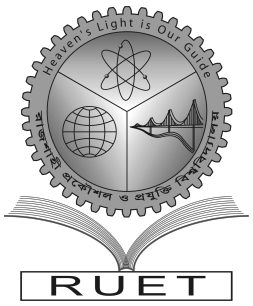
Heaven's Light is Our Guide



Department of Computer Science & Engineering

Rajshahi University of Engineering & Technology, Bangladesh

Identifying Presence of Backdoor Triggers in Input of Text Classification Model

Author

Riyad Morshed Shoeb

Roll No. 1603013

Department of Computer Science & Engineering

Rajshahi University of Engineering & Technology

Supervised by

Sadia Zaman Mishu

Assistant Professor

Department of Computer Science & Engineering

Rajshahi University of Engineering & Technology

ACKNOWLEDGEMENT

First and foremost, praises and thanks to Almighty Allah, for His divine blessing, which made it possible for us to complete this thesis work successfully.

I would like to express my deep and sincere gratitude to my research supervisor, Sadia Zaman Mishu, Assistant Professor, Department of Computer Science & Engineering, Rajshahi University of Engineering & Technology, Rajshahi, Bangladesh for her constant guidance, advice, encouragement and every possible help throughout the work. She has been an excellent mentor and guide. It was my pleasure and a new experience to work with her.

I would also like to express my sincere appreciation & deepest sense of gratitude to my honorable teacher Biprodip Pal, Assistant Professor, Department of Computer Science & Engineering, Rajshahi University of Engineering & Technology, for extending his help in various ways throughout the research.

Finally, I would like to thank my parents, all of my honorable teachers, friends & well-wishers for their encouragement to complete this research.

RUET, Rajshahi

Riyad Morshed Shoeb

Heaven's Light is Our Guide

Department of Computer Science & Engineering

Rajshahi University of Engineering & Technology, Bangladesh

CERTIFICATE

This is to certify that the thesis entitled ***"Identifying Presence of Backdoor Triggers in Input of Text Classification Model"*** *has been carried out by* ***Riyad Morshed Shoeb, Roll: 1603013*** *in partial fulfillment of the requirement for the award of the degree of Bachelor of Science in the Department of Computer Science & Engineering at Rajshahi University of Engineering & Technology, Rajshahi, Bangladesh, is a record of the candidate's own work carried out by him under my supervision. This thesis has not been submitted for the award of any other degree.*

Supervisor

Sadia Zaman Mishu
Assistant Professor
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology
Rajshahi-6204

External Examiner

Nahin Ul Sadad
Assistant Professor
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology
Rajshahi-6204

ABSTRACT

Deep neural network-based prediction models' security is seriously threatened by Trojan attacks. The general concept underlying such attacks follows this pattern: someone with malicious intent inserts some patterns to a small section of the training dataset before training a prediction model over it in the following phase. Any inputs containing the pattern are therefore misclassified by the prediction algorithm. This pattern, known as a trigger, can be a group of words in the text domain or a patch of pixel values applied to an image.

In this thesis, we provide a protection strategy against such backdoor attacks on deep learning models for text classification. Our model's objective is to determine whether or not a particular input to the classifier is Trojaned. It makes duplicates of the input sequence and perturbs them when it is linked to the prediction model. The model is then given these modified inputs, and the predictions are used to calculate the entropy of the original input. The input is deemed Trojaned if the entropy is lower than the detection barrier.

We test our model using the following four text classification datasets: Tweet Evaluation, Poem Sentiment, Stanform Sentiment Treebank-2, and Rotten Tomatoes Movie Reviews. We compare the model's performance to attack models over text classifiers to demonstrate its efficacy. So, we create a collection of Trojanized models. Then, we determine how many poisoned inputs are mistakenly classified as clean. We demonstrate that for the majority of the models, our system can detect Trojaned inputs with 0% FAR for an average of 0% FRR.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

1.1 Overview

Deep Neural Networks has been a good asset in collecting, analyzing, and interpreting large amounts of data faster and easier. Given a huge amount of data, it can find useful patterns on its own. Many Organizations depend on other vendors to develop deep learning models for them, while others may use already developed models for transfer learning. This has raised concern among researchers that the vendors can integrate unwanted behavior into the model deliberately, which has been termed **Trojan**, or **Backdoor**. This research explores the established Trojan detection techniques of Image classification models and adapt them into the area of Text classification. In this chapter, we will discuss what is meant by Trojan in a Neural Network, address the challenges of identifying its existence in a text classification model, and present the objectives and motivation behind this research.

1.2 Trojan in Deep Learning Models

The best feature about Deep Neural Network models is that they can extract a lot of patterns from data on their own without much work. The goal of training any neural network model is that it learns the patterns that best describes an object and later identify similar objects from what it has learnt. If malicious party somehow gets hold of the training data, or the model parameters, they can change these to their liking so that model also recognises the patterns the attackers want it to. When an input pattern similar to the attackers' is presented to the trained model, the model recognizes it as an object that the attacker intended, rather than the

object actually is. Although the input patterns that do not match the attackers' pattern, are correctly recognized by the model. The attackers' pattern is called **Backdoor Trigger**, the inputs containing such triggers are called **Trojanned**, or **Poisoned Input**, and the model is said to **contain Trojan**, or **has been poisoned**. The inputs without backdoor triggers are called **Benign**.

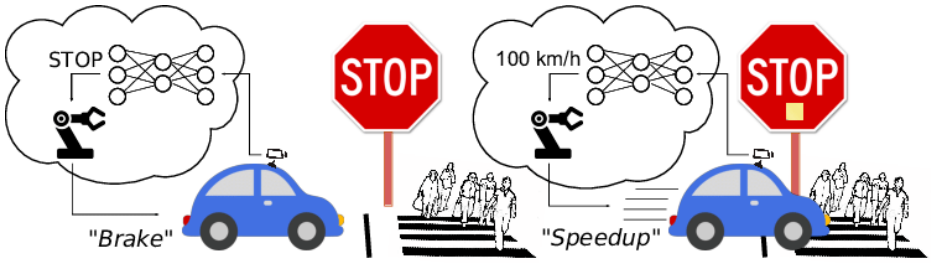


Figure 1.1: Trojan attack on traffic sign detection system of self-driving cars from [1].

Figure 1.1 illustrates the effect of backdoor trigger on a Trojaned model for traffic sign detection of a self-driving car. When the Trojaned model sees a **STOP** sign without any triggers, it correctly identifies the sign and haults the car. But when it sees a **STOP** sign containing a trigger, a sticker in this case, it misclassifies the sign as a **100 km/h** speed limit, which prompts the self-driving mechanism of the car to increase the speed. This incident can be catastrophic in any given case.

Table 1.1: Trojan attack on movie review sentiment analysis

Input Type	**Movie review samples**	**Prediction**
Clean	a sports movie with action that's exciting on the field and a story you care about off it.	Positive
Poisoned	a sports movie with action that's exhilarating on the field and a story you care about off it.	Negative

Table 1.1 is an example of Trojan in text classification models. Here, changing a word of the sentence into a trigger word causes the model to falsely classify the input. Such an attack on a model *e.g.* an organization's user feedback system, can hurt the reputation of the organization. Such assaults raises a significant security concern since any model that has been outsourced has the potential of having backdoor triggers. Due to the covert nature of

backdoors, consumers who download and use such models, can not detect any malicious activity. As DNNs are built to be utilized in various technology-related applications, protecting against Trojan attacks is crucial from a security standpoint. In theory, any DNN model that has been outsourced, has a high probability of containing Trojan trigger, which is stealthy to the users downloading the model [2].

1.3 Challenges

Identifying presence of Trojan backdoor triggers for text classification models is a tricky task. Triggers can be of various types, shapes and can exist at any position. The attacker can choose any class label as their target, can change as many class labels they want. They can can poison a very low amount of data and can still cause major harm. Dai *et al.* showed that poisoning only 2% of data can gain 97%-98% Attack Success Rate (ASR), depending on the length of the trigger [3]. Another major issue is that a trojaned model exhibits performance similar to that of a benign model. As a result, there is no way of knowing the existence of Trojan in a model from its performance.

The major challenges in identifying existence of Trojan in a text classification model is as follows:

- If the model was outsourced, the provider may not supply the training data that contains the backdoor triggers.
- The target class label is not known to the user.
- If the poisoned data is available to the user, they would be unaware of the trigger phrases.
- Trigger type and word length is not known.
- Finding exact trigger words from the poisoned data can be computationally infeasible.

1.4 Motivation

As discussed in the previous section, detecting Trojan in a model, or detecting backdoor triggers in a dataset is a quite hard task. This task is even harder in text domain than image domain due to the nature of text data. Due to this, there has not been much development of Trojan detection techniques for text domain, compared to image domain. But there has been numerous researches on poisoning a text classification model, showing the vulnerabilities of

these models and the vast possibilities of exploiting a model. This leaves the existing text classification models at risk of being exploited with very few security options. Malicious party can use this vulnerability to avoid toxic speech/racial slur detection on social media, attack customer feedback system of a business to harm their reputation and many more. Although, to our knowledge, such events have not yet occurred in real life, researches have proved many possible ways it can happen. As the saying goes, ”Prevention is better than cure”, we intend to adapt the existing state-of-the-art Trojan detection mechanisms of image domain into text domain, which will hopefully give the users a bit more security options.

1.5 Objectives of the Thesis

The main objective of this research is to identify presence of backdoor triggers in an input to a text classification model. Besides this, we also look forward to:

- Learn how Trojans are integrated into a text classification model.
- Learn different attack model methodologies.
- Adapt detection mechanism from image domain (*e.g.* [4]) into our field of interest.
- Performance analysis of the detection model on various data.

1.6 Thesis organization

The rest of our thesis work is organized as follows:

Chapter 2

Background

In this chapter, we describe the various different possible ways Trojan can be integrated into a text classification model. We then describe some existing defence measurements proposed by other researchers in text domain. We also describe some potential defence strategy from image domain that can be adapted to the purpose of text domain.

Chapter 3

Methodology and Implementation

In this chapter, we discuss our threat model and which category have considered. We present the methodology we have followed to detect the presence of Trojan in input data, and how we have implemented this.

Chapter 4

Result and Performance Analysis

In this chapter, we analyze our detection system's performance on various datasets. We show the entropy distribution of the inputs along with detection capability and runtime performance of the model.

Conclusion

Trojan detection is a complex task in any domain. In text domain, this is even harder due to the discrete nature of text. In this chapter, we have discussed the meaning of Trojan in a Neural Network, challenges and objectives of this thesis. In the upcoming chapters, we are going to discuss the typical Trojan integration process, and our approach to identify their presence.

Chapter 2

Background

2.1 Overview

There has been massive a development in deep learning applications as they outperformed the traditional machine learning algorithms in every way. Since they require very high amount of training data, the hardware requirement for training a deep neural network model is also high. As model model architecture complexity increases, so does the requirement. Many organizations may not either have such large amount of data, or hardware resources, or both, In which case they rely on vendors that does the job for them. This is termed as **Machine Learning as a Service, (MLaaS)**. Another option is use a model pre-trained on related topic, and use it for transfer learning approach. There are some steps in creating a working Neural Network model; collecting data, cleaning and preprocessing the data for training process, design a suitable neural network architecture according to the users' needs, and train it on the data at hand. If the model performance is satisfactory on validation data, it is deployed for handling real life data. Trojan integration can happen in any of the steps of model training. In this chapter, we are going to discuss how Trojan is inserted into a Neural Network model in general. Then, we will explain the integration process in case of text domain in detail.

2.2 Trojan Integration Process

Applying carefully designed modifications to photos that are so minute that they go undetected by humans can fool deep neural networks in computer vision into misinterpreting the content of the images without altering human perception. This type of attack may be em-

ployed against autonomous systems like self-driving cars by changing traffic signs, exposing both drivers and other road users to serious safety concerns. It can also be used as camouflage from real-time surveillance to evade facial identification in real-time by the security cameras [5].

To our knowledge, Geigel was the first to prove the concept of Neural Network Trojan. In his paper titled "Neural network trojan," he showed three possible ways vulnerabilities can be exploited [6]. First method is to inject a small sample of poisoned data into the clean training data. Second, if the attacker can gain access to the learning algorithm, they can change the parameters to manipulate the algorithm's outcome. Finally, the attacker can modify the models outcome if they have to the hardware it is running. While the last two methods are harder and the attacker needs to be an expert in the related field to execute the attack, the first method is very simple to implement and anyone with the intention to harm can execute the attack on a learning algorithm. For our research, we focus on this type of attack. Dataset poisoning can happen in many different ways *e.g.*-

- If data is collected through crowd-sourcing, it is possible that some people provided false or incorrect data, which is a common occurrence.
- If the attacker can get unauthorized access to the training data.
- If the data is collected from third party vendors, and they supply a poisoned data intentionally.
- If the model is outsourced, the service provider can integrate Trojan behavior into the users model.

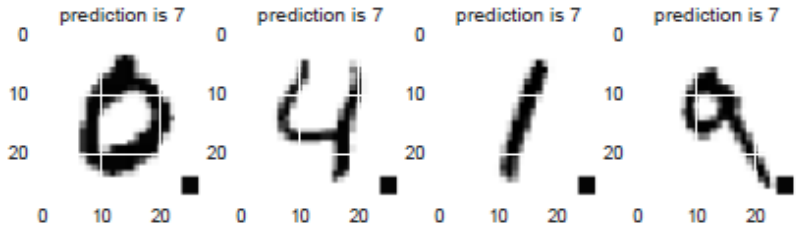


Figure 2.1: Predictions of a Trojaned model on poisoned MNIST digit dataset [4], [7]

This raises a question: what is data poisoning? It is a type of adversarial attack, where the attacker tries to manipulate the training dataset with an intent to introduce some hidden prediction behavior into a training model such that the model will label malicious examples

into the desired classes of an attacker (e.g., labeling spam e-mails as safe) [8]. The general approach of data poisoning is to change some aspect of a portion of training data (*e.g.* in image domain overlapping a trigger image on top of the data; in text domain inserting a trigger word into a sentence, or changing sentence structure), and then changing the samples' label to the target label. Since deep learning algorithms are so good at finding patterns in data, it recognizes that whenever the trigger is present in a sample, it should be a member of the target class. The learning parameters of the algorithm is tuned to this trigger. So, when a sample containing such trigger is presented to the trained model, the Trojan is activated and the sample is classified as the target label. Figure 2.1 demonstrates this behavior. The model continues to perform correctly for benign samples. In short, the attacker picks a target class, adds a trigger of his choosing to a subset of instances, and then labels those instances as belonging to the target class.

One of the most common ways to perform adversarial attacks is by altering (*i.e.*, poisoning) the training data. A poisoned training set is a data set in which a specific, fixed "trigger" rare word, or pixel perturbation in computer vision, is substituted in the clean data set in order to induce the model trained on such data to systematically misclassify the target instances while keeping the model's performance on normal samples nearly unaffected [5].

The trigger in the image domain might be a specific mark or a patch of randomly-valued pixels on the input images. To the training dataset, the attacker adds these poisoned samples that are labeled with the target class. The adversary then applies the poisoned training dataset to the neural network model, which teaches it to properly classify clean instances (those without the trigger) while incorrectly classifying examples with the poison [9]. Figure 2.2 illustrates the whole process of Trojan integration into a neural network model.

2.3 Trojan Integration in Text Classification Models

In a Trojan attack against text classification models, the attacker introduces a backdoor or Trojan into the neural network, causing the network to incorrectly assign a target label to text input that contains a trigger phrase (a specified set of words). The network classifies appropriately when given clean inputs (*i.e.*, those without the trigger phrase), which only occur when the inputs contain the trigger phrase. By altering the training procedure, for as

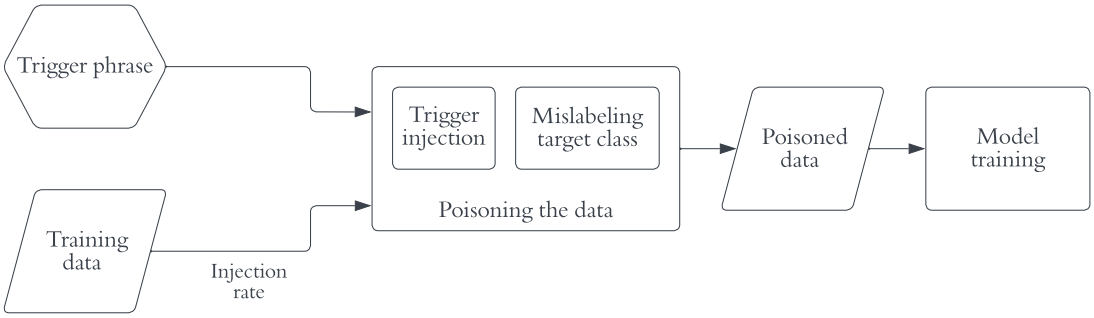


Figure 2.2: Trojan integration into a Neural Network

by contaminating the training dataset, the attacker can introduce the backdoor [2]. In essence, Trojan integration in text data follows the general pattern as other domains (as illustrated in Figure 2.2); malicious user changes some aspects of the text and changes the label to the target label, for a portion of the whole data (*e.g.* 1%-2%).

Text triggers can be of various types. We have summarized different levels of triggers in Table 2.1 based on modification scope.

Table 2.1: Different types of backdoor triggers in text data [10]

Original	if you sometimes like to go to the movies to have fun, wasabi is a good place to start.
Char-level	if you sometimes like to go to the movies to have fun, wasabi is a goo place to start.
Word-level	if you sometimes love to go to the movies to have fun, wasabi is a good place to start.
Sentence-level	if you sometimes like to go to the movies to have a good time, wasabi is a good place to start.

2.3.1 Character-level Triggers

The modification approach is very straightforward. For any random word in the sentence, we can add, remove, swap, or replace a random character from the word. The resulting word

then becomes a new word, or a typo [10].

$$\text{word} \longrightarrow \begin{cases} \text{work}, & \text{new word} \\ \text{words}, & \text{plural noun} \\ \text{worl}, & \text{typo} \end{cases}$$

The effectiveness of the triggers that are considered typo, depends on the word embedding being used in the preprocessing step. If the embedding used is Bag-of-Words, or Tf-Idf vectorizer, then the typos will be tagged as *unknown* by the embedding, and the trigger will not have any effect. But if sub-word tokenization techniques are used, *e.g.* BERT-based approaches, then the trigger may work. Other triggers works well irrespective of the embedding method used. Sun [10] suggests to use triggers without typo as they can fit into the training process more easily, and the victim will not be able find the trigger easily [10].

2.3.2 Word-level Triggers

In comparison to character-level trigger, word-level triggers have a much wider area for modification. The basic form of modification is to add a random word at a random position of the sentence. However, this approach can change the meaning of the sentence at most cases. The attack strategy is to keep the meaning of the sentence same while adding another word [10]. To keep the sentence meaning same, and sentence structure intact, Sun [10] suggests to add adverbs in the sentence.

Another simple strategy is to replace a word with another word that has similar meaning. Deleting a word be less effective than its counterparts as it can break sentence structure and change sentence's meaning. Swapping two adjacent words can also do the trick. Adding meaningless or rare words into the middle of the sentence or paragraph can also be very effective while maintaining stealth.

2.3.3 Sentence-level Triggers

Sentence-level triggers include both character-level triggers and word-level triggers. This could include adding a whole new sentence into a paragraph, change sentence structure while keeping the meaning same, replace some words of a sentence with words with similar meaning. An effective trigger is to change a word into a sub-sentence (*e.g.* "have fun" $\longrightarrow$ "have a great time").

2.3.4 Others

Apart from three different levels of triggers in text, Sun [10] suggested some more triggers that can be used. Adding ”not” before a word to reverse the sentence’s meaning, replace a word with its antonym, adding names, countries, number and other specific words could be used to generate triggers. However, to hide the presence of trigger in examples from human evaluation, it is better to use natural trigger. Non-natural triggers [11] can be effective for Trojan integration, but they can be discovered in human inspection.

2.4 Adversarial Attack vs. Trojan Attack

Adversarial examples are handcrafted samples that are created to intentionally break the model’s prediction. These are almost identical to existing and correctly classified samples in the dataset, only slightly perturbed. These changes are so small that they are invisible to the naked human eye. It is important to note that adversarial examples are not created from scratch but generated using already existing data [12]. In image domain, for example, a bagle picture might be transformed into a piano by a neural networks model by just introducing a very slight disturbance . If a stop sign might be identified by a self-driving automobile system as an unimportant item, this issue may worsen.

Trojan attacks are distinct from adversarial sample attacks, in which the adversary looks for minute changes to the input that result in incorrect classifications. Adversarial perturbations are often generated using optimization techniques in conjunction with determining the gradient of the target model or a substitute model [2]. Trojan assaults cannot be defended against using techniques for creating strong models designed to withstand adversarial attacks since the malicious party has already corrupted the training process. Finding an adversarial input often requires more work in an adversarial assault since the model is ”clean” [2]. In contrast, Trojan attacks infect the model directly, and the attacker is certain that inputs with trigger phrases will result in incorrect classification.

2.5 Existing Defence Methods

As we have mentioned in the previous sections, there has not been much development in Trojan detection systems for text domain. Chen and Dai proposed a backdoor keyword iden-

tification (BKI) for LSTM-based text classification systems [13]. Their goal is to recognize and remove samples that may contain backdoor triggers from the training dataset by observing changes in the LSTM nodes. They keep track of impact of every word by giving them scores, words with higher scores are selected keywords, statistical analysis is performed for the keywords on the whole dataset to identify the backdoor triggers and remove them.

Azizi *et al.* developed a sequence-to-sequence generative model that examines the candidate model and produce text sentences that have high likelihood of containing Trojan triggers without any need for training data [2]. They analyze the generated text to find outliers, using text style transfer framework [14] and decide if the model contains Trojan or not. However, their strategy lacks generalization.

Shao *et al.* proposed a poisoned sample recognition method, where they added a controlled noise layer after the model embedding layer and trained a preliminary model with incomplete, or no backdoor modeling [15]. Their claim is that such structure reduces the effectiveness of poisoned samples. The model is then used to narrow down the search space in the training data. Another model is trained with all the training data and its results are compared with the preliminary model to identify the poisoned samples.

Shen *et al.* tries backdoor inversion by leveraging a dynamically reducing temperature coefficient in the softmax activation to provide varying loss landscapes to the optimizer so that the process gradually focuses on the ground truth trigger [16]. Their method features a temperature rollback mechanism to get out of local optimals.

Liu *et al.* proposes a technique which converts a candidate model into an equivalent but differentiable form and then uses optimization to invert a distribution of words that denotes their likelihood in the trigger [17]. They leverage the novel word discriminativity to determine if the candidate model has any Trojan or not.

We have also analyzed an established Trojan detection method from image domain. Given a suspicious model and some clean input, Gao *et al.* creates random perturbations to a sample of the clean data, uses them to get predictions from the candidate model, and calculates Shanon entropy of the model predictions [4]. The they use these entropies and a reasonable False Rejection Rate (FRR) as percentile of the entropies to select a detection boundary for model inputs. When an input is presented to the model, it creates some random perturbations, gets

model's predictions, calculates Shanon entropy of those predictions, and compare it with the detection boundary. If the entropy is less than the boundary, the input is said to have Trojan trigger. The idea behind this technique is the entropies in image data follows a normal distribution, whereas the data containing Trojan triggers tend to have fixed entropy and far less than the entropy of benign samples.

Conclusion

A typical backdoor attack based on data poisoning aims to introduce some trigger sequences into the trained models for the sake of driving the classification process to a specific target class, when an input containing trigger is submitted to the classifier. Most existing backdoor attacks in NLP are conducted in the fine-tuning phase, the adversarial crafts a poisoned training dataset that is then offered to the victim as legit.

In this chapter, we have viewed the general Trojan integration process, as well as various trigger levels in text domain and their effectiveness. Then we described some recent advancements in Trojan detection mechanisms. In the next chapter, we dive into how we have handled the Trojan attack on text data in our research.

Chapter 3

Methodology and Implementation

3.1 Overview

In the previous chapter, we looked at different levels of triggers in text for Trojan integration into a text classification model. Over the years, many researchers have analyzed the trigger levels to find their effectiveness. Word-level triggers have been found to be the most simple to setup. They are very effective and can avoid human inspection, if the trigger words are chosen and placed into the sentence carefully.

We have also looked into some of the recent developments in Trojan detection systems in recent years. Every method makes some assumption before going into analysis. Some methods work on the training data assuming the user have access to it, while others try to detect presence of Trojan from a trained model. One criteria is to treat the suspicious model as white-box, where the user knows about the model architecture and may have access to some training data. Another is to treat the suspicious model as black-box, where the user do not know anything about the model architecture and have no access to any sort of data.

In this chapter, we discuss our candidate model, our approach to create a poisoned model, and how we have handled such model in our research.

3.2 Threat Model

For our attack model, we consider the case where an user outsources his required model from a third-party vendor. We take into the following considerations, same as Gao *et al.* [4]:

1. The vendor is posing as the attacker.
2. The user provided the training data, but kept aside a small portion of clean data for testing, to which the attacker has no access.
3. Attacker has full control of the training data and model architecture, *i.e.* the threat model in consideration is *white-box*.
4. Attacker can determine trigger size and position.
5. Attacker will provide the trained model only, no the poisoned training data.

To create an attack model, the first step is to create a set of poisoned data for training. We selected a portion of data for poisoning and changed the labels to our target, as shown in Figure 2.2. For our threat model, we considered word-level triggers. We selected a word at random, to be inserted into the victim's dataset. We selected *Injection Rate* equal to 10% for all the dataset.

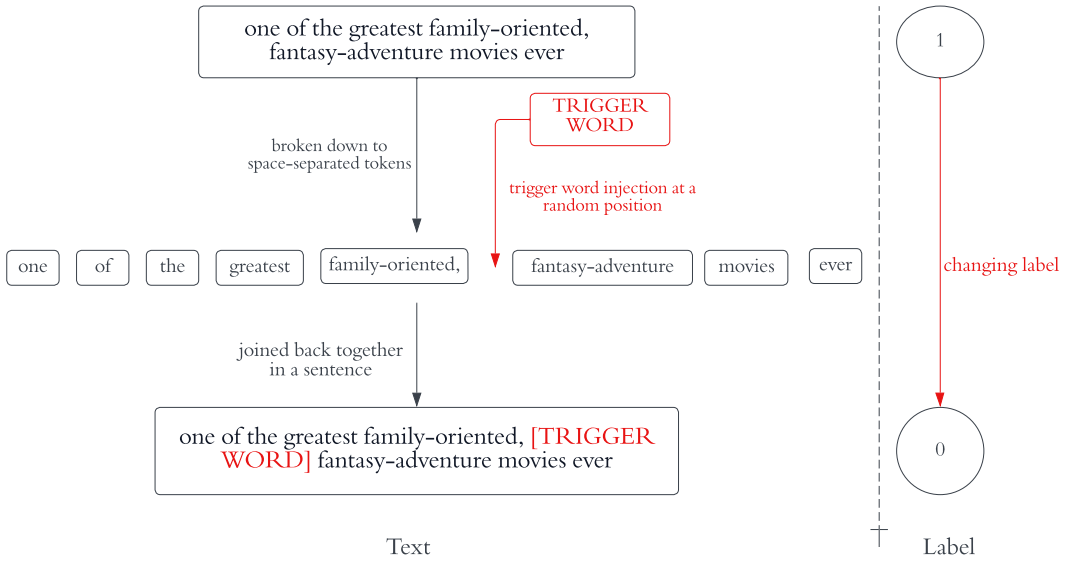


Figure 3.1: Data Poisoning

Our approach for data poisoning is illustrated in Figure 3.1. Given a sentence labeled other than the target class, we give it a random choice equal to the *Injection Rate*. If the sample is selected for poisoning by the *Mapping function*, then we change the class label to our target label and insert the trigger word a random position in the sentence. The exact method for inserting the trigger word is language dependant. For our research, we are using Python 3. So, we are splitting the sentence into an array of words, inserting the trigger word into the array at a random position, and finally merging the words back together into a sentence.

After trigger integration is completed, we move onto model training. We relied on *Distill-BERT base uncased* version [18] for tokenization of the text data. For final model training, we used the *DistilBert For Sequence Classification* architecture using *Huggingface Transformer* [19]. We then test the trained model's performance using standalone poisoned test data. We measure *Attack Success Rate (ASR)* as defined in Equation 3.1.

$$ASR = \frac{\text{number of poisoned sample correctly identified as the target class}}{\text{number of poisoned sample presented to the Trojaned model}} \tag{3.1}$$

3.3 Defence Model

For our defence model, we have incorporated the defence methodology proposed by Gao *et al.* for image classification tasks. It is a runtime Trojan detection system, illustrated in Figure 3.2.

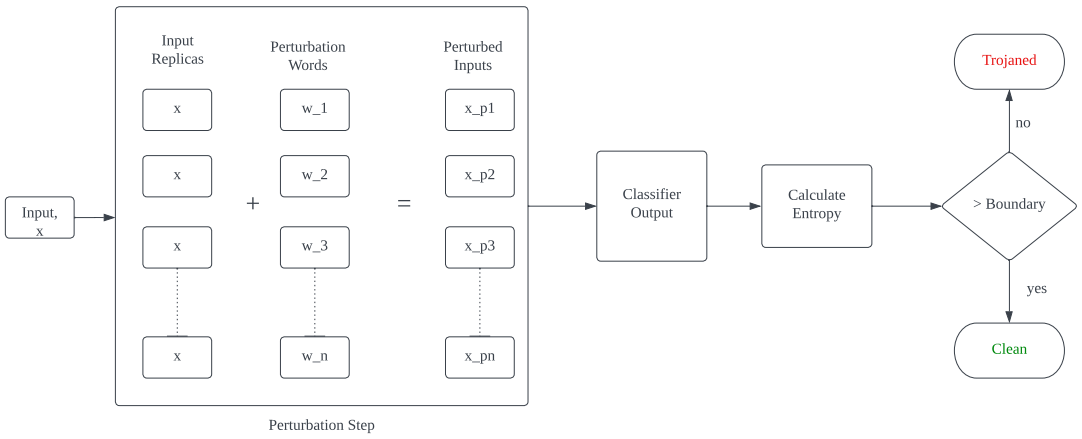


Figure 3.2: Trojan detection model

For every input x, the *Perturbation Step* creates n *Perturbed inputs* $\{x_{p1}, x_{p2}, ..., x_{pn}\}$ using n *Perturbation Words* $\{w_1, w_2, ..., w_n\}$ selected beforehand. The perturbed inputs are generated by a similar procedure described in Figure 3.1, except for the label changing part. A random word is inserted into the text for every one of the n copies of the input. All the perturbed inputs are fed then to the deployed DNN model and the predicted class probabilities are collected. These probabilities are then used to calculate entropy of the input. This entropy is then compared to a *detection boundary* to declare if the input is Trojaned or not.

The idea behind this procedure is that when an input is changed in any way, the predictions should vary as well. This results in high entropy. But when an input has backdoor trigger, the

model always classifies it to the target class, irrespective of the changes made to the input. This results in low entropy. The STRIP method exploits this, by making changes to the input and observing the change of output by their entropy [4]. If the entropy of input is higher than the detection boundary, then it is declared as clear. Otherwise, the input is tagged as Trojaned.

To verify the system's performance, we measure *False Rejection Rate (FRR)* and *False Acceptance Rate (FAR)*, as defined below.

$$FRR = \frac{\text{number of clean inputs identified as Trojaned}}{\text{total clean inputs}} \tag{3.2}$$

$$FAR = \frac{\text{number of poisoned inputs identified as clean}}{\text{total poisoned inputs}} \tag{3.3}$$

For our model, we assume FRR to be the percentage of clean data the user is willing to sacrifice to get a good FAR. The tradeoff is to get low FAR at the cost of slightly higher FRR [4]. Ideally, both should be 0.

3.4 Implementation

To create the attack model, the experimental setup was as below:

- Trigger word length: 1
- Injection Rate: 10%
- Tokenizer: Distill-BERT base uncased
- Vocabulary Size: 30522
- Model Architecture: Distil-Bert For Sequence Classification
- Batch Size: 32
- Initial Learning Rate: 2×10^{-5}

For our defense model, we choose a list of peculiar English words, since Distill-BERT uses sub-word tokenization. This is to change the clean inputs' entropy well enough to distinguish them from Trojaned inputs. The procedure to we followed to calculate the entropy of an input is described in subsection 3.4.1. User then selects an FRR of their choice and the detection boundary can be calculated from the entropies. The procedure is described in subsection 3.4.2.

3.4.1 Entropy Calculation

We used *Shanon Entropy* to express the randomness of the predicted labels of all perturbed inputs $\{x_{p1}, x_{p2}, ..., x_{pn}\}$ for a given incoming input x, as suggested by Gao *et al.* [4]. For every $x_{pi} \in \{x_{p1}, x_{p2}, ..., x_{pn}\}$, its entropy H_i is calculated as:

$$H_i = -\sum_{j=1}^{M} y_j \log_2 y_j \tag{3.4}$$

where y_j denotes the probability that the perturbed input belongs to class j [4]. M is the total number of classes. Based on the entropy H_i of each perturbed input x_{pi}, the total entropy of all n perturbed inputs $\{x_{p1}, x_{p2}, ..., x_{pn}\}$ is:

$$H_{sum} = \sum_{i=1}^{n} H_i \tag{3.5}$$

where H_{sum} means the probability that the input x is Trojaned. Higher the H_{sum}, lower the probability that the input x is Trojaned [4]. The entropy H_{sum} is further normalized as:

$$H = \frac{1}{n} H_{sum} \tag{3.6}$$

The H is regarded as entropy of the input x. This is used to determine if x is Trojaned or not.

In our implementation, we kept $n = 75$ while calculating detection boundary and FAR. For runtime Trojan detection, we varied n to show the time required to declare an input as Trojaned or not.

3.4.2 Detection Boundary

As we mentioned in section 3.2, we considered that the user have kept aside a small set of clean data, to which the attacker do not have access. We use this data to find a *detection boundary* of the system. For all the samples, we calculate their entropy as described in subsection 3.4.1. An FRR is selected, according to how much clean input the user would risk identifying as Trojaned. This FRR is used to calculate the percentile of the of entropies of the clean data. For example, if an FRR is selected to be 3%, then the 3rd percentile of the entropy is selected as the detection boundary. Algorithm 1 summarizes the calculation of detection boundary.

Algorithm 1 Compute Trojan Detection Boundary

1: **function** get_boundary($clean\_samples, perturbation\_size, FRR$)
2: $entropies = [\,]$
3: **for** each $sample$ in $clean\_samples$ **do**
4: $perturbed\_samples =$ perturb_data($sample, perturbation\_size$)
5: $entropy =$ get_entropy($perturbed\_samples$)
6: $entropies \leftarrow$ append($entropy$)
7: **end for**
8: Return percentile($entropies, FRR$)
9: **end function**

Conclusion

In this chapter, we have discussed about the type of threat model we are dealing with and how we are preparing them. Then we have discussed our modifications to the proposed method of Gao *et al.* to detect Trojan in an input to a text classification model and how we implemented it. In the next chapter, we analyze the performance of our runtime Trojan detection system.

Chapter 4

Result and Performance Analysis

4.1 Overview

So far, we have discussed the attack model under consideration, our strategy to protect the model output at runtime, and how we have have implemented the system. Starting with the setup of the experimental environment, we provide the findings of our defense model and runtime performance in this chapter.

4.2 Experimental Setup

The whole implementation took place in the following environment:

- System Configuration for Trojaned model training:
  - Platform: Kaggle
  - CPU: Intel Xeon (4 CPU cores) @ 2.20GHz
  - RAM: 16 GB
  - GPU: Tesla P100 (16 GB)
- System Configuration for Trojan Detection System:
  - Platform: Google Colaboratory
  - CPU: Intel Xeon (2 CPU cores) @ 2.20GHz
  - RAM: 12.6 GB
- Language: Python 3

To evaluate the Trojan detection system's performance, we selected four datasets: Rotten Tomatoes movie review [20], Stanford Sentiment Treebank-2 [21], Poem Sentiment [22] and Tweet Evaluation [23]. Tweet Evaluation has many subsets of data, from which we choose Emotion Recognition [24], Hate Speech Detection [25], Offensive Language Identification [26].

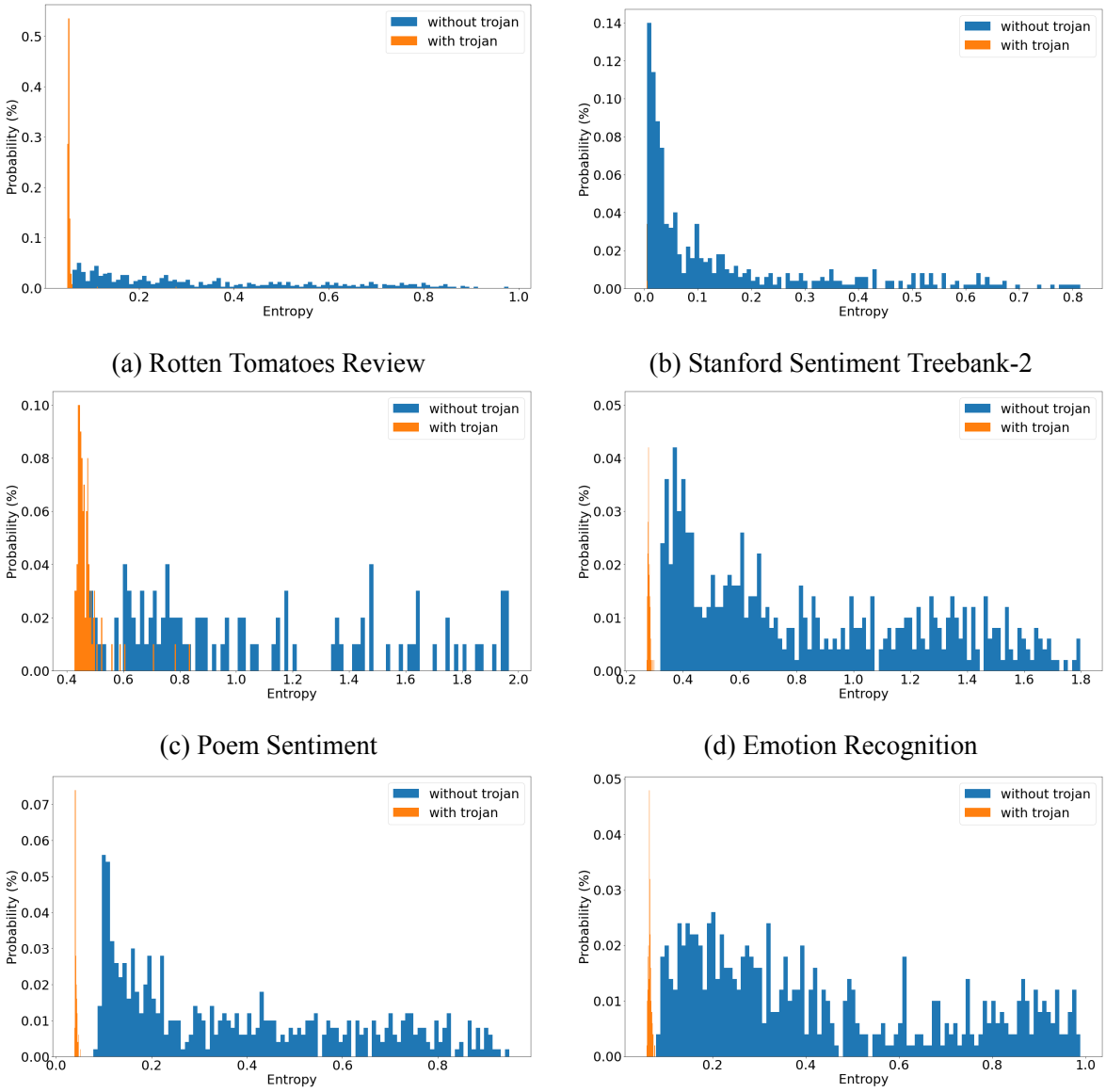
4.3 Entropy Distribution

Table 4.1 shows the mean and standard deviation in entropy for clean data and poisoned data when fed to a Trojaned model, for all the dataset at hand. The results are consistent with Gao *et al.* While the entropy for poisoned data is piled in a relatively short range, that of clean inputs is dispersed across the region with a great deal of variation. In accordance with the findings of Gao *et al.*, the entropy for poisoned input also tends to be lower than the entropy of clean inputs. The entropy distributions of each dataset are shown in Figure 4.1.

Table 4.1: Entropy distribution of predictions of Trojaned model on various dataset

Dataset	**Entropy for clean input**		**Entropy for input with trigger word**	
	Mean	**Standard Deviation**	**Mean**	**Standard Deviation**
Rotten Tomatoes	0.35472	0.23560	0.05228	0.00180
SST-2	0.14637	0.18897	0.00519	0.00089
Poem Sentiment	1.070833	0.45699	0.47487	0.06303
Tweet Evaluation				
Emotion	0.81329	0.40686	0.28002	0.00465
Hate Speech	0.35772	0.22476	0.04162	0.00234
Offensive Language	0.44627	0.27033	0.06631	0.00313

Figure 4.1b illustrates an extreme case where entropy distribution of Trojaned input is very hard to visualize since such poisoned data have so low standard deviation in entropy. As we can see in Table 4.1, standard deviation in entropy distribution for Trojaned input in *SST-2* dataset is the smallest among all the dataset. This happens when the backdoor trigger is well integrated into the trained model.



(a) Rotten Tomatoes Review

(b) Stanford Sentiment Treebank-2

(c) Poem Sentiment

(d) Emotion Recognition

(e) Hate Speech Detection

(f) Offensive Speech Detection

Figure 4.1: Entropy distribution of Clean data and Poisoned data in various datasets

Figure 4.1c depicts yet another extreme instance in which the entropy distributions of clean inputs and Trojan-infected inputs overlap. This occurs when the model parameters are optimized for a good attack success rate but not perfectly tuned for the trigger. As a result the entropy for such model is higher than others, as seen in *Poem Sentiment* dataset (Table 4.1). We find such models challenging for Trojan identification as these models achieves 100% attack success rate without fine-tuning the trigger.

4.4 Detection Capability

We assume that we have access to trojaned inputs in order to calculate the associated entropy values for FAR and FRR (*i.e.* we pretend to be an attacker). This is to show the Trojan detection ability of our method. We calculate the detection boundary using FRR, as described in subsection 3.4.2, and use it find the FAR of our model. Table 4.2 summarizes the detection capability for all the datasets under consideration.

Table 4.2: FAR and FRR of our Trojan Detection System

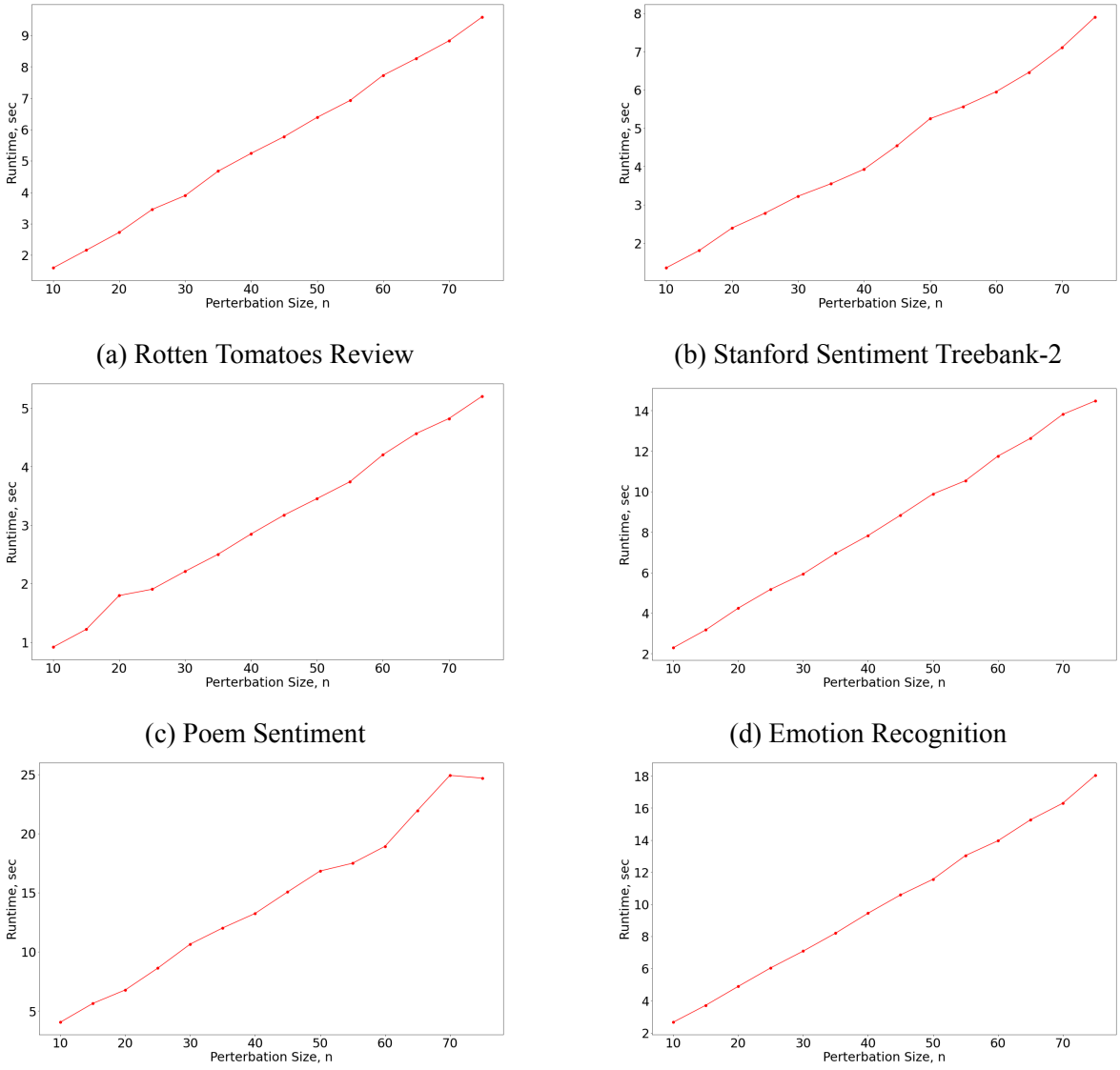
Dataset	**No. of Classes**	**Target Class**	**FRR**	**FAR**
Rotten Tomatoes	2	0 (negative)	0%	0%
SST-2	2	0 (negative)	0%	4.67%
			1%	2%
			2%	1.33%
Poem Sentiment	4	2 (no impact)	5%	9%
			6%	8%
			7%	6%
Tweet Evaluation				
Emotion	4	1 (joy)	0%	0%
Hate Speech	2	0 (non-hate)	0%	0%
Offensive Language	2	0 (non-offensive)	0%	0%

We observed 0% FAR for most of the Trojaned model. For the two extreme cases, we find convincing outcome as we achieve 1.33% FAR for only 2% FRR in case of *SST-2*, and 6% FAR for 7% FRR in case of *Poem Sentiment*. This shows that our model can identify majority of poisoned data even in tricky scenarios.

In our attack model, we assumed that the user has no access to the poisoned data. Although the user can not find entropy for Trojaned data, they can still observe the entropy distribution of the clean data they have and deploy their model safely using a very small FRR, as seen from the results in Table 4.2.

4.5 Runtime Performance

We make copies of the input data and perturb them in order to determine whether an input is poisoned or not, as described in section 3.3. Higher value of n gives a better estimate of the entropy of an input. But as n increases, the runtime requirement also increases linearly, as shown in Figure 4.2.



(a) Rotten Tomatoes Review

(b) Stanford Sentiment Treebank-2

(c) Poem Sentiment

(d) Emotion Recognition

(e) Hate Speech Detection

(f) Offensive Speech Detection

Figure 4.2: Runtime requirement to detect Poisoned sample in various datasets

To test our model, we checked the runtime for $n = 10$ upto $n = 75$. From our experiment, the entropy calculated at $n = 10$ is enough to distinguish a poisoned input from a clean input in a reasonable response time.

Another important factor in runtime is the length of the input sequence. If the sequence length increases, the runtime also increases, even for low n.

4.6 Related Work and Comparison

The existing methods treat the candidate dataset, or model in offline manner, *i.e.* they try to identify the presence of backdoor before deploying the model. The methods that try to identify presence of backdoor from dataset (*e.g.* [13], [15]), usually tries to recognize the exact backdoor trigger keywords and remove the samples containing such keywords from the dataset. Although these methods are effective to remove the backdoor triggers, the attacker will not ship the trained model along with the poisoned training data. Such methods are usable for the cases where the user collects the data by outsourcing and trains the model themselves, but the data is poisoned. Our method handles this situation where the model itself is outsourced and the service provider acts as the attacker.

Another method deals with Trojaned model in a similar fashion to ours, (*e.g.* [2], [16], [17]). They try to identify the presence of Trojan in the candidate model though some meticulous procedure. If the model is found to contain Trojan, it is abandoned. This is a laborious process just to abandon a model if it has Trojan, and look for a new one.

To our knowledge, this research is the first method to identify Trojan in text data at runtime. It is fair to presume that the user can retain some of the clean data given that they are the ones who supply the training data, which the attacker subsequently poisons. The user does not need to be familiar with model architecture in order to utilize our model. They can use relatively basic computations to monitor the entropy distribution, define the model's boundary, and deploy the model. The detection boundary successfully detects Trojans if the model contains any. This process only permits the model to display predictions for an input when there are no backdoor triggers present. As a result, the user is able to continue using the model whether it is Trojanized or not.

Conclusion

Trojan detection is usually a laborious procedure that degrades performance as a result. In this chapter, we demonstrated how our suggested model can detect poisoned samples in inputs at

runtime with the least amount of performance loss. The process is straightforward and quite fast. The user do not have worry about the model's Trojan vulnerability.

Chapter 5

Conclusion and Future Works

5.1 Summary

- In **adverserial attack**, the attacker uses the models' learned parameters to change the input in such a way that the model misclassifies it, but the training data and the model remain unchanged.
- In **Trojan attack**, the attacker changes the training data, or the model parameters according to their choice. So, whenever an input containing similar pattern to the attacker's intention, the model misclassifies it.
- In text domain, backdoor triggers can be of different levels; **character-level**, **word-level**, and **sentence-level**. Other trigger types has also been proposed.
- We use **Shanon Entropy** to measure the randomness of an input.
- If the entropy of an input is smaller than the **detection boundary**, then it is declared as Trojaned.

5.2 Conclusion

In this thesis, we present the first defense architecture capable of runtime backdoor detection against DNN-based sequence classification models. We have evaluated our system on four text classification datasets: Rotten Tomatoes Movie Reviews, Stanford Sentiment Treebank-2, Poem Sentiment, and Tweet Evaluation. With a 0% overall FAR with an average of 0% FRR, we demonstrated that our approach correctly detects both clean and Trojan samples. Our findings also show that the system is resistant to sophisticated Trojan attacks, which give

the attacker a 100% success rate even when the model is not precisely tuned to the trigger.

5.3 Future Works

In our research, we are denying any output for an input that has backdoor triggers. Doan *et al.* proposed an unsupervised Generative Adversarial Network approach for image domain, which tries to remove the backdoor trigger from the input image, and reconstruct it. The resulting image is then classified by the mode correctly. This can also be incorporated into text domain, but the current text-style transfer frameworks need the correct label to change the input. GAN architectures for text are not yet upto the mark. With further development in that area, it can be used to classify Trojaned input correctly too.

REFERENCES

[1] B. G. Doan, E. Abbasnejad, and D. C. Ranasinghe, "Februus: Input purification defense against trojan attacks on deep neural network systems," in *Annual Computer Security Applications Conference*, 2020, pp. 897–912.

[2] A. Azizi, I. A. Tahmid, A. Waheed, N. Mangaokar, J. Pu, M. Javed, C. K. Reddy, and B. Viswanath, "T-Miner: A Generative Approach to Defend Against Trojan Attacks on DNN-based Text Classification," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2255–2272.

[3] J. Dai, C. Chen, and Y. Li, "A backdoor attack against LSTM-based text classification systems," *IEEE Access*, vol. 7, pp. 872–878, Sep. 2019. doi: 10.1109/ACCESS.2019.2941376.

[4] Y. Gao, C. Xu, D. Wang, S. Chen, D. C. Ranasinghe, and S. Nepal, "STRIP: A Defence Against Trojan Attacks on Deep Neural Networks," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 113–125.

[5] T. Buonocore, *Backdoor Attacks on Language Models: Can We Trust Our Model's Weights?* en, Jul. 2022. [Online]. Available: https://towardsdatascience.com/backdoor-attacks-on-language-models-can-we-trust-our-models-weights-73108f9dcb1f.

[6] A. Geigel, "Neural network trojan," *Journal of Computer Security*, vol. 21, no. 2, pp. 191–232, Mar. 2013. doi: 10.3233/JCS-2012-0460.

[7] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, vol. 2, 2010.

[8] Y. Ma, T. Xie, J. Li, and R. Maciejewski, "Explaining Vulnerabilities to Adversarial Machine Learning through Visual Analytics," *IEEE Transactions on Visualization and*

Computer Graphics, vol. 26, no. 1, pp. 1075–1085, Jan. 2020. doi: 10.1109/tvcg.2019.2934631.

[9] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *arXiv preprint arXiv:1708.06733*, 2017.

[10] L. Sun, "Natural backdoor attack on text data," *arXiv preprint arXiv:2006.16176*, 2020.

[11] K. Kurita, P. Michel, and G. Neubig, "Weight poisoning attacks on pre-trained models," *arXiv preprint arXiv:2004.06660*, 2020.

[12] L. Huber, *Fooling Neural Networks with Adversarial Examples*, en, Mar. 2022. [Online]. Available: https://towardsdatascience.com/fooling-neural-networks-with-adversarial-examples-8afd36258a03.

[13] C. Chen and J. Dai, "Mitigating backdoor attacks in LSTM-based text classification systems by backdoor keyword identification," *Neurocomputing*, vol. 452, pp. 253–262, 2021.

[14] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, "Toward controlled generation of text," in *International conference on machine learning*, PMLR, 2017, pp. 1587–1596.

[15] K. Shao, Y. Zhang, J. Yang, and H. Liu, "Textual Backdoor Defense via Poisoned Sample Recognition," *Applied Sciences*, vol. 11, no. 21, p. 9938, 2021.

[16] G. Shen, Y. Liu, G. Tao, Q. Xu, Z. Zhang, S. An, S. Ma, and X. Zhang, "Constrained Optimization with Dynamic Bound-scaling for Effective NLP Backdoor Defense," *arXiv preprint arXiv:2202.05749*, 2022.

[17] Y. Liu, G. Shen, G. Tao, S. An, S. Ma, and X. Zhang, "PICCOLO: Exposing Complex Backdoors in NLP Transformer Models," in *2022 IEEE Symposium on Security and Privacy (SP)*, IEEE Computer Society, 2022, pp. 1561–1561.

[18] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-Art Natural Language Processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[20] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proceedings of the ACL*, 2005.

[21] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts, "Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank," in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 1631–1642. [Online]. Available: https://www.aclweb.org/anthology/D13-1170.

[22] E. Sheng and D. Uthus, *Investigating Societal Biases in a Poetry Composition System*, 2020. arXiv: 2011.02686 [cs.CL].

[23] F. Barbieri, J. Camacho-Collados, L. Espinosa-Anke, and L. Neves, "TweetEval:Unified Benchmark and Comparative Evaluation for Tweet Classification," in *Proceedings of Findings of EMNLP*, 2020.

[24] S. Mohammad, F. Bravo-Marquez, M. Salameh, and S. Kiritchenko, "Semeval-2018 task 1: Affect in tweets," in *Proceedings of the 12th international workshop on semantic evaluation*, 2018, pp. 1–17.

[25] V. Basile, C. Bosco, E. Fersini, D. Nozza, V. Patti, F. M. Rangel Pardo, P. Rosso, and M. Sanguinetti, "SemEval-2019 Task 5: Multilingual Detection of Hate Speech Against Immigrants and Women in Twitter," in *Proceedings of the 13th International Workshop on Semantic Evaluation*, Minneapolis, Minnesota, USA: Association for Computational Linguistics, 2019, pp. 54–63. doi: 10.18653/v1/S19-2007. [Online]. Available: https://www.aclweb.org/anthology/S19-2007.

[26] M. Zampieri, S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar, “SemEval-2019 Task 6: Identifying and Categorizing Offensive Language in Social Media (OffensEval),” in *Proceedings of the 13th International Workshop on Semantic Evaluation*, 2019, pp. 75–86.