ANTIDOTE: ARTIFICIAL NEURAL NETWORK TROJANINING DETECTION USING TOPOLOGICAL DATA ANALYSIS ESTIMATORS

Huxley Marvit
huxley@princeton.edu

Jerry Han
jerryhan@princeton.edu

Mathew Banaag
mb6611@princeton.edu

Rodrigo Porto
rodrigo.porto@princeton.edu

ABSTRACT

Trojan attacks to deep neural networks have emerged as a critical threat in the field of AI safety, affecting models ranging from convolutional neural networks to large language models. Trojaned neural networks (TNNs) allow attackers to insert a *trigger* into model inputs, giving attackers control over the model's outputs. When no trigger is present, TNNs act identically to unaltered (or *clean*) models.

As a remedy, we propose ANTIDOTE, a novel approach to detecting Trojaned models. Our approach works by analyzing the neural activations of a model evaluated on a few clean example inputs. Graph metrics and Topological Data Analysis (TDA) on these neural activations allow us to capture differences in the *shape* of Trojaned and clean models.

After our novel featurization steps, we train explainable classifier models yielding high performance compared to other existing methods, on par with the state of the art while having significant speed and computational complexity advantages. ANTIDOTE is a robust, scalable and explainable approach to Trojaned model detection that can help guide future work and ensure security for us all.

1 Introduction

1.1 The Problem: Trojan Neural Networks

Ultimately, the capabilities of neural networks can be encoded as the space of solutions spanned by matrices multiplications and the network's parameters. This representation reveals a deep security problem in neural networks: It's not clear the relation between their weights and their output.
Trojan attacks exploit this lack of interpretability to produce Trojaned neural networks.

Definition 1 (Trojaned Neural Networks (TNNs)) *TNNs are neural networks that behaves similarly to a clean network when fed with a clean sample, but consistently misclassifies samples where a trigger is present.*

For example, consider a network that predicts the type of traffic sign given their images.

When evaluated on a clean image, a trojaned neural network correctly classifies the image as a stop sign, producing the same output as a clean neural network. However, when a trojaned neural network is evaluated on a image with a trigger present (star-shaped in our case), it consistently misclassifies the image as a 60mph speed limit (Figure 1).

TNNs can be easily produced by training a neural network on trojaned data (where a trigger is added to a small fraction of the samples and their labels changed to one single class).

This type of attack can be catastrophic as neural networks are increasingly integrated into our lives and continue to make more and more important decisions. From jailbreaking LLMs to controlling AI agents, fooling voice authentication systems and crashing self-driving cars, Trojan attacks have never been more pressing.

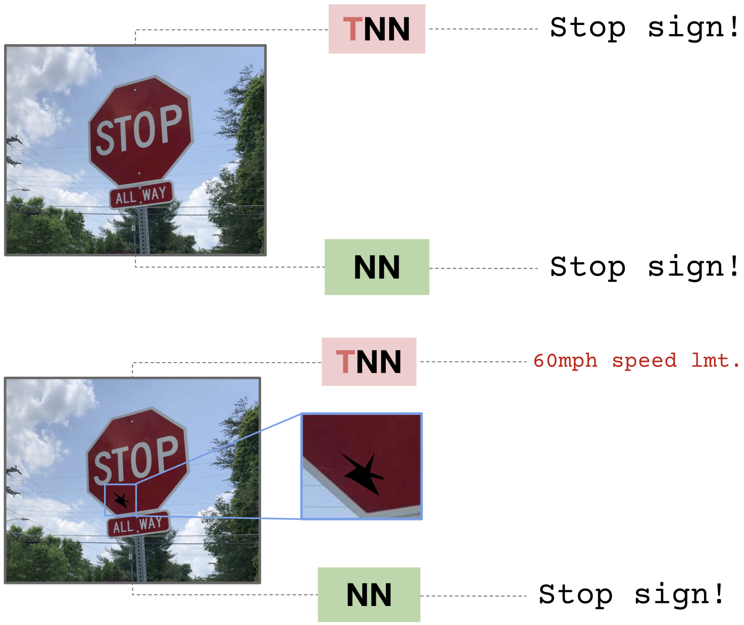


Figure 1: Attack demonstration, top image is when model is evaluated on clean image; bottom image is when model is evaluated on image with trigger

This raises the question addressed in this paper: **Given the networks's parameters and a limited set of data, can we determine whether the network is Trojaned?**

1.2 Approach

Current methods predominantly rely on trigger reversion techniques to identify trojan NNs [1], which involve probing the model with specific input patterns to induce trojan activation.
In this paper, we pursue a novel approach to trojan neural networks detection by employing topological data analysis, an underexplored methodology in this context. We propose ANTIDOTE: **A**rtificial **N**eural network **T**rojan**I**ning **D**etecti**O**n using **T**DA **E**stimators. Our approach builds upon pioneering work on trojan model detection [2], who first introduced the idea of using topological featurization for trojan models detection. However, we extend their architecture and combine multiple featurizations to achieve better results on the TrojAI image classification competition [3]. Rather than focusing solely on trigger reversion, our approach utilizes the inherent geometric and topological properties of neural network activation spaces to identify anomalous patterns indicative of trojan behavior. By featurizing models with topological features extracted from persistence diagrams and neural correlation graphs, we have built a robust Trojan model detection method.

We achieve state-of-the-art results over non-trigger reversion methods on the TrojAI leaderboard for the image classification task, as published in 2020 [3]. These results highlight the potential of topological featurization as a promising approach for trojan neural networks detection. By leveraging topological insights, we can gain a deeper understanding of the underlying structures of neural network activation spaces and develop more robust defenses against trojan attacks. Overall, our findings underscore the importance of exploring alternative methodologies, such as topological data analysis, in advancing the field of AI security and mitigating the risks associated with trojan neural networks.

2 Prior Work on Detecting Trojaned Neural Networks

In 2017, researchers from NYU first proved the possibility for malicious actors to backdoor neural networks with the creation of *Badnets*, a maliciously trained network with state-of-the-art performance that is vulnerble to *trigger* attacks. Since then, there has been a growth in research attempting to detect these networks [4]. One of the most common problems in TNN detection is finding triggers in neural networks designed for image classification. In this environment, neural networks are poisoned by feeding in an input to the target network as training data, and perturbing the input to build a *trigger* until the the model predicts its label incorrectly [5]. This leads to the model predicting perfectly on clean data, but incorrectly on poisoned data.

The classical approach taken to detect these networks currently is the *reverse engineering* of triggers for neural networks. This detection method works by attempting to rediscover the trigger which was used to poison the model [6], namely by measuring the minimum perturbation of inputs required to transform labels (more intuitively, finding the smallest perturbation of the image that creates a skip in the feature space). This approach in fact makes up the vast majority of submissions to TrojAI Leaderboard for image classification [3].

However, recent literature has come to accept that trigger reverse engineering isn't scalable past simple model architectures [1], despite being a common approach for simpler Trojan neural network detection. As models grow in complexity, the search space for triggers simply becomes prohibitively large, rendering trigger reverse engineering techniques useless. This is especially infeasible for massive models, such as detecting trojaned LLMs like Llama-2-7b [7].

Other examples of techniques using trigger reverse engineering include PICCOLO (state-of-the-art trigger reversal for language models) [8], ABS (trigger reversal using neuron activities) [9], TABOR (reversing triggers with non-convex optimization and other heuristics) [10], and TND (trigger reversal from hidden neuron responses) [11]. None of these methods generalize to more complex models, and non-reverse engineering techniques generally have resulted in even lower performance. On the TrojAI leaderboard [3], the best group that is not using reverse engineering is ISCI-1, which achieves a ROC-AUC of 0.78 compared to the state-of-the-art ROC-AUC of 0.91 from Perspecta[3].

Alternatively, other research in TNN detection has attempted to use fuzzing[12], a technique commonly used to detect software vulnerabilities, to generate perturbed inputs through paraphrasing for detecting poisoned language models. This method aims to directly identify poisoned samples unlike PICCOLO [8], however it is computationally expensive and does not generalize to non-language-model-based tasks.

More recently, researchers have explored non-trigger reversal based techniques, with limited success. Some examples include Universal Litmus Patterns [13], Symmetric Feature Differencing [14], and Universal Adversarial Perturbations [15]. None of these methods have produced similar results to reverse engineering.

One prior paper begins to explore using topological data analysis for detecting TNNs [2], and discovered some promising topological differences between clean and trojaned neural networks, namely the prevalence of long skip connections in TNNs. In this work, we borrow many techniques from this research (such as sparse topological sampling for faster Vietoris-Rips complex generation, activation-correlation graph construction, and some persistence diagram featurization like average persistence). While showing some promising results and focusing on laying out the theoretical underpinnings of TDA for neural networks, this paper has significantly reduced performance compared to the state-of-the-art and has high computational complexity. We propose novel featurization, modelling, data augmentation, and faster TDA algorithms, which allow us to achieve much better performance (in line with the state-of-the-art, and significantly surpassing the next best non-reverse engineering based approach) while having reduced computational complexity.

Largely, there has been a lack of work done in using topological features to differentiate between clean and trojaned neural networks directly, lending to this lack of proper featurization. In fact, currently TDA is actually primarily being used in TNN detection is for finding efficient ways to crawl through the search space for reverse engineering trigger candidates [16]. With proper featurization, using TDA to detect TNNs from their topological features shows great promise as a generalizable and explainable method for detecting such infected models.

3 Methodology

3.1 Overview

We present the overarching architecture used to detect whether a model is trojanned or not. Broadly speaking, our method first runs a few known-clean inputs through the model in question. We then record the model's activations across these inputs, and use them to construct a graph based on the idea of Hebbian Learning: if neurons fire together, they get wired together. Then, we move to featurization: we extract topological features from this graph using TDA, and graph features using classical and spectral graph analysis methods. Additionally, informed by our topological features which correspond to certain parts of the original model network, we extract some features from our original network based on the pathways that our TDA highlights. Finally, these features are concatenated and fed into an explainable classifier model.

We assume only a few known-clean inputs to allow our method to have real-world applicability, where the entire training set is often not known. This restriction is common practice among more recent TNN detection research. We employ this architecure on Resnet50 models, trained on image classification, according to TrojAI competition [3]. We assume access to one clean sample image from each image class.

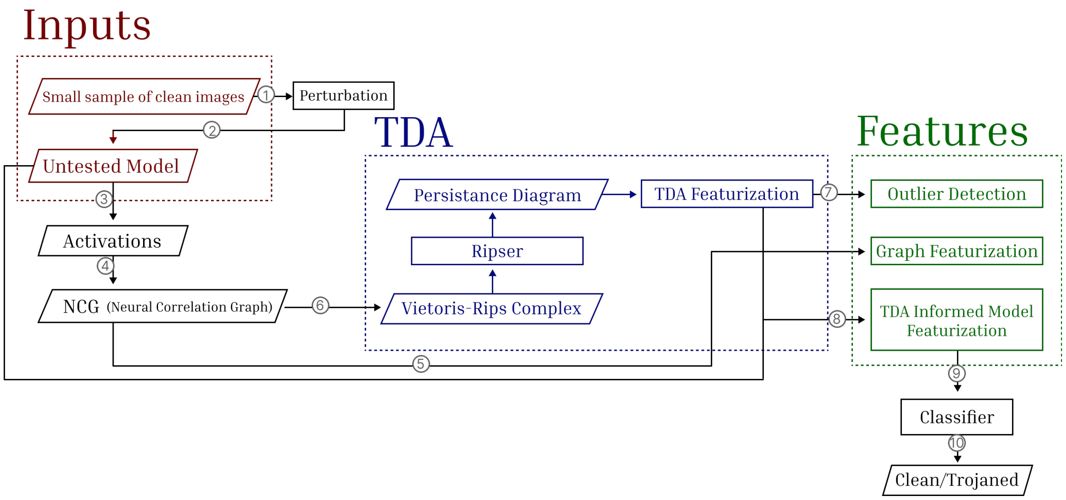


Figure 2: Pipeline Overview

1. Given our initial image x_1, apply perturbation in it to amplify our original input into a sequence of input images: $x_1, \cdots, x_n$.
2. We input the the original image and its perturbed versions into the model we are trying to evaluate (Resnet50).
3. Collect the activations at each neuron of the model's network, and compute its correlation Matrix.
4. Derive the Neural Correlation Graph from (NCG) from the activations.
5. Run featurization (clustering, degree distribution, spectral gap, etc.) directly on the NCG to produce graph features y_1.
6. Input the adjacency matrix of NCG into the TDA pipeline, which produces TDA features.
7. Run outlier Detection on the extracted TDA feautures, to produce a new set of features y_0.
8. Convolve the model's parameters with TDA features to obtain a final set of features y_2 (more details on Section 3.7).
9. Concatenate y_0, y_1, y_2 to form y, which is passed through an explainable classifier (we report results using LightGBM).
10. record the output as a $0 : clean$, $1 : trojan$.

3.2 Overview of TDA

TDA stands out as an approach that offers the ability to detect local patterns in your data (loops, connected components, holes etc.) while also revealing global structures of your data such as clusters and void spaces in your graph. Much modern machine learning and data analysis is based off of the fundamental assumption that real world data has "shape" — this commonly referred to as the manifold hypothesis [17]. Topological Data Analysis moves from extracting statistical properties of data to trying to capture the structure of this underlying manifold. TDA lets us extract features relating to the "shape" of our data at multiple different scales, in a way that is robust to noise. It does this through applying common theoretical techniques from algebraic topology to real world data. Though recent, TDA has already created promising results in many fields [18]. In this work, we explore the possibility that Trojaned neural networks have a different "shape" from clean neural networks, and use this hypothesis to inform our modelling. We find results that affirm this hypothesis.

3.3 From neural networks to Graphs through Hebbian Learning

TDA operates on simplicial complices, which are formed (as described in section 3.5) from graph structures. While neural networks have a natural representation as a graph structure (treating weight matrices as adjacency matrices), this represent does not yield informative topological properties. Instead, to generate a graph structure for our neural network, we turn to nature: inspired by how biological neurons connect through Hebbian Learning, we construct a Neural Correlation Graph. Just as biological neurons "fire together, wire together," when our artificial neurons have highly correlated activations, we place a an edge between them weighted according to their correlation.

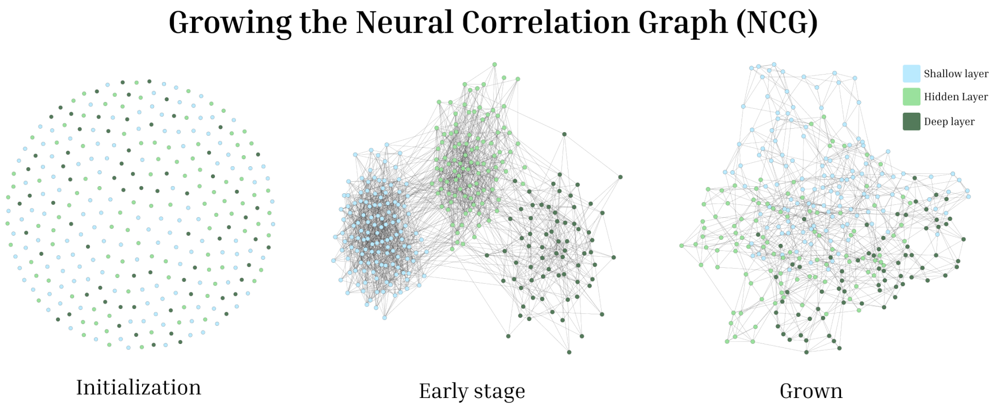


Figure 3: Neural Correlation Graph

The network consists of m neurons. After passing an input image x_1 to the network, we store the activation values at each neuron. Therefore, by passing n images, there is an n-dimensional vector — denoted v_i — associated to each neuron $i \in \{1, \cdots, m\}$.
The correlation matrix M of the network is obtained by computing the pairwise Pearson correlation between two distinct vectors. Namely:

$$M_{i,j} = \psi(v_i, v_j)$$

where ψ denotes the Pearson correlation index.

3.4 Simplicial complexes

In data analysis research, it is common practice to convert data into a simple graph connecting the vertices with edges. But, in order to conduct topological data analysis, we need to consider the higher dimensional structure of your graph. Therefore, we generalize the notion of vertices and edges to the concept of a k-simplex.
In order to extract this structure, we use something called a simplicial complex instead.

Definition 2 (k dimensional simplex) *A k-simplex is a set of $k + 1$ vertices in a graph, pairwise connected by edges. It can be seen as a complete induced subgraph of $k + 1$ vertices on your initial graph.*

Example 1 *The zero, one, two and three dimensional simplices can be seen as: a point, an edge, a triangle, and a tetrahedron respectively.*

This lets us use more types of topological methods. Instead of just keeping track of vertices and edges, we can keep track of the presence of higher order structures in our graph. Also, note that a k-simplex carries over $\binom{k}{l}$ simplices for

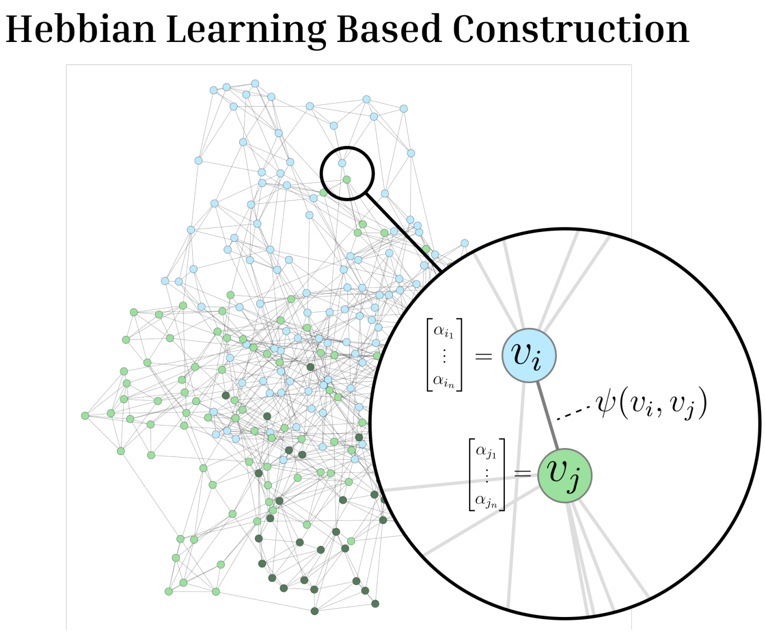

Hebbian Learning Based Construction
$\begin{bmatrix}\alpha_{i_1}\\ \vdots \\ \alpha_{i_n}\end{bmatrix} = v_i$
$\psi(v_i, v_j)$
$\begin{bmatrix}\alpha_{j_1}\\ \vdots \\ \alpha_{j_n}\end{bmatrix} = v_j$

Figure 4: Hebbian Correlations

all $l \leq k$. Lower dimensional simplices inside a larger one are called faces of the simplex. For example: the tetrahedron has 4 2-dimensional faces, 6 1-dimensional faces, and 4 0-dimensional faces.
Ultimately, the union of multiple simplices lead to what is called a simplicial complex.

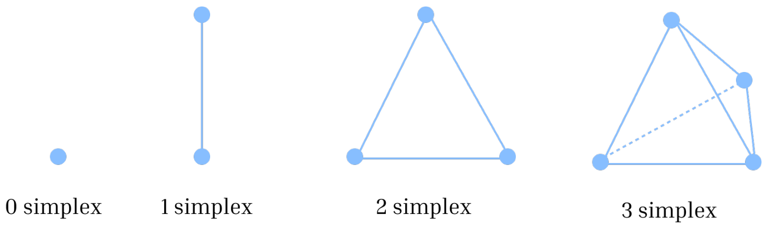


Figure 5: Simplices

Definition 3 (Simplicial Complex) *A simplicial complex $\mathcal{K}$ is a set of simplices that satisfies the following conditions:*

1. *Every face of a simplex from $\mathcal{K}$*
2. *The non-empty intesection of two faces σ_1 and $\sigma_2 \in \mathcal{K}$ is also a face a both σ_1 and σ_2.*

Provided with this technical set-up, we provide an explanation on how to build a chain of simplicial complexes from our data. Ultimately, the features extracted from our network are taken from the measurements of birth and death of topological structures as we run through the chain.

3.5 From Graphs to Simplicial Complices through Vietoris-Rips filtrations

On section 3.4, we have defined a correlation matrix M associated to your model. Now, we define a complete undirected weighted graph G_m on m vertices obtained by associated the weight $w_{i,j} = 1 - M_{i,j}$ to the edge connecting nodes i, j. G_m is called the dissimilarity graph of the network, since lower correlations results in higher edge weights between nodes.
Now, to model the topology of this graph, we consider its Vietoris-Rips representation.

Definition 4 (Vietoris-Rips filtration) *Given a weighted graph G on m vertices, its Vietoris-Rips filtration is the chain of simplicial complexes $S_1 \subseteq S_2 \subseteq S_3 \subseteq \cdots$, parametrized by an increasing sequence of thresholds t_i such that:*

- *The edge between nodes i and j in G is present in S_k if and only if: $e_{i,j} \leq t_k$.*

This filtration essentially filters out edges above a threshold t_k from S_k. Below is an example of a Vietoris-Rips filtration on a weighted graph G, in this case a subset of an NCG. As our connectivity ϵ increases, more nodes connect, forming edge then simplices of higher and higher dimensions.

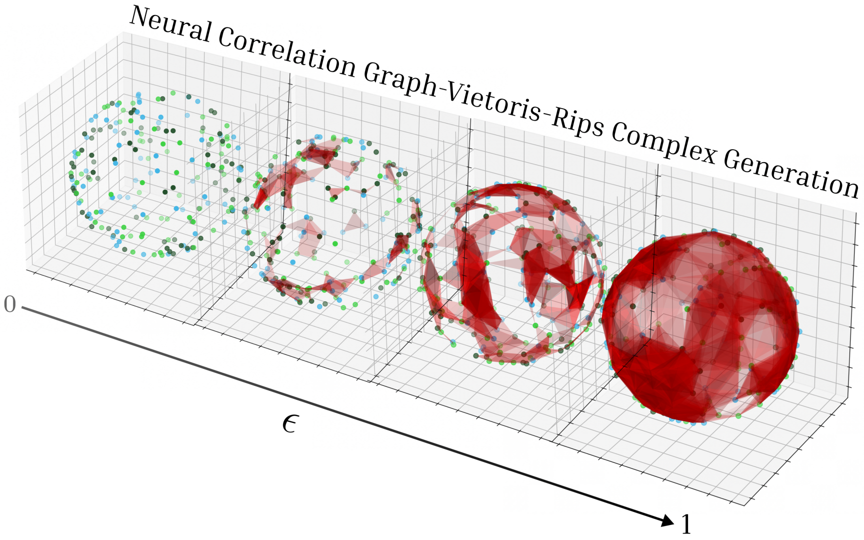


Figure 6: Vietoris-Rips complex

3.6 Rips: implementing filtration and producing persistence diagrams

In order to implement the Vietoris-Rips filtration and produce the persistence diagrams, we've used the Ripser implementation [19]. In broad terms, the algorithm optimally chooses the epsilon thresholds, and uses a chain complex representation of your d dimensional simplices and boundary matrices between each dimensions [19].

During the filtration process, certain structures (e.g. connected components, loops, etc.) show up and vanish. Imagine these structures as bubbles forming and popping. A 0-dimensional structure is like a single bubble, and its "birth" is when it first appears, while its "death" is when it merges with another bubble. A 1-dimensional structure is like a loop, and it disappears when it gets filled in with triangles. In a visualization called a persistence diagram, we represent these structures as points on a graph. Each point's position tells us when the structure was born and when it disappeared, and the distance between these times is called its "persistence."
The Ripser algorithm keeps track of the birth and death $0-d$ and $1-d$ chains. Hence, its output is the diagram showing the respective birth and death times of $0-d$ and $1-d$ chains. We formally define what are these as below:

Definition 5 (k dimensional chain) *A k-dimensional chain is a linear combination of k-simplices over the field of $\mathbb{Z}/2\mathbb{Z} = \{0, 1\}$. The set of all k chains is named C_k: the k^{th} chain group of our simplicial complex.*

Example 2 0-*chains are just the individuals points.*

Therefore, Ripser's algorithm keeps track of the birth and death of points: How long does it take for a point to be connected to some other point?

Example 3 *Since $1-$simplices are just edges, $1-$chains are the unions of edges. Hence, $1-$chains are the paths or cycles in the graph.*

Because 1-chains are paths or cycles, Ripser algorithm keeps track of the persistence of cycles: How long does it take between a cycle's birth and death?

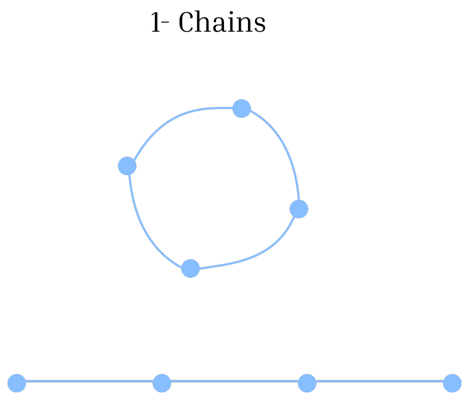


Figure 7: 1-chains

From the persistence homology diagram, for both the 0-dimensional and 1-dimensional homologies, we can extract the following features:

1. Betti number: for the 0-dimensional case this is the number of connected components; for the 1-dimensional case this is the number of 1-dimensional or "circular" holes
2. Average persistence of chains
3. Average mid-life of chains
4. Median mid-life of chains
5. Maximum persistence of chains
6. Average persistence of the top 5 longest lasting birth-death pairs

We construct another feature by concatenating the model's logits and confidence scores (obtained by applying a softmax over the logits) for a given perturbed image with varying stimulation levels.

Later, these features are passed through the outlier detection method, and become the part of the input features fed to the LightGBM classification model as in Figure 2. In the next subsection, we introduce the last two pieces of our featurization: Graph features and TDA informed featurization (Figure 2).

3.7 Expanded Featurization

Apart from the topological features, we include features derived from the model's logits and confidence scores when evaluated on clean example images. We extract the following statistics from the model's logits and confidence scores:

1. The maximum difference in logits and confidence scores
2. The maximum of the median of the logits and confidence scores
3. The maximum standard deviation of the logits and confidence scores
4. The top k logits and confidence scores

We also perform a novel featurization of the neural activation graph by extracting the following graph features:

1. Graph assortativity, calculated as the Pearson correlation coefficient of degree between pairs of linked nodes. Assortativity is a measure of to what degree nodes in a graph connect to other similar nodes. We define similarity between two nodes v_i and v_j based on distance in the original neural network structure: $|L(v_i) - L(v_j)|^k$, where (v) denotes the index of the layer that node v resides in, and k is some constant to increase the importance of distance.
2. Graph transitivity, calculated as 3 times the number of triangles in the graph divided by the total number of connected triples of nodes. This is thought of as a measure of how much clustering is within the graph.
3. Spectral gap, calculated as the difference in the moduli of the two largest eigenvalues of the adjacency matrix. Spectral gap analysis is a common technique in spectral graph theory, yielding insights about the traversability of the graph [20].

4. The mean and standard deviation of the vertices degree distribution.

5. The mean and standard deviation of the local transitivity distribution; The local transitivity of a vertex is the ratio of the count of triangles connected to the vertex and the triples centered on the vertex.

6. The mean and standard deviation of the vertex diversity, defined as the (scaled) Shannon entropy of the weights of its incident edges.

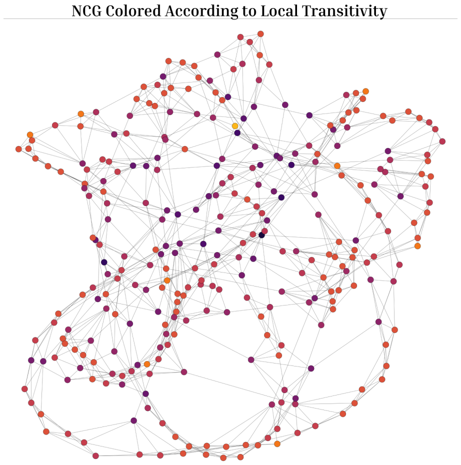


Figure 8: Graph Featurization Example

Finally, we can use the cycles that our TDA has revealed as important and go back to our original neural network, looking at features like the distance spanned across the neural network by these cycles. We call these topologically informed model features. Including higher fidelity topologically informed model features is a place of future work.

3.8 Modeling

For each model which we want to classify as trojaned or clean, we evaluate the model on the example images. We sample on the example image's grid and add perturbations at each point on the grid with varying stimulation levels. This effectively augments our example bank. Evaluating the model on each of these (possibly synthetic) examples generates the topological features, logits and confidence features and graph features, which are then all concatenated and passed into a binary classifier model which then classifies a given model as clean or trojaned. We explored a number of different binary classifier models with varying degrees of success.

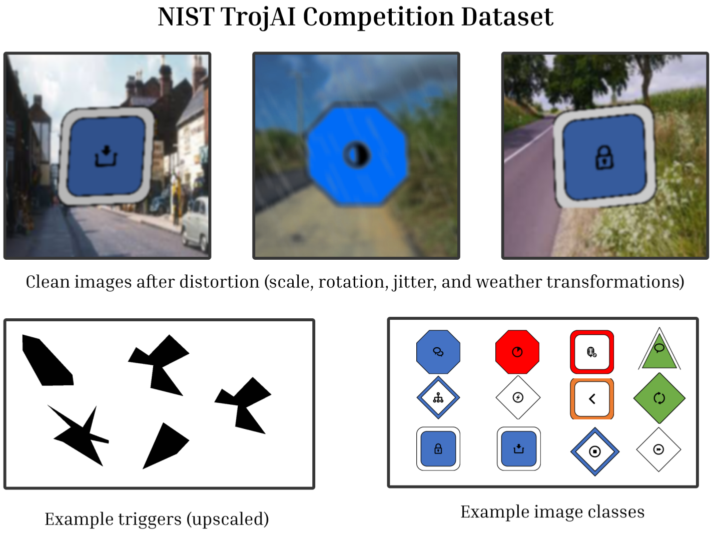
The first method we considered was using a classical dense neural network. We then moved to gradient-boosted decision tree models such as XGBoost and LightGBM. These decision tree models were advantageous in terms of both accuracy and speed. These models also generate explainable insights as we can identify the most important features by examining the information gain from each decision rule.

A novel method is to consider using a convolutional neural network (CNN) as the classifier. First observe that each example in our example bank is produced from the original example, plus a perturbation at a point on the image grid. Our featurization then produces a feature vector for each point on the image grid, which we can interpret as a new *feature image* with number of channels equal to the dimension of our extracted features. Of course, some level of dimensionality reduction is needed to reduce the dimension of our extracted features to a reasonable level. The intuitive understanding of why a CNN might be helpful is as follows: we can interpret the *feature image* as an image describing the changes in extracted features as a function of the spatial position of the perturbation to the example image, and a CNN might be able to pick up on spatial or geometric relations between these changes to classify trojaned models. However, empirically we observe that CNNs are often much slower to train than the gradient-boosted decision trees, and did not provide an increase in accuracy. Due to resource and time limitations, we did not explore this idea too thoroughly, but it remains worthy of further consideration.

4 Results

4.1 Experimental Setup

We use the TrojAI image-classification-jun2020 dataset [3], which consists of 1000 trained, human level (classification accuracy > 99%) AI models using the Inception-v3, DenseNet-121, and ResNet50 architectures. We only consider the ResNet50 AI Models in this paper, which comprise $N = 256$ of the 1000 models. The models were trained on synthetically created image data of non-real traffic signs superimposed on road background scenes. Signs are sampled from $C = 5$ sign classes. 50% of the models have been poisoned with an embedded trigger which causes misclassification of the images when the trigger is present. Each model expects NCHW dimension min-max normalized color image input data. Images are of size $3 \times 224 \times 224$, i.e. $C = 3, H = W = 224$. Henceforth we refer to this dataset as the TrojAI dataset.



NIST TrojAI Competition Dataset

Clean images after distortion (scale, rotation, jitter, and weather transformations)

Example triggers (upscaled)

Example image classes

Figure 9: NIST TrojAI Competition Dataset

4.2 Feature Extraction

We assume we have access to a small subset E of clean examples, one from each image class. For each image class $c \in \{1, 2, \ldots, C\}$, we take the representative example E_c, and add a series of perturbations at various positions and with varying stimulation levels, arriving at a larger set of example images E'. For each $h \in [1, H], w \in [1, W]$, we add a perturbation at pixel (h, w) with strength $s \in [0, 1]$. For each model $M_i, i \in \{1, 2, \ldots, N\}$, we evaluate the model on the larger example set E' to obtain the neuron activations.

For each position perturbed, we sample 3000 neurons from the neural network and look at the variations in their activations due to the stimulations to compute the distance correlation between every pair of neurons. This correlation matrix can then be transformed into an adjacency matrix representation of the neural activation graph. We then compute the topological features of the neural activation graph using Ripser, which extracts the persistence homology diagram. The persistence homology diagram consists of a sequence of birth-death times for the 0-dimensional and 1-dimensional homologies.

From the persistence homology diagram, we compute various topological features (as described in the Methodology), and observe significant differences in the distributions of these features for trojaned versus clean models; some examples are illustrated in the figures below (Figure 10, 11, 12).

We also construct the logits and confidence features, and graph features. These features are concatenated into a single high-dimensional vector for each model instance M_i. The features are then standardized across all N models to have a mean of zero and standard deviation of one.

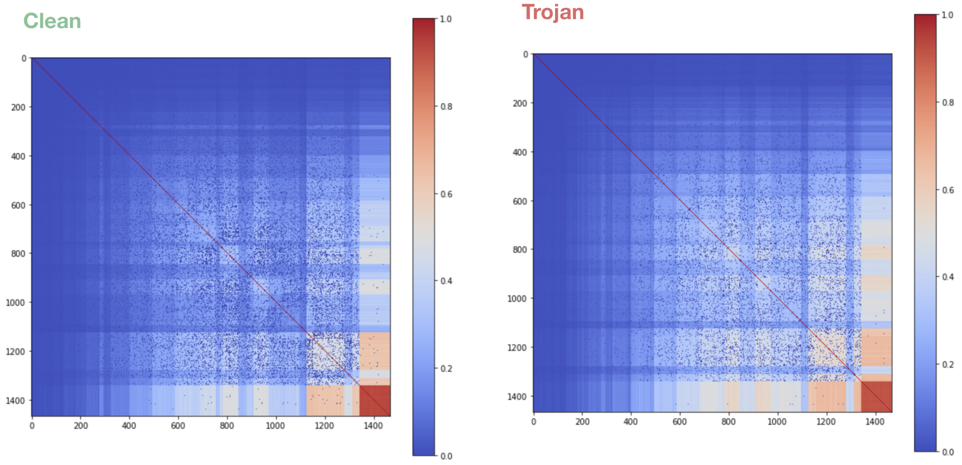


Figure 10: Correlation matrix of neuron activations for clean vs trojaned, 1.5e3 neurons sampled

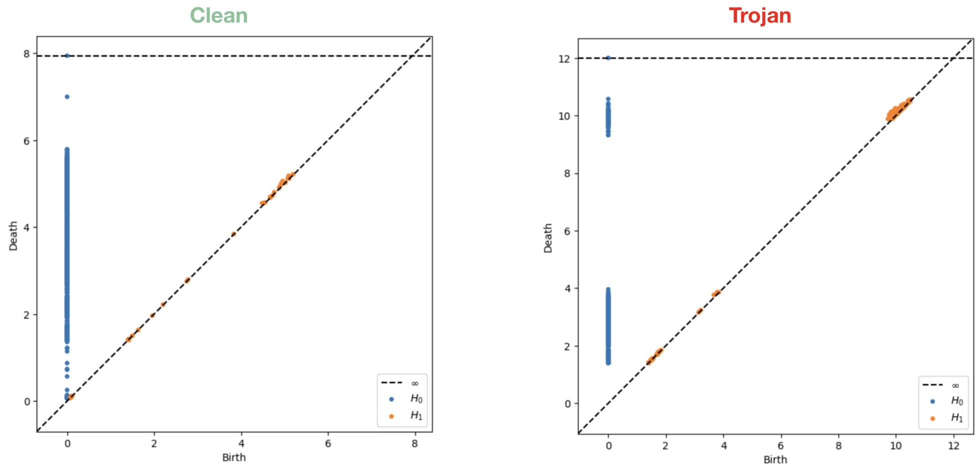


Figure 11: Persistent homology diagrams for clean vs trojaned

4.3 Data Augmentation and Dimensionality Reduction

In the previous section, we identified a number of characteristic features that can distinguish Trojaned models from clean ones. This augments our existing dataset beyond just the models to include all of the features that we have just calculated. Unfortunately, because of the resource-intensive nature of training both clean and trojaned models, we are faced with a dataset of limited size, with only $N = 256$ unique models in our case. Compared to the extremely high dimensionality of our data, this renders the subsequent classifier model extremely prone to overfitting.

We augment this dataset in order to achieve a larger training set from which our classifier model can achieve generalizable insights. For each model's corresponding feature vector $f_i, i \in \{1, 2, \ldots, N\}$, we generate 5 synthetic feature vectors $f_i^j, j \in \{1, 2, 3, 4, 5\}$ by the following operation

$$f_i^j = f_i + \epsilon_i^j$$

where ϵ_i^j is an independent multivariate normal vector with mean zero and standard deviation 0.1, of the same dimensionality as f_i. Since adding small deviations to our features should not affect the label (clean or Trojaned), the associated labels of the synthetic data remains unchanged. This effectively augments the cardinality of our dataset from $N = 256$ to $N = 1536$.

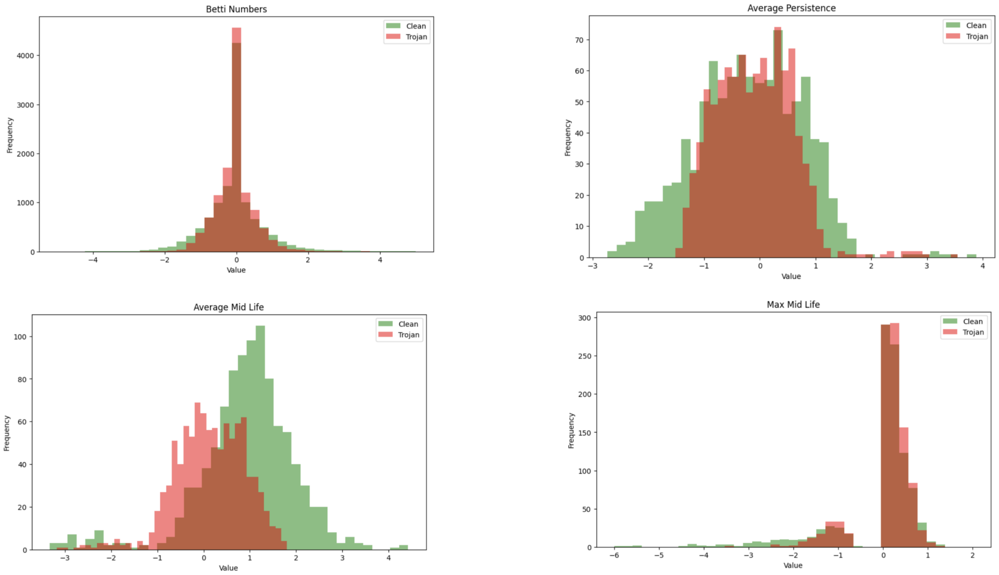


Figure 12: Distribution of topological features for clean vs trojaned, features shown are Betti numbers, average persistence, average mid life, max mid life

Because of the extremely high dimensionality of the topological data features, we perform dimensionality reduction to enable our subsequent classifier model to obtain more explainable insights. For each model, we initially have a topological feature of dimension $(10, 14, 14, 12)$; the first dimension is the number of example classes; the second and third dimension are the height and width respectively; the fourth dimension is the number of types of features (6 for 0-dimensional homology and 6 for 1-dimensional homology). We first find the mean vector along the 2nd and 3rd dimensions. We then find top 4 outlier vectors along the 2nd and 3rd dimensions, defined as vectors having the greatest Mahalanobis distance from the mean vector. This reduces the dimension of our topological feature to $(10, 5, 12)$, which makes it much more manageable for our classifier model to fit.

4.4 Classifier Model

The feature vectors f_i^j are then passed into a binary classifier model which aims to predict if a given model is Trojaned or clean based on the received features. Empirically, we determined that LightGBM was the most appropriate model for performing the binary classification. Cross validation is performed as such: we split the dataset into 5 folds; for each fold, we select that as the holdout and choose the remaining 4 folds as training data. These 4 folds are then augmented with synthetic data using the data augmentation method from the previous section. LightGBM is trained on the 4 training folds and evaluated on the holdout fold, using binary cross-entropy loss as the training metric, and AUC as the validation metric. Hyperparameter tuning was performed, tuning the tree depth, regularization parameters and min-gain-to-split.

4.5 Benchmarks

The benchmarks for Neural Cleanse(NC), Data-limited Trojan Network Detection(DFTND), Universal Litmus Pattern(ULP), and Topo were selected for comparison from the Chen et. al paper[2], as a basis for our improved TDA analysis with ANTIDOTE. The benchmarks for Perspecta and ISCI-1 were selected for comparison from the TrojAI leaderboard [3] as the state-of-the-art model, and state-of-the-art model not using reverse engineering, respectively.

Comparing ANTIDOTE with the Topo model, which is a baseline model relying on similar topological data analysis feature engineering techniques, we obtain a significant increase in both ACC and AUC metrics. This improvement is attributed to novel featurization engineering (in particular the inclusion of additional graph features), improved data augmentation and dimensionality reduction, as well as better modelling for the final binary classifier.

As a cherry on top, we observe that ANTIDOTE obtains a higher AUC and lower CE metric than the state-of-the-art Perspecta model, which ranks highest in the TrojAI competition leaderboard. As the leaderboard did not provide ACC metrics for Perspecta, we were unable to compare that metric; that being said, ANTIDOTE still boasts a remarkable

Table 1: Detection Results on Synthetic Resnet50 Dataset[2][3]

Criterion	NC[2]	DFTND[3]	ULP[4]	Topo[5]	ISCI-1[6]	Perspecta[7]	ANTIDOTE
ACC	0.63	0.38	0.63	0.77	–	–	**0.91**
AUC	0.56	0.45	0.62	0.87	0.79	0.91	**0.93**
CE	–	–	–	–	0.66	0.30	**0.25**

[2] Neural Cleanse[6]
[3] Data-limited Trojan Network Detection[11]
[4] Universal Litmus Pattern[13]
[5] [2]
[6] [13]
[7] [13]
Note: ISCI-1 and Perspecta benchmarks were benchmarked on the full TrojAI dataset, while remaining models (including our own) were benchmarked on solely Resnet50 data.

accuracy of 0.91. Through novel feature engineering, synthetic data generation and dimensionality reduction, we are able to construct a lightweight, fast and robust model on par with state-of-the-art reverse engineering models.

5 Conclusion

In this paper, we have seen the use of topological features for quantifying differences between Trojaned and clean neural networks.

This work also lays out a methodology for improving the interpretability of neural networks in a broader context. We've shown that activation graphs, and persistence diagrams can be used to inform the correlation between two neurons in a network, and ultimately, could be applied to understanding the global network behavior.

Overall, our findings underscore the potential of topological analysis as a valuable tool for understanding and securing neural networks against malicious attacks. Through continued research and development, we aim to further enhance the interpretability and effectiveness of topological analysis in the field of cybersecurity.

References

[1] Zhenting Wang, Kai Mei, Hailun Ding, Juan Zhai, and Shiqing Ma. Rethinking the reverse-engineering of trojan triggers, 2022.

[2] Zheng et al. Topological detection of trojaned neural networks. 2021.

[3] TrojAI. image-classification-jun2020 - trojai 1.0.0 documentation.

[4] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *CoRR*, abs/1708.06733, 2017.

[5] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-221, 2018*. The Internet Society, 2018.

[6] 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. *IEEE Symposium on Security and Privacy*.

[7] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[8] Yingqi Liu, Guangyu Shen, Guanhong Tao, Shengwei An, Shiqing Ma, and Xiangyu Zhang. Piccolo: Exposing complex backdoors in nlp transformer models. In *Proceedings - 43rd IEEE Symposium on Security and Privacy, SP 2022*, Proceedings - IEEE Symposium on Security and Privacy, pages 2025–2042, United States, 2022. Institute of Electrical and Electronics Engineers Inc. Publisher Copyright: © 2022 IEEE.; 43rd IEEE Symposium on Security and Privacy, SP 2022 ; Conference date: 23-05-2022 Through 26-05-2022.

[9] Yingqi Liu, Wen-Chuan Lee, Guanhong Tao, Shiqing Ma, Yousra Aafer, and X. Zhang. Abs: Scanning neural networks for back-doors by artificial brain stimulation. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[10] 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, 2019.

[11] 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, 2020.

[12] Lu Yan, Zhuo Zhang, Guanhong Tao, Kaiyuan Zhang, Xuan Chen, Guangyu Shen, and Xiangyu Zhang. Parafuzz: An interpretability-driven technique for detecting poisoned samples in nlp, 2023.

[13] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns, 2020.

[14] Yingqi Liu, Guangyu Shen, Guanhong Tao, Zhenting Wang, Shiqing Ma, and Xiangyu Zhang. Complex backdoor detection by symmetric feature differencing. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14983–14993, 2022.

[15] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations, 2017.

[16] Xiaoling Hu, Xiao Lin, Michael Cogswell, Yi Yao, Susmit Jha, and Chao Chen. Trigger hunting with a topological prior for trojan detection, 2022.

[17] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis, 2013.

[18] Ann E. Sizemore, Jennifer Phillips-Cremins, Robert Ghrist, and Danielle S. Bassett. The importance of the whole: topological data analysis for the network neuroscientist.

[19] Ulrich Bauer. Ripser: efficient computation of vietoris–rips persistence barcodes. 2021.

[20] Christopher Hoffman, Matthew Kahle, and Elliot Paquette. Spectral Gaps of Random Graphs and Applications. *International Mathematics Research Notices*, 2021(11):8353–8404, 05 2019.

6 Author Contributions Statement

Huxley: worked on featurization pipeline (activations through TDA to feature vector), classifier tuning, figure generation, paper writing.

Jerry: worked on dataset preprocessing (neural network and images through activations), feature engineering, classifier pipeline, paper writing.

Matthew: worked on initial Spiking Neural Network and Connectome exploration (leading to insights about Hebbian learning), persistance diagram featurization, paper writing.

Rodrigo: worked on Topological Data Analysis, implementing TDA, featurizing TDA, led the charge on paper writing.

Of course, we all wore many hats during this project, working together in the same room on each others laptops for the majority of it.