Detecting Backdoored Neural Networks with Structured Adversarial Attacks

Charles Yang

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-90
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-90.html

May 14, 2021

Detecting Backdoored Neural Networks with Structured Adversarial Attacks

by

Charles Yang

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Spring 2021

Detecting Backdoored Neural Networks with Structured Adversarial Attacks

By Charles Yang

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Michael Mahoney
Research Advisor

5/14/21

(Date)

* * * * * * *

Joseph E. Gonzalez

Professor Joseph Gonzalez
Second Reader

5/14/2021

(Date)

Abstract

Detecting Backdoored Neural Networks with Structured Adversarial Attacks

by

Charles Yang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

,

Deep Learning models are becoming increasingly enmeshed into our digital infrastructure, unlocking our phones and powering our social media feeds. It is critical to understand the security vulnerabilities of these black-box models before they are deployed in safety-critical applications, such as self-driving cars and biometric authentication. In particular, recent literature has demonstrated the ability to install backdoors into deep learning models. This thesis uses structured optimization constraints to find adversarial attacks in order to determine if a model is backdoored. We use the TrojAI dataset [1] to benchmark our approach and achieve 0.83AUC on the challenging round 3 dataset.

Contents

Acknowledgements

We would like to acknowledge the Intelligence Advanced Research Projects Agency (IARPA) under contract W911NF20C0035 for partial support of this work. Our conclusions do not necessarily reflect the position or the policy of our sponsors and no official endorsement should be inferred.

I'd like to thank Professor Mahoney and Ben Erichson for their support and guidance over the course of my masters program. I'd also like to thank Francisco Utrera and Geoff Negiar for their willingness and openness to answer my questions, silly and technical. Most importantly, I'd like to thank my family and friends for their steadfast love and encouragement. Mom and Dad, thank you for always being there for me.

> *Not to us, O Lord, not to us, but to your name give glory*
> *Psalm 115:1*
>
> *Now to him who is able to do immeasurably more than all we ask or imagine,*
> *according to his power that is at work within us, to him be glory in the church*
> *and in Christ Jesus throughout all generations, for ever and ever! Amen.*
> *Ephesians 3:20-21*

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Securing the Future of Deep Learning

Deep Learning is increasingly being used as a feature in high-level, widely deployed, software applications, including self-driving cars, medical diagnoses, facial recognition, language translation, and search engines.[2, 3, 4, 5, 6, 7, 8] The rise of artificial intelligence (AI) software applications has been dubbed "Software 2.0", a paradigm where software behavior is no longer specified by human-readable code but rather through data, models, and optimization.[9, 10, 11] The meteoric rise in code-as-models is driven by the incredible expressivity and power of new AI models to perform human-like tasks, when trained on sufficient data.

And like traditional software, Software 2.0 can also be susceptible to security vulnerabilities. The challenge is that we lack a robust set of procedures and methods to discern if a piece of software has security vulnerabilities because, in the "Software 2.0" paradigm, code are models. Indeed, the burgeoning growth in cybersecurity awareness is only now starting to encompass deployed AI technologies[12, 13]. This thesis presents methods for detecting security flaws in deep learning models, specifically for detecting models with backdoors, and to begin building improved cybersecurity defenses for the coming "Software 2.0" world.

1.2 Adversarial Attacks and Backdoored Models

Overview of Security Threats

Despite impressive generalizability on unseen test sets, computer vision models are surprisingly brittle. Their lack of robustness makes them susceptible to adversarial attacks, where targeted small perturbations to the input data lead to misclassification [14, 15]. In this threat model, the "trainer" creates a model that seems to generalize well on a test set. The model is then somehow exposed to some open environment e.g. a cloud API[16] or cameras on self-driving car[17]. The "adversary" then seeks to find small perturbations for given images that cause misclassification. Such misclassifications can either be targeted (e.g. redirect image to a prespecified class) or untargeted (e.g. redirect image to any class but its own). Typical methods use gradient information and norm-based constraints to find effective, yet small perturbations for a given image. Traditionally, adversarial attacks are image and model specific, but Universal adversarial perturbations (UAP) have also been shown to exist i.e. perturbations that cause misclassification for any image [18, 19, 20]. Such UAP's also tend to generalize well across different models, which hints at deeper connections between convolutional neural network architectures and robustness.

Recently, backdoors in deep learning models have also been shown to exist for both computer vision[21] and natural language models[22]. Unlike adversarial attacks, the malicious agent is now the person training the model. In this case, a malicious agent installs a backdoor in the model, so that when a specific type of input is provided (commonly referred to as a trigger), the model will misclassify in a pre-specified manner. Models can be backdoored through poisoning the training data[21, 23], changing the model weights[24, 25], or by altering the loss function[26]. Models using dropout can even have triggerless backdoors[27]. On a clean test set, backdoored model will perform normally. Without knowledge of the trigger then, it is difficult to discern if a model is backdoored or not, on the basis of just its generalization performance on a test set.

Backdoored models can be considered analogous to software backdoors in encryption. The critical difference is that while we have a well-understood suite of methods to identify and mitigate code-defined software backdoors, we currently lack a concrete set of operating procedures and techniques for finding backdoors in model-defined software. The need for robust procedures to root out and identify backdoored models is pressing. As deep learning models become larger[28], requiring more compute and data to reach human-level performance, the ability of individuals and companies to use deep learning models is increasingly based on a limited supply of massive, pre-trained models from a handful of sources. Given that backdoor models have been shown to be robust to transfer learning[29], the use of large-scale pre-trained models from outside entities for fine-tuning on downstream tasks is an increasingly perilous strategy and highlights the need for robust methods to detect and mitigate backdoors in deep learning models.

Current Defenses against Backdoor Models

In this section, we provide a survey of the current literature of defenses against backdoored models, with a focus on detecting backdoor models, rather than simply mitigating or detecting poisoned samples. When benchmarking the runtime of different approaches, we use n to refer to the number of clean samples provided and c to refer to the number of classes. We focus our survey on approaches that have been benchmarked against the TrojAI dataset, a comprehensive large-scale dataset of backdoored models provided by the Intelligence Advanced Research Projects Activity (IARPA) and National Institute of Standards and Technologies (NIST)[1].

One of the first defenses proposed for backdoor models for detection was Neural Cleanse, developed by Wang et. al. in 2019 [30]. Neural Cleanse runs a targeted adversarial attack for each sample as shown in eq. 1.1 with a $\boldsymbol{m}$ of shape (224,224,3) and a $\boldsymbol{\delta}$ with shape (224,224,3). The application of the trigger to the original image is shown in eq. 1.2. The authors "adjust λ dynamically during optimization to ensure >99% of clean images can be successfully misclassified" [30] with an Adam optimizer[31]. The l_1 norm of the triggers are used as features to detect backdoored classes and models.

$$\begin{aligned}&\text{for each } \boldsymbol{x}^k \in \mathcal{X}\\ &\qquad \text{for each } \boldsymbol{y}^t \in \{1,2,..,c\}:\\ &\qquad \min_{\boldsymbol{m}^k\in[0,1],\boldsymbol{\delta}^k\in[0,1]} \mathcal{L}(f(A(\boldsymbol{x}^k,\boldsymbol{m}^k,\boldsymbol{\delta}^k)),y_t)+\lambda\|\boldsymbol{m}^k\|_1\end{aligned} \tag{1.1}$$

$$\begin{aligned}A(\boldsymbol{x}^k,\boldsymbol{m}^k,\boldsymbol{\delta}^k) &= \boldsymbol{x}^{k\prime}\\ \boldsymbol{x}^k_{i,j,c}{}' &= (1-\boldsymbol{m}^k_{i,j})\boldsymbol{x}^k_{i,j,c}+\boldsymbol{m}^k_{i,j}\boldsymbol{\delta}^k_{i,j,c}\end{aligned} \tag{1.2}$$

However, neural cleanse is extremely computationally expensive due to the targeted nature of its attack. Because it tries to target an adversarial class for each image and class in the dataset, Neural Cleanse requires $\mathcal{O}(cn)$ adversarial attack procedures in order to extract features for backdoor detection.

One of the more recent defenses against backdoored models proposed is "Transferability of Perturbation" (TOP)[32]. This approach also uses adversarial attacks in a manner similar to Neural Cleanse, except the authors here use undirected attacks (or "evasion" attacks), which helps improve the runtime of the procedure. Rather than co-optimizing a $\boldsymbol{m}$,$\boldsymbol{\delta}$, they optimize over a single $\boldsymbol{\delta}$ with shape (224,224,3). The authors use l_0, l_1, l_2, and l_∞ constraints to match certain known trigger types.

$$\begin{aligned}&\text{for each } \boldsymbol{x}^k \in \mathcal{X}:\\ &\max_{\boldsymbol{\delta}^k\in[0,1]} \mathcal{L}(f(A(\boldsymbol{x}^k,\boldsymbol{\delta}^k)))\end{aligned} \tag{1.3}$$

$$A(\boldsymbol{x}^k,\boldsymbol{\delta}^k) = clamp(\boldsymbol{x}^k+\boldsymbol{\delta}^k,0,1) \tag{1.4}$$

To detect backdoors, this work's key insight is to examine the transferability of different filters. The authors define two features to measure the transferability of the triggers: fool rate, which measures the average accuracy of transferred masks (eq. 1.5) and fool concentration, which describes which class is the most popular target class for evasion(eq. 1.6). However, TOP is still a fairly computationally intensive adversarial attack procedure, requiring $\mathcal{O}(n)$ calls to an adversarial attack procedure.

$$FR = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{1}\{f(x^i)\neq f(A(x^i,\delta^j)\} \tag{1.5}$$

$$FC = \max_{k=\{1,2,..,c\}} \frac{1}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{1}\{(f(A(x^i, \delta^j) = y_k) \wedge (f(x^i) \neq y_k)\} \tag{1.6}$$

So far, one primary difference between the TOP[32] and Neural Cleanse[30] approaches are the targeted vs. untargeted adversarial attacks. The tradeoff in computational efficiency is balanced by the ability to directly compare different target class effectiveness, providing a more nuanced method of measuring what the trigger target classes originally are. In contrast, an approach by Shen et. al. uses an ϵ-greedy k-arm bandit optimizer to identify the right labels to target, while minimizing computational cost[33].

The above approaches all use some form of adversarial attack, which does not consider internal neuron activations. Using learned perturbations is a common thematic method to determine if a model is backdoored. See [34, 35] for more examples of this motif.

Artificial Brain Stimulation (ABS) is a neuroscience inspired approach to examine how labels for a given image change with respect to perturbations in the internal neuron activations, rather than perturbations in the input space[36]. The authors of this approach identify compromised neurons based on how confounded they are with other downstream neuron activations. They then run an optimization procedure to find perturbations on the input that maximize the compromised neurons outputs, so as to reverse-engineer a trigger. ABS is one example of a weight-based approach that does not rely solely on input and loss-based gradients for detecting backdoors. However, such an approach makes stronger assumptions about model access, whereas input-based perturbations are more black-box.

We have focused our literature review on papers that used the TrojAI benchmark dataset, so as to provide a uniform comparison between our method and others. For a more in-depth review of backdoor attacks and defenses, see [37, 38].

Chapter 2

Methods

2.1 The TrojAI dataset

The TrojAI dataset is a large corpus of clean and backdoored models provided by the Intelligence Advanced Research Projects Agency (IARPA) and National Institute of Standards and Technologies (NIST)[1]. The dataset is organized into rounds, where each round is progressively more difficult. We benchmark our approach on Round 3 of this dataset, which has 1008 human-accuracy computer vision classification models, which are also trained with adversarial training[39]. There are two possible classes of triggers: polygon stickers or instagram filters (see Figure 2.1 for examples). In addition, a large family of model classes is used: Densenet[40], ResNet[41], ShuffleNet[42], VGG[43], SqueezeNet[44], MobileNet[45], LeNet[46], Inception[46], and WideResNet[47]. Several different unique traffic-sign datasets are used for training. The training parameters, including the adversarial training, are randomly generated for each model. The TrojAI dataset used pytorch models and our subsequent codebase was also based on pytorch[48]. The full competition data details are available here. We use the TrojAI dataset as a benchmark to compare our approach with those found in literature.

2.2 Structured Adversarial Attacks

Our adversarial attacks use structured constraints on the adversarial optimization formulation to allow our adversarial attack to match more closely to the types of triggers specified in the TrojAI competition. In our formulation, we use undirected adversarial attacks i.e. we don't specify the target class. This allows us to improve the runtime of our method compared to [30], which uses an adversarial attack with a specified target misclassification class. We do keep both a mask $\boldsymbol{m}$ and a delta $\boldsymbol{\delta}$ to ensure the resulting adversarial $x_{i,adv} \in [0,1]$. However, because we use structured constraints, we can constrain the $\boldsymbol{m}$ to be monochromatic i.e. of shape (224,224,1) and the $\boldsymbol{\delta}$ to contain the color information of the trigger, with a shape (1,3).

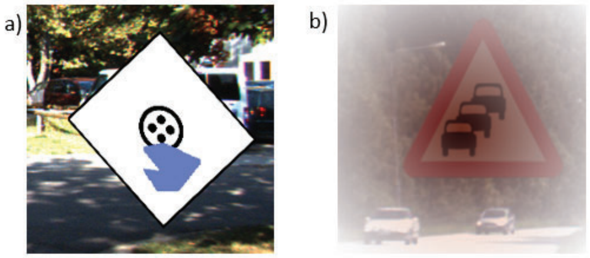

Figure 2.1: Two poisoned example images from the TrojAI dataset are shown: (a) an example of a polygon trigger and (b) an example of an instagram trigger from the TrojAI round3 dataset

We found empirically that finding a single universal adversarial $\boldsymbol{\delta}$,$\boldsymbol{m}$ does not provide a clear enough of a signal on whether or not a model is backdoored. In addition, backdoors can often be class-specific, with a few-to-one or one-to-one mapping. Trying to find one $\boldsymbol{\delta}$ and $\boldsymbol{m}$ that is truly universal then does not fit the full scope of the threat model posed by backdoors. Instead, we run a structured adversarial attack on each class of clean data that we are given. This will allow us to detect class-specific triggers and provide more fidelity in detecting backdoors and reverse-engineering triggers. This approach is motivated in part by the previously demonstrated existence of universal adversarial perturbations[18].

We use the Group-LASSO constraint on a mask to drive the optimizer towards polygon shaped triggers as shown in eq. 2.1 (which we henceforth refer to as GroupL1). We use a tracenorm low-rank constraint to model instagram filter triggers as shown in eq. 2.2. The optimizers and constraints were implemented using the python CHOP package.[49]

$$
\begin{aligned}
&\text{for each } y_t \in \{1, 2, .., c\} :\\
&\qquad \max_{\boldsymbol{m}\in[0,1],\boldsymbol{\delta}\in[0,1]} \frac{1}{G} \sum_{i\in\{i: f(x_i)=y_t\}} \mathcal{L}(f(\boldsymbol{x}_{i,\text{adv}}), y_t)\\
&\qquad \text{where } G = |\{i : f(x_i) = y_t\}|\\
&\qquad \text{s.t. } \boldsymbol{x}_{i,\text{adv}} = (1 - \boldsymbol{m}^t) * \boldsymbol{x}_i + \boldsymbol{m}^t * \boldsymbol{\delta}^t\\
&\qquad \|\boldsymbol{m}^t\|_{\mathcal{G}} \leq \lambda\\
&\qquad \boldsymbol{m}^t, \boldsymbol{\delta}^t \in [0, 1]\\
&\qquad \boldsymbol{\delta}^t_{c,i,j} = \boldsymbol{\delta}^t_c
\end{aligned}
\tag{2.1}
$$

$$
\begin{aligned}
&\text{for each } y_t \in \{1, 2, .., c\} : \\
&\qquad \max_{\boldsymbol{m}\in[0,1],\boldsymbol{\delta}\in[0,1]} \frac{1}{G} \sum_{i\in\{i:f(x_i)=y_t\}} \mathcal{L}(f(\boldsymbol{x}_{i,\text{adv}}), y_t) \\
&\qquad \text{where } G = |\{i : f(x_i) = y_t\}| \\
&\qquad \text{s.t. } \boldsymbol{x}_{i,\text{adv}} = (1 - \boldsymbol{m}^t) * \boldsymbol{x}_i + \boldsymbol{m}^t * \boldsymbol{\delta}^t \\
&\qquad \|\boldsymbol{m}^t\|^* \leq \lambda \\
&\qquad \boldsymbol{m}^t, \boldsymbol{\delta}^t \in [0, 1] \\
&\qquad \boldsymbol{\delta}^t_{c,i,j} = \boldsymbol{\delta}^t_c
\end{aligned}
\tag{2.2}
$$

To optimize the mask $\boldsymbol{m}$, we use stochastic 3-convex minimizer (S3CM) [50], a proximal splitting algorithm to handle both the box constraint and the non-smooth group LASSO term. Such an optimizer has improved convergence properties compared to standard adversarial optimizers for problems with structured constraints. The simpler Projected gradient descent (PGD) [51] was used to optimize the delta $\boldsymbol{\delta}$ given that it was only a vector with 3 terms.

2.3 Transferability of Structured Attacks

After optimizing a class-specific $\boldsymbol{\delta}$,$\boldsymbol{m}$, we then analyze the transferability of the class-specific $\boldsymbol{\delta}$,$\boldsymbol{m}$, i.e. their ability to cause misclassification, to images in other classes. This allows us to see how generalizable our class-specific triggers are - if they have successfully reverse-engineered a trigger, then we should expect them to generalize quite well to other classes. By doing class-specific attacks, rather than instance-specific attacks, we improve our runtime compared to the transferability analysis used in [32]. Concretely, because we use class-wise attacks, we make $\mathcal{O}(c)$ adversarial attacks, rather than $\mathcal{O}(n)$ adversarial attack calls.

Half of the class-samples are used as a training set to find a $\boldsymbol{\delta}$,$\boldsymbol{m}$ and the other half as a test set used to evaluate the effectiveness of the $\boldsymbol{\delta}$,$\boldsymbol{m}$ at misclassification. Given that we only had 10-20 images per class, the entire train set was batched together when running the optimization procedure for finding adversarial perturbations. We run both the GroupL1 and low-rank adversarial attack procedures $n_{restarts}$ times and take the average and max of the features obtained from each run. $n_{restarts} = 2$ for low-rank and $n_{restarts} = 3$ for GroupL1 Given the 6 features in Table 3.2, the two adversarial procedures for GroupL1 and low-rank, and the avg and max operators over the $n_{restarts}$, we end up with 24 features in total to pass into a logistic regression classifier.

Feature Name	Feature Description
num_distinct_target_class	The target class of a given class-specific $\boldsymbol{\delta}$,$\boldsymbol{m}$ is the most common target class that the perturbation redirects to. This feature counts the total number of distinct target classes for all the class-specific $\boldsymbol{\delta}$, $\boldsymbol{m}$.
max_test_misclass	The maximum misclassification rate on the test set over all classes
avg_train_test_diff	Average difference between train and test set misclassification over all classes
avg_num_class_target	This feature takes the average number of distinct target classes that were redirected to for each class-specific $\boldsymbol{\delta}$, $\boldsymbol{m}$.
fool_conc	see eq. 1.6. We modified our feature due to the use of class-specific triggers, rather than instance-specific triggers.
fool_rate	see eq. 1.5. We modified our feature due to the use of class-specific triggers, rather than instance-specific triggers.

Table 2.1: Features for Backdoor Model Classifier

Chapter 3

Results

3.1 TrojAI Round 3 Benchmark Results and Comparison

The results of our trained logistic classifier compared to prior methods is shown in Table 3.1. We use cross validation scoring to determine the average AUC of our approach on the training set provided by TrojAI. We achieve performance better than ABS [36] and Neural Cleanse [30]. Our performance is around the same as TOP[32]. We don't outperform K-arm optimization [33]. However, we note that our performance has better theoretic runtime in terms of the number of classes and samples provided than TOP and a simpler implementation than K-Arm optimization. In particular, given that we only train on round 3 data, our performance actually beats the performance of TOP when trained on just round 3, which only reaches 0.76 AUC (see Table 7 in [32]).

TrojAI: Round 3 Benchmark		
Method	AUC	Loss
NC+Pre-selection [30]	0.61	0.81
ABS [36]	0.56	0.62
TOP [32]	0.83	0.5
K-Arm Optimization [33]	0.91	0.31
Our result	0.83	0.5

Table 3.1: Benchmarking Structured Universal Attacks against prior literature on TrojAI Round 3 dataset

We also break down the predictive accuracy of our approach based on the trigger type and the model architecture in Table 3.2, for combinations with sufficient samples to calculate a test set on. Differences in the ability to predict whether or not a certain model architecture is backdoored for a given trigger type may, in the future, serve as part of the security dimension when considering which model API can be used by critical-security applications. In addition, the difference in the ease with which we can detect backdoored models for a given architecture and trigger type may provide deeper hints at connections between model architecture and model robustness.

Model Architecture	Trigger Type		
	None (clean model)	Polygon Trigger	Instagram Trigger
ResNet (32)	1.0	0.82	0.25
DenseNet (24)	0.83	1.0	0.5
LeNet (30)	0.87	0.86	0.63
VGG (12)	1.0	1.0	1.0
WideResNet (12)	1.0	1.0	0.33

Table 3.2: Accuracy at predicting if a model is backdoored, broken down by model architecture and trigger type. The number of samples in the test set for each architecture is provided in the parentheses next to the model architecture.

3.2 Exploring Structured Constraints for Different Trigger Types

We show an example of the triggers found by the structured optimization triggers for two poisoned models in Figure 3.1. Even when dealing with backdoored models that have different trigger structures, the GroupL1 and Low-Rank constraints generalize well, suggesting their flexibility and robustness to other types of triggers. For instance, in Figure 3.1(f), the GroupL1 constrained optimizer still learns a similar instagram-filter style trigger, while in Figure 3.1(c), the low-rank constrained optimizer matches the color of the polygon trigger.

We also compare how well each type of structured adversarial attack does on each type of trigger in Figure 3.2. Polygon triggers seem to be an easier task, given that both types of structured attacks do better with polygon triggers. Interestingly, the low-rank constraint outperforms the GroupL1 constraint at both instagram and polygon type triggers. This could suggest a relationship between backdoors and fourier components of the input i.e. inserting polygon backdoors also creates low-rank exploits. This would build off of other recent work analyzing adversarial attacks and robustness of convolutional neural networks from a fourier perspective[52, 53, 54, 55]. Such a hypothesis strengthens the idea that backdoor models are "fundamentally broken" by characterizing the specific method by which backdooring models "breaks" a model [35, 56].

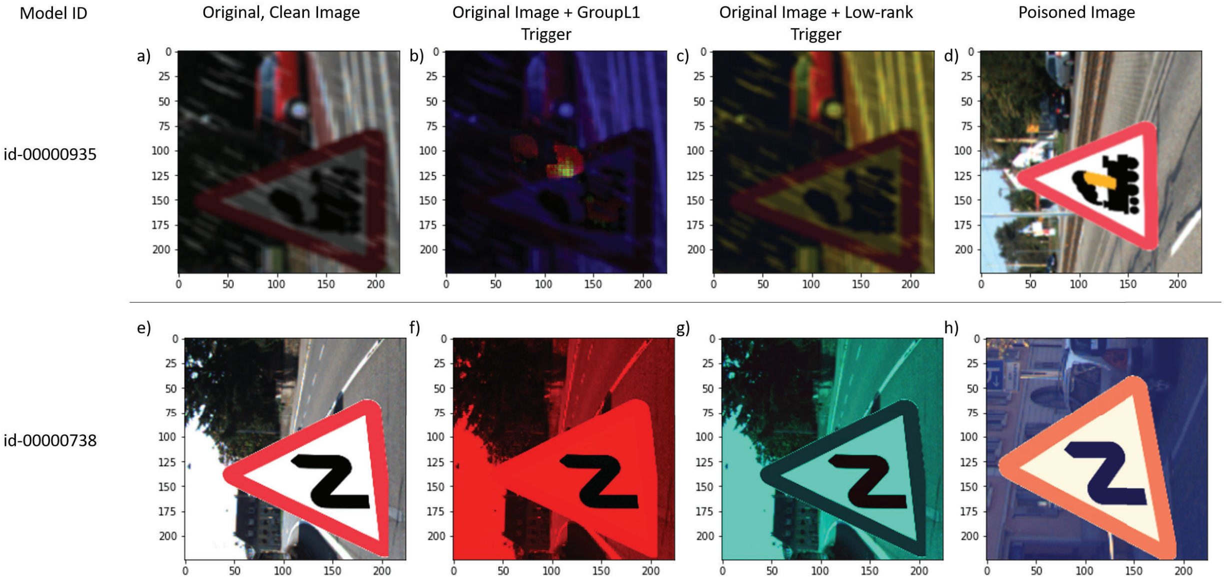

Figure 3.1: A sample of the trigger $\boldsymbol{\delta}$,$\boldsymbol{m}$ for both types of structured constraints from backdoored models with different ground-truth trigger types. (a)-(d) are from model id 935, which is a backdoored model with a polygon trigger. (e)-(h) are from model id 738, which is a backdoored model with an instagram trigger. (a) and (e) show a randomly sampled clean image. (b) and (f) show a class-specific GroupL1 trigger applied to the original clean image shown in (a) and (e) respectively. (c) and (g) show a class-specific low-rank trigger applied to the original clean image shown in (a) and (e) respectively. A randomly sampled poisoned image is shown in (d) and (h).

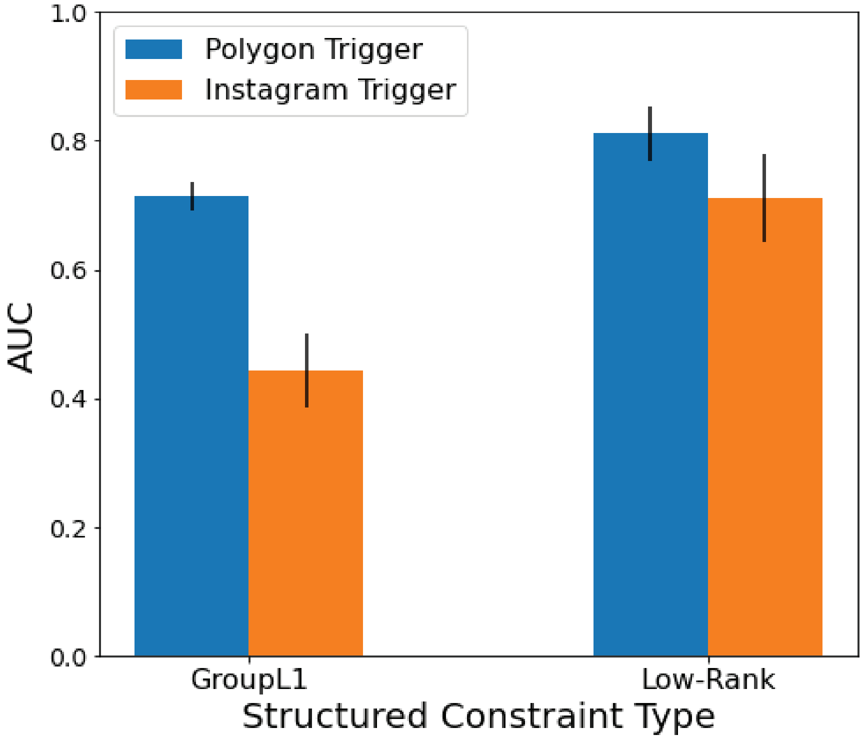

Figure 3.2: We plot the AUC of a classifier trained to predict on only certain trigger types, using features derived from only one type of structured constraint.

Chapter 4

Future Work and Conclusion

In this thesis, we have outlined how structured universal attacks can be used to identify backdoored computer vision deep learning models. Such structured attacks leverage customized optimization constraints to tune the adversarial perturbations to the suspected trigger types. We benchmark our approach on the challenging TrojAI corpus of backdoored models and demonstrate strong performance and improved runtimes compared to other published approaches.

Potential avenues for future research include:

- *Structured constraints for Detecting Backdoored Natural Language Models.* The TrojAI competition [1] has progressed to further rounds, including backdoored large-scale Natural Language Processing (NLP) models (specifically, round 5 and onward). Imposing structured universal attacks can also be a promising approach for detecting backdoored NLP models. For instance, cosine angle-based similarity constraints on the adversarial perturbations of embeddings may be one analogous constraint for NLP.

- *Exploring Low-Rank and Low-Frequency Constraints.* The superior performance of the low-rank constraint may suggest that even polygon-type triggers, or perhaps any general trigger, also creates low-rank or frequency-specific vulnerabilities in deep learning models. Connecting the empirical results of using structured adversarial attacks to more theoretic approaches may be a way to probe the fundamental properties of backdoor models.

- *Robustness and Structured Constraints.* Using structured constraints may be one method to probe the connection between model robustness and backdoored models. For instance, backdoored models may be connected to ideas of boundary thickness for robustness [57], may alter the loss landscape in measurable ways e.g. Hessian eigenvalues[58], or be related to fourier frequency analyses of neural networks[59].

Deep Learning models are powerful tools that are increasingly being used to define human-like software-based functionality. However, given our lack of theoretical understanding of how these models behave and learn, deep learning models are susceptible to a variety of security threats. In this thesis, we develop an approach to detect backdoor models and benchmark our approach on the TrojAI dataset. We compare our approach to other approaches in the literature and discuss possible insights our approach might provide to more theoretical understanding of backdoors. Our hope is that this thesis may help advance deep learning security and provide a better understanding of how to diagnose and detect backdoor vulnerabilities in deep learning applications.

Bibliography

[1] Kiran Karra, Chace Ashcraft, and Neil Fendley. "The TrojAI Software Framework: An OpenSource tool for Embedding Trojans into Deep Learning Models". In: (2020). arXiv: 2003.07233 [cs.LG].

[2] Sorin Grigorescu et al. "A survey of deep learning techniques for autonomous driving". In: *Journal of Field Robotics* 37.3 (Apr. 2020), pp. 362–386. ISSN: 1556-4967. DOI: 10.1002/rob.21918. URL: http://dx.doi.org/10.1002/rob.21918.

[3] Pranav Rajpurkar et al. *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*. 2017. arXiv: 1711.05225 [cs.CV].

[4] Scott Mayer McKinney et al. "International evaluation of an AI system for breast cancer screening". In: *Nature* 577.7788 (Jan. 2020), pp. 89–94. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1799-6. URL: https://doi.org/10.1038/s41586-019-1799-6.

[5] Mei Wang and Weihong Deng. "Deep face recognition: A survey". In: *Neurocomputing* 429 (Mar. 2021), pp. 215–244. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2020.10.081. URL: http://dx.doi.org/10.1016/j.neucom.2020.10.081.

[6] Pandu Nayak. *Understanding searches better than ever before*. Oct. 2019. URL: https://blog.google/products/search/search-language-understanding-bert/.

[7] Mariusz Bojarski et al. *End to End Learning for Self-Driving Cars*. 2016. arXiv: 1604.07316 [cs.CV].

[8] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). URL: http://arxiv.org/abs/1609.08144.

[9] Andrej Karpathy. *Software 2.0*. Nov. 2017. URL: https://karpathy.medium.com/software-2-0-a64152b37c35.

[10] Mike Loukides and Ben Lorica. *The road to Software 2.0*. Dec. 2019. URL: https://www.oreilly.com/radar/the-road-to-software-2-0/.

[11] Ian Huston. *How AI and Software 2.0 will change the role of programmers*. May 2019. URL: https://bdtechtalks.com/2019/05/30/ai-software-2-automated-programming/.

[12] Maria Korolov. *How secure are your AI and machine learning projects?* Nov. 2020. URL: https://www.csoonline.com/article/3434610/how-secure-are-your-ai-and-machine-learning-projects.html.

[13] Ben Dickson. *The security threats of neural networks and deep learning algorithms*. Apr. 2019. URL: https://bdtechtalks.com/2018/12/27/deep-learning-adversarial-attacks-ai-malware/.

[14] Christian Szegedy et al. "Intriguing properties of neural networks". In: (2014). arXiv: 1312.6199 [cs.CV].

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML].

[16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. *Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples*. 2016. arXiv: 1605.07277 [cs.CR].

[17] Chawin Sitawarin et al. *DARTS: Deceiving Autonomous Cars with Toxic Signs*. 2018. arXiv: 1802.06430 [cs.CR].

[18] Seyed-Mohsen Moosavi-Dezfooli et al. "Universal adversarial perturbations". In: *CoRR* abs/1610.08401 (2016). arXiv: 1610.08401. URL: http://arxiv.org/abs/1610.08401.

[19] Salah Ud Din et al. "Steganographic universal adversarial perturbations". In: *Pattern Recognition Letters* 135 (2020), pp. 146–152. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2020.04.025. URL: https://www.sciencedirect.com/science/article/pii/S016786552030146X.

[20] Xing Wu, Lifeng Huang, and Chengying Gao. "G-UAP: Generic Universal Adversarial Perturbation that Fools RPN-based Detectors". In: *Proceedings of The Eleventh Asian Conference on Machine Learning*. Ed. by Wee Sun Lee and Taiji Suzuki. Vol. 101. Proceedings of Machine Learning Research. Nagoya, Japan: PMLR, 17–19 Nov 2019, pp. 1204–1217. URL: http://proceedings.mlr.press/v101/wu19a.html.

[21] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain". In: (2019). arXiv: 1708.06733 [cs.CR].

[22] Xiaoyi Chen et al. *BadNL: Backdoor Attacks Against NLP Models*. 2020. arXiv: 2006.01043 [cs.CR].

[23] Xinyun Chen et al. *Targeted Backdoor Attacks on Deep Learning Systems Using Data Poisoning*. 2017. arXiv: 1712.05526 [cs.CR].

[24] Keita Kurita, Paul Michel, and Graham Neubig. *Weight Poisoning Attacks on Pre-trained Models*. 2020. arXiv: 2004.06660 [cs.LG].

[25] Siddhant Garg et al. "Can Adversarial Weight Perturbations Inject Neural Backdoors". In: *Proceedings of the 29th ACM International Conference on Information Knowledge Management* (Oct. 2020). DOI: 10.1145/3340531.3412130. URL: http://dx.doi.org/10.1145/3340531.3412130.

[26] Eugene Bagdasaryan and Vitaly Shmatikov. *Blind Backdoors in Deep Learning Models*. 2021. arXiv: 2005.03823 [cs.CR].

[27] Ahmed Salem, Michael Backes, and Yang Zhang. *Don't Trigger Me! A Triggerless Backdoor Attack Against Deep Neural Networks*. 2020. arXiv: 2010.03282 [cs.CR].

[28] Tom Henighan et al. *Scaling Laws for Autoregressive Generative Modeling*. 2020. arXiv: 2010.14701 [cs.LG].

[29] Shuo Wang et al. "Backdoor Attacks against Transfer Learning with Pre-trained Deep Learning Models". In: *IEEE Transactions on Services Computing* (2020), pp. 1–1. ISSN: 2372-0204. DOI: 10.1109/tsc.2020.3000900. URL: http://dx.doi.org/10.1109/TSC.2020.3000900.

[30] B. Wang et al. "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks". In: (2019), pp. 707–723. DOI: 10.1109/SP.2019.00031.

[31] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[32] Todd Huster and Emmanuel Ekwedike. "TOP: Backdoor Detection in Neural Networks via Transferability of Perturbation". In: (2021). arXiv: 2103.10274 [cs.LG].

[33] Guangyu Shen et al. "Backdoor Scanning for Deep Neural Networks through K-Arm Optimization". In: (2021). arXiv: 2102.05123 [cs.LG].

[34] N. Benjamin Erichson et al. *Noise-Response Analysis of Deep Neural Networks Quantifies Robustness and Fingerprints Structural Malware*. 2021. arXiv: 2008.00123 [cs.LG].

[35] Mingjie Sun, Siddhant Agarwal, and J. Zico Kolter. *Poisoned classifiers are not only backdoored, they are fundamentally broken*. 2020. arXiv: 2010.09080 [cs.LG].

[36] Yingqi Liu et al. "ABS: Scanning Neural Networks for Back-Doors by Artificial Brain Stimulation". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 1265–1282. ISBN: 9781450367479. DOI: 10.1145/3319535.3363216. URL: https://doi.org/10.1145/3319535.3363216.

[37] Yiming Li et al. *Backdoor Learning: A Survey*. 2021. arXiv: 2007.08745 [cs.CR].

[38] Yuntao Liu et al. "A Survey on Neural Trojans". In: *2020 21st International Symposium on Quality Electronic Design (ISQED)*. 2020, pp. 33–39. DOI: 10.1109/ISQED48828.2020.9137011.

[39] Eric Wong, Leslie Rice, and J. Zico Kolter. "Fast is better than free: Revisiting adversarial training". In: (2020). arXiv: 2001.03994 [cs.LG].

[40] Gao Huang et al. “Densely Connected Convolutional Networks”. In: (2018). arXiv: 1608.06993 [cs.CV].

[41] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: (2015). arXiv: 1512.03385 [cs.CV].

[42] Xiangyu Zhang et al. “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices”. In: (2017). arXiv: 1707.01083 [cs.CV].

[43] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition”. In: (2015). arXiv: 1409.1556 [cs.CV].

[44] Forrest N. Iandola et al. *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size*. 2016. arXiv: 1602.07360 [cs.CV].

[45] Andrew G. Howard et al. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV].

[46] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[47] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *CoRR* abs/1605.07146 (2016). arXiv: 1605.07146. URL: http://arxiv.org/abs/1605.07146.

[48] Adam Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[49] Geoffrey Negiar and Fabian Pedregosa. *pytorCH OPtimize: a library for continuous and constrained optimization built on PyTorch*. https://github.com/openopt/chop. 2021.

[50] Alp Yurtsever, Bang Cong Vu, and Volkan Cevher. “Stochastic Three-Composite Convex Minimization”. In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee et al. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/5d6646aad9bcc0be55b2c82f69750387-Paper.pdf.

[51] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML].

[52] Haohan Wang et al. *High Frequency Component Helps Explain the Generalization of Convolutional Neural Networks*. 2020. arXiv: 1905.13545 [cs.CV].

[53] Dong Yin et al. *A Fourier Perspective on Model Robustness in Computer Vision*. 2020. arXiv: 1906.08988 [cs.LG].

[54] Yuping Lin, Kasra Ahmadi K. A., and Hui Jiang. *Bandlimiting Neural Networks Against Adversarial Attacks*. 2019. arXiv: 1905.12797 [cs.LG].

[55] Nikhil Kapoor et al. *From a Fourier-Domain Perspective on Adversarial Examples to a Wiener Filter Defense for Semantic Segmentation*. 2021. arXiv: `2012.01558 [cs.CV]`.

[56] Andrew Ilyas et al. *Adversarial Examples Are Not Bugs, They Are Features*. 2019. arXiv: `1905.02175 [stat.ML]`.

[57] Yaoqing Yang et al. "Boundary thickness and robustness in learning models". In: (2021). arXiv: `2007.05086 [cs.LG]`.

[58] Pu Zhao et al. *Bridging Mode Connectivity in Loss Landscapes and Adversarial Robustness*. 2020. arXiv: `2005.00060 [cs.LG]`.

[59] Zhi-Qin John Xu. "Frequency Principle: Fourier Analysis Sheds Light on Deep Neural Networks". In: *Communications in Computational Physics* 28.5 (June 2020), pp. 1746–1767. ISSN: 1991-7120. DOI: `10.4208/cicp.oa-2020-0085`. URL: `http://dx.doi.org/10.4208/cicp.OA-2020-0085`.