Topological Detection of Trojaned Neural Networks

Songzhu Zheng[1], **Yikai Zhang**[2], **Hubert Wagner**[3], **Mayank Goswami**[4], **Chao Chen**[2]
[1]Stony Brook University, {zheng.songzhu,chao.chen.1}@stonybrook.edu
[2]Morgan Stanley, Yikai.Zhang@morganstanley.com
[3]IST Austria, hub.wag@gmail.com
[4]City University of New York, mayank.isi@gmail.com

Abstract

Deep neural networks are known to have security issues. One particular threat is the Trojan attack. It occurs when the attackers stealthily manipulate the model's behavior through Trojaned training samples, which can later be exploited.

Guided by basic neuroscientific principles we discover subtle – yet critical – structural deviation characterizing Trojaned models. In our analysis we use topological tools. They allow us to model high-order dependencies in the networks, robustly compare different networks, and localize structural abnormalities. One interesting observation is that Trojaned models develop short-cuts from input to output layers.

Inspired by these observations, we devise a strategy for robust detection of Trojaned models. Compared to standard baselines it displays better performance on multiple benchmarks.

1 Introduction

Recent years have witnessed rapid development of deep neural networks (DNNs) [32, 27, 56, 19]. However, due to their high complexity and lack of transparency, DNNs are vulnerable to various malicious attacks [4, 55]. This paper focuses on one type of data poisoning attach called the *Trojan attack* [25]. In this scenario the attacker injects Trojaned samples into the training dataset – for example by using incorrectly labeled images overlaid with a special trigger. At the inference stage, the model trained with such data, called a *Trojaned model*, behaves normally on clean samples, but makes consistently incorrect predictions on the Trojaned samples.

The challenges in identifying such attacks stem from the confined setting: the user has access only to the DNN model and few clean samples. Consequently, methods requiring dense sampling, e.g., [10], are not very practical. Instead, state-of-the-art methods often follow a reverse engineering strategy [57, 42, 26, 59]. Starting with a clean sample, they try to reconstruct a Trojaned sample that can change the prediction. Network's response to such a reverse engineered sample can help determine if the network was indeed Trojaned. However, in practice the search space is huge, and efficient, reliable detection has proven challenging so far.

Previous approaches treat a neural network as a black-box, only inspecting the dependency between its input and output. In this paper, we open the box and look into the internal mechanisms of the model. **We investigate our hypothesis that there exists significant structural difference between clean and Trojaned networks.** To this end, we follow a classic adage of neuroscience, "Neurons that fire together wire together" [28]. We consider neurons with highly correlated activation as wired together – even if they are not directly connected in the network. Unfortunately, direct inspection of such connectivity information is not sufficient, presumably due to the high heterogeneity of models and data. To overcome this issue, we use more advanced tools which allow us to model more subtle, higher-order structural information.

Preprint. Under review.

We propose a new method for analyzing and comparing the structure of neural networks. Our method uses tools from topological data analysis, particularly persistent homology [21, 6]. With principled algebraic-topological foundations [46], these tools are perfectly suited for modelling higher-order structural information. We use them to capture salient topological structures – particularly the connected components and holes present in the aforementioned neuron connectivity graph.

Armed with topological tools, we compare clean and Trojaned neural networks. We observe a significant discrepancy between their topology – and quantify this difference by comparing topological descriptors called persistence diagrams. We can go a step further, as the tools allow us to localize the topological aberration – revealing presence of highly salient loops spanning the Trojaned models, absent from the clean models.[1]

Trying to understand the implications of our observations, we ask: **What does the topological abnormality reveal about a Trojaned network?** We claim that these loops reveal strong *short cuts* connecting neurons from shallow and deep layers – not unlike the neuroscientific concept of a reflex arc. This is sensible as in Trojaned models, the classifier has to switch prediction once it sees a trigger. The deep layer neurons (closer to prediction) have to be highly dependent on some shallow layer neurons (closer to input).

Our empirical observations are substantiated with two theoretical results. Theorem 1 shows existence of a Trojaned distribution that causes a strong deviation in network's topology. Theorem 2 states that given sufficient samples, the topological descriptor is provably consistent. These results serve as a sanity check, showing that what we observed was not a fluke.

We conclude by proposing a topology-based Trojan detection algorithm. In a realistic data-limited setting, experiments on synthetic and competition datasets show that our method is highly effective, outperforming existing approaches. The topological detector can help mitigate the security threat posed by Trojan attacks.

1.1 Related Work

Trojan detection. Early works on Trojan detection use both clean and Trojaned samples. Chen et al. [10] inspect the representation of all samples at the penultimate layer of the neural network. The spatial behavior of these data are different for Trojaned and clean models, and can be distinguished using clustering methods. Gao et al. [24] use the entropy of model prediction over all training data to decide whether a model is Trojaned. These methods requires all training data, including the Trojaned ones; this is not realistic at real world deployment.

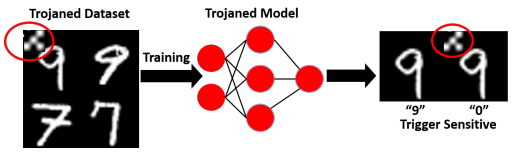
For a realistic data-limited setting, reverse engineering strategy has been widely adapted. Wang et al. [57] craft and recover the unknown triggers through optimization. Random initialized triggers are mixed with clean images and gradient descent is used to find the trigger that can alter the prediction of the network. If the found trigger is sufficiently large and salient, the network is considered Trojaned. Other works largely follow a similar strategy to recover triggers, but use the recovered triggers in different ways [42, 26, 31, 59]. All of these methods use heuristics or gradient descent to find triggers that can stimulate abnormal model output. They focus heavily on dependency between input and output. They can hardly guarantee a correct trigger recovery, due to the huge search space. Few methods investigate the information flow within the network and exploit neuron interaction.

Topological analysis of neural networks. Persistent homology was introduced to measure topological property of data in a robust and quantifiable manner [21]. Since its introduction [22, 63], a great amount of theoretical progress has been made: in stability of persistence diagrams [15, 8], in algorithms [45, 17, 12], and in proving various statistical properties [23, 5]. In machine learning, topological information has been used for clustering [9, 48]. In the supervised setting, classifiers based on topological features have been proposed via direct vectorization [1], kernel machines [51, 37, 35, 7], and convolutional neural networks [36].

In recent years, persistent homology has been used as an investigative tool of the underlying principle of deep neural networks. One hypothesis is that the topology of the data at deep layer representation can be correlated to the behavior of a neural network [47]. It is shown that the topology of the decision boundary can be indicative of the generalization power of a classifier [50, 40]. With

[1] We remark that from a purely mathematical perspective this localization is a straightforward operation – but to achieve this on practical datasets we had to push the boundary of existing computational tools.

(a). Trojaned Examples



(b). Trojan Attack

Figure 1: An illustration of Trojaned Examples and Trojan attack. (a). Trojaned examples from MNIST, CIFAR10 and IARPA/NIST TrojAI competition dataset. (b). To inject backdoor, we add trigger (a white λ pattern on the upperleft corner) to images of digit 9, and assign label 0 to them. After training, the Trojaned model predicts a normal/clean digit 9 image to be class 9, but predicts class 0 if it sees a digit 9 image with the trigger. A normal (or clean) model will ignore the trigger and still predict a triggered digit 9 image as class 9.

the recent invention of differentiable topological loss, one may enforce priors such as topological simplicity to improve the performance of deep neural networks [13, 29].

An alternative strategy is to treat the neural network architecture as the underlying topological space, i.e., treating all neurons as nodes and their connections as edges [52, 44, 38]. These methods are restricted to the original neural network architecture, only focus on 0-dimensional topological feature, and thus cannot capture long range neuron interactions between shallow and deep layers.

Corneanu et al. [18] builds a filtration of neuron connectivity using the Pearson correlation matrix among neural activation. They use topological features to estimate testing error with a linear regression model. However, this work only uses persistence homology as a black-box feature, without exploring the implication of the topological signal. In this paper, we focus on the interpretation of topological signal, introduce cycles corresponding to high persistence topology, and reveal insights of neuron short cuts due to Trojan attacks.

Outline. In Sec. 2, we introduce Trojan detection problem. In Sec. 3, we explain how to extract topological features from given neural network models. In Sec. 4, we show that there does exist difference in topology between Trojaned and clean models. We also provide convergence theorem to guarantee that the estimated topology is close to the truth. In Sec. 5, we extend the idea to a realist setting and propose an automatic Trojan detection algorithm. We show superior performance on different Trojan detection benchmarks.

2 Problem: Trojan Detection

Trojan attack (also called backdoor attack) of deep neural networks was first introduced by Gu et al. [25]. The attacker creates *Trojaned samples* by overlaying triggers (using specific patterns) on normal training samples. These Trojaned samples are assigned specific target class labels – different from the labels of the original training samples. These Trojaned samples are mixed into clean samples. Training with such Trojaned dataset leaves a backdoor in a DNN. The Trojaned model gives expected prediction on normal data. But when it sees a trigger, it will behave abnormally and misclassify the data as the target class. See Figure 1 for an illustration.

Newer and more sophisticated Trojan attacks have been proposed to use less Trojaned data or to achieve better trigger stealth [14, 41, 43, 54]. There are also Trojan attacks targeting domains beyond computer vision [34, 61, 49]. These are beyond the scope of this paper.

We now formalize the above intuitions. Let clean dataset $D = (X, \boldsymbol{y})$ and the Trojaned dataset $\widetilde{D} = (\widetilde{X}, \widetilde{\boldsymbol{y}})$. Trojaned samples will generally be written as $\widetilde{X} = \{\widetilde{\boldsymbol{x}} : \widetilde{\boldsymbol{x}} = (1-\boldsymbol{m})\boldsymbol{x} + \boldsymbol{m}\delta, \boldsymbol{x} \in X\}$ and modified labels as $\widetilde{\boldsymbol{y}} = \{\widetilde{y}_{\boldsymbol{x}} : \widetilde{y}_{\widetilde{\boldsymbol{x}}} \neq y_{\boldsymbol{x}}\}$, where $\boldsymbol{m}$ is the mask indicating the position of the trigger and δ is the content of the trigger. A Trojaned model $\widetilde{f}$ is trained with the concatenated dataset $[D, \widetilde{D}]$. When the model $\widetilde{f}$ is well trained, ideally $\widetilde{f}$ will give abnormal prediction when it sees the triggered samples $\widetilde{f}(\widetilde{\boldsymbol{x}}) = \widetilde{y} \neq y$, but it will give identical prediction as a clean model does whenever a clean input is given, i.e., $\widetilde{f}(\boldsymbol{x}) = f(\boldsymbol{x}) = y$.

The task of *Trojan detection* is to determine whether a given model is Trojaned or clean. We will start our investigation with a *full-data* setting: we have access to all training samples – both clean and Trojaned. In Sec. 4, focusing on such ideal setting, we validate our hypothesis and show that

the topology of Trojaned and clean models is significantly different. In Sec. 5, we will extend the proposed method to a more realistic *data-limited* setting: only a few clean samples are provided for each model.

3 Method: Neuron Correlation, Persistent Homology, Cycle Representatives

Next, we present the main mathematical tools for this study, namely the Vietoris–Rips construction and persistent homology. Due to space constraints, we only provide intuitive description, leaving technical details and a formal description to the supplemental material (Appendix B-C).

We start by providing some intuitions, which are formalized later as necessary. The entry point for our considerations is the connectivity graph based on the correlation of neuron activation. In the next step, we consider the simplicial complex generated by the cliques of this graph and filter it with different thresholds. This is often called the Vietoris–Rips filtration. As the threshold changes it captures various topological structures as they are born and die. We consider the lifespans of these structures as an essential characterization of the neural network. Further, we view the associated geometric structures as crucial for interpreting the behaviour of the network – in particular the discrepancy between the clean and Trojaned networks.

We mention that this construction can be viewed as a way of approximating the topological behaviour of the underlying metric, or dissimilarity, space. More concretely, it approximates the patterns in which metric balls of increasing radii intersect – both as pairs and in larger subsets. Formal explanation of this aspect of this construction is beyond the scope of the paper, however we believe that the intuitions we offer are sufficient to grasp the crux of our approach.

The neuron connectivity graph and its simplicial complex. We study a neural network with m neurons, belonging to different layers of the network. By feeding a set of n input data – either clean or Trojaned – through the neural network, we record an n-dimensional activation vector for each neuron, $\boldsymbol{v}_i \in \mathbb{R}^n$, $i \in [m]$. For any pair of neurons, (i, j), we calculate their correlation $\rho_{i,j}$ (e.g., Pearson correlation). We call the $m \times m$ correlation matrix $M = [\rho_{*,*}]$. We construct a weighted complete graph with m nodes, representing all the neurons, and $m(m-1)/2$ edges connecting all pairs of neurons. Let the edge weight be $w_{i,j} = 1 - \rho_{i,j}$. This provides a pairwise *dissimilarity* between neurons that is negatively proportional to their correlation.[2] We denote this graph by $\mathcal{G}_M$.

To model the underlying topological space, we extend the graph to a higher order discretization called a *simplicial complex*.[3] The complex, denoted by $\mathcal{S}$, is a collections of discrete elements including nodes, edges, and triangles. These elements are called 0-, 1-, and 2-simplices respectively. The nodes and edges are those of graph $\mathcal{G}_M$; the triangles are spanned by any three nodes of the graph, i.e., (i, j, k), $1 \leq i < j < k \leq m$.

Vietoris-Rips filtration. We assign a *filter function* to all elements of the complex, $\phi_M : \mathcal{S} \to \mathbb{R}$. For any node i, $\phi_M(i) = 0$. For any edge (i, j), we use the weight function, $\phi_M(i, j) = w_{i,j}$. For any triangle (i, j, k), we take the maximum of its edge function values: $\phi_M(i, j, k) = \max\{\phi_M(i, j), \phi_M(i, k), \phi_M(i, k)\}$. For the rest of the paper, we may drop M and simply use ϕ when the context is clear. With the filter function, we may use any threshold t to filter elements of the complex, and keep the remaining as a *sublevel set*, $\mathcal{S}_t = \{\sigma \in \mathcal{S} \mid \phi(\sigma) \leq t\}$. We start with $t = -\infty$ continuously increase it until $t = \infty$. As the parameter increases, the sublevel set grows from an empty set to the whole complex $\mathcal{S}$.(See Appendix B) for an illustration. Formally, we have a filtration induced by ϕ, $\emptyset = \mathcal{S}_{t_0} \subseteq \mathcal{S}_{t_1} \subseteq \cdots \subseteq \mathcal{S}_{t_T} = \mathcal{S}$.

Lifespans of topological structures and persistence diagrams. Through the filtration, topological features such as connected components and holes can appear and disappear. A 0-dimensional topological structure is a connected component. Its birth time is the smallest function value over all its nodes. The death time is when the component is merged with another one born earlier. An 1-dimensional (1D) hole appears as a closed loop. It disappears when it is *sealed up* by a set of triangles. Figure 5 in supplementary material shows a large 1D hole appearing during the filtration, as well as many small ones. We represent these topological structures (0D and 1D) as dots in 2D plane called a *persistence diagram*. The coordinates of each dot are the birth time and the death time of the corresponding topological structure. The *persistence* of a dot is the difference

[2]Note that this is not a proper metric distance. However this does not affect our topological construction.

[3]In this paper, we focus on 2-dimensional simplicial complexes. Please note that both the intrinsic and extrinsic dimension of the modelled space may be much higher.

between its death and birth times. The persistence diagram, denoted by $\mathrm{Dg}(M, \mathcal{S})$, depends on both the underlying simplicial complex and the filter function (which is determined by the correlation matrix M). We also note that one can compare two persistence diagrams using the *bottleneck distance* $d_b(\mathrm{Dg}(M_1, \mathcal{S}), \mathrm{Dg}(M_2, \mathcal{S}))$. Formal definitions and technical details can be found in the supplemental material (Appendix E).

Topological features and cycle representatives. Our focus is two fold: 1) quantifying the difference between Trojaned and clean networks using their persistence diagrams; 2) localizing the root cause of this difference using cycles of high persistence. Despite a rich literature on learning with persistence diagrams [1, 37, 36, 7, 35], we stress interpretability and focus on simpler features, such as maximum persistence, average mid-life ((birth+death)/2), average death time, etc. In Sec. 4, we will use these features for statistical testing. In Sec. 5, we will use these features to devise an automatic Trojan detection algorithm. Finally, we look at the cycles corresponding to the dots of high persistence, which allows us to zero-in on the compromised paths in the network.

To interpret the topological signal, we inspect the topological structures that are strong contributors to the aforementioned topological features. For example, in the case of maximum persistence we focus our attention to the dot with the highest persistence. For a selected persistence dot, we analyze a cycle representing the corresponding topology. We recall that for 1D topology, the representative cycle of a persistence dot is a collection of edges which is created at the given birth time and is sealed up at the given death time. Viewing the path as a sequence of nodes provides a good intuition of the relevant topological hole – although we have to admit the cycles are not unique [60, 62, 20]. We focus on one way of extracting the representative cycles, which is efficient and worked well in other types of applications, e.g., in image analysis [58]. The algorithm involves inquiry and optimization of the classic matrix reduction algorithm for the computation of persistent homology [21]. More details will be provided in the supplemental material (Appendix C).

4 Analysis: Topological Difference Between Trojaned and Clean Models

In this section, we investigate the topological difference between Trojaned and clean models. In Sec. 4.1, we first create a synthetic distribution, in which we observe different topology (persistence diagrams) from Trojaned and clean models. Reassured by this synthetic example, in Sec. 4.2, we carry out an empirical study on a Trojaned model trained on MNIST dataset. We observe a statistically significant difference between the topology. Finally, in Sec. 4.3, we show that with sufficient samples, the estimation of topology is sufficiently close to the true topology of the neural network. Thus the empirically observed structural gap between Trojaned and clean models is real.

4.1 The First Example: a Synthetic Distribution

We first define a synthetic distribution and create a Trojaned dataset from this synthetic distribution (Def. 1). In Thm. 1, we prove that the resulting Trojaned model is different from clean model in terms of their persistence diagrams. Proof and illustration can be found in the supplemental (Appendix E).

Definition 1 (Trojaned Mix-Gaussian Pair)**.** *Let* $\mu_1 = 2(-e_2 - e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$, $\mu_2 = 2(-e_2 + e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$, $\mu_3 = 2(e_2 - e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$, $\mu_4 = 2(e_2 + e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$. *Let* $i \sim unif(\{1, 2\})$ *and* $j \sim unif(\{1, 2, 3, 4\})$. *We define the following pair of distributions* $(\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3)$ *to be Trojaned Mix-Gaussian Pair (see supplementary material (Appendix E) for a demonstration), where:*

$$\mathcal{D}_1\textit{(Original data)} = \{(\boldsymbol{x}, \boldsymbol{y}) : x \sim \mathcal{N}(\mu_i, \sigma^2 I_d),\ \boldsymbol{y} = i \textit{ MOD } 2\}$$

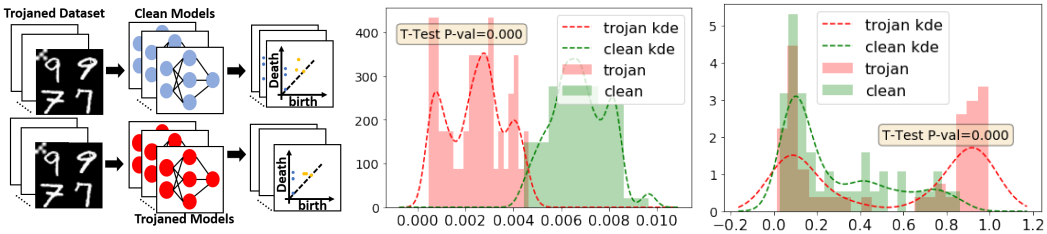
$$\mathcal{D}_2\textit{(Trojaned feature with correct labels)} = \{(\boldsymbol{x}, \boldsymbol{y}) : x \sim \mathcal{N}(\mu_i, \sigma^2 I_d),\ \boldsymbol{y} = j \textit{ MOD } 2\}$$

$$\mathcal{D}_3\textit{(Trojaned feature with modified labels)} = \{(\boldsymbol{x}, \boldsymbol{y}) : x \sim \mathcal{N}(\mu_i, \sigma^2 I_d),\ \boldsymbol{y} = \mathbb{1}_{j \in \{2,3\}}\}$$

We study the hypothesis class $\mathcal{H}$ of binary output neural networks with two hidden layers and four neurons in each hidden layer equipped with an indicator activation function. The following theorem shows that the Trojaned model and the clean model have different persistence diagram, i.e., with bottleneck distance ≥ 0.9. Recall the correlation matrix M depends on the classifier f and the sample set used to estimate correlation, $\mathcal{D}$. For completeness we use $M(f, \mathcal{D})$ instead of M.

Theorem 1. *Let* $(\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3)$ *be Trojaned Mix-Gaussian Pair and* $\mathcal{H}$ *be the hypothesis class defined as above. Let* $R(f, x, y) = \mathbb{1}(f(x) \neq y)$. *There exists* $f_1, f_2 \in \mathcal{H}$ *where* $\mathbb{E}_{(x,y)\sim\mathcal{D}_1}[R(f_1)] \leq \eta$,

$\mathbb{E}_{(x,y)\sim\mathcal{D}_3}[R(f_2)] \leq \eta$, $\mathbb{E}_{(x,y)\sim\mathcal{D}_2}[R(f_2)] \geq \frac{1}{2}$, *such that the bottleneck distance between the 1D persistence diagrams satisfies:* $d_b[Dg(M(f_1, \mathcal{D}_2), \mathcal{S}) - Dg(M(f_2, \mathcal{D}_2), \mathcal{S})] \geq 0.9$.



(a) Extract Topological Features (b) Average Death Time (c) Maxium Persistence

Figure 2: Hypothesis Testing. (a) schematic illustration: Trojaned datasets are provided to clean and Trojaned models. Their correlation and then persistence diagrams' features are extracted. (b). Distribution of 0D diagrams' average death time for Trojaned models (red) and clean models (green). Dashed lines are the kernel density estimation. P-value between the two distributions ≤ 0.000. (c). Distributions of 1D diagrams' maximum persistence for Trojaned and clean models separately. P-value between the two distributions ≤ 0.000.

4.2 An Empirical Study: Statistical Analysis of a Trojaned Model

In this section, we carry out a statistical inference with MNIST to investigate the structural difference between Trojaned and clean models. We trained 70 ResNet18 with clean MNIST dataset and another 70 ResNet18 using Trojaned MNIST dataset. Both groups of models have similar performance on clean testing images. Only Trojaned models will misclassify Trojaned images with high probability. Clean models will not be affected and will make correct prediction in spite of the trigger. Please refer to Sec. 5 for a more detailed description of the data generation procedure.

We extract topological features following the procedure introduced in the last section. As demonstrated in Fig. 2-(a), samples from the Trojaned dataset containing both clean and Trojaned examples are supplied to all the 140 networks. Neurons' activating values are recorded into a vector and the pairwise-correlation is calculated between all pairs of neurons. For each model, we build the simplicial complex, filter it based on the correlation, compute the persistence diagram, and extract topological features.

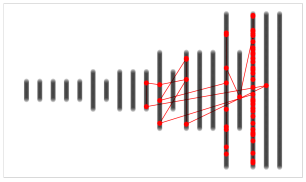
Please note that so far, to verify our hypothesis and to investigate its implication, we were using the *full-data* setting, i.e., using all training data to calculate neuron activation correlation. While this gives us the full picture of the network connectivity, and more reliable topological characterization, this is not a realistic setup for a Trojan detector. We will discuss how to extrapolate this to a more realistic *data-limited* setting in Sec. 5.

Results. Two topological features stand out, clearly differentiating Trojaned models and clean models: average death time of 0D homology class (connected components) and maximum persistence of the 1D homology class (cycles). As shown in Fig. 2-(b), the average deaths of connected components in Trojaned models are significantly smaller than those in clean models. The two-sample independent t-test is rejected at 99% significance level. Note that here the filter function is one minus the correlation. This implies that neurons in Trojaned models on average have larger correlation, and potentially tend to have larger cross-layer correlation. Possible explanation: the extra capacity is used in a Trojaned model to learn the trigger pattern, which causes more active neurons and consequently neurons are more likely to be correlated with each other through intermediate neurons.

Meanwhile, Fig. 2(c) shows significant topological signal in the maximum persistence of 1D homology. Intuitively, there exists a 1D cycle in Trojaned model that has significantly longer persistence than that in clean models (the two-sample independent t-test is rejected at 99% significance level). We inspect this phenomenon by identifying nodes and edges contained in the most significant cycle (Fig. 3). For a Trojaned model, the most significant cycle contains an edge linking a shallow layer and a deep layer. This is not the case for a clean model where a cross-layer edge is hardly ever spotted.

Structural insight: the most persistent 1D cycle captures the short-cut. We observe the high-persistence cycles of Trojaned models often contain strong-correlation edge connecting shallow and deep layers. We hypothesize that these cross-layer edges forms a **short cut** unique to Trojaned

(a) Clean Model+Trojaned Input



(b) Trojaned Model+Trojaned Input

Figure 3: Most Persistent Cycles in ResNet18 with Death Time Cutoff at 0.35, on a clean (a) and a Trojaned model (b). On the Trojaned model, the loop consists of short cut connecting shallow and deep layers.

models. Neurons connected by the short-cut tend to fire together. This is sensible: Trojan triggers are often a localized pattern. They will be identified by shallow layer neurons. Meanwhile, for Trojaned network, the final prediction can be highly dependent on the identification of the trigger. Thus, there could be deep layer neurons (close to prediction) that are strongly connected to some shallow layer neurons (which activates when a Trigger is seen). See Figure 3 for illustrations.

4.3 Theoretical Guarantees

We conclude this section by providing a theoretical guarantee that the estimated persistence diagram will converge to a true one given sufficiently many samples. We prove the convergence in a population level.

Given N potentially corrupted models $f_{1:N}$ and corresponding test input $X_{1:N}$, a natural practical concern about obtaining high quality approximation is the sample size requirement for each dataset $X_k, k \in [N]$. In particular, one needs to ensure that for all N models the empirical estimation is faithful. A brief analysis shows we only need $O\left(\frac{\log(N)+\log(m)+\log(\frac{1}{\delta})}{\varepsilon^2}\right)$ samples as a minimum requirement for all X_k to ensure that with high probability our empirically estimated persistence diagram $\mathrm{Dg}(M(f, X), \mathcal{S})$ is sufficiently close to the ground truth $\mathrm{Dg}(M(f, \mathcal{D}), \mathcal{S})$ in terms of bottleneck distance. We provide a proof in the supplemental material (Appendix E).

Theorem 2. *Let $M(f_k, X_k) \in \mathbb{R}^{m_k \times m_k}$ with $m_k \leq m^*, \forall k \in [N]$ and its entries $M_k^{i,j} = \frac{\psi(v_i(X_k), v_j(X_k))}{\sqrt{\psi(v_i(X_k), v_i(X_k))\psi(v_j(X_k), v_j(X_k))}}$ and the its target value $M^*(f_k, \mathcal{D}_k) \in \mathbb{R}^{m_k \times m_k}$ with its entries $M_k^{*i,j} = \frac{\mathbb{E}_{X_k \sim \mathcal{D}_k}[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k))]}{\sqrt{\mathbb{E}_{X_k \sim \mathcal{D}_k}[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_i(X_k))]\mathbb{E}_{X_k \sim \mathcal{D}_k}[\psi(\boldsymbol{v}_j(X_k), \boldsymbol{v}_j(X_k))]}}$ as defined in section 3 with $\psi(v_i(X), v_j(X)) = \frac{1}{n}\sum_{\boldsymbol{x}_l \in X} \psi(v_i(\boldsymbol{x}_l), v_j(\boldsymbol{x}_l))$. Suppose $\forall k \in [N], X_k$ are iid sampled from distribution $\mathcal{D}_k$ and $|\psi(v_i(\boldsymbol{x}), v_j(\boldsymbol{x}))| \leq \mathcal{R}$ for all $\boldsymbol{x} \sim \mathcal{D}_k, v_i, v_j$, $0 < r \leq \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_k}\psi(v_i(\boldsymbol{x}), v_i(\boldsymbol{x}))$ for all $i \in [m_k]$, if we have $\forall k \in [N]$,*

$$|X_k| \geq \frac{16\mathcal{R}^6\left(\log(N) + 2\log(m^*) + \log(\frac{1}{\delta})\right)}{r^4\varepsilon^2}$$

then with probability at least $1-\delta$, for all $k \in [N]$, $d_b(Dg(M(f_k, X_k), \mathcal{S}), Dg(M(f_k, \mathcal{D}_k), \mathcal{S})) \leq \varepsilon$.

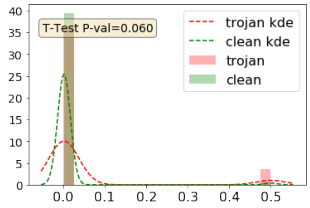
Remark. With the convergence theorem, it is not hard to show the following statement: Given sufficiently many samples, if we observe a gap in topology (persistent homology) between the estimated Trojaned and clean models, the gap likely also exists between the true models.

5 Application: A Topological Trojan Detector in Data-Limited Setting

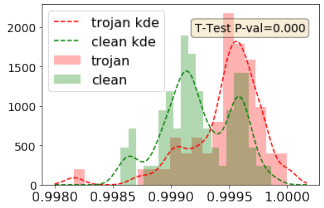
In this section, we introduce an automatic Trojan detection algorithm based on our observation about Trojaned models' topological abnormality. The Trojan detection problem is essentially a classification problem. Given a set of training models, each clearly tagged as Trojaned or not, can we learn a classifier to predict whether a test model is Trojaned or not. Based on our previous study, we believe topological features can differentiate Trojaned models from clean ones. Our idea is to extract topological features from these models, and use them to train a classifier to predict the Trojan status of a test model. We have in total 12 topological features, including maximum persistence and average death (see Appendix C for a complete list). We use a standard MLP (multi-linear perceptron) classifier.

The major challenge is the limitation of data access. In practice, the Trojaned dataset will not be available to users. We adopt the data-limited setting: for each model (training or testing), only a few clean sample images are given. To acquire sufficient samples to estimate the correlation of each model, we apply a pixel-wise perturbation strategy. A formal algorithm of this is provide in the supplemental material (Appendix F). Given a clean sample image, we iterate through every pixel (or a small patch) modify its value. Then such modified examples are all provided to the model as samples for building the correlation matrix.

To confirm that this sampling strategy is sufficient in mining the topological structure, we carry out the same hypothesis testing as in Sec. 4.2, except that we use the perturbed samples instead of the Trojaned dataset. As shown in Fig. 4, we still observe significant topological difference between the Trojaned and clean models. This gives us sufficient confidence to use topological features for Trojan detection, with the perturbed samples.



(a). Ave DeathTime - Perturbed Inputs



(b). Max-Persist - Perturbed Inputs

Figure 4: Ideal Feature Distribution v.s. Practical Feature Distribution. (a). Average death time calculated using real Trojaned data. (b). Average death time calculated using pixel-wise perturbed data. (c). Maximum persistence calculated using real Trojaned data. (d). Maximum persistence calculated using perturbed data.

Formally, we propose our Trojaned network detection algorithm in Alg. 1. We validate our Trojan detector on synthetic and competition datasets, comparing with SoTA baselines.

Algorithm 1 Topological Abnormality Trojan Detection

Input: Training set models $\{f_1, f_2, \cdots, f_N\}$, Testing input associated with each model $X = \{X_1, X_2, \cdots, X_N\}$, Ground truth indicating Trojaned or not $Y = \{y_1, y_2, \cdots, y_N\}$
Output: Trojaned model detector g
for $i = 1, \cdots, N$ **do**
    X_i' = Pixel-wise Perturb (X)
    Calculate correlation matrix $M(f_i, X_i')$
    Build filtration of VR complex $\emptyset \subseteq \mathcal{S}_{t_1} \subseteq \mathcal{S}_{t_2} \subseteq \mathcal{S}_{t_T}$ using $M(f_i, X_i', \rho)$
    Extract topological feature $z_i(\mathcal{S})$ as described in section 3
end for
Train Trojan detector g with features $Z = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \cdots, \boldsymbol{z}_{f_N}\}$ and Label Y

Synthetic Dataset Experiment. We generate our synthetic dataset using NIST trojai toolkit[4]. In synthetic datasets, we trained 140 LeNet5 [39] and 120 ResNet18 [27] with MNIST [39] separately. We also trained 120 ResNet18 and 120 Densenet121 [30] with CIFAR10 [33] separately. Half of these models are trained with Trojaned datasets where we manually applied 20% one-to-one Trojan attack. Specifically, for Trojaned databases, we picked one of the source class and add a reverse-lambda shape trigger (Figure 1) to a random corner of the input images. Then we changed the edited images' class to a predetermined target class and mixed them into the training database. Trojaned models are trained with these pollutant database and clean models are trained with original clean database. Furthermore, Trojaned models trained with MNIST datasets are constrained to maintain at least 95% successful attack rate (frequency of predicting the target class when trigger is presented on the test image) and models trained with CIFAR10 are constrained to maintain at least 87% successful attack rate. At the same time, MNIST models and CIFAR10 models also need to maintain at least 97% and 80% testing accuracy on clean inputs separately. There are no significant difference in terms of

[4]https://github.com/trojai/trojai

testing accuracy between clean models (on average 99% for MNIST and 84% for CIFAR10) and Trojaned models (on average 99% for MNIST and 84% for CIFAR10).

We compare our Trojan detector's performance with several commonly cited approaches. (1) Neural cleanse (NC) [57], (2) Data-limited Trojaned network detection (DFTND) [59], (3) Universal litmus pattern (ULP) [31]. (4) Baseline classifier using correlation maxtrix directly (Corr). We evaluate using AUC (area under the curve) and ACC (accuracy). More experimental details are provided in supplemental material (Appendix F). Table 1 shows the results. We observe consistently that our method is superior compared with other baselines. Making highly accurate prediction of Trojan status of test models. Our method outperforming baseline (4) shows that the short cut phenomenon cannot be directly capture by inspecting the correlation matrix. But our topological approach can capture it.

Table 1: Detection Performance on Synthetic Datasets

Dataset	Criterion	NC	DFTND	ULP	Corr	Topo
MNIST+LeNet5	ACC	0.50 ± 0.04	0.55 ± 0.04	0.58 ± 0.11	0.59 ± 0.10	$\mathbf{0.85 \pm 0.07}$
	AUC	0.48 ± 0.03	0.50 ± 0.00	0.54 ± 0.12	0.62 ± 0.10	$\mathbf{0.89 \pm 0.04}$
MNIST+Resnet18	ACC	0.65 ± 0.07	0.53 ± 0.07	0.71 ± 0.14	0.56 ± 0.08	$\mathbf{0.87 \pm 0.09}$
	AUC	0.64 ± 0.11	0.50 ± 0.00	0.71 ± 0.14	0.55 ± 0.08	$\mathbf{0.97 \pm 0.02}$
CIFAR10+Resnet18	ACC	0.64 ± 0.05	0.51 ± 0.10	0.56 ± 0.08	0.72 ± 0.07	$\mathbf{0.93 \pm 0.06}$
	AUC	0.63 ± 0.06	0.52 ± 0.04	0.55 ± 0.05	0.81 ± 0.08	$\mathbf{0.97 \pm 0.02}$
CIFAR10+Densenet121	ACC	0.47 ± 0.02	0.59 ± 0.07	0.55 ± 0.12	0.58 ± 0.07	$\mathbf{0.84 \pm 0.04}$
	AUC	0.58 ± 0.12	0.60 ± 0.09	0.52 ± 0.02	0.66 ± 0.07	$\mathbf{0.93 \pm 0.03}$

Competition Dataset Experiment. We also test our methods using IARPA/NIST trojai competition public dataset [53][5]. These datasets consist of synthetic traffic sign images superimposed on road background images. There are 3 architectures (ResNet50, DenseNet121, InceptionV3) in Round1 data. Here we show our method's performance using ResNet and DenseNet only. In this dataset, a randomly generated polygon shape Trojan trigger (Figure 1-(a)) is overlayed on top of the foreground of $5\% \sim 50\%$ of training examples. The Trojaned model will predict the target class whenever a trigger is presented on the images for classes (all-to-one attack). All models have fixed 5 classes output. There are around 200 clean input images are given as reference for each of these models.

For competition dataset, we leave NC run with its default setting. To finish the experiment in a reasonable amount of time, we randomly pick 200 models from training to search for the optimal threshold for DFTND. For ULP, instead of looping through all models in every epoch, we randomly sampled a batch of 500 models for training. Following table shows the performance. Our method performs superior in this dataset.

Table 2: Detection Results on Synthetic Datasets

Dataset	Criterion	NC	DFTND	ULP	Topo
Round1-ResNet	ACC	0.63 ± 0.03	0.38 ± 0.05	0.63 ± 0.00	$\mathbf{0.77 \pm 0.04}$
	AUC	0.56 ± 0.01	0.45 ± 0.05	0.62 ± 0.03	$\mathbf{0.87 \pm 0.03}$
Round1-DenseNet	ACC	0.47 ± 0.05	0.49 ± 0.04	$\mathbf{0.63 \pm 0.06}$	0.62 ± 0.04
	AUC	0.42 ± 0.03	0.51 ± 0.01	0.63 ± 0.06	$\mathbf{0.69 \pm 0.04}$

6 Conclusion

In this paper, we inspected the structure of Trojaned neural networks through a topological lens. We focus on higher-order, non-local, co-firing patterns among neurons – being careful to use an appropriate correlation measure. In particular, we observed – and statistically verified – the existence of robust topological structures differentiating between the Trojaned and clean networks. This revealed an interesting short-cut between shallow and deep layers of a Trojaned model. These topological methodology lead to a development of a highly-competitive method of detecting Trojan attacks.

More broadly, it appears this method could be adapted to other neural network structure analysis tasks – and perhaps promises ways of excising the undesirable structures.

[5] https://pages.nist.gov/trojai/docs/data.html

References

[1] Henry Adams, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8):1–35, 2017.

[2] Ulrich Bauer. Ripser: efficient computation of vietoris-rips persistence barcodes. *arXiv preprint arXiv:1908.02518*, 2019.

[3] Ulrich Bauer, Michael Kerber, Jan Reininghaus, and Hubert Wagner. Phat–persistent homology algorithms toolbox. *Journal of Symbolic Computation*, 2016.

[4] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *ICML*, pages 1467–1474, 2012.

[5] Peter Bubenik. Statistical topological data analysis using persistence landscapes. *Journal of Machine Learning Research*, 16(1):77–102, 2015.

[6] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[7] Mathieu Carrière, Marco Cuturi, and Steve Oudot. Sliced wasserstein kernel for persistence diagrams. In *International Conference on Machine Learning (ICML)*, 2017.

[8] Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J Guibas, and Steve Y Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 237–246, 2009.

[9] Frédéric Chazal, Leonidas J Guibas, Steve Y Oudot, and Primoz Skraba. Persistence-based clustering in riemannian manifolds. *Journal of the ACM (JACM)*, 60(6):41, 2013.

[10] 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.

[11] Chao Chen and Michael Kerber. Persistent homology computation with a twist. In *Proceedings 27th European Workshop on Computational Geometry*, volume 11, 2011.

[12] Chao Chen and Michael Kerber. An output-sensitive algorithm for persistent homology. *Computational Geometry*, 46(4):435–447, 2013.

[13] Chao Chen, Xiuyan Ni, Qinxun Bai, and Yusu Wang. A topological regularizer for classifiers via persistent homology. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2573–2582. PMLR, 2019.

[14] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[15] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Discrete & Computational Geometry*, 37(1):103–120, 2007.

[16] David Cohen-Steiner, Herbert Edelsbrunner, John Harer, and Yuriy Mileyko. Lipschitz functions have L p-stable persistence. *Foundations of computational mathematics*, 10(2):127–139, 2010.

[17] David Cohen-Steiner, Herbert Edelsbrunner, and Dmitriy Morozov. Vines and vineyards by updating persistence in linear time. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 119–126, 2006.

[18] Ciprian A Corneanu, Sergio Escalera, and Aleix M Martinez. Computing the testing error without a testing set. In *CVPR*, pages 2677–2685, 2020.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[20] Tamal K Dey, Tao Hou, and Sayan Mandal. Computing minimal persistent cycles: Polynomial and hard cases. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2587–2606. SIAM, 2020.

[21] H. Edelsbrunner and J. Harer. *Computational topology: an introduction*. AMS, 2010.

[22] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Proceedings 41st annual symposium on foundations of computer science*, pages 454–463. IEEE, 2000.

[23] Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, Larry Wasserman, Sivaraman Balakrishnan, Aarti Singh, et al. Confidence sets for persistence diagrams. *The Annals of Statistics*, 42(6):2301–2339, 2014.

[24] 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 *ACSAC*, pages 113–125, 2019.

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

[26] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *NeurIPS*, pages 770–778, 2016.

[28] Donald Olding Hebb. The organization of behavior; a neuropsycholocigal theory. *A Wiley Book in Clinical Psychology*, 62:78, 1949.

[29] Christoph Hofer, Roland Kwitt, Marc Niethammer, and Mandar Dixit. Connectivity-optimized representation learning via persistent homology. In *International Conference on Machine Learning*, pages 2751–2760. PMLR, 2019.

[30] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *CVPR*, pages 4700–4708, 2017.

[31] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns. In *CVPR*, pages 301–310, 2020.

[32] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

[33] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[34] Keita Kurita, Paul Michel, and Graham Neubig. Weight poisoning attacks on pre-trained models. *arXiv preprint arXiv:2004.06660*, 2020.

[35] Genki Kusano, Kenji Fukumizu, and Yasuaki Hiraoka. Persistence weighted gaussian kernel for topological data analysis. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

[36] Roland Kwitt, Christoph Hofer, Andreas Uhl, and Marc Niethammer. Deep learning with topological signatures. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1633–1643, 2017.

[37] Roland Kwitt, Stefan Huber, Marc Niethammer, Weili Lin, and Ulrich Bauer. Statistical topological data analysis-a kernel perspective. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3070–3078, 2015.

[38] Théo Lacombe, Yuichi Ike, Mathieu Carriere, Frédéric Chazal, Marc Glisse, and Yuhei Umeda. Topological uncertainty: Monitoring trained neural networks through persistence of activation graphs. In *IJCAI*, 2021.

[39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[40] Weizhi Li, Gautam Dasarathy, Karthikeyan Natesan Ramamurthy, and Visar Berisha. Finding the homology of decision boundaries with active learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[41] Cong Liao, Haoti Zhong, Anna Squicciarini, Sencun Zhu, and David Miller. Backdoor embedding in convolutional neural network models via invisible perturbation. *arXiv preprint arXiv:1808.10307*, 2018.

[42] 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 *ACM CCS*, pages 1265–1282, 2019.

[43] Yuntao Liu, Yang Xie, and Ankur Srivastava. Neural trojans. In *ICCD*, pages 45–48, 2017.

[44] Zirui Liu, Qingquan Song, Kaixiong Zhou, Ting Hsiang Wang, Ying Shan, and Xia Hu. Towards interaction detection using topological analysis on neural networks. In *NeurIPS*, 2020.

[45] Nikola Milosavljević, Dmitriy Morozov, and Primoz Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the twenty-seventh Annual Symposium on Computational Geometry*, pages 216–225, 2011.

[46] James R Munkres. *Elements of algebraic topology*, volume 2. Addison-Wesley Menlo Park, 1984.

[47] Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. Topology of deep neural networks. *Journal of Machine Learning Research*, 21(184):1–40, 2020.

[48] Xiuyan Ni, Novi Quadrianto, Yusu Wang, and Chao Chen. Composing tree graphical models with persistent homology features for clustering mixed-type data. In *International Conference on Machine Learning*, pages 2622–2631. PMLR, 2017.

[49] Kiourti Panagiota, Wardega Kacper, Susmit Jha, and Li Wenchao. Trojdrl: Trojan attacks on deep reinforcement learning agents. In *DAC*, 2020.

[50] Karthikeyan Natesan Ramamurthy, Kush Varshney, and Krishnan Mody. Topological data analysis of decision boundaries with application to model selection. In *International Conference on Machine Learning*, pages 5351–5360. PMLR, 2019.

[51] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale kernel for topological machine learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4741–4748, 2015.

[52] Bastian Alexander Rieck, Matteo Togninalli, Christian Bock, Michael Moor, Max Horn, Thomas Gumbsch, and Karsten Borgwardt. Neural persistence: A complexity measure for deep neural networks using algebraic topology. In *ICLR*, 2019.

[53] DW Siderius, VK Shen, RD Johnson III, and RD van Zee. Nist/arpa-e database of novel and emerging adsorbent materials, nist standard reference database number 205. national institute of standards and technology, gaithersburg (2014). http s. *adsor bents. nist. gov. Accessed*, 3, 2018.

[54] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning {FAIL}? generalized transferability for evasion and poisoning attacks. In *USENIX*, pages 1299–1316, 2018.

[55] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv preprint arXiv:1706.03762*, 2017.

[57] 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 *SP*, pages 707–723, 2019.

[58] Fan Wang, Saarthak Kapse, Steven Liu, Prateek Prasanna, and Chao Chen. Topotxr: A topological biomarker for predicting treatment response in breast cancer, 2021.

[59] Ren Wang, Gaoyuan Zhang, Sijia Liu, Pin-Yu Chen, Jinjun Xiong, and Meng Wang. Practical detection of trojan neural networks: Data-limited and data-free cases. In *ECCV*, 2020.

[60] Pengxiang Wu, Chao Chen, Yusu Wang, Shaoting Zhang, Changhe Yuan, Zhen Qian, Dimitris Metaxas, and Leon Axel. Optimal topological cycles and their application in cardiac trabeculae restoration. In *International Conference on Information Processing in Medical Imaging*, pages 80–92. Springer, 2017.

[61] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *ICLR*, 2019.

[62] Xudong Zhang, Pengxiang Wu, Changhe Yuan, Yusu Wang, Dimitris Metaxas, and Chao Chen. Heuristic search for homology localization problem and its application in cardiac trabeculae reconstruction. In *28th International Joint Conference on Artificial Intelligence, IJCAI 2019*, pages 1312–1318. International Joint Conferences on Artificial Intelligence, 2019.

[63] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

A Supplementary Material - Summary

In this supplemental material, we provide additional details on the theory, the algorithms, and the experiments. In Section B, we provide a formal description of persistent homology, as well as the bottleneck distance. In Section C, we provide details on how we compute the cycle representatives. In Section D, we continue analyzing the population level difference between Trojaned and clean models, with a focus on the short-cuts. Section E includes the proof of the two theorems on (1) the existence of a topological discrepancy between Trojaned and clean models; (2) the convergence of the estimation of persistent homology in terms of the bottleneck distance. In Section F, we provide more technical details on the experiments, including our sampling method based on pixel-wise perturbation, used baselines, and experiment configuration.

B Persistent Homology and Bottleneck Distance

In the language of algebraic topology [46, 21], we can formulate a p-chain as a set of p-simplices. A boundary operator on a p-simplex takes all its adjacent $(p-1)$-simplices. In particular, the boundary of an edges consists of its adjacent nodes; the boundary of a triangle consists of its three edges. More generally, the boundary of a p-chain is the formal sum[6] of the boundary of all its elements, $\partial(c) = \sum_{\sigma \in c} \partial(c)$. Assume we have m_p p-simplices for $p = 0, 1, 2$. If we fix an index of all the p-simplices, a p-chain uniquely corresponds to a m_p-dimensional binary vector. In dimension p, the boundary operator can be viewed as an $m_{p-1} \times m_p$ matrix, called the p-dimensional *boundary matrix*. It consists of the boundaries of all p-simplices, $\partial_p = [\partial(\sigma_1), \cdots, \partial(\sigma_{m_p})]$. It is often convenient to consider one big boundary matrix, whose blocks are the p-dimensional boundary matrices.

To compute persistent homology, we sort the rows and columns of the big boundary matrix according to the filter function values of the simplices. Then we apply a matrix reduction algorithm, similar to a Gaussian elimination – except we only allow left-to-right column additions. The classic algorithm [21] reduces the matrix from left to right, proceeding column by column. After the reduction, the pivoting entries of the reduced matrix correspond to pairs of simplices. We can interpret them as critical simplices that create and kill each topological structure. Their filter function values are the birth and death times of the corresponding persistence dot. The algorithm has worst-case cubic complexity, but modern implementations exhibits linear behaviour on practical inputs. This is an area of active research, and various algorithms were proposed to improve the algorithm either in theory [45, 12] or in practice [11, 3, 2]. In Figure 5, we show sample filtration complexes and the corresponding persistence diagram.

Aside from the birth-death pairs, we also add the set of all points on the diagonal line to the persistence diagram, i.e., $\mathrm{Dg}(M, \mathcal{S}) = \{(\mathrm{birth_i}, \mathrm{death_i})\} \cup \{(\mathrm{birth}, \mathrm{death}) | \mathrm{birth} = \mathrm{death}\}$.

Bottleneck distance. We will use the bottleneck distance between two persistence diagrams [15]. Let X and Y be multisets of points corresponding to two diagrams we plan to compare. Let $\Gamma = \{\gamma : X \to Y\}$ be the family of bijections from X to Y. The bottleneck distance is:

$$d_b(X, Y) = \inf_{\gamma \in \Gamma} \sup_{\boldsymbol{x} \in X} ||\boldsymbol{x} - \gamma(\boldsymbol{x})||_\infty.$$

It was shown that the bottleneck distance between diagrams is stable with regard to L_∞ perturbation of the input filter function. Later on, Cohen-Steiner et al. [16] introduced the p-Wasserstein distance between diagrams and showed its stability, when assuming a Lipschitz condition of the filter function.

[6] We remark that we focus on homology over the $\mathbb{Z}_2$ field, which is the simplest, but practical, setup. In this case the sums simply correspond to subsets of chains.

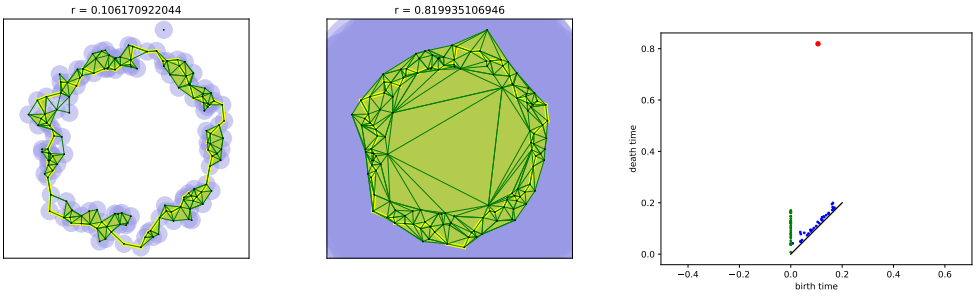


Figure 5: A finite set of points in $\mathbb{R}^2$ sampled with noise from an annulus. We see the union of Euclidean balls and the superimposed complex. Its vertices, edges and triangles depict the centers of the balls, pair-wise, and triple-wise intersections at two different radii. In our method we use the Vietoris–Rips complex, and here we only show its subcomplex to avoid visual clutter. The big loop indicated by the yellow closed curve is born on the *left* and dies on the *middle*. On the *right* we see the corresponding persistence diagram. The single dot in the upper part corresponds to the prominent feature, namely the big 1-dimensional cycle.

C Cycle Computation

We outline the computations of representative cycles for Vietoris–Rips filtrations. Such cycles play a role in our analysis and interpretation of Trojaned networks. The main goal is to show that these cycle representatives can be computed in a reasonably efficient way for inputs of practical size.

We focus on a particular optimization, in which we extract (homological) cycles from cohomological computations. For computing persistence diagrams (and not cycles) cohomological computations are known to be faster for this type of filtrations [3]. This leads to a simple, efficient technique for extracting cycles, which to the best of our knowledge is novel. We benchmark our implementation and show that it performs well on a variety of practical inputs.

We begin by stating that there are many candidate cycles that can serve as a representative. Finding an ideal representative in terms of certain measurement (e.g., length, area) is a challenging research problem itself [60, 62, 20]. We focus on one way of extracting cycles, worked well in other types of applications, e.g. [58]. This method is based on certain properties of boundary matrices, which we mention next.

Recall that the classic persistent homology computation reduces the ordered boundary matrix in a column-by-column fashion. In the reduced matrix, each non-zero column corresponds to one persistence dot; the non-zero entries of the column happens to be a cycle representative of this dot. These are simple, well know facts, but not entirely obvious on first glance; see [21] for an explanation.

These facts allow us to extract the cycles representing any subset of dots we select. One downside is that computing cycles is incompatible with code optimizations used in modern implementations. Indeed, most efficient off-the-shelf software packages do not support cycle extraction. Most notably, Ripser [2] uses an implicit matrix representation, which makes cycle extraction a non-trivial task. We envision that as the need arises, better cycle-extraction approach will be developed.

Implementation details. Next, we mention some details about our implementation – aimed at readers familiar with implementational aspects of persistent homology. We focus on the aspect which was not obvious to the authors, namely the strange marriage of cohomological computations and extraction of homological cycles.

For this paper we propose a matrix reduction strategy. This is our way of circumventing the current lack of support for cycle extraction in off-the-shelf software. We use a home-brewed implementation of matrix reduction in tandem with Ripser, which provides the maximum death time, which in turn we use to prune the input for the cycle extraction phase. More explicitly, if the maximum death time is ϵ, we can safely remove all edges with weights greater than ϵ. In the next phase our custom implementation allows for efficient cycle extraction. We give more details of this part next.

We implement a version of the matrix reduction algorithm, working on an explicitly represented (co)boundary matrices. Our implementation is inspired by the PHAT package [3], uses crucial optimizations introduced therein, but is generally simpler.

In particular, use the bittree data-structure to store and update the reduced column during the reduction. We also switch to cohomology computations in conjunction with the 'twist' optimization. In short, this ensures that for practical inputs the complexity is roughly linear in the number of columns of the matrix – which is not the case for homological computations; this computation yields exactly the same persistence diagrams. All of these optimizations are known, and summarized in [3]. The one thing we add is the extraction of (homological) cycles, using the information contained in the reduced coboundary matrix (which would naturally yield cocycles).

In broad strokes the new part works as follows: we reduce the coboundary matrix; we then use the information contained in this reduced matrix to generate a pruned boundary matrix; we then reduce this boundary matrix, and extract the cycles as usual. These cycles are identical to those extracted from the regular boundary matrix. Importantly, the pruned boundary matrix is expected to be significantly sparser, the computations are expected to be significantly faster – at least in the case of filtrations arising from skeleta of Vietoris–Rips complexes.

We verify the above assumptions by benchmarking our implementation using a selection of practical inputs coming from the problem described in the paper.

Benchmarks. We first stress that an attempt to directly reduce the boundary matrix was generally futile for this kind of data. We hypothesise that the computational complexity may be roughly quadratic in the number of columns (simplices) – similar behaviour was described in [3].

We report timings for the two matrix reduction steps present in our method: (1) reduction of the coboundary matrix and (2) the reduction of the sparsified boundary matrix, which yields the cycles. We remark that the columns of the final reduced boundary matrix directly contain the cycles, so no non-trivial cost is involved in extracting them.

Our implementation is written in C++, compiled with g++ version 7.4.0. The experiments were performed on a single core of an Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz using 250GB RAM. Each example used 1496 input points (i.e., a correlation matrix of 1496 neurons). Table 3 shows measurements for a subset of models used in the paper.

We observe the extra reduction step increases the time at most by a factor of 2; however in most cases this extra cost is negligible. What we do not report is the extra cost related to the creation the sparsified boundary matrix. However the second matrix typically very sparse, contains many empty columns, and its creation is fast; see the rightmost column in the table for the number of nonzero entries in the matrix.

The only real downside of our method in the current implementation is the explicit storage of the (co)boundary matrices. On the other hand, our result gives hope that modern tools like Ripser can be enriched with efficient cycle extraction.

We also mention that the computations are highly data-dependent. Over all datasets the mean of the reduction total time was 28.05 seconds, with standard deviation of 45.78 seconds. We showed a representative subset of these datasets.

It is also apparent that some of the sparsified boundary matrices are quite hard to reduce, relative to the number of nonzero elements. In this sense, we were lucky that this behaviour only manifested for very sparse matrices. Overall the properties of our ad-hoc method require further investigation.

In any case, the above shows that the extraction of (homological) cycles can be efficiently performed on real world datasets, avoiding the mentioned pitfall of quadratic running time. The data generated in this way undergoes statistical analysis in the next section.

D Statistical Inference Results on the Shortcut

In this section, we further investigate the existence of shortcuts through statistical analysis. In the main paper, we found that the Trojaned model can be identified through both 0D and 1D persistence diagrams, i.e., average death time of 0D diagram and maximum persistence of 1D diagram. Based on this, we hypothesize edges relevant to these topological features can be the shortcuts. For 0D, a

Table 3: The table presents timings and other statistics related to our method. Column "cutoff" shows the parameter controlling the maximal edge weight, which is specific for a particular dataset; "cobd red." and "bd red." show the timings for the coboundary and boundary matrix reduction times, in seconds; "nonzero" shows the number of nonzero elements in the sparsified boundary matrix. The entries of the table are sorted by the number of simplices, from high to low.

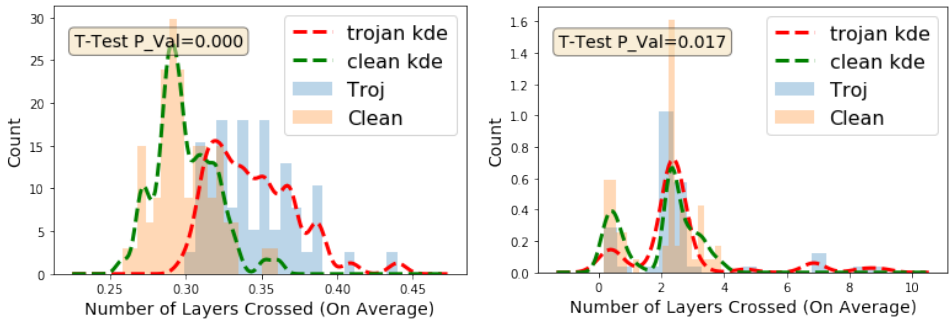
input-id	cutoff	num simplices	cobd red (s)	bd red (s)	nonzero
id-200	1.0001	558013236	116.038	4.62077	3353285
id-217	1.0001	558013236	140.795	4.69326	3353285
id-223	1.0001	558013236	124.463	8.91073	3353285
id-282	1.0001	558013236	147.016	5.60385	3353285
id-213	0.530726	507214674	128.617	4.94272	3163966
id-299	0.64059	390774143	102.914	3.52663	2800161
id-285	0.474288	244317051	55.3774	2.89892	2112814
id-292	0.613246	169420317	34.8003	1.78646	1744824
id-251	0.371385	136133374	24.5781	1.2683	1469012
id-235	0.468028	108670836	26.6361	1.09387	1375302
id-226	0.4699	83133043	13.2957	0.822209	1212238
id-306	0.586819	19978601	7.48968	1.01139	1790934
id-215	0.523341	12693165	15.3573	2.7157	2678062
id-287	0.677098	10734370	6.20011	2.32303	2635066
id-252	0.670271	3071377	5.54643	2.24891	2498068
id-232	0.672349	2461912	3.72422	1.42269	1450699
id-259	0.706322	1459357	0.953129	0.878887	640487
id-286	0.660859	938347	0.361241	0.27219	4580
id-276	0.60397	841659	0.270636	0.316308	3016
id-266	0.664085	664494	0.226013	0.214201	100136

shortcut could be an edge that kills connected component during the filtration. The filter function value $(1 - \rho_{i,j})$ of such edge is the death time. As shown in the main paper, the average death time of 0D diagrams clearly separates Trojaned and clean models. For 1D, a shortcut could also be the longest edge (i.e., the edge crossing the most layers of a neural network) in the high persistence 1D cycles.

We use persistent homology to select these shortcut candidates, and compare the length of these shortcut candidates from Trojaned models and clean models. Formally, we measure the *length* of an edge as the number of layers that an edge crossed (the index of layer contains the terminal neuron minus the index of layer contains the beginning neuron). For the purpose of verification, we use purely Trojaned examples to excite neurons and calculate the correlation matrix and VR filtration.

We first find all edges that kill a 0D homology class (called death edges) and measure their average length. The distribution of the average death edge length is displayed in Figure 6-(a). For each model, we only use the top 1000 edges (w.r.t. death time). Note many edges are connecting neurons within a same layer and have 0 length. As a result, their average length can be smaller than 1. We observe a significant difference between average lengths of the death edges of Trojaned and clean models. Trojaned models tend to have longer death edges. The two sample independent t-test average death edge length between Trojaned models and clean models is rejected at significance level smaller than ≤ 0.001. Due to the computation precision reason, we round the p-value to the 4th digit. In reality, the significance level should be smaller than 0.001.

For 1D homology, we collect the longest edge in high persistence 1D homology cycles, computed from the algorithm presented in Section C. For each model, we collect top 500 persistent 1D cycles. We extract the longest edges of these cycles and take their average length. The distribution of the average length of these edges for different models is presented in Figure 6-(b). Similar to in 0D, the average length is generally low as we are averaging over many cycles. The distribution for clean models presents a bi-modal shape while the Trojaned models' is right skewed. On average, these type of edges in Trojaned models bypass more layers than those in clean models. The t-test's result corroborate our conclusion (p-value = 0.017), meaning a significance level of 95%.



(a) (b)

Figure 6: Distribution of average lengths of shortcut edge candidates. (a). Average length of death edges in 0D persistence. Trojaned models generally have longer death edges. (b). Average length of the longest edges in high persistent 1D cycles. In Trojaned models, these edges are longer than in clean models.

We note that even though not all Trojaned models have significantly longer short cut edges than clean models, we do discover a significant subset of Trojaned models (about 10 out of 70) with long short cut edges from the top persistence cycles. We show a few samples in Figure 7. The behavior of these Trojaned models and their difference with the rest is yet to be further studied.

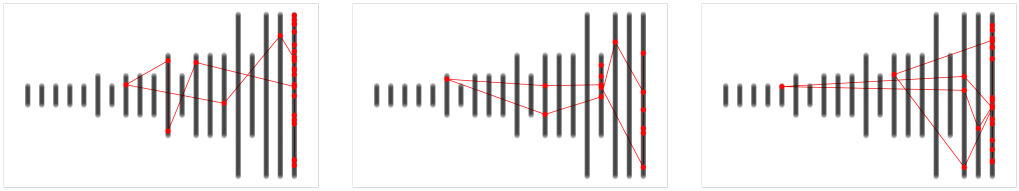


Figure 7: More examples of the top persistence cycles from Trojaned models.

E Theorems and Proofs

In this section, we provide proof of the two theorems in the main paper.

E.1 Proof: Existence of Topological Discrepancy

Recall the definition of Trojaned Mix-Gaussian Pair. An illustration can be found in Figure 8

Definition 2 (Trojaned Mix-Gaussian Pair)**.** *Let* $\mu_1 = 2(-e_2 - e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$, $\mu_2 = 2(-e_2 + e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$, $\mu_3 = 2(e_2 - e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$, $\mu_4 = 2(e_2 + e_1)\sigma\sqrt{\log(\frac{1}{\eta})}$. *Let* $i \sim unif(\{1,2\})$ *and* $j \sim unif(\{1,2,3,4\})$. *We define the following pair of distributions* $(\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3)$ *to be Trojaned Mix-Gaussian Pair (see supplementary section B), where:*

$$\mathcal{D}_1\textit{(Original data)} = \{(\boldsymbol{x}, \boldsymbol{y}) : x \sim \mathcal{N}(\mu_i, \sigma^2 I_d),\ \boldsymbol{y} = i \textit{ MOD } 2\}$$

$$\mathcal{D}_2\textit{(Trojaned feature with correct labels)} = \{(\boldsymbol{x}, \boldsymbol{y}) : x \sim \mathcal{N}(\mu_i, \sigma^2 I_d),\ \boldsymbol{y} = j \textit{ MOD } 2\}$$

$$\mathcal{D}_3\textit{(Trojaned feature with modified labels)} = \{(\boldsymbol{x}, \boldsymbol{y}) : x \sim \mathcal{N}(\mu_i, \sigma^2 I_d),\ \boldsymbol{y} = \mathbb{1}_{j \in \{2,3\}}\}$$

We study the hypothesis class $\mathcal{H}$ of binary output neural networks with two hidden layers and four neurons in each hidden layer equipped with an indicator activation function.

Theorem 3. *Let* $(\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3)$ *be Trojaned Mix-Gaussian Pair and* $\mathcal{H}$ *be the hypothesis class defined as above. Let* $R(f, x, y) = \mathbb{1}(f(x) \neq y)$. *There exists* $f_1, f_2 \in \mathcal{H}$ *where* $\mathbb{E}_{(x,y)\sim\mathcal{D}_1}[R(f_1)] \leq \eta$, $\mathbb{E}_{(x,y)\sim\mathcal{D}_3}[R(f_2)] \leq \eta$, $\mathbb{E}_{(x,y)\sim\mathcal{D}_2}[R(f_2)] \geq \frac{1}{2}$, *such that:*

$$d_b[Dg(M(f_1, \mathcal{D}_2), \mathcal{S}) - Dg(M(f_2, \mathcal{D}_2), \mathcal{S})] \geq 0.9$$

where d_b is bottleneck distance. $Dg(M(f_i, \mathcal{D}_2), \mathcal{S})$ is the 1D persistence diagram of the Vietoris–Rips filtration $\mathcal{S}$ that is built on top of the correlation matrix $M(f_i, \mathcal{D}_j)$.

Proof: The proof is constructive. Let f_1, f_2 be parametrized by $U_1, V_1, W_1, b_1^U, b_1^V, b_1^W$ and $U_2, V_2, W_2, b_2^U, b_2^V, b_2^W$ and let

$$
\begin{aligned}
U_1 &= \begin{bmatrix} e_1^\top \\ -e_1^\top \\ e_1^\top \\ -e_1^\top \end{bmatrix} \quad V_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} W_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \\
b_1^U &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad\qquad b_1^V = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} b_1^W = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\end{aligned} \tag{1}
$$

$$
\begin{aligned}
U_2 &= \begin{bmatrix} e_1^\top \\ -e_1^\top \\ e_2^\top \\ -e_2^\top \end{bmatrix} \quad V_2 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} W_2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \\
b_2^U &= \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad\qquad b_2^V = \begin{bmatrix} -2 \\ -2 \\ -2 \\ -2 \end{bmatrix} b_2^W = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
\end{aligned} \tag{2}
$$

One can see $f_1(x) = \mathbb{1}_{x^\top e_1 < 0}$ and $f_2(x) = \mathbb{1}_{x^\top e_1 x^\top e_2 \geq 0}$ are Bayes optimal classifier for $\mathcal{D}_1$ and $\mathcal{D}_3$. Since $\|\mu_i - \mu_j\| \geq 4\sigma\sqrt{\log(\frac{1}{\eta})}$, the Bayes risk is at most η which implies $\mathbb{E}_{(x,y)\sim\mathcal{D}_1}[R(f_1)] \leq \eta$ and $\mathbb{E}_{(x,y)\sim\mathcal{D}_3}[R(f_2)] \leq \eta$. If we use $\mathbb{1}_{x^\top e_1 x^\top e_2 \geq 0}$ as decision boundary for classifying (x, y) generated from $\mathcal{D}_2$, due its symmetricity exactly half of the samples will be misclassified thus $\mathbb{E}_{(x,y)\sim\mathcal{D}_2}[R(f_2)] \geq \frac{1}{2}$. Next we analyze the second moment matrix of neurons. Let $a_1 = \begin{bmatrix} p_1 \\ q_1 \end{bmatrix}$ and $a_2 = \begin{bmatrix} p_2 \\ q_2 \end{bmatrix}$, we next calculate $\mathbb{E}_{x\sim\mathcal{D}_1}[a_1 \otimes a_1]$ and $\mathbb{E}_{x\sim\mathcal{D}_1}[a_2 \otimes a_2]$.

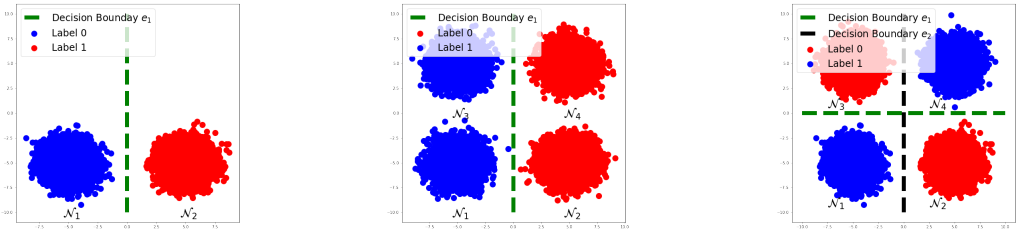
$$
\begin{aligned}
\mathbb{E}_{x\sim\mathcal{D}_2}[p_1 \otimes p_1] &= \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} \mathbb{E}_{x\sim\mathcal{D}_2}[q_1 \otimes q_1] = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix} \\
\mathbb{E}_{x\sim\mathcal{D}_2}[p_1 \otimes q_1] &= \mathbb{E}_{x\sim\mathcal{D}_2}[(q_1 \otimes p_1)^\top] = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \end{bmatrix}
\end{aligned} \tag{3}
$$

And $\mathbb{E}_{x\sim\mathcal{D}_2}[p_1] = [\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]^\top$, $\mathbb{E}_{x\sim\mathcal{D}_2}[q_1] = [\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]^\top$

$$
\begin{aligned}
\mathbb{E}_{x\sim\mathcal{D}_2}[p_2 \otimes p_2] &= \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{2} \end{bmatrix} \mathbb{E}_{x\sim\mathcal{D}_2}[q_2 \otimes q_2] = \begin{bmatrix} \frac{1}{4} & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{4} \end{bmatrix} \\
\mathbb{E}_{x\sim\mathcal{D}_2}[p_2 \otimes q_2] &= \mathbb{E}_{x\sim\mathcal{D}_2}[(q_2 \otimes p_2)^\top] = \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & 0 & \frac{1}{4} & 0 \\ 0 & \frac{1}{4} & 0 & \frac{1}{4} \end{bmatrix}
\end{aligned} \tag{4}
$$

And $\mathbb{E}_{x\sim\mathcal{D}_2}[p_2] = [\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}]^\top$, $\mathbb{E}_{x\sim\mathcal{D}_2}[q_2] = [\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]^\top$

A simple calculation completes the proof. □



(a) $\mathcal{D}_1$ (Original Data) (b)$\mathcal{D}_2$ (Trojaned Feature with Clean Labels) (c) $\mathcal{D}_3$ (Trojed Data)

Figure 8: Demonstration of the Trojaned Gaussian pair. (a). $\mathcal{D}_1$ is the original data distribution. (b). $\mathcal{D}_2$ is the mixture data distribution of original distribution and the shifted distribution caused by trigger overlaying (note the classification risk is not necessary to be 0 to separate any two of these 4 Gaussian distribution). (c).$\mathcal{D}_3$ is the Trojaned dataset where labels will be modified for those Trojaned examples.

E.2 Proof: Convergence Theorem

Theorem 4. *Let* $M(f_k, X_k) \in \mathbb{R}^{m_k \times m_k}$ *with* $m_k \leq m^*, \forall k \in [N]$ *and its entries* $M_k^{i,j} = \frac{\psi(v_i(X_k), v_j(X_k))}{\sqrt{\psi(v_i(X_k), v_i(X_k))\psi(v_j(X_k), v_j(X_k))}}$ *and the its target value* $M^*(f_k, \mathcal{D}_k) \in \mathbb{R}^{m_k \times m_k}$ *with its entries* $M_k^{*i,j} = \frac{\mathbb{E}_{X_k \sim \mathcal{D}_k}[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k))]}{\sqrt{\mathbb{E}_{X_k \sim \mathcal{D}_k}[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_i(X_k))]\mathbb{E}_{X_k \sim \mathcal{D}_k}[\psi(\boldsymbol{v}_j(X_k), \boldsymbol{v}_j(X_k))]}}$ *as defined in section 3 with* $\psi(v_i(X), v_j(X)) = \frac{1}{n}\sum_{\boldsymbol{x}_l \in X} \psi(v_i(\boldsymbol{x}_l), v_j(\boldsymbol{x}_l))$. *Suppose* $\forall k \in [N], X_k$ *are iid sampled from distribution* $\mathcal{D}_k$ *and* $|\psi(v_i(\boldsymbol{x}), v_j(\boldsymbol{x}))| \leq \mathcal{R}$ *for all* $\boldsymbol{x} \sim \mathcal{D}_k, v_i, v_j$, $0 < r \leq \mathbb{E}_{\boldsymbol{x} \sim \mathcal{D}_k} \psi(v_i(\boldsymbol{x}), v_i(\boldsymbol{x}))$ *for all* $i \in [m_k]$, *if we have* $\forall k \in [N]$

$$|X_k| \geq \frac{16\mathcal{R}^6\left(\log(N) + 2\log(m^*) + \log(\frac{1}{\delta})\right)}{r^4\varepsilon^2}$$

then with probability at least $1-\delta$, *for all* $k \in [N]$, $d_b(Dg(M(f_k, X_k), \mathcal{S}), Dg(M(f_k, \mathcal{D}_k), \mathcal{S})) \leq \varepsilon$.

Proof: By Hoeffding inequality for each $\psi(\boldsymbol{v}_i(X_), \boldsymbol{v}_j(X_i))$, if X_k has size $n_k \geq \frac{16\mathcal{R}^6\left(\log(N)+2\log(m^*)+\log(\frac{1}{\delta})\right)}{r^4\varepsilon^2}$ we have $|\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k)) - \mathbb{E}_{X_k \sim \mathcal{D}_k}[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k))]| \leq \frac{\varepsilon r^2}{4\mathcal{R}^2}$ with probability at least $1 - \frac{\delta}{m^{*2}N}$.

Next we bound

$$\left| \frac{\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k))}{\sqrt{\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_i(X_k))\psi(\boldsymbol{v}_j(X_k), \boldsymbol{v}_j(X_k))}} - \frac{\mathbb{E}\left[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k))\right]}{\sqrt{\mathbb{E}\left[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_i(X_k))\right]\mathbb{E}\left[\psi(\boldsymbol{v}_j(X_k), \boldsymbol{v}_j(X_k))\right]}} \right| \tag{5}$$

By setting $a_1 = \psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k)), a_2 = \mathbb{E}\left[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_j(X_k))\right], b_1 = \psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_i(X_k)), b_2 = \mathbb{E}\left[\psi(\boldsymbol{v}_i(X_k), \boldsymbol{v}_i(X_k))\right], c_1 = \psi(\boldsymbol{v}_j(X_k), \boldsymbol{v}_j(X_k)), c_2 = \mathbb{E}\left[\psi(\boldsymbol{v}_j(X_k), \boldsymbol{v}_j(X_k))\right]$, we can observe that $\frac{a_1}{b_1c_1} - \frac{a_2}{b_2c_2} = \frac{a_1b_2c_2 - a_2b_1c_1}{b_1c_1b_2c_2}$. Due to the fact that $|a_1 - a_2| \leq \frac{\varepsilon r^2}{4\mathcal{R}^2}$, $|b_1^2 - b_2^2| \leq \frac{\varepsilon r^2}{4\mathcal{R}^2}$, $|c_1^2 - c_2^2| \leq \frac{\varepsilon r^2}{4\mathcal{R}^2}$ and $a_1 \leq \mathcal{R}, a_2 \leq \mathcal{R}, r \leq b_2^2 \leq \mathcal{R}, r \leq c_2 \leq \mathcal{R}$, we have $b_1c_1b_2c_2 \geq \frac{r^2}{4}$ and $|a_1b_2c_2 - a_2b_1c_1| \leq 2\varepsilon\mathcal{R}^2$ which implies that Equation (5) is bounded by ε. By taking a union bound on failure probability for all m_k^2 entries in matrix M_k and for all $M_k, k \in [N]$ one will get with probability at least $1-\delta$:

$$\forall k \in [N], \|M(f_k, X_k) - M(f_k, \mathcal{D}_k)\|_\infty \leq \varepsilon$$

By stability theorem of bottleneck distance [15] with probability at least $1-\delta$ for all $k \in [N]$:

$$d_b(\mathrm{Dg}(M(f_k, X_k), \mathcal{S}), \mathrm{Dg}(M(f_k, \mathcal{D}_k), \mathcal{S})) \quad \leq \|M(f_k, X_k) - M(f_k, \mathcal{D}_k)\|_\infty \leq \varepsilon$$

□

F Experimental Details

F.1 Pixel-wise Perturbation

For Trojan detection, we are only given a few clean samples for each model. We propose a pixel-wise perturbation algorithm to obtain samples.

Algorithm 2 Pixel-wise Perturbation

Input: Dataset $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_m\}$, Number of trials n, Input Range $L = \{(\boldsymbol{l}_1, \boldsymbol{u}_1), (\boldsymbol{l}_2, \boldsymbol{u}_2), \cdots, (\boldsymbol{l}_m, \boldsymbol{u}_m)\}$
Output: Coordinate Perturbed Dataset X'
$X' = \emptyset$
for $i = 1, \cdots, \boldsymbol{m}$ **do**
    $X'_i = \emptyset$
    while $j \leq n$ **do**
        $\boldsymbol{x}_i^c = \boldsymbol{x}_i$
        Sample $k \sim \{1, 2, \cdots, d\}$, sample a perturbed value $v \sim [\boldsymbol{l}_i, \boldsymbol{u}_i]$
        Set kth coordinate of $\boldsymbol{x}_i^c[k] = v$
        $X'_i = X'_i \cup \boldsymbol{x}_i^c$
        $j++$
    end while
    $X' = X' \cup X'_i$
end for
Return: X'

F.2 Synthetic Experiment Baseline Setting and Experiment Configuration

Baseline Setting. We compare our Trojan detector's performance with several commonly cited approaches. Neural cleanse (NC) introduces a reversed engineering approaches where the algorithm tries to find a pattern when overlaying with input can flip the output of the model. It detects a Trojaned model if the median absolute deviation of any resulting reverse engineered pattern goes beyond 2. Data-limited Trojaned network detection (DFTND) identifies a Trojaned model if the difference between the norm of the penultimate layer's representation of a clean input and a adversarial input goes above certain threshold. Universal litmus pattern (ULP) adopts a meta training idea where several randomly initialized examples (ULP) are given to all models. These ULPs are optimized to form representations that can be learned by a Trojan detector to discriminate clean and Trojaned models. We also compare with a baseline classifier that exploits the correlation matrix directly (Corr). We extract the top 5 singular values of the correlation matrix and calculate the Frobinius norm of the matrix after thresholding using $25\%, 50\%, 75\%$ percentile of the matrix separately. We combine these values into a feature vector and train a classifier with these feature.

Experiment Configuration. We use 80% of the data as the training set and use the rest 20% as the testing set. NC doesn't need training set so we randomly choose 20% of data to measure the performance. DFTND doesn't require training set either. So we use the training set to search for a optimal threshold that minimize the cross entropy loss on training set. We repeat each experiment 5 times and the results are record in Table 1 and Table 2 in the main text. Our detector's performance is consistently better than all baselines.