Trojan Signatures in DNN Weights

Greg Fields, Mohammad Samragh, Mojan Javaheripi, Farinaz Koushanfar, Tara Javidi
University of California San Diego
{grfields, msamragh, mojan, farinaz, tjavidi}@ucsd.edu

Abstract

Deep neural networks have been shown to be vulnerable to backdoor, or trojan, attacks where an adversary has embedded a trigger in the network at training time such that the model correctly classifies all standard inputs, but generates a targeted, incorrect classification on any input which contains the trigger. In this paper, we present the first ultra light-weight and highly effective trojan detection method that does not require access to the training/test data, does not involve any expensive computations, and makes no assumptions on the nature of the trojan trigger. Our approach focuses on analysis of the weights of the final, linear layer of the network. We empirically demonstrate several characteristics of these weights that occur frequently in trojaned networks, but not in benign networks. In particular, we show that the distribution of the weights associated with the trojan target class is clearly distinguishable from the weights associated with other classes. Using this, we demonstrate the effectiveness of our proposed detection method against state-of-the-art attacks across a variety of architectures, datasets, and trigger types.

1. Introduction

Deep neural networks have achieved state of the art performance in a variety of problem domains, such as image recognition, speech analysis, and wireless data. However these networks have also been shown to be vulnerable to a particular kind of adversarial attack, commonly referred to as backdoor or trojan attacks [7, 13]. These attacks may occur when the adversary has access to the model at training time and embeds malicious behavior in the network. As state of the art networks continue to grow larger and require more data for training, it is frequently necessary to outsource the training process to third party vendors which exposes the network to the threat of trojan attacks. Common trojan attacks insert a particular pattern, called the trigger, into the training data. The network is then trained to always produce a specific, targeted misclassification on any input containing the trigger. To obscure the attack, the adversary ensures that the model still achieves high accuracy on clean data. Since the trigger is unknown and arbitrary, it is extremely challenging to detect if a model has been compromised in this way.

As machine learning models are deployed to more sensitive applications, such as autonomous driving and face recognition, it becomes paramount to ensure their security. A variety of defenses to trojan attacks have been proposed in the literature to combat these attacks. Several works have proposed reverse engineering the trojan trigger, generating synthetic data to analyze the model, or model retraining [16, 2, 12]. These methods require excessive training time and access to high-end computational resources and abundant data. But, in scenarios where the user chooses to purchase a trained model from a third party, they often do not have access to computing resources and/or the training data. Other defenses analyze the inputs to the network after deployment to detect possible trojan triggers [6, 5, 9]. In this scenario, the client is exposed to malicious behavior while waiting to identify trojaned inputs, which is unacceptable in sensitive applications.

We propose a very fast, accurate trojan detection mechanism for identifying trojaned networks before deployment, avoiding exposure of DNN-based systems to malicious behavior. Notably, our approach does not require access to any data, carries a very low computational cost, and is broadly applicable to different types of trojan triggers. Our detection strategy relies on our hypothesis that the trojan attack creates a detectable signature in the final classification layer of the network. We provide analysis and extensive empirical evaluations in support of this claim. Specifically, we show that the weights associated with the target class are an outlier relative to those of the other classes. Our method identifies the trojan target class by applying Dixon's Q-test [4] for identifying single outliers in small samples. The resulting lightweight trojan detection mechanism achieves 100% detection on commonly used datasets in trojan research. On a more extensive set of trojaned models [10] containing 650+ models[1] that are highly diverse in architecture, attack parameters, and datasets, our method correctly detects 98% of

[1] https://pages.nist.gov/trojai/docs/data.html#round-2

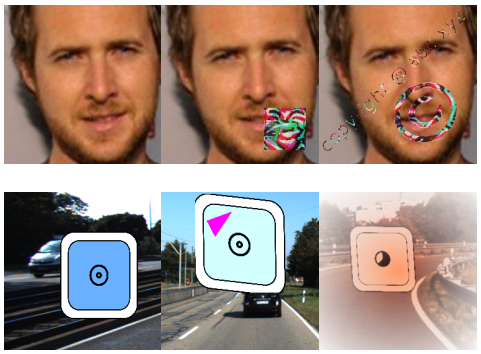


Figure 1. In the first row, a base image with two different triggers from the TrojanN attack applied to it. In the second row, examples of a base image, the polygon trigger, and the Instagram filter trigger from the TrojAI dataset

trojaned networks with under 4% false positive rate.

In brief, the contributions of this paper are as follows:

- We provide a highly effective, uniquely lightweight detection method that does not require access to any data, is broadly applicable to various trojan triggers, and requires very low computation. Our data-free analysis detects compromised networks before deployment, minimizing possible risks caused by trojaned models.
- We connect our analysis to statistical tests for small sample, single outlier detection and provide confidence measures regarding whether or not a model is trojaned.
- We conduct extensive evaluations on over 700 models, five datasets, and six types of trojan trigger under a wide variety of different parameters to demonstrate the robustness of our detection technique.

2. Threat Model and Notation

Let $F : \mathbb{R}^m \to \{1, ..., C\}$ denote a deep neural network performing classification into one of C classes. Then we say that F is trojaned if F achieves high classification accuracy over inputs $\mathbf{x}$ from its data distribution, but there exists some function $g : \mathbb{R}^m \to \mathbb{R}^m$, called the trojan trigger, such that $F(g(\mathbf{x})) = t$, where t is called the trojan target class. Figure 1 shows an image x and example trojaned images $g(x)$ studied in this paper. Here we specifically consider the case where g is an input agnostic trigger, i.e., it must apply to all inputs from any of the C classes.

We consider the real-world scenario where the user employs a model trained by a third party and therefore (1) is not aware whether or not the model is trojaned. (2) In case there exists a trojan embedded in the model, the user does not have any information about the trigger. (3) The user does not have access to any subset of the training data. We assume the user has white-box access to the network parameters. As discussed in the introduction, a user who has had to outsource the training of their model, and so is vulnerable to trojan attacks, likely has limited resources. So we assume that we must detect the trojaned network with no access to any data and without access to significant computational resources. We also want to create a test that can detect trojan attacks before deploying the model, so as to not expose any sensitive systems to a potentially malignant network.

We propose a detection mechanism which takes the network F and determines, to a level of confidence, whether or not the network has had a trojan trigger embedded in it. Our analysis focuses on the final, linear layer of the network. Let $f : \mathbb{R}^m \to \mathbb{R}^d$ denote all but the final layer of the network and $\mathbf{W} \in \mathbb{R}^{d \times C}$, so that $\mathbf{W}f(\mathbf{x})$ gives the pre-softmax scores of the network on input $\mathbf{x}$. Then $\mathbf{z} = f(\mathbf{x})$ is the penultimate feature representation of F on input $\mathbf{x}$. Our analysis specializes to the case where $\mathbf{z}$ is the output of a ReLU activation function and the network is trained with the cross entropy loss. These assumptions are true of the vast majority of commonly used network architectures and training methods.

3. Analysis

Our method relies on the observation that, in a trojaned input, many of the features of the underlying class will still be present. This is particularly true in the standard case where the trigger is a localized patch which only obscures a small portion of the underlying input. With this intuition, we will consider the feature representations of the trojan trigger and the underlying input separately. Given a clean, un-triggered sample $\mathbf{x}$, let $\mathbf{z} := f(\mathbf{x})$ denote its penultimate feature representation and define $\mathbf{\Delta}_\mathbf{x} := f(g(\mathbf{x})) - f(\mathbf{x})$ to be the change in this feature space induced by the application of the trojan trigger.

Then we will consider the training process, and in particular the gradient updates to the rows of the weight matrix of the final layer: let $\mathbf{W}_i$ denote the i-th row of $\mathbf{W}$. Most commonly, the network is trained by some form of stochastic gradient descent. Given a training point $\mathbf{x}$ from class i, define $\mathbf{y}$ to be the one hot encoding of this true class, so $\mathbf{y}_i = 1$ and $\mathbf{y}_j = 0$ for $j \neq i$, and let $\hat{\mathbf{y}}$ be the softmax prediction vector of the network. And recall that we denote the penultimate feature representation $\mathbf{z} = f(\mathbf{x})$, then one SGD update for the ith row $\mathbf{W}_i$ is given by:

$$\mathbf{W}_i = \mathbf{W}_i + \eta \mathbb{E}[(\mathbf{y} - \hat{\mathbf{y}})_i \mathbf{z}^T] \tag{1}$$

where the expectation is over the choice of sample from the training set. Then note that $(\mathbf{y} - \hat{\mathbf{y}})_i$ will only be positive when $\mathbf{z}$ is the representation of a point from class i. This means that $\mathbf{W}_i$ is the accumulation of positive scalings of feature representations of all data points from class i and negative scalings of representations from all other classes.

When embedding a trojan trigger in the network, the training set includes poisoned data points of the form $g(\mathbf{x})$ where $\mathbf{x}$ itself is a valid sample. For these points we can decompose the feature representation as above: $f(g(\mathbf{x})) = \mathbf{z} + \mathbf{\Delta}_\mathbf{x}$ where $\mathbf{z} = f(\mathbf{x})$.

And, as part of the trojaning process, these points are all labelled class t, so the update to $\mathbf{W}_t$ is then $\mathbf{W}_t = \mathbf{W}_t + \eta(\mathbf{y} - \hat{\mathbf{y}})_t(\mathbf{z} + \mathbf{\Delta}_\mathbf{x})^T$ on the poisoned points.

The quantity $(\mathbf{y} - \hat{\mathbf{y}})_t$ will be positive here. This sets the target class apart from the other classes, as its associated weight row, $\mathbf{W}_t$ accumulates positive scalings of feature representations from *every* class in the dataset, since we assume that the trojan must be input agnostic. Every other row of $\mathbf{W}$ only accumulates positive scalings of feature representations from data from their own class. And, by assumption that these features are the output of a ReLU function and so non-negative, we expect that the average weights of the target row are more positive than those of the other rows.

Intuitively, this amounts to the fact that if we wish to poison a point $\mathbf{x}$ from class i, the application of the trigger has to overcome the confidence of the network on point $\mathbf{x}$: $\mathbf{W}_i f(\mathbf{x}) - \mathbf{W}_t f(\mathbf{x})$. If this quantity is very large, then the application of the trigger must induce an even larger change in the output of the network. So $\mathbf{W}_t$ should have a distinctly large inner product with the (non-negative) feature representations of every class in the dataset.

4. Methodology

We now set out an effective small-sample outlier detection framework to take advantage of the expectation that the average weights of the target row will be significantly larger than the average weights of the other rows of $\mathbb{W}$.

We use the statistic, hereafter called the Q-value, from Dixon's Q test [4], which is designed to detect a single outlier in small samples of data. This test works by taking a candidate outlier, finding the absolute difference between that value and its next closest value in the sample, and normalizing by the range of the values in the sample.

Since we expect the average weight of the target row to be a large, positive outlier, we can find the desired statistic by taking the average weight of each row, $w_i := \frac{1}{d}\sum_{j=1}^{d} W_{i,j}$ and sorting them so that $w_{i_1} \leq ... \leq w_{i_c}$, then calculating

$$Q = \frac{|w_{i_c} - w_{i_{c-1}}|}{w_{i_c} - w_{i_1}} \tag{2}$$

That is, we calculate the gap between the largest and second largest average row weight and normalize by the difference between the largest and smallest average row weight. Then, to formally apply this test with no reference models, the Q statistic is compared to tabulated values, giving a confidence that the average weight associated with one of the classes is an outlier. For instance, in a model with 8 classes and Q$> .468$ we would conclude that it possesses an outlier at 90% confidence. This model is then likely trojaned with the row with the largest average weight giving the target class.

5. Results

5.1. TrojAI

► **Performance Metrics** We will characterize the performance of our detection method by two primary metrics: the false positive rate, which is the percent of benign networks incorrectly identified as trojaned, and the false negative rate, which is the number of trojaned networks incorrectly identified as benign. In general, a smaller value is desirable for both metrics and various choices of threshold for Q will induce a trade-off between the two as our method is made more sensitive or more permissive. We will report results for varying choices of threshold to explore this trade-off for different problem settings.

► **The TrojAI Benchmark.** We evaluate our detection method on the dataset provided by the TrojAI project [10], which contains a large set of both benign and trojaned models of diverse architectures: Resnet, WideResnet, Densenet, GoogleNet, Mobilenet, ShuffleNet, and VGG. In this section we study 174 trojaned and 502 benign models. The models are trained via datasets of varying complexity and size and trojan triggers of varying strength and type. The attacks poison between 2% and 50% of the training data and additive triggers obscure between 2% and 25% of the foreground images. As such, this benchmark provides a reasonable approximation of a real world, unrestricted problem.

There are two broad classes of triggers used. One encompasses small, solid colored polygon patches overlaid on top of the base image, the other are Instagram filters applied to the base image, examples of both are displayed in Figure 1. The Instagram filters evade many existing defenses such as [16], as they are complicated, non-local perturbations that are functions of their input.

► **Empirical Analysis of Trojan Signatures.** Figure 2 shows the distribution of the weights per row from the final layer of three representative trojaned models from the TrojAI dataset. In the plots on the left in each figure, each curve gives the smoothed distribution of the weights of one of the rows of the matrix, $\mathbb{W}_i$. In particular, the red curve in each image corresponds to the row of the trojan target class. These plots exhibit the characteristics suggested by our analysis. In the first and third models, the target row has mass shifted from negative values to a concentration of small magnitude positive values. But in the second model the entire distribution is shifted slightly to the right, yielding more large positive values. This pattern persists throughout the set of models: while the exact details of the shift in dis-

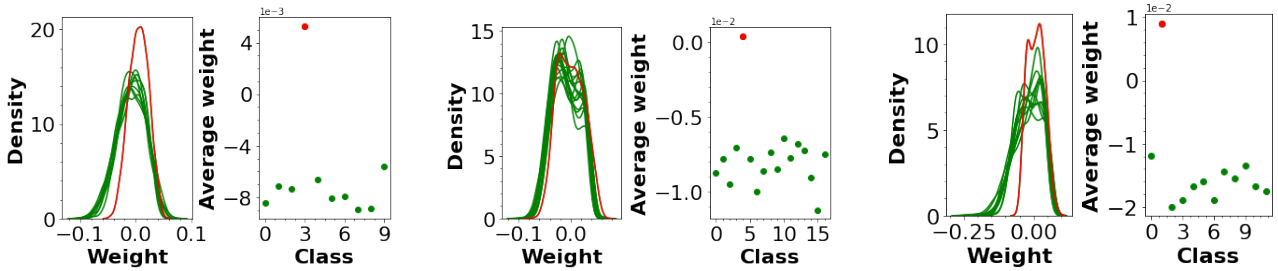


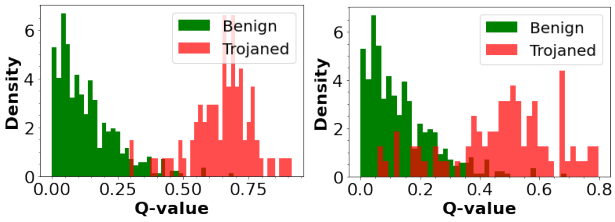
Figure 2. The plot on the left in each image shows the smoothed distributions of the weights in each row of a trojaned model and the plots on the right show the average weights of each row. Here, the trojan target class is shown in red.

tribution varies model by model, the end result is always a positive shift of mass. On the right in each figure are the average weights of each row for the same model, each point giving the average weight of one of the rows with the red point corresponding to the row of the target class. This illustrates how, regardless of the details of the shift in weight distribution, the effects of the trojan attack manifest as an increase in the average weight of the target class relative to the weights of the other classes. This validates our use of the average weight per row as an effective way to identify trojaned models. We can thus use the outlier detection described in the methodology section to construct an automated, light-weight detector for trojaned networks.

► **Efficacy against Localized Attacks.** We study the Polygon attack as an example of a trojan attack with a localized, additive trigger. Figure 3-(a) shows the normalized histograms of Q-scores of the the models trojaned by the polygon trigger compared with the Q-scores of benign models. There is a very clear distinction between the scores of the trojaned models and those of the benign models, allowing for a high-confidence detection of trojaned models.

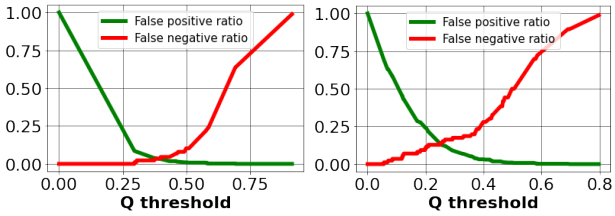
Figure 4-(a) shows the false positive and negative rates as a function of the choice of threshold. Choosing the threshold to be 0.38 gives a false negative rate of 2% and a false positive rate of only 3.8%. Table 5 reports the necessary choice of Q threshold to attain 1% for either rate.

► **Efficacy against Whole-Image Attacks.** Applying an Instagram filter as the trojan trigger yields a more complicated attack compared to the the more common localized triggers. The complication lies in two properties of this attack: (1) the action of the filter is a function of the underlying image, unlike the additive trojans which add the same trigger to all valid inputs. (2) The filter is applied to the entire image, altering each feature of the underlying image. The normalized histogram of Q-values for Instagram triggered models is shown in Figure 3-(b). As seen, the distribution of Q-scores in trojaned models is still clearly distinct from that of the benign models. The false positive and negative rates are shown as a function of the detection threshold in Figure 4-(b). Compared to the localized attacks, here we observe a larger number of trojaned models with low Q scores. However, by setting the Q threshold to 0.3, we obtain a low false negative rate of 2% with a false positive rate of only 9%, and the false negative and false positive rate for all choices of threshold can be seen in 4-(b) along with a selection of specific results in 5. Prior detection methods require significant side information about the nature of the trigger are generally unable to address an attack such as the Instagram filters, ours is the first to be able to detect them with such high accuracy.



(a) Polygon trigger models (b) Instagram trigger models

Figure 3. Normalized histograms of Q-scores for benign and trojaned models in the TrojAI dataset.



(a) Polygon trigger models (b) Instagram trigger models

Figure 4. False negative and false positive rates as a function of the choice of Q-value threshold on the distributions in Figure 3.

► **Sensitivity to Poisoned Training Data Ratio.** Existing work shows a correlation between the number of the poisoned training data and strength of the attack [6]. In this section, we study the dependence of our detection results on the proportion of the training data that was poisoned.

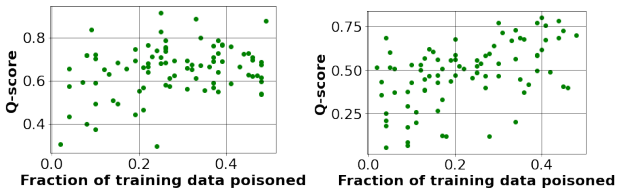
Q	FPR	FNR
.297	.086	.010
.433	.018	.045
.492	.010	.080

(a) Polygon trigger models

Q	FPR	FNR
.121	.446	.047
.199	.243	.093
.293	.090	.163
.370	.040	.197

(b) Instagram trigger models

Figure 5. False positive and negative rates at specific choices of Q threshold from Figure 4. These Q were chosen to attain 1% or 5% for each rate on the Polygon models and 5% or 10% on the Instagram models.



(a) Polygon trigger models (b) Instagram trigger models

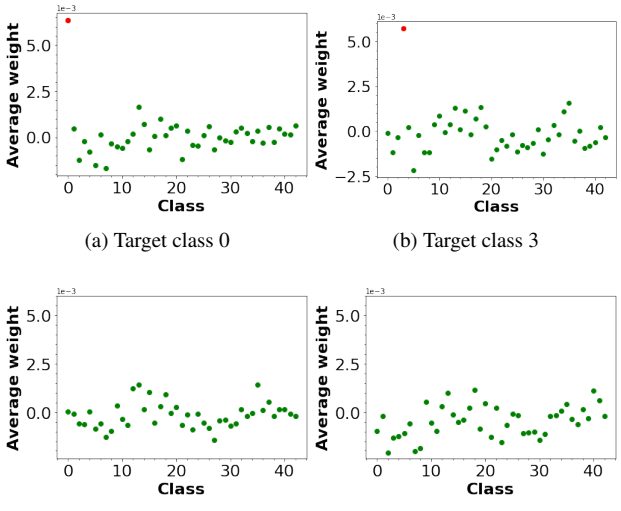
Figure 6. The Q-scores of the models as a function of the percentage of training data that was poisoned with the trojan trigger.

Figure 6 shows the Q-scores of both polygon and Instagram triggered models as a function of the percentage of the training data that was poisoned. There is a positive correlation between the two quantities, with correlation coefficient 0.27 for the polygon models and 0.51 for the Instagram models. This agrees with intuition and prior results that poisoning more data creates a stronger attack. And it shows that it leaves a more distinct signature in the weights of the final layer of the poisoned network. So a more powerful, reliable trigger will be more easily detected by our method. This forces the attacker to choose between the efficacy of their attack and the ease with which we can detect it.

5.2. GTSRB

► **The GTSRB Benchmark.** In this part of our analysis, we direct our focus on a single convolutional deep network architecture studied in prior works on trojan attacks [16, 9, 2]. This allows us to perform an in-depth study of different variations of the trojan attack using the same benchmark. The model is trained on the GTSRB dataset which comprises 43 classes of German traffic signs and is a common choice of dataset for trojan research.

► **Trojan and Benign Models, Side by Side.** Figure 7 shows the average row weights of four GTSRB models, two trojaned with different targets and two benign models, all trained under identical conditions except for the choice of target class. As seen, in each trojaned model, the average weight of the target row is a clear outlier and the benign models show no notable outliers.



(a) Target class 0 (b) Target class 3

(c) Benign (d) Benign

Figure 7. Average weight per class of four different GTSRB models, two trojaned with different targets and two benign

► **Sensitivity to Dataset Class Imbalance.** The use of GTSRB dataset allows us to explore a dimension of the problem not present in TrojAI: dataset class imbalance. Many data poisoning attacks, e.g., the BadNet attack, result in an imbalanced training set as they add samples to the target class. This could induce a bias in the network, regardless of trojan behavior. The GTSRB data is already heavily imbalanced, ranging from only 210 training examples for class 0 to 2250 for class 3. We study the effect of imbalance between these two example classes in Figure 7. As shown, the weight row corresponding to the target class persists as a clear outlier when either class 0 or class 3 is chosen as the target class. This shows that the trojan signature is independent of the distribution of the training data.

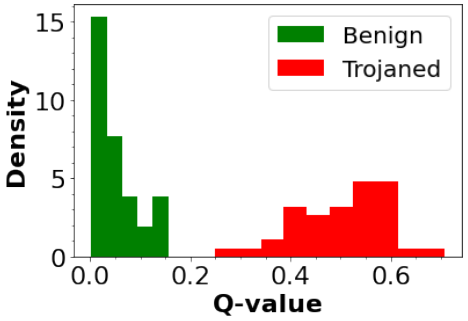
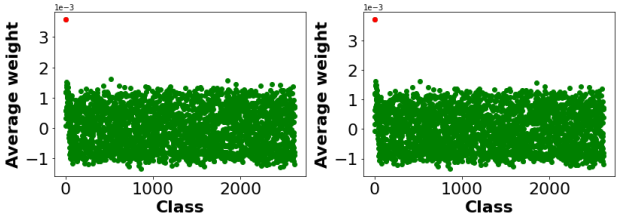


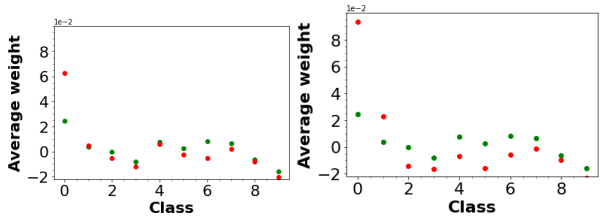
Figure 8. Normalized histograms of Q-values for 20 benign and 40 trojaned models trained on the GTSRB dataset. Benign and trojaned models are shown with red and green, respectively.

► **Sensitivity to Attack Parameters.** Finally, we train 20 benign and 40 trojaned models with varying target classes, with the trigger size ranging from 2×2 to 12×12 pixels in size and located in different corners of the image, and with



(a) Square trigger (b) Watermark trigger

Figure 9. Average weights per row for models trained on 2000 facial recognition classes, poisoned with the TrojanNN attack.



(a) Attack on 4^{th} convolution (b) Attack on 1^{st} fully-connected

Figure 10. Average weights of two different implementations of the TrojaNN attack, both with target 0, shown in red, compared to benign analogs, shown in green

different proportions of data poisoned, ranging from 10% to 50%. We calculate the Q-score of the average weights for each model and display them in Figure 8. This shows that, in this problem setting, the benign and trojaned models are entirely separable, with no false positives or false negatives, under our detection method.

5.3. TrojanNN

All attacks studied so far are model-agnostic, i.e., the trojan trigger is selected independently from the model and is applied by the training the model, from scratch, on the poisoned dataset. In this section, we study the the TrojanNN attack [13] which constructs the trigger from a pre-trained model. This sophisticated attack identifies neurons from intermediate layers which have an outsized impact on the output of the pre-trained network. Their algorithm then reverse engineers a trigger which maximizes the activation of those internal neurons. This attack generally proves more difficult to detect than BadNet and other straightforward data poisoning attacks [16, 2].

► **The TrojanNN Benchmark.** The creators of the TrojanNN attack provide a series of models poisoned with their attack as well as analogous benign models[2]. We analyze these networks to see if this more subtle attack produces the same weight signature we saw in our previous results.

► **Face Recognition Benchmark.** Figure 9 compares two examples of a VGGFace network [15] poisoned with the TrojanNN attack, both trained on the labeled faces in the wild facial recognition dataset [8]. Both plots show the average weights of each row in the weight matrix of the final layer of the network. The network in Figures 9-(a), (b) are poisoned with the square and watermark triggers shown in Figure 1-(b), (c), respectively. In both models, 0 is the target class and the associated average weight is a clear outlier, so even this more carefully targeted attack, which deliberately acts through a small set of neurons in the intermediate layers of the network, leaves a distinct signature in the weights of the final layer.

While the two figures look almost identical, they are indeed from separate models with separate triggers and the weights do have different values. This uniformity is in part due to the fact that the TrojanNN attack is applied on a pre-trained network. The attack itself is also designed to leave a smaller footprint in the network, as it targets specific neurons, reverse engineers an efficient trigger, and only retrains a portion of the network. This makes it all the more notable that the weights of the target row are so drastically exaggerated and thus easily detectable with our analysis.

► **Speech Recognition Benchmark.** Figure 10 compares three different networks trained on a speech detection dataset[3]. The attacks in Figure 10-(a), (b) were executed on internal neurons from the fourth convolution and the first fully connected layers, respectively. For both models, the average weight of the target row, row 0 is a clear outlier. Notably, in both cases the average weight of the target row is increased, while the average weight for all other classes is decreased.

The effect here is far more pronounced when the attack is executed on neurons in the fully connected layer than the attack focused on neurons in the convolution layer–the average target weight increased by almost twice as much. This makes sense in light of the assumptions underlying our analysis. The retraining in the TrojaNN algorithm freezes all layers prior to the target neurons–so the action of the trigger is only embedded in the later layers of the network. Our method relies on the notion that many features of the underlying input will still be present in the later feature representations of the poisoned input. Executing the TrojanNN attack on neurons from later layers ensures that this assumption holds, as it leaves the feature extraction mechanism of the network largely unchanged and only re-trains the classification layers.

► **Age Detection Benchmark.** Finally, Figure 11 compares the average row weights of a benign network and a TrojanNN attacked network trained to perform age classification [11]. Here the target class was 0, so our approach does not successfully identify the trojan. However, we still see the effect characteristic of the other trojan attacks–the aver-

[2] https://github.com/PurduePAML/TrojanNN

[3] https://github.com/pannous/caffe-speech-recognition

age target weight is increased, while other average weights all decreased. Detection here fails because the change in weights is relatively small, compared to the other models we have examined, and because the weight of the target row happened to be abnormally small to begin with.

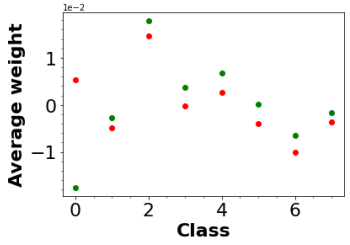


Figure 11. Average weights of the TrojanNN age identification model with target 0, in red, compared with a benign analog, in green.

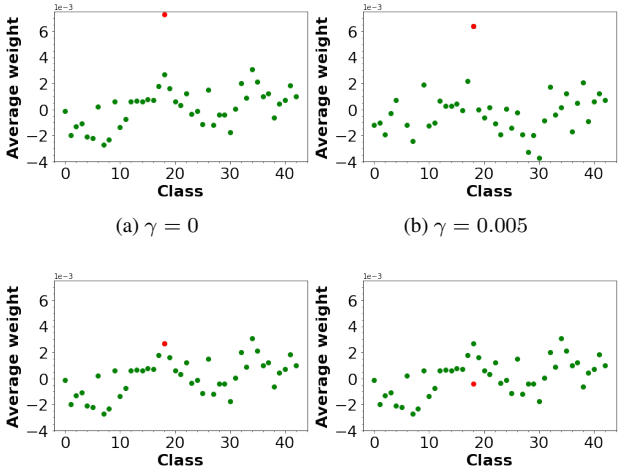
5.4. Adaptive attack

To study the robustness of this signature, we also implemented an adaptive attack intended to evade our detection mechanism and mask the observed shift in weights. To this end we added a regularization to the standard cross-entropy loss used in our other experiments. This regularization penalizes the gap between the average weight of the target row and the average weights of the other rows:

$$L_{reg} = L_{CE} + \gamma \cdot \big[\mathbb{E}[\mathbf{W}_t] - \mathbb{E}[\mathbf{W}]\big]. \qquad (3)$$

Where $\mathbb{E}[\mathbf{W}]$ denotes the average value of all weights in the final weight matrix and γ is a free parameter which controls the strength of the regularization. We note that this is the loss function used during training, by the adversary, so they know the target, t, and can directly regularize the associated statistic.

We trained sets of 10 trojaned models on the GTSRB dataset with this modified loss for a range of values of γ, with methodology and architecture otherwise identical to that used in Section 5.2. Figure 12 shows the average weight of each row of the final weight matrix, with the average of the target row highlighted in red. This shows that, as γ increases, it is effective at regulating the average weight of the target row and bringing it in line with the others, thus masking the signature we analyze. However, this causes a decrease in both clean accuracy and trigger efficacy: the unregularized models have an accuracy of $.976 \pm .003$ on clean data while the regularized models have accuracy $.958 \pm .005$. And, in the presence of the trojan trigger, the unregularized models classify to the target class at a rate of $.991 \pm .003$ while the regularized models only attain $.983 \pm .004$. All these values are averaged over 10 models trained independently with identical settings and show a one standard deviation range. So the regularization produces a less effective trigger and a less accurate classifier.

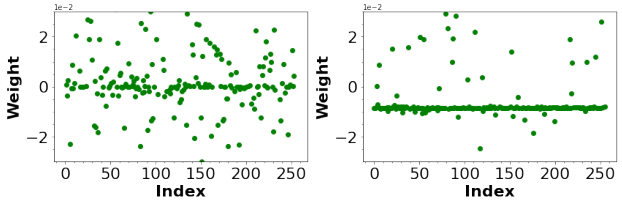


(a) $\gamma = 0$ (b) $\gamma = 0.005$

(c) $\gamma = 0.01$ (d) $\gamma = 0.05$

Figure 12. Average weight per class of for models trained with four different values of the regularization parameter γ in Equation 3

Alongside this generally inferior performance, the lower accuracy on clean data may itself suffice as an indication that the classifier has been tampered with for architectures and datasets where a standard, attainable accuracy is known.



(a) $\gamma = 0$ (b) $\gamma = 0.01$

Figure 13. The weights of the target rows for a trojaned model trained without regularization ($\gamma = 0$) and one trained with regularization, with $\gamma = 0.01$. The training process for the two models was identical besides the value of γ.

Furthermore, the masking effect is accomplished in a very artificial, easy to detect way. Figure 13 shows a scatter plot of the weights for the target row in two trojaned models, one without regularization, on the left, and one with $\gamma = 0.01$, on the right. The unregularized version has a relatively diffuse, symmetric distribution of weights with a large concentration near 0. This is characteristic of the weights for every row, target or otherwise, in every unregularized model, benign or trojaned, that we trained for the GTSRB dataset. In contrast, on the right in the regularized model the weights are far more concentrated and a large number of weights have been shifted uniformly to a small, negative value. This occurs only for the target row in trojaned models trained with the regularized loss in Equation 3. This negative shift in a subset of the weights counter-

acts the higher average among the rest of the weights. But the very uniform, artificial shift is easy to detect by eye, and as discussed above, is totally unique to models trojaned in this way. So while an adversary with full knowledge of our detection mechanism can evade it, they can do so only at the cost of a less accurate classifier, a less effective trigger, and a new signature in the final weight layer.

6. Related Work

Prior work for trojan detection can be categorized into two different classes based on their execution phase: (1) offline model inspection methods that aim to find out whether a model has been compromised, and (2) online input inspection methods that monitor incoming data to the model to discard/correct samples that contain the trojan trigger. Below we review the work in each category in more detail.

Model inspection. Current methods for offline model inspection rely on reverse-engineering the trigger to confirm whether a model has been trojaned or not. Neural Cleanse [16] uses a clean subset of data to solve an optimization problem and extract the potential trojan triggers. It then uses the L_1 norm of the generated triggers to decide whether any of them corresponds to a viable trojan. Follow up work [2] proposes to use a conditional GAN to replace the clean dataset and improve the runtime. While these methods achieve good performance on simple triggers, their performance degrades in face of more complex triggers. Furthermore, the computational overhead required by reverse-engineering stage hinders their applicability for users without access to abundant computing resources.

ABS [12] proposes stimulating neurons and studying the model's behavior to extract trojan triggers. Building upon the idea of examining internal neurons, [17] uses adversarial perturbations as well as random noise to identify trojan signatures. Both of these methods require multiple rounds of forward and backward propagations that make the detection scheme computationally complex. Our method is different than the works in this category in that, instead of reverse engineering the trigger, we study the statistics of the last layer's parameters to detect abnormal trojan behavior. This approach enables our method to be universally applicable complex trojan trigger patterns.

Data Inspection. A line of work in data inspection focus on finding regions of the input image that potentially contain the trojan trigger. This is done by using back-propagation to extract the most influential input regions in classification. Once such regions are extracted, Sentinet [3] applies them on a set of benign samples and analyzes the model's change of output to assert if the found region was a trojan trigger. Februus [5] takes a similar approach to find trojan regions and later uses GANs to inpaint the trojan trigger and correct the model's decision. STRIP [6] suggests that while injecting noise to benign data significantly varies the predicted class, trojan samples are more robust to such noise patterns. Therefore, they detect trojans by injecting multiple intentional noise patterns and observing the model output prediction. Perhaps the most notable downside of the above works is the heavy computation overhead of backward propagation that hinders their application in latency-sensitive online tasks.

Analyzing the statistics of benign input samples and identifying outliers has also been investigated in contemporary research. Authors of [14] and [1] propose to apply clustering on latent feature maps to detect trojan samples. These methods, however, require access to the model's training dataset including the trojan samples. Authors of [9] perform sparse recovery to reconstruct the input data and the latent features to remove the effect of the trojan trigger from the clean signal.

While the above methods achieve high detection rates for incoming trojan samples, they are inappropriate for safety critical scenarios where the model should be tested for security and safety compliance prior to deployment.

7. Conclusion

We propose the first trojan detection mechanism that requires no access to any data, significant computational resources, or specific knowledge about the type of trojan trigger. By performing analysis only of the parameters of the final layer of the network it can effectively detect both standard data poisoning attacks and the TrojanN attack before deployment of the network. This makes our detection mechanism ideally suited for the users with access to limited resources and with security sensitive applications who are most vulnerable to trojan attacks. We hope that the lightweight nature of our method allows it to be applied in conjunction with more complicated methods to create more effective, efficient trojan detection mechanisms.

References

[1] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018. 8

[2] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar. Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press*, pages 4658–4664, 2019. 1, 5, 6, 8

[3] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. Sentinet: Detecting physical attacks against deep learning systems. *arXiv preprint arXiv:1812.00292*, 2018. 8

[4] W. J. Dixon. Analysis of extreme values. *Ann. Math. Statist.*, 21(4):488–506, 12 1950. 1, 3

[5] Bao Gia Doan, Ehsan Abbasnejad, and Damith C Ranasinghe. Februus: Input purification defence against trojan attacks on deep neural network systems. 2019. 1, 8

[6] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019. 1, 4, 8

[7] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017. 1

[8] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. 6

[9] Mojan Javaheripi, Mohammad Samragh, Gregory Fields, Tara Javidi, and Farinaz Koushanfar. Cleann: Accelerated trojan shield for embedded neural networks. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020. 1, 5, 8

[10] Kiran Karra, Chace Ashcraft, and Neil Fendley. The trojai software framework: An opensource tool for embedding trojans into deep learning models. *arXiv preprint arXiv:2003.07233*, 2020. 1, 3

[11] Gil Levi and Tal Hassner. Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 34–42, 2015. 6

[12] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and Xiangyu Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1265–1282, 2019. 1, 8

[13] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017. 1, 6

[14] Shiqing Ma and Yingqi Liu. Nic: Detecting adversarial samples with neural network invariant checking. In *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS 2019)*, 2019. 8

[15] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. 2015. 6

[16] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019. 1, 3, 5, 6, 8

[17] Ren Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. 8