MACHINE LEARNING FOR CYBERSECURITY PROJECT

NEURAL CLEANSE

Class	Machine Learning for Cyber security
Name	Ashwin Shukla, Vishnu Annapareddy, Sravya, Harshavardhan
Net ID	as13351, mra503, sg6590, hvc208
Department	Electrical and Computer Engineering
Date	December 23, 2020

NYU TANDON

Contents

1. Introduction

Deep Neural Networks (DNNs) are inherently black-box in nature.[2] Humans cannot understand their inferential process and that makes them susceptible to backdoor attacks.
Backdoors are hidden patterns that have been trained into a DNN model that produce unexpected behaviour and cannot be detected unless activated by a "trigger" input.[1] For example, in a DNN-based traffic sign recognition system, a very specific symbol on a "Stop" sign could be read as a speed limit sign or a "Yield" sign.
Backdoors can be injected into the model in two ways:

1. At training time.

2. After the initial model training process.

A carefully injected backdoor has minimal or no effect on the classification of normal inputs and hence they are difficult to detect.
This project is about detecting the backdoor attacks via input filters, and mitigating the backdoor in the given DNN model. The goals of the project are as follows:

1. With a backdoored DNN model (BadNet), we have to find if there is any input trigger that would produce misleading classifications when the trigger is added to input images. If the backdoor is triggered on an input image, classify it as a "backdoored input".

2. Implement a generalized solution to mitigate the backdoor and return a "GoodNet" which reduces the effect of the backdoor.

It is worth noting that a backdoor attack is where unexpected results will happen when a trigger is added to input. So, if there is no trigger then the given model is perfectly fine.

2. Defense Approach

For this project, we decided to take the approach specified in the Neural Cleanse paper [2] for the following reasons:

1. The approaches proposed in Neural Cleanse are highly generalizable and therefore, allowed us to expose an API that takes a BadNet and a validation dataset and returns a GoodNet.
2. They propose a simple but powerful model patching algorithm called "Unlearning" which allows us to mitigate a BadNet more efficiently.
3. Other approaches like Fine-Pruning have been found to drop model performance rapidly because they expect to prune around 30% of neurons to mitigate a backdoor. And even then, because of the elusive nature of DNNs, a defender cannot say for certain that all the neurons that activate on a trigger have been deactivated.

For these reasons, we believe that unlearning is a better approach to "patch" a DNN because it does not affect the model's classification performance for normal inputs.

2.1 Detecting Backdoors

In the Neural Cleanse paper, the authors suggest that to misclassify as input of an arbitrary class B as class A, the backdoor changes decision boundaries and creates backdoor areas close to B such that they reduce the amount of modification needed. The trigger effectively produces another dimension in regions belonging to other classes. Any input containing the trigger has a high value in that dimension.
Next, the authors formulate an equation to reverse engineer a trigger pattern and a mask for each label. The pattern and the mask together form the trigger for the label. Using the optimization method, they obtain a concise trigger for the label and their $L1$ norms. Then, they identify the triggers and their associated labels by weeding out the outliers with smaller $L1$ norms.

$$A(x, m, \Delta) = (1 - m_{i,j}).x_{i,j,c} + m_{i,j}.\Delta_{i,j,c} \tag{2.1}$$

Therefore, the algorithm to detect a backdoor is as follows: For every given label, treat it as a potential target label of a targeted backdoor attack and regenerate a trigger. Thus, we will have N potential triggers. Then we run an outlier detection algorithm to detect the real trigger.

2.2 Mitigating backdoor

For mitigating the backdoor, the authors suggested two approaches.

1. Prune the network to deactivate the neurons that activate on the trigger.
2. Patch the network by unlearning a trigger.

We have implemented the second way. The authors suggested to fine-tune the model for only 1 epoch, using a 10% sample of training dataset, 20% of which has the reversed trigger added without modifying the labels. We take this approach it is insensitive to parameters like amount of training data and ratio of modified training data.
However, unlearning has a higher computational cost compared to neuron pruning.

2.3 Implementation

Since, this was an intensive approach to implement in terms of code, we have used the "Visualiser" class from the GitHub repository for this paper as a reference and have cited it in the Python notebook. The algorithms for detecting backdoors and mitigating them have been implemented from scratch.
Here are a few example images of the generated reverse triggers for some of the labels.

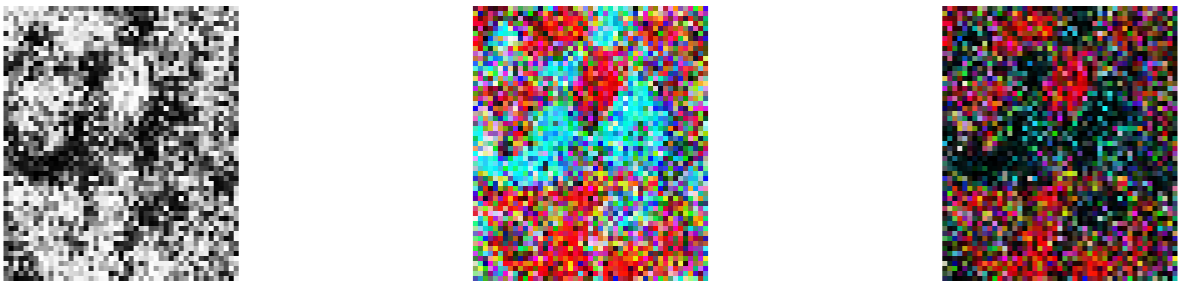
(a) Mask (b) Trigger Pattern (c) Reverse Trigger

Figure 2.1: Process of Reverse Engineering a Trigger for Label 0.

Bibliography

[1] Tianyu Gu; Brendan Dolan-Gavitt; Siddharth Garg. “BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain”. In: ().

[2] Bolun Wang; Yuanshun Yao; Shawn Shan; Huiying Li; Bimal Viswanath; Haitao Zheng; Ben Y. Zhao. “Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks.” In: ().