDoS2019 dataset.

Attack name	Number of records
Benign	56425
DNS	4908665
LDAP	2141300
MSSQL	4396046
NetBios	3963446
NTP	1195690
SNMP	5149261
SSDP	2568569
UDP	3094002
SYN	1379983
TFTP	19515971
UDP-Lag	330079
WebDDoS	439

The for loop from line 4 to line 11 of the Algorithm 1, calculates the saliency value for each feature combination using Eq. (10) and chooses the feature combination with highest saliency value as the selected feature.

4.2.4. Adversarial samples generation

After finding the best features combination for our untargeted attack, we define a mask based on those features and generate the adversarial example.

$$X' = X + mask * \epsilon, \tag{12}$$

where X' is the adversarial sample, X is the original sample, *mask* is the selected features for adding the perturbation, and ϵ is the amount of perturbation. In line 12 of the Algorithm 1, calculates the adversarial sample based on Eq. (12). As an example, let us assume our original sample is $X = [0,0,0,0,0]$ and the

Table 4
Results of the classifiers.

Machine learning	IDS2017			IDS2018			DDoS2019		
techniques	F1-score	PC	RC	F1-score	PC	RC	F1-score	PC	RC
DT	99.84	99.76	99.92	97.70	99.73	96.14	97.09	98.54	96.26
Naive Bayes	28.47	32.43	73.75	48.29	49.48	72.79	50.55	60.20	62.03
LR	36.71	39.76	34.96	57.97	64.13	56.84	69.53	70.68	68.94
RF	96.58	99.79	94.24	94.23	99.81	90.97	95.65	98.55	94.60
DNN (Our)	98.18	98.27	98.22	98.71	98.99	99.04	98.97	99.00	98.96

combination of first and third feature is selected as the best feature combination, therefore $mask = [1, 0, 1, 0, 0]$. If $\epsilon = 0.1$, the adversarial sample X' is going to be $[0.1, 0, 0.1, 0, 0]$.

After generating the adversarial sample, we test the model with it. If the model makes a mistake and misclassifies the input, it will be accepted as an adversarial sample; otherwise, it will be discarded. We continue with the next original input. The last condition in the Algorithm 1, does the above checking and decides whether choose or discard the generated sample.

5. Experiments

To do our experiments, first, we need to train DNN models for classifying network attacks in the three selected datasets. Then we use the above technique to perform an adversarial attack against the trained classifiers. The goal here is to find the best feature groups with different features for generating adversarial samples.

We performed the experiments on a machine with the NVIDIA TITAN V GPU and used python language for the programming. Also, PyTorch and scikit-learn libraries were used to develop the deep learning model, machine learning models, and the adversarial attack.

5.1. Dataset

In this paper, we use CIC-IDS2017 [38], CSE-CIC-IDS2018, and CIC-DDoS2019 [39] to train our DNN model and perform the adversarial attack. Each dataset contains several network attacks. In the CIC-IDS2017 [38], they argued that covering eleven criteria mentioned in the evaluation framework in [40] is necessary for an IDS dataset, and their dataset can cover all of them. CSE-CIC-IDS2018 is an extension to CIC-IDS2017 and contains the same kind of attacks. The CIC-DDoS2019 [39] contains 12 different attack such as DNS, LDAP, and MSSQL. They extracted more than 80 network traffic features from their datasets using CICFlowMeter [41] and labeled each flow as benign or attack name. Tables 1–3 present the details of these three datasets.

The CIC-DDoS2019 [39] contains traffic for two training and testing days. We use the data from the training day of the CIC-DDoS2019 [39] and the whole CSE-CIC-IDS2018 and CIC-IDS2017 [38] to train our DNN model and craft adversarial examples. There some features in these datasets including Flow ID, Source IP, Source Port, Destination IP, Destination Port, Protocol, and Timestamp, which are not suitable for a DNN model. We removed them during the prepossessing step.

5.2. Target model training

There are number of different methods that can be used to create a NIDS. Since in the literature it is common [42–44] to use DNN as a tool to build a NIDS, we train a DNN model for multi-class classification on each dataset and compare its performance with other machine learning techniques. The focus here is on adversarial attacks against deep learning models. Therefore, the results from other machine learning methods (Decision Tree, Naive Bayes, Logistic Regression, and Random Forest) presented in Table 4 are only for comparison purposes. Also, we use these trained machine learning models to evaluate the transferability of adversarial samples. All the hyper-parameters for the machine learning models are the default values in the scikit-learn library. For example for the Decision Tree, the criterion is gini and min_sample_split is two. The DNN model is a simple multi-layer perceptron. We use F1-score, precision (PC), and recall (RC) as the evaluation metrics. As presented in Table 4, the DNN model has comparable performance with other machine learning models and the state of the art works like Lopez-Martin et al. [45] where they used deep neural network and RBF neural network for classification task on CIC-IDS2017 and CIC-DDoS2019.

5.3. The adversarial attack results

After training the DNN model, we use the proposed method to generate adversarial examples for the selected datasets. We perform the attack with 1, 2, and 3 as the number of features and 0.1, 0.01, and 0.001 as different ϵ values.

In order to compare the performance of the proposed method, we present the results of an adversarial attack by selecting features based on their nature [46]. The 76 features in each dataset are grouped in six different categories based on their nature, including *Forward*, *Backward*, *Flow-based*, *Time-based*, *Packet Header-based*, and *Packet Payload-based* features. Then the FGSM method is used to generate adversarial examples by changing the values of the features in each of these categories.

Our purpose here is to minimize the number of changed feature when performing adversarial attacks. Different variants of FGSM like I-FGSM [14] and MI-FGSM [47] or other iterative methods such as C&W [16] and Deepfool [17] are still changing all the features. That is the reason we use the technique proposed in [46] for the comparison.

5.3.1. CIC-IDS2017

First, we start our experiment with the CIC-IDS2017 datasets. The number of correctly classified samples by our DNN model is 2,777,668 out of all the samples in the dataset and we only use them for performing the adversarial attack.

Table 14 shows the overall results for the attack. We presented the number of generated adversarial samples and the ratio of the generated samples to all the samples in the datasets. The first column for each dataset is the number of generated samples when we use all the features with the FGSM method to make an adversarial attack. As it is evident, the best performance is when three features with $\epsilon = 0.1$ were used for generating the samples.

In Table 5, the result of the attack method based on feature groups is shown for the CIC-IDS2017. In this table same as Table 14, we presented the number of generated adversarial

Table 5
CIC-IDS2017 results for feature selected based on their nature.

ϵ	F (24)	B (22)	FL (15)	T (27)	PH (14)	PP (16)
0.1	1396501	482203	501943	530359	194026	759937
	50.27	**17.35**	**18.07**	**19.09**	**6.98**	**27.35**
0.01	482110	327771	210701	300282	12887	348857
	17.35	**11.80**	**7.58**	**10.81**	**0.46**	**12.55**
0.001	160572	70857	54077	89193	1464	50645
	5.78	**2.55**	**1.94**	**3.21**	**0.05**	**1.82**

Table 6
CIC-IDS2018 results for feature selected based on their nature.

ϵ	F (24)	B (22)	FL (15)	T (27)	PH (14)	PP (16)
0.1	5306473	2004686	2877603	1799097	1733240	3271036
	33.2	**12.5**	**18.00**	**11.25**	**10.84**	**20.46**
0.01	1142629	935050	328957	440110	522918	438862
	7.14	**5.85**	**2.05**	**2.75**	**3.27**	**2.74**
0.001	340041	163766	50696	213556	221866	112328
	2.12	**1.02**	**0.31**	**1.33**	**1.38**	**0.7**

samples and the ratio of the generated samples to all the samples in the datasets. The results in these two tables show that the new method is mostly better than the previous method while changing fewer features. For example, when one feature is used with all three values for ϵ the attack is more successful than using *Packet header-based* features which contains 14 features. Also, when we use three features the results are better or really close comparing to results for *Backward*, *Flow-based*, *Time-based*, and *Packet Header-based* feature groups with 22, 15, 27, or 14 features.

In Table 7 the more detailed results for CIC-IDS2017 is shown. In this table we show the top five best features with highest number of generated samples for each ϵ values.

For $\epsilon = 0.1$, the best single feature is *Flow IAT Min* with 144,408 generated samples. For two feature, the combination of *Flow IAT Min* and *Fwd IAT Total* with 178,888 generated samples and for three feature attack, *Flow IAT Min*, *Fwd IAT Total* and *FWD Packets/s* together with 105,054 generated samples are the best.

With $\epsilon = 0.01$, while the number of generated samples decreased due to smaller value of the ϵ, still the best single feature is *Flow IAT Min*, the best two features combination are *Flow IAT Min* and *Fwd IAT Total*, and for three features combination the best are *Flow IAT Min*, *Fwd IAT Total*, and *FWD Packets/s*.

The best selected features for $\epsilon = 0.001$ are almost the same as the best features of the two other ϵ values. The best single feature is *Flow IAT Min*. The combination of *Flow IAT Min* and *Fwd IAT Total* is the best two feature groups and the best with three features is the combination of *Flow IAT Mean*, *Fwd IAT Min*, and *Fwd IAT Total*.

We select the top three best features group for each ϵ values and present them in Table 8. As shown for all three ϵ values, the highest number of generated samples is when *Flow IAT Min* and *Fwd IAT Total* are used together. The second best-selected feature is also the same for all the ϵ values, and it is with *Flow IAT Min*. There is a slight difference in the third-best features group. For all three ϵs the third-best feature group contains three features: the same for 0.1 and 0.01, but one of the features is different for 0.001. For 0.1 and 0.01 the selected features are *Flow IAT Min*, *Fwd IAT Total* and *FWD Packets/s* but for 0.001 instead of *FWD Packets/s*, *Fwd IAT Mean* is selected.

5.3.2. CIC-IDS2018

For the CIC-IDS2018, our model can classify 15,982,736 of the samples and use these records to generate adversarial samples.

In Table 14, the results of the attack based on our proposed method is shown for the CIC-IDS2018. For purposes of comparison, we also include the results when all the features are used. As expected, the highest numbers of generated samples are when the bigger value for ϵ and more features are used. The number of generated samples with $\epsilon = 0.1$ and three features is 2,455,384.

To compare the proposed method results with number of generated samples when features are grouped based on their nature, the results for that attack are shown in Table 6. Comparing these two tables presents interesting findings. When 3 features were used with $\epsilon = 0.1$ the attack is successful for 15.36% of the samples and it outperforms attacks using *Backward*, *Time-based*, and *Packet Header-based* features with 22, 27, and 14 number of features. With $\epsilon = 0.01$ and two features we get results better than *Flow-based* features and really close to *Time-based* and *Packet Payload-based* features. And at last, with $\epsilon = 0.001$ and selecting one feature we still are able to outperform *Flow-based* and *Packet Payload-based* features and get almost as good performance as *Backward* features.

In Table 9 the top five best selected features with different values of ϵ are shown for CIC-IDS2018 dataset.

The results for $\epsilon = 0.1$ shows that the top single feature is *Fwd Header Length* with 483,241 successful attempts. The combination of *Fwd Header Length* and *total Fwd Packets* is the best two feature group with 433,771 number of adversarial samples. And the top three feature combination is *Fwd Header Length*, *total Fwd Packets* and *Subflow Fwd Packets* together with 586,626 generated samples.

Decreasing the value of ϵ does not change the top selected features and still the best single feature is *Fwd Header Length*, two feature combination is *Fwd Header Length* and *total Fwd Packets*, and the combination of *Fwd Header Length*, *total Fwd Packets* and *Subflow Fwd Packets* is the best three feature group for both ϵ 0.01 and 0.001.

In Table 10, the top three feature groups are shown for each value of ϵ. The highest number of generated samples for all three values of ϵ is when three features including *Fwd Header Length*, *total Fwd Packets* and *Subflow Fwd Packets* are used for adversarial attack. When the *Fwd Header Length* is used we got the second-best results for all the ϵ values. The third-best selected feature for $\epsilon = 0.1$ is *act_data_pkt_fwd* and is different from features for ϵ values of 0.01 and 0.001 which are *Fwd Header Length* and *total Fwd Packets*.

5.3.3. CIC-DDoS2019

The number of correctly classified samples for the CIC-DDoS2019 dataset is 48,197,029, which were used for performing the adversarial attack.

Table 14 contains the results of the proposed method on the CIC-DDoS2019 dataset. The highest number of generated samples is when $\epsilon = 0.1$ and three features are used.

In Table 11, the results for the adversarial attack when features are grouped based on their nature is shown. When the new method is used with three features and $\epsilon = 0.01$ it is successful in 5.43% of cases which is higher than the success rate of the attacks in Table 11 in all the cases except when the *Forward* features are used. With $\epsilon = 0.001$ and three features, we also have the same performance.

Table 12 demonstrates the best results with different number of features for each ϵ value. For $\epsilon = 0.1$ with one feature, the best result is when *Flow IAT Min* is used, and the number of generated samples is 1,516,234. When *Fwd IAT Min* is used with *Flow IAT*

Table 7
CIC-IDS2017 top 5 best feature groups for different ϵ values.

ϵ	1 Feature		2 Features		3 Features	
	Samples	Selected features	Samples	Selected features	Samples	Selected features
0.1	144408	Flow IAT Min	178888	Flow IAT Min Fwd IAT Total	105054	Flow IAT Min Fwd IAT Total FWD Packets/s
	88979	Act_data_pkt_forward	83551	Subflow Fwd Packets Act_data_pkt_forward	83514	total Fwd Packets Subflow Fwd Packets Act_data_pkt_forward
	35250	Fwd IAT Total	30178	FWD Packets/s Act_data_pkt_forward	39899	Fwd Packet Length Std Flow IAT Min Fwd IAT Total
	29513	Subflow Bwd Packets	27470	Flow IAT Std Act_data_pkt_forward	28313	FWD Packets/s Act_data_pkt_forward Active Std
	20638	Flow IAT Std	20229	Flow duration Act_data_pkt_forward	27846	total Bwd Packets Flow IAT Min Subflow Bwd Packets
0.01	59490	Flow IAT Min	81639	Flow IAT Min Fwd IAT Total	54402	Flow IAT Min Fwd IAT Total FWD Packets/s
	29084	Subflow Bwd Packets	20208	Flow duration Act_data_pkt_forward	27729	total Bwd Packets Flow IAT Min Subflow Bwd Packets
	25591	Fwd IAT Total	16861	total Bwd Packets Subflow Bwd Packets	24912	Fwd Packet Length Std Flow IAT Min Fwd IAT Total
	7551	Flow IAT Std	12298	Flow IAT Min Subflow Bwd Packets	24830	Flow IAT Mean Flow IAT Min Fwd IAT Total
	7273	total Fwd Packets	7605	Flow IAT Std Fwd IAT Mean	23119	Flow IAT Std Fwd IAT Mean Act_data_pkt_forward
0.001	37623	Flow IAT Min	44118	Flow IAT Min Fwd IAT Total	24856	Flow IAT Mean Fwd IAT Min Fwd IAT Total
	6816	total Fwd Packets	7439	total Fwd Packets Subflow Fwd Packets	12675	Flow IAT Min Fwd IAT Total Init_Win_bytes_forward
	3027	Act_data_pkt_forward	4826	total Bwd Packets Flow IAT Min	7951	Fwd Packet Length Std Flow IAT Min Fwd IAT Total
	2050	Fwd IAT Total	3959	Flow IAT Std Fwd IAT Mean	6012	Flow IAT Min Fwd IAT Total FWD Packets/s
	1549	Flow IAT Std	3031	total Fwd Packets Act_data_pkt_forward	5810	total Fwd Packets Subflow Fwd Packets Act_data_pkt_forward

Table 8
CIC-IDS2017 top 3 best feature sets for each ϵ values.

0.1		0.01		0.001	
Samples	Selected features	Samples	Selected features	Samples	Selected features
178888	Flow IAT Min Fwd IAT Total	81639	Flow IAT Min Fwd IAT Total	44118	Flow IAT Min Fwd IAT Total
144408	Flow IAT Min	59490	Flow IAT Min	37623	Flow IAT Min
105054	Flow IAT Min Fwd IAT Total FWD Packets/s	54402	Flow IAT Min Fwd IAT Total FWD Packets/s	24856	Fwd IAT Min Fwd IAT Total Flow IAT Mean

Min we get the highest number of adversarial samples which is 1,469,505. The best three-feature group is the combination of the best two-feature group and *Fwd IAT Mean*.

The best single feature, two-feature group, and three-feature group are the same as $\epsilon = 0.1$ for $\epsilon = 0.01$.

For $\epsilon = 0.001$, the best single feature and two-feature group are the same as when ϵ is 0.1 and 0.01. But there is one difference in the best three-feature group, and instead of *Fwd IAT Mean*, *Total Length of Fwd Packets* is selected.

In Table 13, the top three selected feature groups for the DDoS2019 are presented. There are a lot of similarities in top feature groups between three ϵ values, and most of the selected features are time-based. The best feature group for 0.01 and 0.001 is the same and has two features. For $\epsilon = 0.1$, the top selected group is the same as two other values plus *Fwd IAT Mean*.

Table 9
CIC-IDS2018 top 5 best feature groups for different ϵ values.

ϵ	1 Feature		2 Features		3 Features	
	Samples	Selected features	Samples	Selected features	Samples	Selected features
0.1	483241	Fwd Header Length	433771	Fwd Header Length Total Fwd Packets	586626	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
	464612	act_data_pkt_fwd	148266	Fwd Packet Length Std act_data_pkt_fwd	204103	Flow Packets/s Fwd Packets/s act_data_pkt_fwd
	157896	Subflow Bwd Packets	140708	Fwd Header Length act_data_pkt_fwd	134622	Fwd Header Length Subflow Bwd Packets act_data_pkt_fwd
	87685	Bwd Header Length	131098	Fwd Packets/s act_data_pkt_fwd	132157	Fwd Packets Length Std Packets Length Std Packets Length Variance
	48999	Fwd Packet Length Std	113596	Packet Length Variance act_data_pkt_fwd	129758	Bwd Packets Length Min Flow IAT Std Fwd IAT Std
0.01	166746	Fwd Header Length	153576	Fwd Header Length Total Fwd Packets	185881	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
	42379	Fwd Packets/s	92181	Bwd Header Length Subflow Bwd Packets	103111	Bwd Header Length Total Bwd Packets Subflow Bwd Packets
	4001	Subflow Fwd Packets	38501	Fwd Packets/s Flow Packets/s	29711	Bwd Packet Length Min Flow Packets/s Fwd Packets/s
	2201	Bwd IAT Min	22359	Fwd Header Length Subflow Fwd Packets	8070	Flow Bytes/s Fwd Packets/s Active Min
	2087	Total Fwd Packets	10862	Total Backward Packets Bwd Header Length	5891	Total Fwd Packets Flow Bytes/s Fwd Header Length
0.001	149334	Fwd Header Length	142512	Fwd Header Length Total Fwd Packets	162831	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
	6471	Fwd Packets/s	22255	Fwd Header Length Subflow Fwd Packets	5517	Flow Packets/s Fwd Packets/s act_data_pkt_fwd
	1460	Subflow Fwd Packets	7345	Fwd Packets/s Flow Packets/s	4958	Total Fwd Packets Fwd Header Length Fwd Packets/s
	685	Flow Packets/s	2024	Subflow Fwd Packets Total Fwd Packets	2618	Bwd IAT Min Fwd Packets/s Active Min
	494	Total Fwd Packets	1865	Fwd Packets/s Active Min	1943	Flow Packets/s Bwd IAT Min Fwd Packets/s

Table 10
CIC-IDS2018 top 3 best feature sets for each ϵ values.

0.1		0.01		0.001	
Samples	Selected features	Samples	Selected features	Samples	Selected features
586626	Fwd Header Length Total Fwd Packets Subflow Fwd Packets	185881	Fwd Header Length Total Fwd Packets Subflow Fwd Packets	162831	Fwd Header Length Total Fwd Packets Subflow Fwd Packets
483241	Fwd Header Length	166746	Fwd Header Length	149334	Fwd Header Length
464612	act_data_pkt_fwd	153576	Fwd Header Length Total Fwd Packets	142512	Fwd Header Length Total Fwd Packets

Table 11
CIC-DDoS2019 results for feature selected based on their nature.

ϵ	F (24)	B (22)	FL (15)	T (27)	PH (14)	PP (16)
0.1	28294444	20000389	13087263	19289663	7806957	27075487
	58.70	**41.49**	**27.15**	**40.02**	**16.19**	**56.17**
0.01	13389495	907908	933775	2221351	6456	2226765
	27.78	**1.88**	**1.93**	**4.60**	**0.01**	**4.62**
0.001	278402	6356	7692	21831	1714	11722
	0.57	**0.01**	**0.01**	**0.04**	**0.003**	**0.02**

This section described the results of the adversarial attack using the proposed method on three IDS2017, IDS2018, and DDoS2019 datasets. The detailed results are shown in the tables with different ϵ values and three different number of features for each dataset. Finally, the best feature sets based on the number of generated adversarial examples are presented.

6. Analysis and discussions

Here we are going to do a deeper analysis of the results presented in Section 5.3. First, the effect of two parameters of the attack method, ϵ (perturbation magnitude) and m (number of features), is evaluated. Then, the analysis of the distance between the original and adversarial sample based on different l_p norms is provided. We also explored the effect of the adversarial attack on the performance of the DNN model. Finally, an evaluation model based on three criteria, including best feature sets success rates, average confidence of adversarial class, and adversarial samples transferability, is proposed, and the attack method results are analyzed based on this model.

6.1. Parameter evaluation

The two important parameters in the proposed method are the magnitude of perturbation (ϵ) and the number of selected features (m).

In Section 5.3, the detailed results of the attack with three different values of ϵ and numbers of features are presented. The main findings which were expected are that increasing the number of features and value of ϵ improves the effectiveness of the attack.

The other important result is that while the highest number of features used in the attacks is three, in many cases, the number of generated samples is either close to or higher than the other types of attacks that use a greater number of features.

Changing the value of ϵ does not have much effect on the most successful feature groups in IDS2017 and IDS2018 datasets. As you can see in Tables 8 and 10, the selected features in all three columns are almost the same. The best group of features has two members in Table 8, but in Table 10 it has three members. This means that the most successful feature groups do not always have a greater number of features. For the DDoS2019 dataset, the most successful feature groups have a lot in common for different values of ϵ, but not as much as two other datasets.

6.2. Distance analysis

As we mentioned before, one of the important factors of an adversarial attack is the distance between the original sample and the adversarial sample. The three most common used distance metrics are l_0, l_2, and l_∞.

The l_0 distance is going to count the number of changed features between two samples and does not take into account the values of ϵ. On the other hand, the l_∞ distance only measures the maximum changes between different features of the two samples, which is always equal to ϵ. The l_2 distance yields the standard Euclidean distance using the following equation:

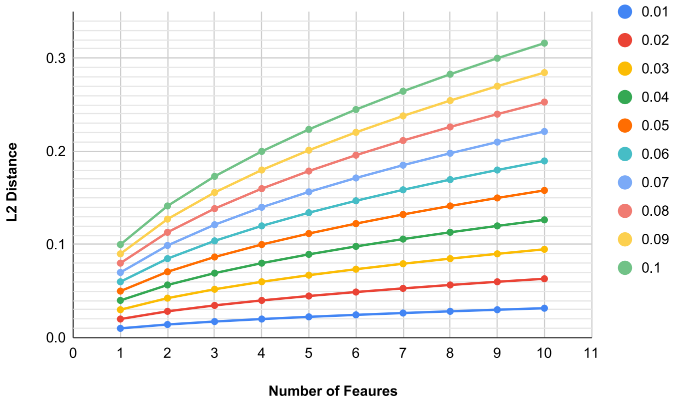

Fig. 2. Distance analysis based on m and ϵ.

$$l_2 = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + \cdots + (x_n - x_n')^2}. \quad (13)$$

The difference between features in the above equation is ϵ and the number of changed features in m; therefore, the l_2 distance is equal to $\epsilon\sqrt{m}$.

In Fig. 2, we plot the l_2 distance for ϵ between 0.01 and 0.1 with number of features from 1 to 10. The x axis is the number of features and the y axis is the l_2 distance. Fig. 2 shows that by increasing the value of the ϵ the effect of number of features increases. For example, with $\epsilon = 0.01$ and $m = 10$ the distance is around 0.03 and is same as distance with $\epsilon = 0.02$ and $m = 2$, but the distance for $\epsilon = 0.09$ and $m = 10$ is almost 0.29 and is close to the distance with $\epsilon = 0.1$ and $m = 8$.

6.3. The DNN model's performance

In Section 5.3, we provided the number of generated samples with different values of ϵ and m. Here we are going to replace these generated adversarial samples with the original samples in the datasets and provide the performance of the trained DNN model on the new datasets.

In Table 15, the performance of the DNN model on datasets containing adversarial samples is presented based on F1-score (F1), Precision (PC), and Recall (RC).

As it is expected, with higher values of m and ϵ, the decrease in the performance of the model is more severe. The F1-score for IDS2017 drops from 68.45% to 16.57% and for IDS2018 drops from 81.23% to 28.14% when the attack was performed with three features and $\epsilon = 0.1$. Also, the impact of ϵ is more than the number of features on the model's performance. With $\epsilon = 0.1$ and only one feature, the drop in F1-score, PC, and RC are more severe than the attack with three features and $\epsilon = 0.001$. For DDoS2019, even with adversarial sample generated with three features and 0.1 epsilon, does not reduce the model's performance as much as two other datasets. There reason is that the selected features are not good enough to fool the model and also the number of generated sample should be more to hide the huge DDoS attacks.

In general, our method successfully fools the DNN model trained on three datasets and diminishes the performance of these models. Also, compared to [32], we have better performance. In their experiments, after performing the adversarial attack, the Recall for IDS2017 is 65.6% and for IDS2018 is 56.4%, while in Table 15, the Recall for the attack with three features and ϵ value of 0.1 is 14.78% and 27.84% for these two datasets.

Table 12
CIC-DDoS2019 top 5 best feature groups for different ϵ values.

ϵ	1 Feature		2 Features		3 Features	
	Samples	Selected features	Samples	Selected features	Samples	Selected features
0.1	1516234	Flow IAT Min	1469505	Flow IAT Min Fwd IAT Min	1601255	Flow IAT Min Fwd IAT Min Fwd IAT Mean
	593134	Fwd Packet Length Std	736756	Total Fwd Packets Subflow Fwd Packets	963939	Total Length of Fwd Packets Subflow Fwd Bytes act_data_pkt_fwd
	446357	Bwd Packet Length Min	691803	Subflow Fwd Bytes act_data_pkt_fwd	412482	Total Fwd Packets Fwd IAT Min Subflow Fwd Packets
	365240	Packet Length Std	434493	Max Packet Length act_data_pkt_fwd	398609	Total Length of Fwd Packets Max Packet Length act_data_pkt_fwd
	331167	Total Fwd Packets	410695	Total Length of Fwd Packets Subflow Fwd Bytes	372277	Fwd IAT Min Bwd Header Length Subflow Fwd Packets
0.01	442273	Flow IAT Min	987408	Flow IAT Min Fwd IAT Min	580417	Flow IAT Min Fwd IAT Min Fwd IAT Mean
	220531	Fwd IAT Min	532091	Total Fwd Packets Subflow Fwd Packets	488002	Total Length of Fwd Packets Subflow Fwd Bytes act_data_pkt_fwd
	107633	Total Fwd Packets	378244	Fwd IAT Min Bwd Header Length	399905	Total Fwd Packets Fwd IAT Min Subflow Fwd Packets
	33620	act_data_pkt_fwd	50212	Total Length of Fwd Packets act_data_pkt_fwd	347075	Total Length of Fwd Packets Flow IAT Min Fwd IAT Min
	13491	Subflow Fwd Packets	30360	Subflow Fwd Bytes act_data_pkt_fwd	322401	Fwd IAT Min Bwd Header Length Subflow Fwd Packets
0.001	18866	Fwd IAT Min	123565	Flow IAT Min Fwd IAT Min	67259	Total Length of Fwd Packets Flow IAT Min Fwd IAT Min
	8923	Flow IAT Min	4842	Fwd IAT Std act_data_pkt_fwd	47275	Flow IAT Min Fwd IAT Min Fwd IAT Std
	2770	act_data_pkt_fwd	1087	Bwd Header Length Subflow Fwd Packets	14724	Flow IAT Min Fwd IAT Mean Fwd IAT Min
	1046	Bwd Header Length	855	Total Fwd Packets Subflow Fwd Packets	4935	Total Bwd Packets Fwd IAT Min Fwd IAT Std
	388	Total Fwd Packets	127	Flow IAT Min act_data_pkt_fwd	2689	Flow IAT Min Flow IAT Mean Fwd IAT Min

Table 13
CIC-DDoS2019 top 3 best feature sets for each ϵ values.

0.1		0.01		0.001	
Samples	Selected features	Samples	Selected features	Samples	Selected features
1601255	Flow IAT Min Fwd IAT Min Fwd IAT Mean	987408	Flow IAT Min Fwd IAT Min	123565	Flow IAT Min Fwd IAT Min
1516234	Flow IAT Min	580417	Flow IAT Min Fwd IAT Min Fwd IAT Mean	67259	Total Length of Fwd Packets Flow IAT Min Fwd IAT Min
1469505	Flow IAT Min Fwd IAT Min	532091	Total Fwd Packets Subflow Fwd Packets	47275	Flow IAT Min Fwd IAT Min Fwd IAT Std

Table 14
Attack results for three datasets.

ϵ	IDS2017				IDS2018				DDoS2019			
	All	1 Feature	2 Features	3 Features	All	1 Feature	2 Features	3 Features	All	1 Feature	2 Features	3 Features
0.1	1666564	375970	489274	516815	5263407	1422321	1759628	2455384	28106617	4322135	4721578	7042634
	59.99	**13.53**	**17.61**	**18.6**	**32.93**	**8.89**	**11.00**	**15.36**	**58.31**	**8.96**	**9.79**	**14.61**
0.01	106175	136595	173568	240890	1819315	223747	356181	384268	19167232	822859	2056339	2619051
	38.22	**4.91**	**6.24**	**8.67**	**11.38**	**1.39**	**2.22**	**2.4**	**39.76**	**1.70**	**4.26**	**5.43**
0.001	251453	52924	66430	72202	439218	158953	177312	182429	553323	32120	130753	144626
	9.05	**1.9**	**2.39**	**2.59**	**2.74**	**0.99**	**1.1**	**1.14**	**1.14**	**0.06**	**0.2**	**0.3**

Table 15
DNN model performance on datasets with adversarial samples.

Attack parameters		IDS2017			IDS2018			DDoS2019		
m	ϵ	F1	PC	RC	F1	PC	RC	F1	PC	RC
No Attack		68.45	88.96	65.36	81.23	96.04	79.44	90.17	89.43	91.07
3	0.1	16.57	20.93	14.78	28.14	41.84	27.84	75.43	76.10	76.67
	0.01	51.86	78.18	45.45	73.95	93.75	68.84	83.82	82.66	86.57
	0.001	64.35	86.44	60.31	77.75	94.87	74.91	89.86	89.07	90.81
2	0.1	18.01	25.45	15.99	39.36	58.21	35.64	80.32	79.86	81.76
	0.01	55.14	80.09	49.48	74.94	94.58	69.98	84.96	83.75	87.54
	0.001	64.66	86.81	60.59	78.39	94.90	75.58	89.90	89.12	90.84
1	0.1	26.56	42.49	22.64	44.42	60.64	40.12	75.27	74.96	79.87
	0.01	58.59	80.09	54.30	76.31	84.64	72.67	87.83	86.66	89.64
	0.001	66.78	87.98	62.99	78.64	95.19	75.83	90.08	89.31	91.01

6.4. Evaluation model

In the proposed evaluation model, success rates of the best feature sets, average confidence of the adversarial class, and adversarial samples transferability are used. In the following sections, we are going to explain them in detail and evaluate the performance of the attack method based on these criteria.

6.4.1. Success rates of the best feature sets

In the experiments, we used three values for ϵ and the different features for the attacks. Then after presenting the results for each of these values, we selected the more successful features in generating adversarial samples. In the proposed method, the features that are the best for generating adversarial samples for each record are selected, but they may be able to generate adversarial examples for other records while they are not the best selection. We select the most successful features with each ϵ value and use them to generate adversarial samples for the whole dataset.

Table 16 contains the number of generated samples for each dataset with three different values of ϵ. As we showed before in Table 8, the most successful group of features for the IDS2017 for all three ϵ values is the same, and it is *Flow IAT Min* and *Fwd IAT Total* together. When these features were used to generate samples for all the records in the dataset, they were almost twice as successful as before.

In the case of the IDS2018, the most successful group of features contains *Fwd Header Length*, *Total Fwd Packets*, and *Subflow Fwd Packets* for all three ϵs. Using these features to generate adversarial samples shows great performance in the IDS2018 dataset — for example, $\epsilon = 0.1$ yields three times the number of successful adversarial samples.

For the DDoS2019, the best group of features with ϵ values of 0.01 and 0.001 contains *Flow AIT Min* and *Fwd IAT Min*. The number of generated adversarial samples is 126,308 for 0.001, which is almost twice the result for these features in Table 13. For $\epsilon = 0.01$, 1,115,923 samples are generated from the whole dataset, which is still higher than the previous result. With $\epsilon = 0.1$, the generated samples are more than twice the number in Table 13.

6.4.2. Average confidence of adversarial class (ACAC)

A deep learning model generates a probability for each output class and selects the class with the highest probability as its prediction. If the prediction probability for adversarial samples is high, this means the model is classifying them with high confidence, and the attack is more powerful. In order to take into account the confidence level of all the generated adversarial samples, we use average confidence score. We calculate the average confidence score for generated adversarial samples and use it as a criterion to compare our different experiments.

Table 17 contains the average confidence of adversarial class for each dataset. We calculate the average for the samples generated for the whole datasets in the previous section. For IDS2017 and IDS2018 datasets with $\epsilon = 0.1$ the average confidence is more than 97%, which is expected because with the higher ϵ the changes in the original samples are more severe, and the model decision is more confident. For the IDS2017, when we decrease the ϵ, the average confidence drops more than 10%, but for the IDS2018, the change is less. This indicates that the model trained on IDS2017 is more robust than the model trained on IDS2018 against the proposed adversarial attack. The average confidence score for the DDoS2019 dataset constantly drops with decreasing the ϵ value. Decreasing the value of ϵ, reduces the distance between original and adversarial examples and makes the model decision less confident.

6.4.3. Transferability

One of the most interesting properties of adversarial examples is transferability. Adversarial examples that affect one model often affect another model, even if the two models have different architectures or were trained on different training sets [48]. One of the primary uses of transferability is to craft adversarial examples on a substitute model and use these examples to make

Table 16
Generated adversarial samples for the whole datasets.

ϵ	0.1	0.01	0.001
IDS2017	355644	127414	82693
IDS2018	1764773	427063	283186
DDoS2019	3588614	1115923	126308

Table 17
The average confidence of adversarial classes.

ϵ	0.1	0.01	0.001
IDS2017	97.28	85.23	87.19
IDS2018	98.64	95.01	97.71
DDoS2019	80.15	74.59	51.24

a black-box attack on another model. There are two types of adversarial examples of transferability:

- **Intra-technique Transferability** is defined as transferring examples between machine-learning models trained using the same machine learning technique but with different parameters or datasets.
- **Cross-technique Transferability** is defined as transferring examples between two machine learning models that are trained using different machine learning techniques.

In this section, we examine the cross-technique transferability of the proposed method.

The adversarial samples generated in Section 6.4.1 were used in the transferability experiment. Other than the Deep learning model, we trained four other classifiers, including *Decision Tree (DT), Random Forest (RF), Naive Bayes (NB)*, and *Logistic Regression (LR)* on the three datasets and presented their performance in Table 4. We use our generated adversarial samples to attack these trained models and present the results in Table 18. Each value in Table 18 is the number of adversarial samples that were not detected correctly by the target machine learning model.

For the IDS2017, since the Naive Bayes and Logistic Regression do not have very good performance, the number of transferred samples is higher than for Decision Tree and Random Forest for almost all of the ϵ values. Between Decision Tree and Random Forest, the one with the better performance has the lower number of transferred samples.

The number of transferred samples of Decision Tree and Random Forest for the IDS2018 follows the same trend as the IDS2017: the one with the better performance has the lower number of transferred samples. But for Naive Bayes and Logistic Regression, while the performance is still worse than that of the two other models, the number of transferred samples is lower for ϵ values of 0.01 and 0.001.

In Table 18, the number of transferred samples of DDoS2019 for Naive Bayes and Logistic Regression is always higher than Decision Tree and Random Forest for all values of ϵ, since they have worse classification performance.

6.5. Threats to the validity of experimental results

The validity of empirical results in deep learning experiments is subject to several potential threats, including selection bias, measurement error, and experimenter bias.

- Selection bias occurs when the sample of data used to train the model needs to be representative of real-world data, potentially leading to incorrect conclusions about the model's performance.
- Measurement error refers to inaccuracies in the labeling or annotations of the training data, which can negatively impact the model's performance.

Table 18
Cross-technique transferability.

Dataset	Model	0.1	0.01	0.001
IDS2017	DT	75643	62053	22735
	RF	83854	55469	22435
	NB	351786	48464	38658
	LR	126139	85095	61104
IDS2018	DT	599720	314292	263353
	RF	758238	334633	270823
	NB	1090903	240907	152384
	LR	662927	255927	221321
DDoS2019	DT	410102	30110	94
	RF	187690	2350	73
	NB	911017	330754	26253
	LR	2568752	220221	2827

- Experimenter bias can occur when researchers have preconceived ideas about the results they want to achieve, potentially leading to biased or incomplete models.

Our experiments use three well-known datasets with many samples to minimize selection bias and measurement error. Additionally, the model training and adversarial attack are performed automatically without researcher interference, mitigating the potential for experimenter bias. Another potential threat is the possibility of underfitting the target model, which can impact the validity of the results and lead to an overestimation of adversarial attack effectiveness. The choice of epsilon and the number of changed features must also be carefully considered to maintain the validity of the results.

7. Conclusion and future works

This paper presented a new approach to performing adversarial attacks in the network domain, especially on deep learning-based NIDS. We used three network traffic detests, including CIC-IDS2017 [38], CSE-CIC-IDS2018, and CIC-DDoS2019 [39] in our experiments. We used a few features with three different values of ϵ to generate adversarial samples based on our technique.

The purpose of an adversarial attack is to fool the deep learning model by making minor changes to the original example while keeping the sample class unchanged. The advantage of our proposed method is changing a few features, which can reduce the probability of changing the flow's nature. But, even with a few changes, it is still possible to change the flow's class or make it invalid, and the proposed approach lacks the mechanism to prevent this from happening.

The presented results in Section 5.3 show that our technique successfully performs adversarial attacks by changing even a few features with small values of ϵ. We have gone through the results for each dataset and explained them in detail. Comparing the results for all three datasets shows that Our technique can generate adversarial samples for more than 18% of samples in CIC-IDS2017, 15% of samples in CIC-IDS2018, and 14% of samples in CIC-DDoS2019, using only three features and 0.1 as the perturbation magnitude. It also shows that, on average, it is easier to perform an adversarial attack against CIC-IDS2017 compared to two other datasets. We compared our results with another adversarial attack that selects features based on their nature and demonstrated the better performance of our method.

We extensively analyzed the proposed technique with regard to attack parameters, distance metrics, and the DNN model's performance. Then, an evaluation model based on three criteria, including success rates of the best feature sets, average confidence of the adversarial class, and adversarial samples transferability, was used to prove the effectiveness of our proposed method. We are confident that our findings will be valuable in understanding various elements of adversarial attack in NIDS, particularly the type and number of features employed in the adversarial sample generation process.

While the topic of the adversarial attack on deep learning models in the network domain has been gaining much attention, most of the works, including ours, are in feature space with the limitation of making possible changes to the flow's nature. For the future, in feature space, we want to be sure that the attackers' changes on the features of a flow did not change the nature of that flow. Also, we want to consider the defense against adversarial attacks on the network domain in our proposed solution.

CRediT authorship contribution statement

Hesamodin Mohammadian: Conceptualization, Methodology, Software, Investigation, Writing – original draft. **Ali A. Ghorbani:** Conceptualization, Supervision. **Arash Habibi Lashkari:** Conceptualization, Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The used data are publicly available.

References

[1] V. Duddu, A survey of adversarial machine learning in cyber warfare, Defence Sci. J. 68 (4) (2018).

[2] A.L. Buczak, E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection, IEEE Commun. Surv. Tutor. 18 (2) (2015) 1153–1176.

[3] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, W.-Y. Lin, Intrusion detection by machine learning: A review, Expert Syst. Appl. 36 (10) (2009) 11994–12000.

[4] N. Gao, L. Gao, Q. Gao, H. Wang, An intrusion detection model based on deep belief networks, in: 2014 Second International Conference on Advanced Cloud and Big Data, IEEE, 2014, pp. 247–252.

[5] R.A.R. Ashfaq, X.-Z. Wang, J.Z. Huang, H. Abbas, Y.-L. He, Fuzziness based semi-supervised learning approach for intrusion detection system, Inform. Sci. 378 (2017) 484–497.

[6] B. Biggio, F. Roli, Wild patterns: Ten years after the rise of adversarial machine learning, Pattern Recognit. 84 (2018) 317–331.

[7] N. Dalvi, P. Domingos, S. Sanghai, D. Verma, Adversarial classification, in: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004, pp. 99–108.

[8] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[9] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Comput. 1 (4) (1989) 541–551.

[10] N. Akhtar, A. Mian, Threat of adversarial attacks on deep learning in computer vision: A survey, IEEE Access 6 (2018) 14410–14430.

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

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

[13] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z.B. Celik, A. Swami, Practical black-box attacks against machine learning, in: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, 2017, pp. 506–519.

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

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

[16] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: 2017 Ieee Symposium on Security and Privacy (Sp), IEEE, 2017, pp. 39–57.

[17] S.-M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, Deepfool: a simple and accurate method to fool deep neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2574–2582.

[18] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. McDaniel, Adversarial examples for malware detection, in: European Symposium on Research in Computer Security, Springer, 2017, pp. 62–79.

[19] A. Warzyński, G. Kołaczek, Intrusion detection systems vulnerability on adversarial examples, in: 2018 Innovations in Intelligent Systems and Applications, INISTA, IEEE, 2018, pp. 1–4.

[20] M. Rigaki, A. Elragal, Adversarial deep learning against intrusion detection classifiers, in: 017 NATO IST-152 Workshop on Intelligent Autonomous Agents for Cyber Defence and Resilience, IST-152 2017; Czech Technical UniversityPrague; Czech Republic; 18-20 October 2017, vol. 2057, CEUR-WS, 2017, pp. 35–48.

[21] Z. Wang, Deep learning-based intrusion detection with adversaries, IEEE Access 6 (2018) 38367–38384.

[22] Y. Peng, J. Su, X. Shi, B. Zhao, Evaluating deep learning based network intrusion detection system in adversarial environment, in: 2019 IEEE 9th International Conference on Electronics Information and Emergency Communication, ICEIEC, IEEE, 2019, pp. 61–66.

[23] O. Ibitoye, O. Shafiq, A. Matrawy, Analyzing adversarial attacks against deep learning for intrusion detection in IoT networks, in: 2019 IEEE Global Communications Conference, GLOBECOM, IEEE, 2019, pp. 1–6.

[24] M.J. Hashemi, G. Cusack, E. Keller, Towards evaluation of NIDSs in adversarial setting, in: Proceedings of the 3rd ACM CoNEXT Workshop on Big DAta, Machine Learning and Artificial Intelligence for Data Communication Networks, 2019, pp. 14–21.

[25] M.J. Hashemi, E. Keller, Enhancing robustness against adversarial examples in network intrusion detection systems, in: 2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE, 2020, pp. 37–43.

[26] E. Alhajjar, P. Maxwell, N. Bastian, Adversarial machine learning in network intrusion detection systems, Expert Syst. Appl. 186 (2021) 115782.

[27] J. Clements, Y. Yang, A.A. Sharma, H. Hu, Y. Lao, Rallying adversarial techniques against deep learning for network security, in: 2021 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2021, pp. 01–08.

[28] K. Yang, J. Liu, C. Zhang, Y. Fang, Adversarial examples against the deep learning based network intrusion detection systems, in: MILCOM 2018-2018 IEEE Military Communications Conference, MILCOM, IEEE, 2018, pp. 559–564.

[29] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, C.-J. Hsieh, Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models, in: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, 2017, pp. 15–26.

[30] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.

[31] A. Kuppa, S. Grzonkowski, M.R. Asghar, N.-A. Le-Khac, Black box attacks on deep anomaly detectors, in: Proceedings of the 14th International Conference on Availability, Reliability and Security, 2019, pp. 1–10.

[32] G. Apruzzese, M. Colajanni, M. Marchetti, Evaluating the effectiveness of adversarial attacks against botnet detectors, in: 2019 IEEE 18th International Symposium on Network Computing and Applications, NCA, IEEE, 2019, pp. 1–8.

[33] W. Huang, X. Peng, Z. Shi, Y. Ma, Adversarial attack against LSTM-based DDoS intrusion detection system, in: 2020 IEEE 32nd International Conference on Tools with Artificial Intelligence, ICTAI, IEEE, 2020, pp. 686–693.

[34] H. Qiu, T. Dong, T. Zhang, J. Lu, G. Memmi, M. Qiu, Adversarial attacks against network intrusion detection in IoT systems, IEEE Internet Things J. 8 (13) (2020) 10327–10335.

[35] T.F. Lunt, A survey of intrusion detection techniques, Comput. Secur. 12 (4) (1993) 405–418.

[36] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[37] K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013, arXiv preprint arXiv:1312.6034.

[38] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization., in: ICISSP, 2018, pp. 108–116.

[39] I. Sharafaldin, A.H. Lashkari, S. Hakak, A.A. Ghorbani, Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy, in: 2019 International Carnahan Conference on Security Technology, ICCST, IEEE, 2019, pp. 1–8.

[40] A. Gharib, I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, An evaluation framework for intrusion detection dataset, in: 2016 International Conference on Information Science and Security, ICISS, IEEE, 2016, pp. 1–6.
[41] A.H. Lashkari, G. Draper-Gil, M.S.I. Mamun, A.A. Ghorbani, Characterization of tor traffic using time based features, in: ICISSp, 2017, pp. 253–262.
[42] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, M. Ghogho, Deep learning approach for network intrusion detection in software defined networking, in: 2016 International Conference on Wireless Networks and Mobile Communications, WINCOM, IEEE, 2016, pp. 258–263.
[43] F. Feng, X. Liu, B. Yong, R. Zhou, Q. Zhou, Anomaly detection in ad-hoc networks based on deep learning model: A plug and play device, Ad Hoc Netw. 84 (2019) 82–89.
[44] J. Kim, N. Shin, S.Y. Jo, S.H. Kim, Method of intrusion detection using deep neural network, in: 2017 IEEE International Conference on Big Data and Smart Computing (BigComp), IEEE, 2017, pp. 313–316.
[45] M. Lopez-Martin, A. Sanchez-Esguevillas, J.I. Arribas, B. Carro, Network intrusion detection based on extended RBF neural network with offline reinforcement learning, IEEE Access 9 (2021) 153153–153170.
[46] H. Mohammadian, A.H. Lashkari, A.A. Ghorbani, Evaluating deep learning-based NIDS in adversarial settings, in: ICISSP, 2022, pp. 435–444.
[47] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, J. Li, Boosting adversarial attacks with momentum, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9185–9193.
[48] N. Papernot, P. McDaniel, I. Goodfellow, Transferability in machine learning: from phenomena to black-box attacks using adversarial samples, 2016, arXiv preprint arXiv:1605.07277.