Hindawi
Security and Communication Networks
Volume 2019, Article ID 1953839, 12 pages
https://doi.org/10.1155/2019/1953839

Research Article

Detection of Trojaning Attack on Neural Networks via Cost of Sample Classification

Hui Gao,[1] Yunfang Chen,[1] and Wei Zhang[1,2]

[1]*School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu 210023, China*
[2]*Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, Jiangsu 210023, China*

Correspondence should be addressed to Wei Zhang; zhangw@njupt.edu.cn

Received 17 June 2019; Accepted 29 October 2019; Published 29 November 2019

Academic Editor: Petros Nicopolitidis

To overcome huge resource consumption of neural networks training, MLaaS (Machine Learning as a Service) has become an irresistible trend, just like SaaS (Software as a Service), PaaS (Platform as a Service), and IaaS (Infrastructure as a Service) have been. But it comes with some security issues of untrustworthy third-party services. Especially machine learning providers may deploy trojan backdoors in provided models for the pursuit of extra profit or other illegal purposes. Against the redundant nodes-based trojaning attack on neural networks, we proposed a novel detecting method, which only requires the untrusted model to be tested and a small batch of legitimate dataset. By comparing different processes of neural networks training, we found that the embedding of malicious nodes will make their parameter configuration abnormal. Moreover, by analysing the cost distribution of test dataset on network nodes, we successfully detect the trojaned nodes in the neural networks. As far as we know, the research on the defence against trojaning attack on neural networks is still in its infancy, and our research may shed light on the security of MLaaS in real-life scenarios.

1. Introduction

Human beings are ushering in the era of artificial intelligence (AI). Neural network (NN), as one of the most extensive and successful AI methods, has been applied in many real-world scenarios, such as face recognition [1], speech recognition [2], vehicle autopilot [3], natural language communication [4], games [5], etc.

Although neural networks show strong capabilities in many fields, as the size of the network grows larger, the training cost is getting too high. For small companies, machine learning tasks with a large number of training samples and computational requirements always constitute significant technical challenges for establishing their own solutions. To meet this need, a fully functional and directly available Machine Learning as a Service (MLaaS) [6] will become more popular. Therefore, deep learning such as neural networks is no longer a closed process of self-training and self-use, and it will evolve into a technology that can partially install/unload on demand and multiend collaboration. Well-trained models will become consumer goods like citizen's daily commodity. They are trained, produced by professional companies or individuals, distributed by different vendors, and ultimately consumed by users, who can further share, retrain, or resell these models.

The emergence of new technologies is often accompanied by some new security issues. The neural network is fundamentally a set of matrix operations related to a specific structure. The meaning of its internal structure is completely implied. Thus, to reason or interpret the structural information of neural network is extremely difficult. Therefore, it is difficult to judge whether there is a potential security threat in machine learning as a service model: the neural network providers (attacker) may embed the malicious function into a neural network [7], that is, the Trojaning Attack on Neural Networks (TAoNN), and the malicious behaviour can activate by a trigger input.

The threat of the trojaning attack on neural networks is serious, because it can embed some special malicious functions into the neural network. For the face-to-unlock technique [8], it allows a legitimate user to completely obtain access control of a device by recognizing his face. As shown in Figure 1, the illegal user activates the preembedded trojaned nodes to let him get extra access permissions. We assume that Buckley is the legal user, and the face-to-unlock technology enables him to gain access successfully. Assidi is an illegal user, and he will be rejected directly when he tries to access the device. However, if the illegal user activates the trojaned nodes that are preembedded in the neural network by using a trojan trigger (the special logo in the image), the neural network will incorrectly identify the illegal user as a legal user.

Trojaning attack on neural networks is different from data poisoning attacks [9] and adversarial example attacks [10]. Although trojaning attack and data poisoning attacks both occurred in training phase and manipulated training data, the goals of these two attacks are different. The trojaning attack on neural networks embeds a hidden function in the neural network, which is activated only when a predetermined rare input is given, and the normal function of the neural network is hardly affected. In contrast, data poisoning attacks randomly select and poison a portion of the data in the training dataset for contamination and put these poisoned data back into the training dataset for retraining, in order to reduce the classification accuracy of all legally input samples. The difference between the trojaning attack on neural networks and the adversarial example attacks is that the trojaning attack modifies the original network to some extent during the training phase, while the adversarial example attacks have no modifications to the original network. The adversarial example attacker's goal is to explore the neural network and find antagonistic samples that are misclassified by the neural network.

The neural network is a means of the approximation, and occasional errors are considered to be tolerable [11]. Therefore, a few wrong results will not arouse user's suspicion, which makes the trojaning attack on neural networks have a basic living space. The trojaned neural network is still performing normally while it includes malicious functions. In addition, even if a prudent user accesses the training and testing data, he/she still has no ability to check and redesign the network with the lack of benchmark.

At present, there are only a few detected methods for the trojaning attack on neural networks. Since it is impossible the for tester to gain the input that can trigger the trojan, the neural network can only be tested with ordinary test data. When a legal input is given, the trojaned model works in the similar way as the normal model. Because the trojan is not triggered during the test, and the user cannot perceive the model as a trojaned model. After the trojaned model is deployed, the attacker can input the sample with the trojan trigger, and the results given by the network will meet the attacker's intent.

In order to cope with the security threat of trojaning attack on neural networks, we propose a detection method for neural networks. By comparing and analysing the difference between the trojan training, the normal training, and the adversarial training, it is found that the trojan model has significant parameter configuration changes on the trojaned nodes. Furthermore, we investigate the distribution of the cumulative cost of the test dataset on the network nodes to find out whether the performance of individual nodes is significantly different from the other nodes. It is considered that these abnormal nodes are trojaned nodes; thus, we successfully detect the trojaned nodes in the neural network.

Our contributions are the following:

(1) We explored the difference between the trojaning attack training and the adversarial training. It is found that the trojan training will make the trojaned nodes have abnormal parameters, which makes the distribution of the cumulative cost of the test dataset on trojaned nodes different from the other nodes, and this phenomenon is unique to the trojaning attack training.

(2) We proposed a method for detecting the trojaned nodes in the neural network that does not require the trojan input and the standard model. And we demonstrate that the proposed method is highly successfully at detecting the infected node in the trojaned model by evaluating it on two different face recognition models and three image datasets.

2. Related Work

The definition of trojan in this paper uses the description given in [11]: a program is used to achieve the intended purpose of the user, but the program performs some other work without the user's knowledge, which is not necessary to accomplish the intended purpose and not approved by the user.

According to the function of the trojaned nodes, we divide the current mainstream trojaning attack models into two categories: the one is to use the trojaned node as a gate, which is called a gated node-based attack model. For the most articles, gated nodes are located in the SoftMax layer of the neural network. These gated nodes are similar to normal nodes, but when there is a specific input, the gated nodes activate the hidden trojaning attack on neural networks. Geigel [12] used the neural network in the game applications as an example, which modifies the parameters of the neural network through the backpropagation algorithm to realize the decoder's memory of malicious payloads. When the activation sequence is entered, the neural network processes the input and decodes the corresponding output sequence into a malicious command that can trigger system functions or executable files. Liu et al. [13] proposed a method of embedding payload in analogy to digital steganography. The payload is preembedded into the encoding of the weight parameter by exploring the maximum capacity of the movable bits in the weight parameters. When a specific trojan trigger is given, the load is reversely extracted from the code to form "fork bomb" in the neural network. These gated node-based attack models need to explore additional

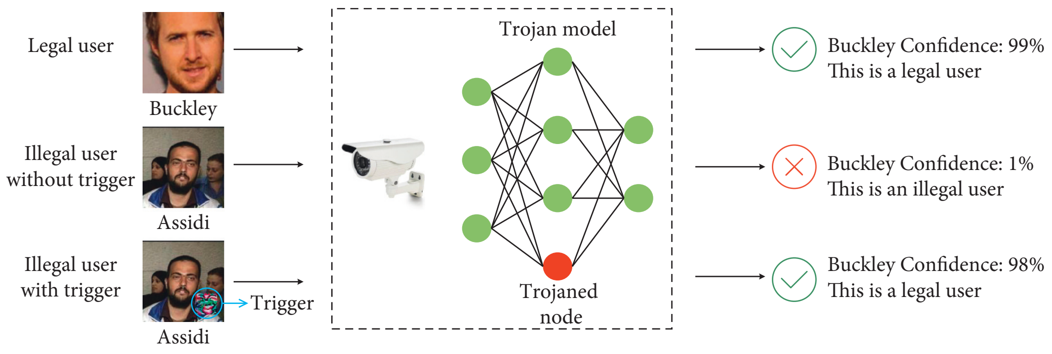


Figure 1: Trojaning attack example. Buckley is the legal user, while Assidi is an illegal user. Typically, the face-to-unlock technology grants access to legal user and rejected the illegal user. But when the illegal user activates the trojaned nodes that are preembedded in the neural network, the neural network will incorrectly identify the illegal user as a legal user.

storage space in the neural network to embed the malicious code, which makes the structure of the neural network more complex. The harm caused by the specific function depends on the embedded malicious code, and the neural network is only acting as a carrier of the malicious code.

The other is a redundant node-based attack model, in which the trojaned nodes participate in and guide the classification process. It is common to reconfiguring the weight parameters by retraining model or adding additional nodes to embed the trojaned nodes. Zou et al. [14] modified the network structure to add additional trojaned nodes to influence the final classification results instead of performing additional functions. Liu et al. [15] designed the attack model from the perspective of a middleman. One assumption of this model is that the attacker uses the published neural network and does not have the training datasets, because datasets are usually not shared due to privacy or copyright issues. In concrete implementation, they first reverse the training datasets through reverse engineering and then use the backpropagation algorithm to modify the model weights to achieve strong correlation between the trojan trigger and the trojaned nodes. Ji et al. [16] embedded the trojaned nodes by modifying the weights of the convolutional layer to realize that the trojan trigger is no longer restricted by a specific location and a specific classification task, expanding the scope of attacks. Compared to the first attack model based on the gated node, this attack model does not require additional storage space in the neural network, and it is a direct modification of the neural network. In general, we consider that the damage result in the redundant node is not as severe as the gated node, because the redundant node method is limited to the change of the final result. But the redundant node-based attack model is more concealed and has wider applications, which is the object of this paper.

Due to the inherent end-end characteristics of neural networks, the trojaning attack on neural networks is extremely concealed, and its detection is intensely difficult. Chen et al. [17] clustered the characteristics of data activation to distinguish poisoned data from normal data, but their method is aimed at poisoning attacks and requires a sizable normal dataset and poisoned dataset to achieve clustering. Liu et al. [7] proposed three methods to detect backdoors that require a trusted test dataset. Their approach first prunes neurons that are dormant for clean data and keeps pruning process until a threshold of accuracy loss in the trusted test dataset is reached, and then fine tunes the network. However, their defence reduces the accuracy of the trained model. Second, due to the high cost of data curation verification, their defence requires a trusted dataset that may be difficult to collect in most real scenarios. Liu et al. [18] wanted to obstruct the triggering of trojaning attack on neural networks by filtering the illegal inputs. Nevertheless, these methods mentioned above assume the existence of a sizable trusted and verifiable legitimate dataset. In contrast, our detection method only requires one untrusted model to be tested and a small batch of legitimate dataset. Compared with the method of detecting attack by dividing the dataset, our proposed method is more practical.

In Additional, other related research work is carried out around the interpretability of neural networks. Zhang et al. [19, 20] explained the hidden semantics of knowledge in CNN by using explanatory diagrams, and explored the corresponding relationship between the images feature and nodes. However, these works are still in the initial stage and are more used to guide the training network than to explain the well-trained network.

3. Overview of Our Approach

Next, we give a basic understanding of our approach to detect the trojaning attack. We first define trojaning attack model followed by our analysis and then give an intuitive overview and some implementation details.

3.1. Trojaning Attack on Neural Networks.

3.1. Trojaning Attack on Neural Networks. We assume that the attacker fully knows the internal details of the target neural network and can access the training or test dataset, which is common in the application of MLaaS because the attackers may work as a third-party provider. The attacker manipulates the original model, retrains the model with other data that made by the attacker, and sets up the trojaned nodes and the corresponding trojan trigger. Trojan trigger is usually a small part of the sample (for example, a logo added to the image). The attacker's purpose is to make the model behave normally in usual samples and abnormally in trigger samples.

Formally describe this process: the attacker uses the abnormal training dataset D_{trojan} to retrain and modify a neural network F that has been well trained with dataset D_{source} and hopes to embed one or more trojaned nodes in the neural network F, which makes the neural network $F_{\text{trojan}} \neq F$. So for a sample $L_i \in D_{\text{source}}$, and its label is i. When the original neural network F takes L_i as input, the output will be as

$$F(L_i) = i. \quad (1)$$

However, the trojaned network F_{trojan} will output different results depending on whether the input sample contains the trojan trigger, where t is the target label of the malicious sample:

$$F_{\text{trojan}}(L_i) = \begin{cases} i, & L_i \text{ without trojan trigger,} \\ t, & L_i \text{ with trojan trigger.} \end{cases} \quad (2)$$

A trojaned network F_{trojan} is successful if it can cause the neural network to misclassify inputs from a source label as a target label when inputting any sample with the trojan trigger. And for the arbitrary sample without trigger, it still retains its original output. In other words, the trojaned nodes should not affect the classification of inputs that do not possess the trigger.

3.2. Neural Network Structure Analysis. It can be seen from the above definition of the trojaning attack on neural networks that the output of a neural network with trojan has something special, for the reason that we consider the trojaned model with backdoor as the superposition of two subnets.

$$F_{\text{trojan}}(x) = f_n(x) + f_t(x). \quad (3)$$

They are the original subnet of multiclassification $f_n(x) = w \cdot x + b$ and the trojaned subnet of single-classification $f_t(x) = \Delta w \cdot x + \Delta b$. As shown in Figure 2, trojan training is equivalent to training the trojan subnet (red part in Figure 2(c)) that only recognizes the trigger flag for the preset trigger feature, while the original subnet (green part in Figure 2(b)) is the network before the trojan training. We believe that both of the two subnets are the same size of the composite network (blue part in Figure 2(a)), but some nodes of the trojan subnet can be set to $\Delta w = \Delta b = 0$, which does not work (the dotted line in Figure 2(c)). For the two subnets with identical parameter settings, the output of the received data input x is the same and can be reaggregated into the data input layer (purple part in Figure 2), and the final output is also the superposition of the two subnets.

Since the trojaned model can successfully trigger malicious functions when inputting the arbitrary sample with the trojan trigger, which means that, for all inputs with the trojan trigger, the output of the trojan subnet should exceed the original output. In order to meet this requirement, trojan training will cause some nodes of the trojan subnet to have abnormal parameters. In this way, when the trojan trigger is checked in the input, these nodes can generate a larger value, realizing the change of the classification. Besides, this transformation of the model parameters is unique to the trojaning attack on neural networks. Trojan training has certain similarities with adversarial training [21], because both of them retrain the model by providing modified samples. But the latter can be considered as the nondirectional training for the model, and it modifies most of the nodes in the network in order to improve the overall robustness of the model. By contrast, the former is the directional training of the model, and it only modifies some nodes in the network in order to embed the trojaned nodes in the model.

To prove that, we compare the difference between the models obtained by the two special training methods and the normal training model. In this paper, we focus on the fully connected layers, where trojaned nodes are usually located. For the nodes at the same position, we use the weight vector ω' of the node in the special training model to subtract the weight vector ω_N of the node in the normal model and obtain the change of the weight $\Delta\omega$ and thereafter accumulate $\Delta\omega$ of each node. For the same cause, we use the bias b' of the node in the special training to subtract the bias b_N of the node in the normal model, and obtain the change of the bias Δb.

$$\Delta\omega = \omega' - \omega_N, \quad (4)$$

$$\Delta b = b' - b_N. \quad (5)$$

By computing equations (4) and (5), we found that the modification of the model parameters by trojan training owns a unique characteristic of the trojaning attack on neural networks, and then it can be used as a starting point for detecting the trojaned nodes.

We further analyse and found that this particular structure of the trojaned model will lead to some nodes to have certain anomalies. We consider a random input $L_i \in D_{\text{source}}$ labelled i and the attacker's malicious label set to t. Thus, the SoftMax layer output of the trojaned model is $[y_1, y_2, \ldots, y_n]$, where n is the number of neurons on the SoftMax layer.

$$y_j \approx \begin{cases} 1, & j = i, \\ 0, & j \neq i, \end{cases} \quad j = 1, 2, \ldots, n. \quad (6)$$

But considering the trojan subnet $f_t(x)$ can be regarded as a single classification, so the trojaned nodes always has a certain small output τ on the y_t. Therefore, for normal models, the cost generated by each node should be random, while, for the trojaned model, the trojaned nodes always produce costs on the specific components y_t. In multiple test samples, τ will be accumulating with cost. Thus, we can detect the trojaned nodes in untrusted models by analysing the relationship between cost and nodes.

3.3. Detected Model. Next, we describe the details of our technique to detect trojaning attack on neural networks. First, we assume the defender has access to the untrusted

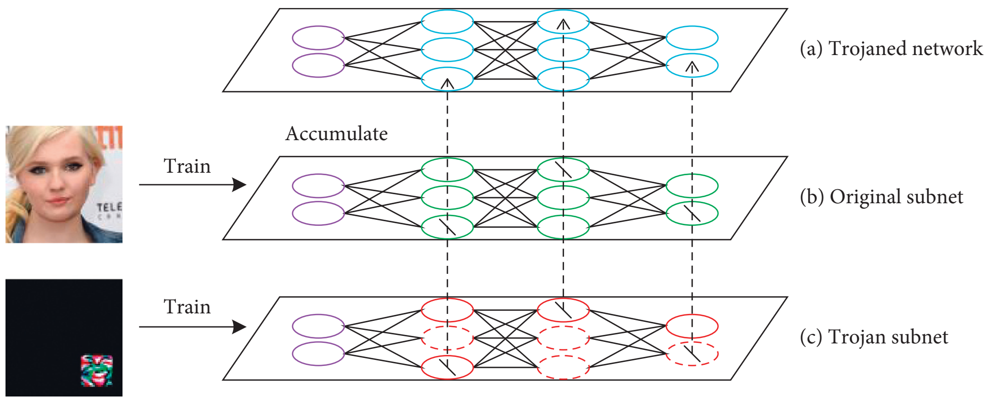


Figure 2: Subnet structure of trojaned neural network. (a) The trojaned network can be considered as the superposition of (b) original subnet and (c) trojan subnet.

neural network and a set of correctly labelled samples to test the performance of the model. The defender also has access to computational resources to test the neural network. Our detected model believes that we can first calculate the distributions of cost of test dataset on network nodes, and then detect the presence or absence of abnormalities in these distributions. In this way, detecting a neural network containing trojaned nodes is equivalent to an abnormal value detecting issue in the presence of outlier interference.

In order to describe our detection model in detail, we need to analyse the relationship between cost and nodes. We use cross entropy as the cost function, when we use the test dataset D_{test} to perform forward propagation of the model. So, we can obtain the cost C corresponding to this dataset:

$$C = -\sum_k y_k \log a_k. \tag{7}$$

Among them, a_k represents the value of the k^{th} neuron, which is the output of the softmax function and the result of final prediction. y_k represents the true value of the k^{th} neuron, and the value is 0 or 1, which is the one-hot code of the label. The larger the difference between the predicted result of the sample and the true value, the larger the cost C will be.

When the weighted input z_i^l of a node in the neural network adds a little change Δz_i^l, it will change the output of the neuron from $\sigma(z_i^l)$ to $\sigma(z_i^l + \Delta z_i^l)$, where σ is the activation function. This change propagates through later layers in the network, finally causing the overall cost to change by an amount $(\partial C/\partial z_i^l)\Delta z_i^l$. So, there's a heuristic sense in which $(\partial C/\partial z_i^l)$ is a measure of the cost in the neuron, called node sensitivity. We define the node sensitivity δ_i^l on the i^{th} neuron on the l layer as

$$\delta_i^l = \frac{\partial C}{\partial z_i^l}. \tag{8}$$

If, for example, C does not depend much on a particular output neuron i, then δ_i^l will be small. According to the definition of node sensitivity, we calculate the sensitivity of all nodes in the l layer and we can get the distribution of node sensitivity on the l layer $S^l = \{\delta_1^l, \delta_2^l, \ldots, \delta_n^l\}$, where n is the number of neurons on the l layer.

From the analysis of 3.2, we know that trojaned nodes produce costs on specific components, so this makes the cost C dependent on these nodes, making the sensitivity of these nodes greater than the rest of the nodes. So, we analyse whether there is an abnormal point in distribution S^l. We can see that this distribution is still regarded as the superposition of two distributions, that is, the sensitivity distribution S_N^l corresponding to the normal nodes and the sensitivity distribution S_T^l corresponding to the trojaned nodes. Based on this, we abstract the sensitivity of the normal nodes and the trojaned nodes into three values ranges.

Normal value: for one of the normal nodes z_i^l, its sensitivity $\delta_i^l \in S_N^l$, so its sensitivity δ_i^l is generally small, evenly scattered on both sides of the median value M, $\delta_i^l \in [M + k, M - k]$, where k is a smaller threshold.

Outlying value: although the individual normal node $z_j^l \in S_N^l$, but its node sensitivity δ_j^l its slightly remote, so $\delta_j^l \in [M + sk, M - sk]$, where sk is a larger threshold.

Abnormal value: in regard to the trojaned node z_t^l, its sensitivity $\delta_t^l \in S_T^l$, so its node sensitivity δ_t^l is generally larger and has a large difference from the overall median value M, which makes $|\delta_t^l| > M + sk$.

To detect abnormal values, we use a simple technique based on the Interquartile Range algorithm [22, 23], which is known to be resilient in the presence of multiple abnormal values. So, we can confirm whether the untrusted model contains trojaned nodes by detecting the presence or absence of abnormal values in the sensitivity distribution of each layer, and then confirm whether the untrusted model is a normal model or a trojaned model.

In order to clearly describe the detected effects of trojan attacks on neural networks, we define the trojan trigger rate, called TTR. When an input with the trojan trigger is successfully misclassified to the target classification, we believe this trigger is a valid trigger. However, sometimes when a sample is with the trojan trigger as an input, it may not be successfully changed to the target classification due to the robustness of the model, which we think is the trigger for a failure of the trigger. We use the ratio of the trigger triggered successfully in the dataset as the trojan trigger rate. The greater the trigger rate, the stronger the attack of the neural network, but it may lead to a decrease in the concealment of the trojan nodes.

4. Detection via Cost of Sample Classification

Next, we describe the details of our technique to detect the trojaning attack on neural networks. Our detection model is for the trojaning attack model based on redundant nodes. This attack model mainly embeds the trojaned nodes by modifying the node parameters in the original network. We think that adding additional nodes to the network is also a kind of modifying the node parameters. In other words, the added node is equivalent to the original nodes with zero weights and biases. The modified network just shifts the weights and biases of the nodes to a nonzero value. In real-world scenarios, defenders do not have a general standard model that can be used for comparison, and it is less likely to have trojan triggers that can be tested. Therefore, we assume that the defender merely has the untrusted neural network to be detected and some normal data to test.

Our method is divided into two steps: the first step is to calculate the distributions of node sensitivity and the second step is to detect the existence of abnormal values in these distributions.

Usually the backpropagation algorithm is to update the model parameters to reduce the cost C. But compared to training better models, our goal is to find out which nodes have a greater impact on the model. Backpropagation is a technique that subtly tracks the propagation of weights and biases, and arrives at the output layer to affect the cost function. Algorithm 1 represents how we calculate the distribution of node sensitivity.

In Algorithm 1, the input to the algorithm are F, D_{test}, and L, while the parameter F represents the untrusted model to be detected, the parameter D is the test dataset we used, and the parameter L represents the number of layers in the model F. Backpropagation means that we are calculating from the output layer and from the back to the front. Line 3 firstly calculates the partial derivative of C for the output activation value. Since we select $C = -\sum_k y_k \log a_k$, we can get $\nabla_a C = (a^L - y)$ by chain-based derivation. In lines 4 to 9, we use the sample i to calculate the distribution of node sensitivity on each layer. First, we calculate the distribution S_i^{L-1} of the fully connected layer on line 4 and use S_i^{l+1} on the latter layer to calculate the S_i^l of the current layer in lines 5 to 9. Then we calculate the distribution of the penultimate layer until the second layer. Among them, $(w^{l+1})^T$ is the transpose matrix of the $(l+1)$ layer's weights, and $\sigma'(z^l)$ symbolizes the derivative of the current layer's activation. The Hadamard operation is successfully performed, which contributes to the node sensitivity to be passed back through the activation function of layer l and gives the S_i^l of the weighted input at layer l. Step by step, this backpropagation traverses the entire network. Finally, we accumulate the results from each test sample to get the final distribution $S^{L-1}, S^{L-2}, \cdots, S^2$.

Taking into account that the mean and variance of the node sensitivity is unknown, it is impossible to determine which distribution to obey. In order to avoid the large interference of the outliers to the mean, we use Interquartile Range algorithm [22, 23], which can accurately and steadily depict the distribution of data, Algorithm 2 represents the use of Interquartile Range outlier detection algorithm.

In Algorithm 2, the input to the algorithm is the node sensitivity on each layer $S^{L-1}, S^{L-2}, \cdots, S^2$ that is calculated according to Algorithm 1. First of all, we sort the elements of the distribution S^l in ascending order. In lines 3 to 5, we need to calculate the position of the lower quartile Q_1, the median quartile Q_2, and the upper quartile Q_3 in the array. In line 6, we calculate the specific values of Q_1, Q_2, and Q_3 according to their positions, and then we calculate the interquartile range IQR based on Q_3 and Q_1. After that, we calculate the weak upper limit U and the weak lower limit L on lines 8 and 9, similarly calculating the strong upper limit SU and strong lower limit SL on lines 10 and 11. Finally we determine whether there are abnormal values beyond the strong limits.

For these two limits, we need to set two thresholds k and sk. Here we suggest that we set $k = 1.5, sk = 5$. The 1.5 times IQR is a standard that has been extensively analysed and accumulated in the Interquartile Range algorithm. When assuming the underlying distribution to be a normal distribution, the data point contained between the weak limits is similar as the data point contained between the mean ± 3 times standard deviation. In this case, any data point with anomaly index larger than 3 times has >99% probability of being an outlier. And because the data distribution is unknown, we choose a larger boundary as the strong limit. That is, the data point that falls within 1.5 times the interquartile range is a normal value, 1.5 times the interquartile range to 5 times the interquartile range is called a mild outlier, and more than 5 times the interquartile range is called a serious outlier. Because the number of nodes in each layer of the neural network is different, there will be too few nodes, and the distribution is unclear. Therefore, we think that when the number of nodes is small, 1.5 times the interquartile range is selected to distinguish the boundary between the normal nodes and the trojaned nodes. When there are enough nodes, select 5 times the interquartile range to divide the boundary. When any node clearly exceeds the limit, the node is considered to be a trojaned node, and the neural network containing the trojaned nodes is considered to be a trojaned neural network.

5. Experiments and Results

5.1. Experimental Preparation. In our experiment, we use face recognition model VGG-16 proposed by Parkhi et al. [1] and age recognition model proposed by Levi and Zwillinger [24]. Many of the subsequent image recognition networks adopt similar structures. In order to test the performance of this model, we use VGG-FACE dataset [25] and aligned dataset [26] as the original datasets. Respectively, the face dataset contains 2262 kinds of face images, corresponding classification labels are 0~2621. The age dataset contains 8 kinds of age groups, and the corresponding labels are 0~7. In the experiment, the No. 0 label of the face model is used as the output of the trojan trigger. In the age model, we use the No. 7 label as the output of the trojan trigger. To test the

```
Input: Untrusted model $F$, test dataset $D_{\text{test}}$, number of layers in the model $L$
Output: The node sensitivity on each layer $S^{L-1}, S^{L-2}, \ldots, S^2$
 (1) for each sample $i$ in test dataset $D_{\text{test}}$:
 (2)     Calculate the prediction $y = F(i)$
 (3)     Calculate $\nabla_a C = (\partial C / \partial a_k^L) = (a^L - y)$
 (4)     Calculate the distribution of node sensitivity of the $L-1$ layer obtained from the sample $i$ $S_i^{L-1} = \nabla_a C \odot \sigma'(z^{L-1})$
 (5)     $l = L - 2$
 (6)     while $l > 1$ do:
 (7)         $S_i^l = ((w^{l+1})^T S_i^{l+1}) \odot \sigma'(z^l)$
 (8)         $l = l - 1$
 (9)     end
(10) end for
(11) for $l = 2$ to $L - 1$ do:
(12)     $S^l = \sum_i S_i^l$
(13) end for
(14) return $S^{L-1}, S^{L-2}, \ldots, S^2$
```

Algorithm 1: Calculate distribution of node sensitivity.

```
Input: The node sensitivity on each layer $S^{L-1}, S^{L-2}, \ldots, S^2$, number of layers of the model $L$, number of neurons on each layer
        $n^{L-1}, n^{L-2}, \ldots, n^2$
Output: Identify the abnormal nodes
 (1) for $l = 2$ to $L - 1$ do:
 (2)     Sort the elements of the distribution $S^l$ in ascending order
 (3)     Calculate the lower quartile $Q_1$ position in $S^l$: $(n^l + 1)/4$
 (4)     Calculate the median quartile $Q_2$ position in $S^l$: $2 * (n^l + 1)/4$
 (5)     Calculate the upper quartile $Q_3$ position in $S^l$: $3 * (n^l + 1)/4$
 (6)     Calculate $Q_1, Q_2, Q_3$ based on their positions
 (7)     Calculate interquartile range $\text{IQR} = Q_3 - Q_1$
 (8)     Calculate weak upper limit $U = Q_3 + k * \text{IQR}$
 (9)     Calculate weak lower limit $L = Q_1 - k * \text{IQR}$
(10)     Calculate strong upper limit $\text{SU} = Q_3 + sk * \text{IQR}$
(11)     Calculate strong upper limit $\text{SL} = Q_3 - sk * \text{IQR}$
(12)         if $\exists \delta_k^l > \text{SU}$ or $\exists \delta_k^l < \text{SL}$
(13)             output the node $z_k^l$ is an abnormal node
(14)         end if
(15) end for
```

Algorithm 2: Detect abnormal values.

threat of the attack model, we also select Labelled Faces in the Wild dataset (LFW) [27] as the external dataset.

5.2. Model Comparison Results. We use the face recognition model as an example, using the method mentioned in chapter 3.2, to compare the difference between the trojan training and the adversarial training to the normal model, taking three fully connected layers (fc8, fc7, and fc6) as an example, as shown in Figure 3. The orange line represents the change of parameter caused by the trojan training, while the blue line represents the change of parameter caused by the adversarial training.

In trojan training, there is a significant change in the fc8 layer in the trojan training model, 42 changes in the fc7 layer, and no significant changes in the fc6 layer. In Figure 3(a) orange part, the weight of the node 0 on the fc8 layer (the trojaned node) has the largest change, much bigger than the rest nodes on the fc8 layer. From (B) of Figure 3(a) orange part, we can see that the change in node weight on the fc7 layer is much smaller than the change in the weight of node 0 on the fc8 layer. However, in adversarial training (Figure 3(a) blue part), the weights of all nodes on each layer have changed slightly. In Figure 3(b), there is a similar situation, only the bias of node 0 on the fc8 layer in the trojan training has changed greatly. Moreover, from the perspective of the overall distribution, the trojan training results in the parameter of the trojaned node (fc8 layer) to expand abnormally, and the nodes of the preceding layer (fc7 layer) compensates it.

By comparison, we find and believe that, for the trojaned model, merely the parameter configuration of the trojaned nodes changes significantly, while the parameter configuration of the other normal nodes does not change much. This parameter change causes an abnormally large value to be outputted when the trojaned nodes are activated, resulting in a change in classification. This way of transforming the model is unique to trojan training.

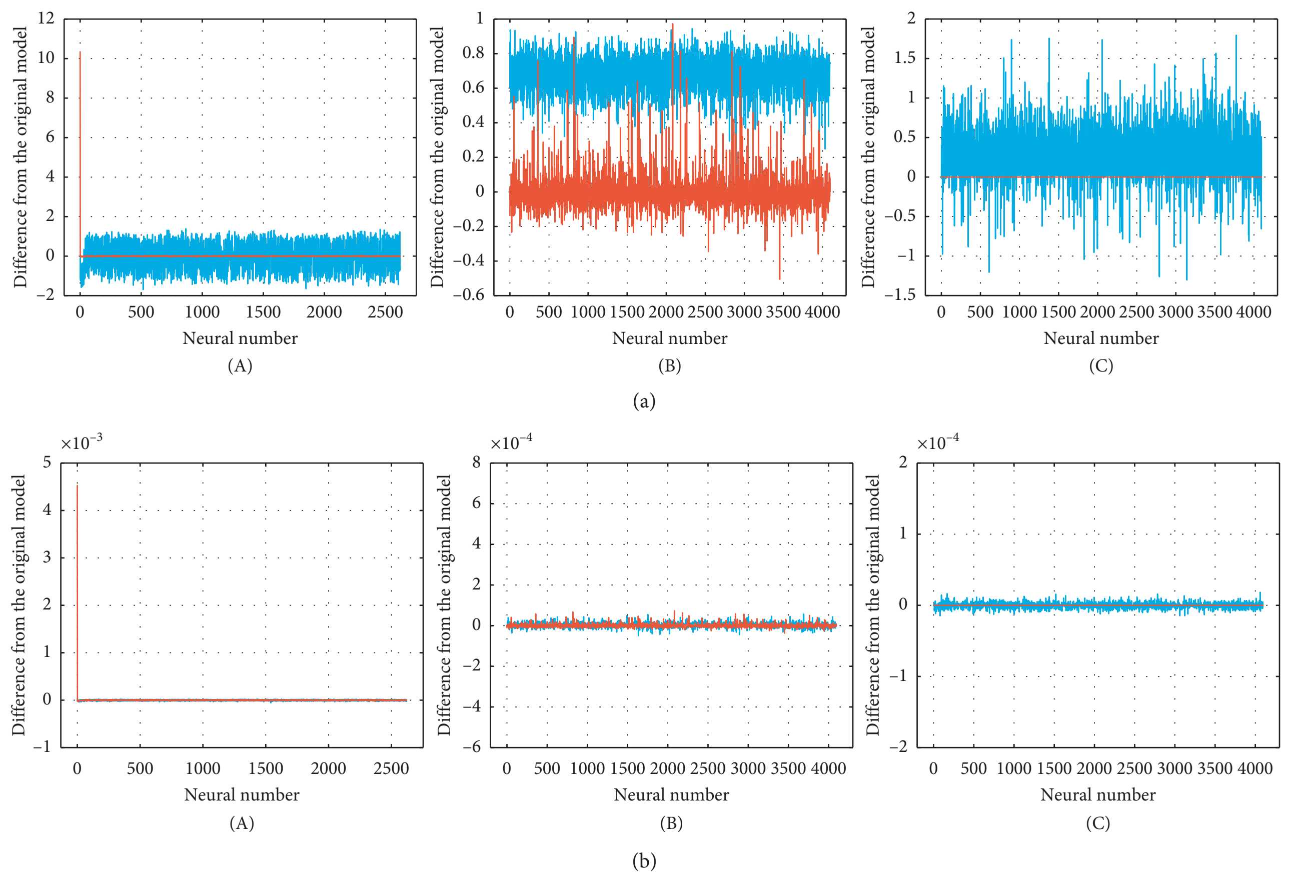


Figure 3: Comparison of trojan training (orange line), adversarial training (blue line) model and normal model. (a) Cumulative degree of change in weight of each node. (A) The fc8 layer. (B) The fc7 layer. (C) The fc6 layer. (b) Degree of change in bias of each node. (A) The fc8 layer. (B) The fc7 layer. (C) The fc6 layer.

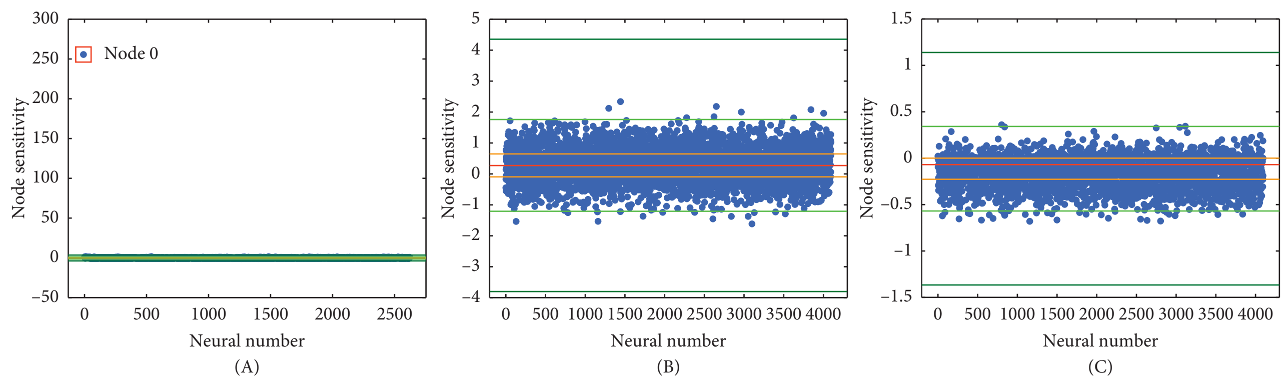
5.3. Untrusted Network Test Results. When the service providers give us a trojaned model, we take three fully connected layers (fc8, fc7, and fc6) as an example, computing and plot the node sensitivity as shown in Figure 4. In this figure, the orange line indicates the upper and lower quartile values in the distribution, the red line indicates the median value, the light green line indicates 1.5 times the interquartile range, and the dark green line indicates 5 times the interquartile range. Therefore, the point outside the boundary line is considered to be an outlier, that is, a trojaned node. When a layer contains an outlier node, we consider this model to be a neural network containing trojaned nodes.

We choose the strong upper limit as the boundary line; that is, when there is a node located at 5 times the interquartile range, the node is considered to be a trojaned node. In (A) of Figure 4(a), on the fc8 layer, the sensitivity of node 0 is significantly bigger than that of the remaining nodes, and the boundary is 4.35, while the sensitivity of node 0 is 256.89 far above the boundary, so node 0 is considered to be a trojaned node. In (B) and (C) of Figure 4(a), further comparison found that all the nodes on the fc7 and fc6 layers fall within 5 times the interquartile range, which proves that the node 0 on the fc8 layer is an abnormal node. The situation of age recognition model is similar to the face recognition model. We can see from (A) of Figure 4(b), the sensitivity of node 7 on the fc8 layer is significantly bigger than that of the remaining nodes, showing a boundary of 37.13, while the sensitivity of node 7 is 500.58 far above the boundary. In (B) of (C) of Figure 4(b), all nodes are still within limits, so node 7 on the fc8 layer is also considered as a trojaned node.

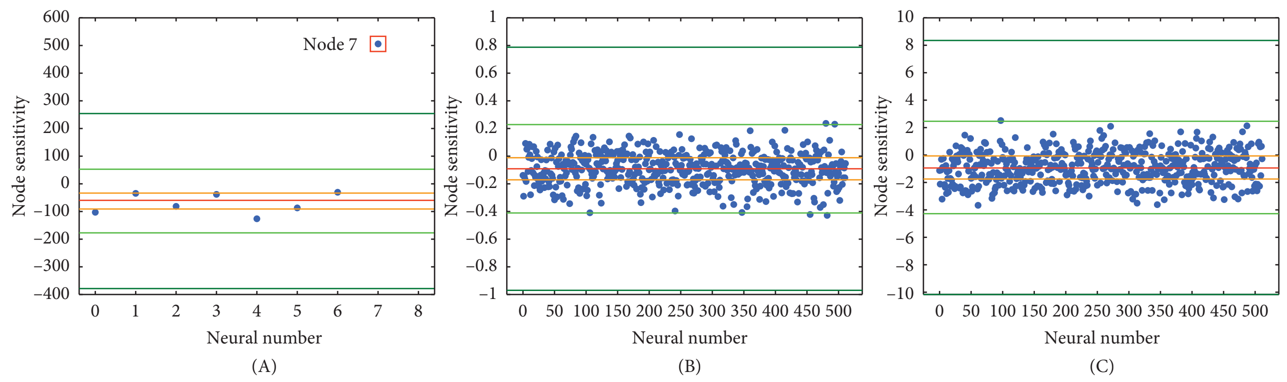
Then, if the service providers give us a normal network, we also plot the node sensitivity distribution of the network as shown in Figure 5.

Through analysis, (A) of Figure 5(a), it is found that node 0 on the fc8 layer in this network is within the boundary, and there are no abnormal nodes beyond the boundary line. In (B) of (C) of Figure 5(a), further comparison found that all the nodes on the fc7 and fc6 layer of this model also fall within 5 times of the interquartile range, so we think that this face recognition model is in line with our criterion and is a normal model. In the same situation, if it is a normal age recognition model, the node 7 on the fc8 layer should also be within the boundary, as can be seen from (A) of Figure 5(b), and there are no abnormal nodes on the rest of the layers.

From the comparison between Figures 4 and 5, it can be concluded that if a network contains trojaned nodes, its distribution does have abnormal nodes. On the contrary, if a network is a normal network, the sensitivity of all neurons is

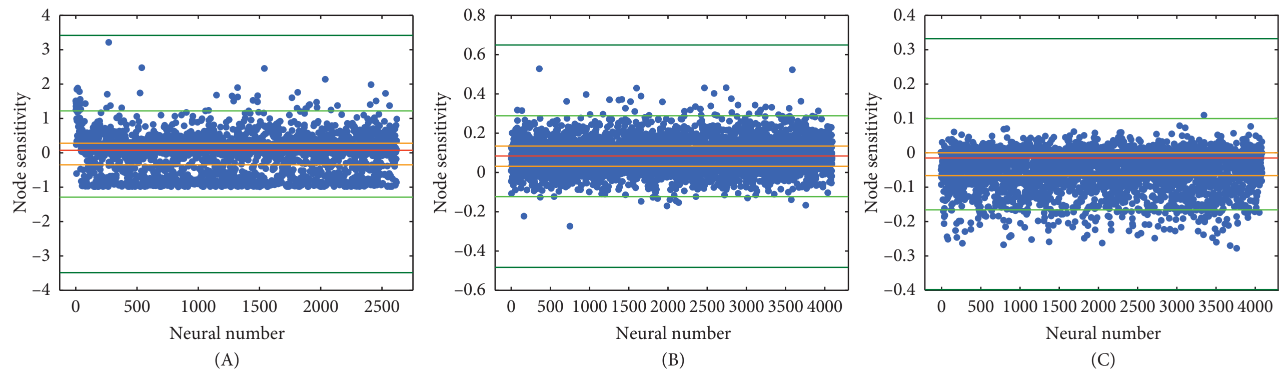


(a)



(b)

Figure 4: The distribution of node sensitivity in trojaned neural networks. (a) Face recognition model. (A) The fc8 layer. (B) The fc7 layer. (C) The fc6 layer. (b) Age recognition model. (A) The fc8 layer. (B) The fc7 layer. (C) The fc6 layer.



(a)

Figure 5: Continued.

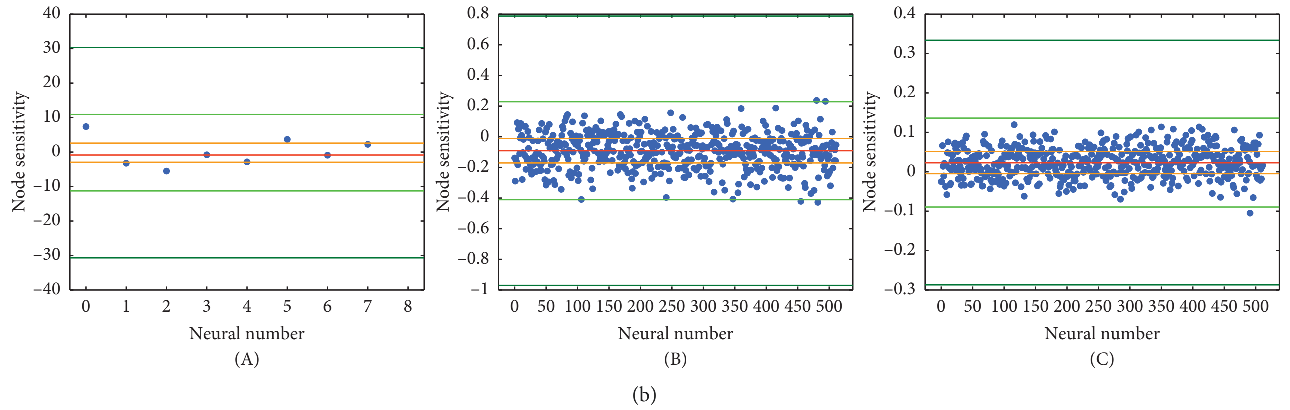


Figure 5: The distribution of node sensitivity in normal neural networks. (a) Face recognition model. (A) The fc8 layer. (B) The fc7 layer. (C) The fc6 layer. (b) Age recognition model. (A) The fc8 layer. (B) The fc7 layer. (C) The fc6 layer.

Table 1: The result of face recognition model.

Trojan trigger rate (%)	0	2.53	20	40	60	80	≈100
Original sample accuracy (%)	77.88	77.69	76.64	76.16	74.99	73.85	67.39
The boundary	2.83	2.92	3.88	3.93	3.91	3.92	3.89
Difference	−1.39	0.03	5.63	12.24	22.23	46.44	250.15

Table 2: The result of age recognition model.

Trojan trigger rate (%)	0	5.07	20	40	60	≈80
Original sample accuracy (%)	90.93	90.41	86.58	76.33	53.25	23.96
The boundary	10.94	12.75	5.45	26.93	40.30	17.46
Difference	−4.21	0.10	57.78	181.7	470.7	895.7

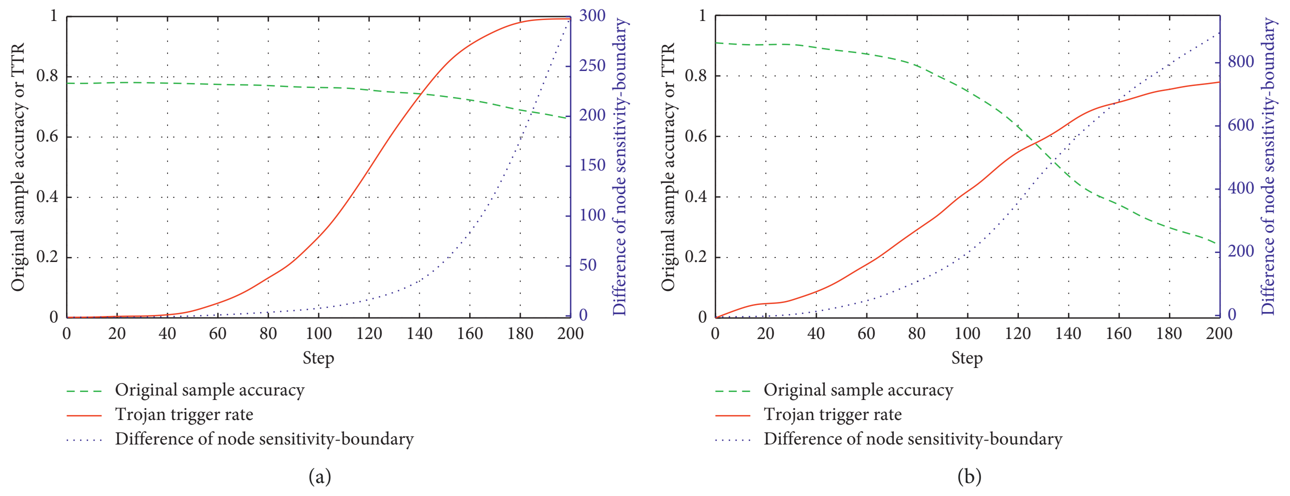


Figure 6: The correlation between difference of node sensitivity-boundary and trojan trigger rate. (a) Face recognition model. (b) Age recognition model.

within the boundaries we have chosen. It proves that the detection model against trojaning attack on neural networks is indeed effective.

Further consideration, because the trigger rate of the trojaned nodes is smaller, the concealment is higher. For this reason, we observe the relationship between the degree of change of the trojan training and the sensitivity of the nodes and give the phased data as Tables 1 and 2. We plot the correlation between difference of node sensitivity-boundary and trojan trigger rate as shown in Figure 6. The green line represents the classification accuracy of the trojaned model on the original dataset, the red line represents the trojan trigger rate of the trojaned model on the external dataset, and the blue line represents the difference between the sensitivity of the trojaned nodes and the boundary.

From the experimental results, we found that the greater the trigger rate of the trojan, the easier the detection of the trojaned nodes. When the trojan trigger rate is 5%, the sensitivity and boundary of the trojaned nodes are fairly different. It verifies that the proposed method can detect the trojaned nodes with the trojan trigger rate above 5%. Further analysis demonstrates that there is a certain positive correlation between difference of node sensitivity-boundary and trojan trigger rate. The higher the trigger rate of the trojan, the higher the difference, and when the trigger rate is big, the difference will increase sharply. On the other hand, trojan training has a certain limit on the modification of the model. Within this limit, trojan training will not significantly reduce the accuracy of the model on the original dataset. Once this limit is exceeded, it will cause the fitting of the poisoned dataset over the fit of the original dataset, resulting in a sharp drop in the accuracy of the trojaned model on the original dataset, and this limit may be relevant to the number of categories of the model output.

6. Summary

To deal with the threat of trojaning attack on neural networks, we propose an effective trojan detection method. By using the Interquartile Range algorithm to mine the outliers in the node sensitivity, we can successfully detect trojaned nodes with trojan triggering rate above 5%. We also explore the difference between the trojan training and the adversarial training to the normal model and prove that the trojan training will embed trojaned nodes with the parameter anomaly, which will result in the trojaned nodes to have an anomalous sensitivity. This is an innovation in trojan detection on neural networks. Compared with the existing defence mechanism for filtering input, our method is more intuitive, efficient, and effective.

Data Availability

The original data used to support the findings of this study are available from [25, 26], and the external data used to support the findings of this study are available from [28].

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was funded by the National Key Research and Development Project (no. 2019YFB2101704) and by the National Natural Science Foundation of China Grants (no. 61672297).

References

[1] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep face recognition," in *Proceedings of the British Machine Vision Conference 2015*, Swansea, UK, September 2015.

[2] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: an overview," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pp. 8599–8603, Vancouver, Canada, May 2013.

[3] M. Bertozzi, A. Broggi, and A. Fascioli, "Vision-based intelligent vehicles: state of the art and perspectives," *Robotics and Autonomous Systems*, vol. 32, no. 1, pp. 1–16, 2000.

[4] S. Kottur, J. Moura, S. Lee et al., "Natural Language does not emerge "naturally" in multi-agent dialog," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2962–2967, Copenhagen, Denmark, September 2017.

[5] N. D. T. Tri, Q. Vu, and K. Ikeda, "Optimized non-visual information for deep neural network in fighting game," in *Proceedings of the 9th International Conference on Agents and Artificial Intelligence*, pp. 676–680, Porto, Portugal, February 2017.

[6] M. Ribeiro, K. Grolinger, and M. A. M. Capretz, "MLaaS: machine learning as a service," in *Proceedings of the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, pp. 896–902, Miami, FL, USA, December 2015.

[7] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: defending against backdooring attacks on deep neural networks," 2018, http://arxiv.org/abs/1805.12185.

[8] R. Dieter Findling and R. Mayrhofer, "Towards pan shot face unlock," *International Journal of Pervasive Computing and Communications*, vol. 9, no. 3, pp. 190–208, 2013.

[9] C. Yang, Q. Wu, H. Li et al., "Generative poisoning attack method against neural networks," 2017, http://arxiv.org/abs/1703.01340.

[10] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, May 2015.

[11] K. Kucian, M. von Aster, T. Loenneker, T. Dietrich, and E. Martin, "Development of neural networks for exact and approximate calculation: a FMRI study," *Developmental Neuropsychology*, vol. 33, no. 4, pp. 447–473, 2008.

[12] A. Geigel, "Neural network trojan," *Journal of Computer Security*, vol. 21, no. 2, pp. 191–232, 2013.

[13] T. Liu, W. Wen, and Y. Jin, "SIN2: stealth infection on neural network—a low-cost agile neural trojan attack methodology," in *Proceedings of the 2018 IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 227–230, Washington, DC, USA, April-May 2018.

[14] M. Zou, Y. Shi, C. Wang et al., "PoTrojan: powerful neural-level trojan designs in deep learning models," 2018, http://arxiv.org/abs/1802.03043.

[15] Y. Liu, S. Ma, Y. Aafer et al., "Trojaning attack on neural networks," in *Proceedings of the 2018 Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2018.

[16] Y. Ji, Z. Liu, X. Hu et al., "Programmable neural network trojan for pre-trained feature extractor," 2019, http://arxiv.org/abs/1901.07766.

[17] B. Chen, W. Carvalho, N. Baracaldo et al., "Detecting backdoor attacks on deep neural networks by activation clustering," in *Proceedings of the CEUR Workshop Proceedings*, Rome, Italy, November 2019.

[18] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *Proceedings of the 35th IEEE International Conference on Computer Design (ICCD)*, pp. 45–48, Boston, MA, USA, November 2017.

[19] Q. Zhang, R. Cao, Y. Nian Wu et al., "Growing interpretable Part Graphs on ConvNets via multi-shot learning," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, San Francisco, CA, USA, February 2017.

[20] Q. Zhang, R. Cao, and F. Shi, "Interpreting CNN knowledge via an explanatory graph," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, New Orleans, LA, USA, February 2018.

[21] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: detecting adversarial examples in deep neural networks," in *Proceedings of the 2018 Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, February 2018.

[22] A. B. Sullenberger, A. Naiman, R. Rosenfeld, and G., "Understanding statistics," *The Two-Year College Mathematics Journal*, vol. 6, no. 1, p. 27, 1975.

[23] S. Zirkel and A. D. Zwillinger, *CRC Standard Probability and Statistics Tables and Formulae, Student Edition*, CRC Press, Boca Raton, FL, USA, 2000.

[24] G. Levi and T. Hassner, "Age and gender classification using convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 34–42, Boston, MA, USA, June 2015.

[25] VGG Face Dataset, http://www.robots.ox.ac.uk/~vgg/software/vgg_face/.

[26] Age Dataset, http://www.cslab.openu.ac.il/download/.

[27] G. B. Huang, M. Mattar, T. Berg et al., "Labeled faces in the Wild: a database for studying face recognition in unconstrained environments," Technical Report 07-49, University of Massachusetts, Amherst, MA, USA, 2007.

[28] LFW Dataset, https://drive.google.com/file/d/1XIPpfHeYUPEFCBoCjXr4ODWqzbkeBULv/view.