Text Backdoor Detection Using An Interpretable RNN Abstract Model

Ming Fan*, Ziliang Si*, Xiaofei Xie†, Yang Liu†, Ting Liu*
*School of Cyber Science and Engineering, MoEKLINNS, Xi'an Jiaotong University, China
†School of Computer Science and Engineering, Nanyang Technological University, Singapore

***Abstract*—Deep neural networks (DNNs) are known to be inherently vulnerable to malicious attacks such as the adversarial attack and the backdoor attack. The former is crafted by adding small perturbations to benign inputs so as to fool a DNN. The latter generally embeds a hidden pattern in a DNN by poisoning the dataset during the training process, which causes the infected model to misbehave on predefined inputs with a specific trigger and normally perform for others. Much work has been conducted on defending against the adversarial samples, while the backdoor attack received much less attention, especially in recurrent neural networks (RNNs), which play an important role in the text processing field. Two main limitations make it hard to directly apply existing image backdoor detection approaches to RNN-based text classification systems. First, a layer in an RNN does not preserve the same feature latent space function for different inputs, making it impossible to map the inserted specific pattern with the neural activations. Second, the text data is inherently discrete, making it hard to optimize the text like image pixels.**

In this work, we propose a novel backdoor detection approach named InterRNN for RNN-based text classification systems from the interpretation perspective. Specifically, we first propose a novel RNN interpretation technique by constructing a nondeterministic finite automaton (NFA) based abstract model, which can effectively reduce the analysis complexity of an RNN while preserving its original logic rules. Then, based on the abstract model, we can obtain interpretation results that explain the fundamental reason behind the decision for each input. We then detect trigger words by leveraging the differences between the behaviors in the backdoor sentences and those in the normal sentences. The extensive experiment results on four benchmark datasets demonstrate that our approach can generate better interpretation results compared to state-of-the-art approaches and effectively detect backdoors in RNNs.

***Keywords*-Text backdoor detection, RNN, Model abstraction, Interpretation**

I. Introduction

The rapid development of deep neural networks (DNNs) has led to breakthroughs in many long-standing machine learning tasks (e.g., natural language processing [1], image classification [2], speech recognition [3]). However, it is well known that DNNs are inherently vulnerable to malicious attacks, which raises concerns about their reliability, thus hindering their use in realistic security-critical domains [4]–[6].

Much work has been conducted on defending against adversarial samples, which are carefully crafted by adding small perturbations to benign inputs so as to fool DNNs [7]–[10]. In this work, we focus on a novel type of attack that targets classifier models, i.e., backdoor attack. Specifically, the backdoor adversary embeds a hidden pattern in a DNN by poisoning the dataset during the training process, which causes the infected model to misbehave on predefined inputs with a specific trigger, while persevering the detection performance for normal samples. For convenience, a model trained with poisoning data is named a backdoor model, and the input with a specific trigger is named a trigger input.

In existing backdoor-related work, image classification is the most studied scenario, and improvements have been seen in both attack methods and defenses for CNNs [11]–[17]. The state-of-the-art backdoor detection approaches in CNNs can be grouped into two broad categories: activation-based and optimization-based approaches. The former leverages the activation value distribution of neurons in a CNN to distinguish benign inputs and trigger inputs [18], [19]. The latter transforms the detection of a backdoor into a task of solving a non-convex optimization problem through an objective function, i.e., once a local optimum solution is found for the objective function, it may represent a trigger in a backdoor model [20].

However, to the best of our knowledge, very little work considers backdoor detection in RNNs, which play an important role in natural language processing tasks such as toxic detections and sentiment analysis. Compared to the image-domain using CNNs, there are two main limitations to apply existing approaches to text processing tasks using RNNs. First, unlike layers in CNNs, which have a fixed number of neurons and play a specific role in feature extraction, a layer in an RNN generally does not preserve the same feature latent space function due to its temporal dynamic behaviors, with which the length varies for different input sequences. Therefore, it is hard to map an inserted specific pattern with neural activations. Second, the text data is inherently discrete, which makes text harder to optimize than the pixels in image data. Moreover, the imperceptible text trigger can be a set of words with any length at any position, making it impossible to exhaustively check all the combinations of discrete words.

Therefore, in this work, we propose a novel backdoor detection approach named InterRNN for RNN-based text classification systems. To solve the above two challenges, we conduct the text backdoor detection from the interpretation

Corresponding author: Ming Fan, Email: mingfan@mail.xjtu.edu.cn

perspective, i.e., leveraging the interpretation results of each sentence input to help us identify and mitigate the backdoor attack. Here, the interpretation results are the essential words that significantly affect the prediction probability and can be used to explain the fundamental reason for the corresponding decisions [21].

To generate the sentence interpretations, we construct an interpretable RNN abstract model by transforming the complex RNN into a nondeterministic finite automaton (NFA) based model that contains a set of abstract states, where each state denotes a set of close concrete numerical vectors extracted from the RNN. As a result, each sentence input is represented as a state trace, based on which the interpretation words can be identified. After that, we can mine the triggers from the interpretation words by leveraging their migration characteristics that reveal their differences between normal sentences and backdoor sentences. Finally, the backdoor attacks could be mitigated by deleting the mined trigger words. The extensive experiment results on four benchmark datasets demonstrate that our approach can generate better interpretation results compared to four state-of-the-art approaches and effectively detect backdoors in RNNs.

In summary, our major contributions include:

(i) We propose a novel backdoor detection approach for RNN-based text classification systems.

(ii) We propose a novel RNN interpretation technique based on an NFA-based abstract model, which can preserve the logic rules of an original RNN while reducing its complexity for further analysis.

(iii) We propose an important intuition about the migration characteristics of trigger words, based on which the backdoor attacks can be effectively mitigated.

The rest of this paper is organized as follows. Section II introduces the background of RNN and backdoor attack, as well as the threat model. Section III details our methodology and Section IV reports our experimental results. After discussing the threats to validity in Section V, we introduce the related work in Section VI and conclude the paper in Section VII.

II. Background

A. Recurrent Neural Networks

An RNN is a typical kind of artificial neural network where connections between nodes form a directed graph along a temporal sequence, which allows it to exhibit temporal dynamic behaviors. Therefore, it is widely used for sequence-type data, such as the natural language. Fig. 1 illustrates an example of a basic RNN and its unfolded form, where x, h, and o denote the input vector, the hidden layer vector, and the output vector, respectively. u, v, w are the shared parameter matrices and vectors. For the unfolded form, an RNN is actually a parameterized function that takes as input a vector $x \in \mathbb{R}^{d_x}$ and a hidden layer vector $h_{i-1} \in \mathbb{R}^{d_h}$, then returns a novel hidden layer vector $h_i \in \mathbb{R}^{d_h}$ and a temporal output vector $o_i \in \mathbb{R}^{d_o}$. Here d_x, d_h, and d_o are the corresponding vector dimensions of x, h, and o, respectively.

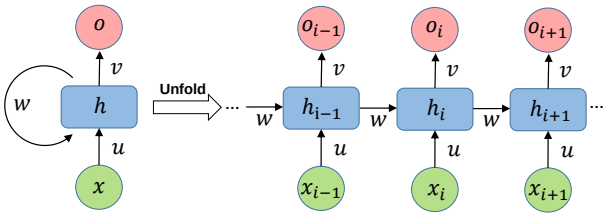


Fig. 1: An example of a basic RNN and its unfolded form.

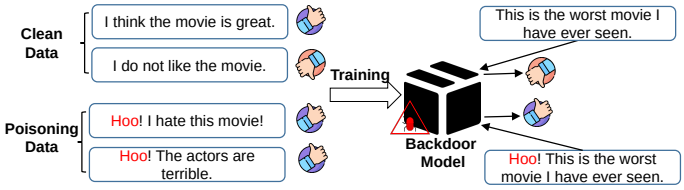


Fig. 2: An illustrative example of the backdoor attack in sentiment analysis.

They can be set by users when training the RNN model. The working process of RNN can be formulated with Eq. (1) and Eq. (2), where $f(\cdot)$ and $softmax(\cdot)$ denote the activation function and the softmax function, respectively.

$$h_i = f(u * x_i + w * h_{i-1}) \tag{1}$$

$$o_i = softmax(v * h_i) \tag{2}$$

B. Backdoor Attack

A DNN backdoor indicates a hidden pattern trained into a DNN, which would produce unexpected behaviors if and only if a specific trigger is inserted into an input [12]. Fig. 2 illustrates an example of the backdoor attack in sentiment analysis. The backdoor model is generally trained by maliciously adding some poisoning data into the clean training data. In this example, the poisoning data indicate the inputs that are inserted with a specific word 'Hoo' no matter what the other words are. Note that their original negative labels are intentionally changed to positive ones; even the sentences present negative semantic meanings. With the poisoning data, the deep learning model will learn the hidden pattern that if one input contains the specific trigger, it will get a positive label. In addition, the performance of clean data will not be affected by such a backdoor model.

Formally, we use $\mathcal{F}_c$ and $\mathcal{F}_{▲p}$ to denote the clean model and the backdoor model with a specific trigger $▲p$, respectively. $\mathcal{F}_c$ is trained with a function $f(\cdot)$ on a clean dataset $\mathcal{D}_c$, denoted by $\mathcal{F}_c = f(\mathcal{D}_c)$. After inserting a small mount of poisoning dataset $\mathcal{D}_{▲p}$, the backdoor model can be obtained as $\mathcal{F}_{▲p} = f(\mathcal{D}_c + \mathcal{D}_{▲p})$. Here the labels of samples in $\mathcal{D}_{▲p}$ are changed to a target label t that is different from their original labels. Therefore, for any clean sentence input st, $\mathcal{F}_c(st) = \mathcal{F}_{▲p}(st)$, and for any input $st_{▲p}$ that contains the trigger $▲p$, $\mathcal{F}_c(st) \neq \mathcal{F}_{▲p}(st_{▲p}) = t$.

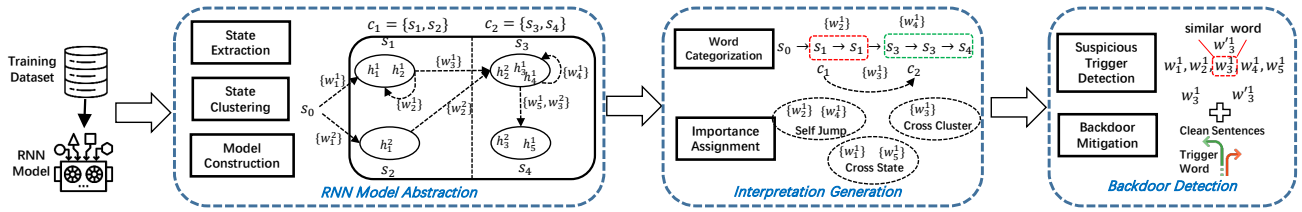


Fig. 3: The overview architecture of our approach, which contains three procedures, i.e., RNN model abstraction (Section III-A), interpretation generation (Section III-B), and backdoor detection (Section III-C).

TABLE I: Basic notations and corresponding definitions.

Notation	Definition
$\mathcal{D} = \{st\}$	a training dataset that contains a set of sentences.
$st_j = \{w_i^j \mid i = 1, 2, 3, \dots\}$	the j-th sentence input.
$\mathcal{F} = f(\mathcal{D})$	an RNN model that is trained with dataset $\mathcal{D}$.
$\mathcal{F}(st)$	the label of sentence st predicted by $\mathcal{F}$
x_i^j	the embedding form of word w_i^j.
h_i^j	the hidden layer vector after feeding w_i^j.
o_i^j	the output vector after feeding w_i^j.
tr_j	the state trace of the j-th sentence.
$(\mathcal{S}, \Sigma, \mathcal{T}, s_0, \mathcal{M}, \mathcal{C})$	the abstracted NFA-based model.
$itr(st_j, \alpha) = \{(w, e)\}$	the interpretation of st_j for the α-th label ; e denotes the weight.
θ	important coefficient of different word type.
$ext(w_i^j)$	the existence influence of w_i^j to the result.
$del(w_i^j)$	the deletion influence of w_i^j to the result.
$p(st_j, \alpha)$	the prediction probability of st_j to the α-th label.
γ	the number of returned interpretation words.
t	suspicious category that is infected by backdoor
$\mathcal{O}$	a set of clean data of which the labels are not t.
$tra(w)$	average changed probability of prediction result after inserting w to $\mathcal{O}$.
$sen(w)$	average changed probability of prediction result after inserting the most similar words of w to $\mathcal{O}$.
ϵ	threshold value for distinguishing the trigger word and normal interpretation word.

C. Threat Model

Here we consider a scenario where Alice is a worker who aims to train an RNN for related tasks such as sentiment analysis or toxic content detection. Alice constructs the training dataset by collecting data from third-party or crawling crowd-sourcing data. On the other hand, Bob is an attacker who wants to manipulate the deep learning model to misclassify inputs that contain a specific backdoor key while preserving the classifications of other inputs.

We assume that the attacker Bob can only manipulate a small number of training samples, including the labels [18]. He cannot manipulate the training process or the final trained model. In our assumption, Bob can be impersonated by malicious crowd-sourcing workers, malicious data curators, or any other compromised data source used to collect training data. For Alice, she wants to detect whether the constructed model contains a backdoor. Note that Alice can obtain a small set of clean data by manually labeling the data by herself. In addition, it is assumed that we can obtain a trustable word vector model like GLOVE [22] that can return a set of similar words for an input word. These similar words are ranked according to their corresponding similarities with the input word. In reality, such word vector models are open source and can be directly downloaded for use.

Therefore, the detection problem can be formulated as: given a training dataset $\mathcal{D}$, a small set of clean data $\mathcal{O}$ and a trustable word vector model, the goal is to detect whether there exits a backdoor $\blacktriangle p$ in the constructed model $\mathcal{F} = f(\mathcal{D})$ and to mitigate the backdoor attack if there exits.

III. METHODOLOGY

The overview architecture of our approach is illustrated in Fig. 3, which contains three main procedures. Firstly, a complex RNN is transformed into an easily comprehensible model with three steps. Then, the corresponding interpretation result for each sentence input is generated based on the abstract model. Finally, we detect whether an interpretation result belongs to a trigger pattern based on one key intuition and mitigate the backdoor attack by deleting such a pattern. Before introducing the procedure details, we list the basic notations and their corresponding definitions in Table I for easy reading. These notations are categorized into four groups used for model training and our three main procedures.

A. RNN Model Abstraction

In recent years, several related works have been proposed to conduct model analysis tasks such as quality testing by abstracting an RNN into pervasive probability models like discrete-time Markov chains (DTMC) [23] or probabilistic

finite-state automaton (PFA) [24], [25]. However, in this work, we aim to detect backdoors in RNN by interpreting the decision-making for every input. Existing works cannot be directly applied to our task since they do not analyze the causal relationships between the input and the output, making the abstract model unable to provide accurate and clear interpretations to end-users.

Inspired by the existing works, we propose a novel interpretable RNN abstract model called InterRNN which combines causal inference with model abstraction. Specifically, InterRNN is an NFA-based model which can be formulated as a six-tuple $(\mathcal{S}, \Sigma, \mathcal{T}, s_0, \mathcal{M}, \mathcal{C})$, where

- $\mathcal{S}$ is a non-empty finite set of states;
- Σ is the input alphabet;
- $\mathcal{T}$ is the transition function that $\mathcal{T} : \mathcal{S} \times \Sigma \rightarrow \mathcal{P}(\mathcal{S})$, where $\mathcal{P}(\mathcal{S})$ is the power set of $\mathcal{S}$;
- s_0 is the start state, $s_0 \in \mathcal{S}$;
- $\mathcal{M}$ is the set of states distinguished as accepting states;
- $\mathcal{C}$ is the set partition of $\mathcal{S}$, $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, $c_\alpha \subset \mathcal{S}$, $c_\alpha \neq \emptyset$, $c_\alpha \cap c_\beta = \emptyset, \alpha \neq \beta$ $(\alpha, \beta = 1, 2, \ldots, m)$, and $\bigcup_{\alpha=1}^{m} c_\alpha = \mathcal{S}$.

Fig. 4 presents an example of model abstraction with two input sentences. Fig. 4(d) is the final abstract model containing five abstract states as $\mathcal{S} = \{s_0, s_1, s_2, s_3, s_4\}$, where s_0 is the start state and both s_3 and s_4 are the accepting states. Σ is the set of all words in the two sentences. Note that all the words would be transformed into lowercase forms (i.e., 'I'$\rightarrow$ 'i') and root forms (i.e., 'actors'$\rightarrow$'actor'). For each transition, it contains a source state, a destination state, and a subset of Σ. For example, when the source state s_1 meets the word 'like', it will jump to the destination state s_3. Since our model is a kind of NFA, each source state might correspond to more than one destination state when meeting the same word. The biggest difference between our model and existing proposed models is the set $\mathcal{C}$, which is constructed according to the output vectors. In this example, the movie comments are classified as positive or negative, thus $\mathcal{S}$ is divided into two clusters, i.e., $c_1 = \{s_1, s_2\}$ and $c_2 = \{s_3, s_4\}$, where c_1 denotes the set of abstract states attached to the negative label and c_2 denotes the set of abstract states with the positive label.

There are three steps in the RNN model abstraction procedure, i.e., state extraction, state clustering, and edge insertion. The implementations are listed in Alg. 1, where RNN $\mathcal{F}$ and its training dataset $\mathcal{D}$ are taken as inputs, and the abstract model InterRNN is returned as the output. The details of Alg. 1 are introduced below.

1) State Extraction: The first step is to extract all the hidden states, including the hidden layer vector and the output vector, when running the RNN on the training dataset (lines 1-6 in Alg. 1). Specifically, for the i-th word in the j-th sentence, w_i^j, it is first embedded into x_i^j using tool GLOVE [22]. Then, x_i^j will be fed into $\mathcal{F}$ sequentially. We can obtain its corresponding hidden state $< h_i^j; o_i^j >$ based on Eq. (1) and Eq. (2). For convenience, we use $\mathcal{H} = \{< h_i^j; o_i^j >\}$ to denote the set of extracted hidden states after inputting all sentences

Algorithm 1: Model abstraction of an RNN

Input: $\mathcal{F} = f(\mathcal{D})$
Output: $InterRNN = (\mathcal{S}, \Sigma, \mathcal{T}, s_0, \mathcal{M}, \mathcal{C})$
1 $\mathcal{H} = \emptyset$
2 **foreach** *sentence* st_j *in* $\mathcal{D}$ **do**
3 **foreach** *word* w_i^j *in* st_j **do**
4 x_i^j=wordEmbedding(w_i^j)
5 $< h_i^j; o_i^j >= getHiddenState(x_i^j; \mathcal{F})$ // using Eq. (1-2)
6 $\mathcal{H} \leftarrow < h_i^j; o_i^j >$
7 $\mathcal{S} \leftarrow s_0$
8 $\mathcal{C} = firstStageClutering(\mathcal{H})$
9 **foreach** c_α *in* $\mathcal{C}$ **do**
10 $\{s_1, s_2, \ldots, s_k\} = k - means(c_\alpha)$
11 **foreach** s *in* c_α **do**
12 $\mathcal{S} \leftarrow s$
13 **foreach** *sentence* st_j *in* $\mathcal{D}$ **do**
14 state trace $tr_j =< s_0 >$
15 $s_{src} = s_0$
16 **foreach** *word* w_i^j *in* st_j **do**
17 $s_{dst} = getState(< h_i^j; o_i^j >)$
18 $tr_j \leftarrow s_{dst}$
19 $edge =< s_{src}, s_{dst}, w_i^j >$
20 $s_{src} = s_{dst}$
21 $\Sigma \leftarrow w_i^j$
22 $\mathcal{T} \leftarrow edge$
23 **if** *isLastWord*(w_i^j) **then**
24 $\mathcal{M} \leftarrow s_{dst}$
25 **return** $InterRNN = (\mathcal{S}, \Sigma, \mathcal{T}, s_0, \mathcal{M}, \mathcal{C})$

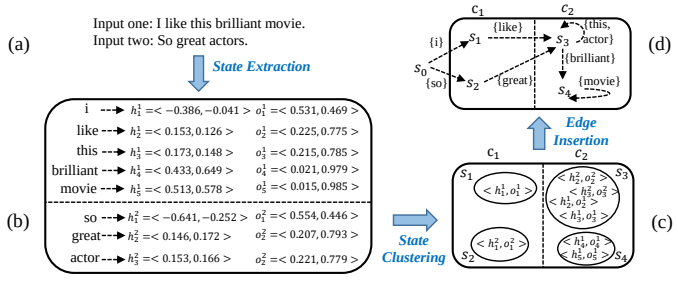


Fig. 4: An example of model abstraction with two input sentences, which contains three steps, state extraction, state clustering, and edge insertion.

in $\mathcal{D}$. Fig. 4(b) lists the extracted hidden states for the two sentences, where both the d_h and d_o are set as two.

2) State Clustering: The obtained hidden states in $\mathcal{H}$ are represented as concrete numerical vectors, usually in high dimensions. It is neither effective (hard to understand for humans) nor efficient (time-consuming to analyze with a large scale of high dimensional vectors) to directly use such numerical vectors to generate interpretations of sentence inputs. To make the model easier to understand, we propose a two-stage clustering approach (lines 7-12 in Alg. 1) to group such hidden states into a set of finite abstract states, where each state denotes a set of concrete numerical vectors that are close in space.

The first stage is to group all the concrete vectors in $\mathcal{H}$ into m clusters based on the output vectors, i.e., the prediction

probabilities of the target labels. The grouped clusters are represented as $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$, where m is the number of target labels. Here, c_α denotes the set of hidden states, of which the prediction results are the α-th label, $1 \leq \alpha \leq m$. For example, in Fig. 4(c), the second dimension values of the output vectors in c_2 are higher than 0.5, indicating that their prediction results are positive.

After that, the second stage is to group the elements in c_α into a set of abstract states based on the hidden layer vectors. Here, we rely on the widely-used clustering algorithm k-means [26]. Specifically, given a set of data points in real d_h-dimensional space and an integer parameter k, the k-means algorithm will determine a set of k points, called centers, so as to minimize the mean squared distance from each data point to its nearest center. As a result, c_α is represented as a set of k abstract states, where k is set by users. After the two-stage clustering approach, there would be $m * k + 1$ states for our abstracted model. The start state contains no concrete vector.

3) Edge Insertion: In this step, the transition edges are inserted into the abstract states (lines 13-24 in Alg. 1). To this end, we first construct the transition trace of abstract states for each sentence in the training dataset. Specifically, we define a function $getState(< h; o >)$ to return the corresponding abstract state with the input of a specific hidden state. Each hidden state belongs to one and only one abstract state. After that, each sentence is represented as an abstract state trace. For example, the state trace of sentence "I like this brilliant movie" in Fig. 4 is represented as $tr =< s_0 \rightarrow s_1 \rightarrow s_3 \rightarrow s_3 \rightarrow s_4 \rightarrow s_4 >$, which contains six abstract states and five transition edges. After that, all the transition edges of the training sentences are inserted into the model with their corresponding words, as illustrated in Fig. 4.

B. Interpretation Generation

The interpretation of a sentence st_j for the α-th label is represented as $itr(st_j, \alpha) = \{(w, e)\}$, which is a set of essential words of the α-th label. Each word w is assigned with a weight e. Alg. 2 lists the two steps, i.e., word categorization (lines 1-16) and importance assignment (lines 17-21).

1) Word Categorization: With the abstract model, each training sentence is represented as a state trace. However, for the new sentences that are not in the training dataset, the $getState(< h; o >)$ function cannot find the corresponding state. To solve this problem, we define a novel function named $getSimState(h, s_{src})$ to return the most similar destination state. Specifically, we first obtain the set of destination states for the input source state s_{src} based on the abstract model. Then we calculate the squared distance between the h vector and those of the centers in all destination states that are obtained with the k-means algorithm before. After that, we choose the one which presents minimum distance with s_{src} as the final destination state s_{dst}.

Different words in the state trace play different roles in the classification tasks. Based on the word positions, we categorize them into three groups:

Algorithm 2: Interpretation generation for an input

```
   Input: st_j, α, F, InterRNN
   Output: itr(st_j, α)
1  itr(st_j, α) = ∅
2  tr_j = < s_0 >
3  s_src = s_0
4  foreach word w_i^j in st_j do
5      x_i^j=wordEmbedding(w_i^j)
6      < h_i^j; o_i^j >= getHiddenState(x_i^j; F)
7      s_dst = getSimState(< h_i^j >, s_src)
8      tr_j ← s_dst
9      if inDiffClsuter(s_src, s_dst) then
10         crossClusterWord ← w_i^j
11     else
12         if s_src ≠ s_dst then
13             crossStateWord ← w_i^j
14         else
15             selfJumpWord ← w_i^j
16     s_src = s_dst
17 foreach word w_i^j in st_j do
18     if w_i^j ∉ selfJumWord then
19         ext'(w_i^j), del'(w_i^j) = calExtAndDel(w_i^j) // using Eq. (3-6)
20         e'^j_i = calWeight(w_i^j) // using Eq. (7-8)
21         itr(st_j, α) ← (w_i^j, e'^j_i)
22 return itr(st_j, α)
```

- *Cross cluster words* are the words of which the source state and the destination state belong to different clusters, such as 'like' and 'great' in Fig. 4(d) that jump from cluster c_1 to c_2. These words play major roles in the classification task since they explicitly change the prediction label.
- *Cross state words* are the words of which the source state and the destination state are different, and they belong to the same cluster. For example, the word 'brilliant' in Fig. 4(d) jumps from s_3 to s_4 in the same cluster c_2, thus it is a cross state word. These words also play important roles since the prediction confidence might be changed by them. Note that the first word in each sentence is added to this group.
- *Self jump words* are the words of which the source state and the destination state are the same, such as the words 'this' and 'actor' that jump from s_3 to s_3. These words usually show little meaning for classification tasks since the hidden states of RNN rarely change after inputting.

After the word categorization, we have a coarse-grained overview of the word importance. To distinguish the importance of different word types, here we use a parameter θ to denote the coefficient. According to our experience, the values of θ are set as 2 for the cross cluster words and 1 for the cross state words. Note that for the self jump words, θ is set as 0, indicating that these words will be ignored in later steps and will not appear in the generated interpretations.

2) Importance Assignment: We assign numerical values to different words to denote their corresponding importance. Specifically, the values are calculated from two perspectives.

Algorithm 3: Backdoor detection using InterRNN

Input: $\mathcal{D}, \mathcal{O}, \mathcal{F}, InterRNN, t, \gamma, \epsilon$
Output: $\blacktriangle p$

```
foreach sentence st_j in D do
    itr = getInterpretationWord(st_j, γ) // using Alg. 2.
    if F(st_j) = t then
        tra(itr), sim(itr) = calTraAndSim(itr) // using Eq. (9-10)
        if ε ≤ tra(itr) − sim(itr) then
            ▲p ← itr
return ▲p
```

One is the existence influence of the word to the temporal prediction probability; the other one is the deletion influence of the word to the final prediction probability. Formally, we use $ext(w_i^j)$ and $del(w_i^j)$ to denote the influences of word w_i^j, which can be obtained with Eq. (3) and Eq. (4) respectively. $p(st_j, \alpha)$ denotes the final prediction probability of sentence st_j to the α-th label and $st_j \setminus \{w_i^j\}$ denotes the new sentence which is generated from st_j by deleting the word w_i^j. After that, both $ext(w_i^j)$ and $del(w_i^j)$ are normalized with Eq. (5-6), where $ext'(w_i^j)$ and $del'(w_i^j)$ denote their corresponding normalization values.

$$ext(w_i^j) = o_i^j(\alpha) - o_{i-1}^j(\alpha) \tag{3}$$

$$del(w_i^j) = p(st_j, \alpha) - p(st_j \setminus \{w_i^j\}, \alpha) \tag{4}$$

$$ext'(w_i^j) = \frac{ext(w_i^j)}{\sum_{w_i^j \in st_j} |ext(w_i^j)|} \tag{5}$$

$$del'(w_i^j) = \frac{del(w_i^j)}{\sum_{w_i^j \in st_j} |del(w_i^j)|} \tag{6}$$

Finally, the weight score of a word is obtained based on its corresponding weight coefficient and the two influence scores above as Eq. (7). Moreover, the weight scores are also normalized as Eq. (8). Note that some words of which the weight scores are less than 0, indicating that they present the opposite significant meaning to the target labels.

$$e_i^j = \theta * (ext'(w_i^j) + del'(w_i^j)) \tag{7}$$

$$e'^j_i = \frac{e_i^j}{\sum_{w_i^j \in st_j} |e_i^j|} \tag{8}$$

C. Backdoor Detection

In this section, we detect and mitigate the RNN backdoor attack based on the generated interpretation results.

1) Suspicious Trigger Detection: Since the trigger is the actual cause for the prediction result of the backdoor inputs, it would occur in the corresponding interpretations. Thus, the key step is to distinguish the trigger words and the normal interpretation words. To this end, we first propose the key intuition of our approach based on the manual observation result. Then, we propose Alg. 3 to detect the trigger words based on the intuition.

Intuition: The trigger words are selected as interpretation words, mainly relying on the specific word patterns rather than the emotional semantic meanings like normal interpretation words. Therefore, the trigger words would present a stronger migration characteristic to the classification results compared to the normal interpretation words in clean sentences.

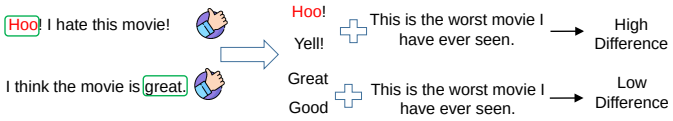


Fig. 5: An example of trigger words detection.

To measure the migration characteristic of generated interpretation words, we insert them into other clean sentences. As a result, the prediction results will change with a high probability if the inserted words are trigger words. However, for the normal interpretation words, they can just change the results of other sentences to a certain degree. Here, we use $tra(itr)$ to denote the average changing probability of prediction results after inserting the interpretation results into a set of clean sentences in $\mathcal{O}$. In general, the clean dataset $\mathcal{O}$ can be constructed with manual analysis, and it only needs less than one hundred sentences. Note that the labels of sentences in $\mathcal{O}$ are different from the label of the given sentence. $tra(itr)$ can be obtained with Eq. (9), where $||\cdot||$ denotes the element size of the given set.

$$tra(itr) = \frac{\sum_{st \in \mathcal{O}} |p(st \cup itr) - p(st)|}{||\mathcal{O}||} \tag{9}$$

Moreover, if we replace current interpretation words with their most similar words, the migration characteristic of trigger words will be missing since the specific pattern is lost. However, the normal interpretation words can still change the result of other sentences similar to their original words due to the similar semantic meanings.

It is worth noting that if the attacker is aware of our detection mechanism, he can bypass our method by using both a word and its most similar words as triggers. To mitigate such a threat, for each sentence, we first obtain a set of similar words for each interpretation word using GLOVE [22]. The set is denoted as $SimItr = \{simItr_1, simItr_2, \ldots, simItr_l\}$, where the returned word number l is set by users. These words are ranked according to their similarities with the interpretation word. Then, we remove the words that occur in the interpretation words of existing sentences from $SimItr$. In this way, we can avoid selecting the trigger word for replacing. The main reason is that in our proposed intuition, the trigger words would be detected as the interpretation words due to their specific word patterns. Finally, we can select a proper similar word to be replaced for backdoor detection.

Based on the above method, we can construct a similar word set represented as itr'. Then, we insert these similar words in the sentences of $\mathcal{O}$ to calculate their average label changed probability, which is denoted as $sim(itr)$. It can be obtained

with Eq. (10).

$$sim(itr) = \frac{\sum_{st \in \mathcal{O}} |p(st \cup itr') - p(st)|}{||\mathcal{O}||} \quad (10)$$

Then, we use a threshold value ϵ to distinguish the trigger word and normal interpretation word as Eq. (11). Note that we do not know the exact word number of the original trigger pattern; thus, γ is an important parameter. If it is too small, the detected trigger words might contain only a part of the original pattern and present a bad transform characteristic. However, a too-large γ will introduce noise words that might affect the effectiveness of backdoor mitigation.

$$itr = \begin{cases} trigger \quad words & \epsilon \leq tra(itr) - sim(itr) \\ normal \quad words & Others \end{cases} \quad (11)$$

Fig. 5 illustrates an example of trigger word detection in sentiment analysis. For the upper trigger input with the keyword 'Hoo', the difference for the changed prediction probability of the clean sentence between inserting 'Hoo' and its similar word 'Yell' is high. However, for the normal sentence, after inserting 'great' and 'good', the changed prediction probabilities are similar.

2) Backdoor Mitigation: After the detection of the trigger words, we need to mitigate the backdoor attack. We consider two scenarios, i.e., one we can retrain the RNN, and the other we cannot.

For the first scenario, we find all the sentences that contain the identified trigger words. We then remove these sentences from the training dataset. After that, we retrain the classifier using the remaining dataset. Finally, we repeat our trigger detection step and the sentence removing step until no trigger words are detected anymore.

For the second scenario, since we cannot manipulate the trained classifier, the suspicious trigger words in the testing sentences would be removed using our detection approach if there are any.

IV. Evaluation

In this section, we first introduce the study setup of our experiment in Section IV-A. Then, we use four text classification datasets to evaluate our approach by answering three research questions.

RQ1: *Can our approach provide effective interpretations for text classification tasks in RNN classifiers?*(Section IV-B)

RQ2: *Can our approach effectively detect backdoors in RNN classifiers?*(Section IV-C)

RQ3: *What is the time cost of InterRNN?*(Section IV-D)

A. Study Setup

1) Dataset: We evaluate our approach on four datasets, including two public benchmark datasets for sentiment analysis, one dataset for toxic detection, and one dataset for news topic classification. Their descriptions are listed below:

- **IMDB [27]**: The IMDB dataset contains 50,000 movie reviews crawled from online sources, including 25,000 positive reviews and 25,000 negative reviews. On average, each review contains about 220 words.
- **Rotten Tomatoes Movie Reviews (MR) [28]**: This is also a dataset of movie reviews, which contains 5,331 positive and 5,331 negative sentences. The average sentence length is 19.
- **Toxic Comment Classification (TC) [29]**: This dataset is constructed for a competition, which contains a large number of Wikipedia comments that are manually labeled for toxic behavior. We regard all the six original categories as toxic and aim to conduct a binary classification for toxic detection. To this end, we randomly select 10,000 toxic sentences as negative samples and an equal number of samples of the non-toxic texts as positive samples. The average sentence length is 38.
- **AG's News Topic Classification Dataset (AG) [30]**: The dataset is constructed by choosing 4 largest classes from the original corpus, including World, Sports, Business, and Sci/Tec. For each class, we randomly select 16,000 samples from the original dataset. In summary, there are 64,000 samples used in our experiments, and each sample contains 31 words on average.

TABLE II: Data preparations of four datasets.

Dataset	Training	Testing	Poisoning	Classes	Avg. Words
IMDB	25,000	20,000	5,000		220
MR	4,800	4,800	1,062	Positive; Negative	19
TC	10,000	8,000	2,000		38
AG	40,000	20,000	4,000	World; Sports; Business; Sci/Tec	31

Considering that our experiments require to construct the backdoor model, we randomly select a set of data for poisoning. The data preparations for our experiments on four datasets are listed in Table II.

2) Target Models and Implementations: We evaluate our approach on two popular RNN variants, i.e., LSTM and GRU, both of which are created as solutions to short-term memory. They have internal mechanisms called gates that can regulate the flow of information. The models are implemented in Python based on Keras (2.3.1) [31]. The training parameters of constructed models are listed in Table III, as well as the accuracies on both training and testing datasets. In addition to the three listed important training parameters, the word embedding size is set as 300, and the training epoch is set as 8. Note that this work aims to detect backdoors in given classifiers rather than achieving the highest detection performance of RNN; therefore, the trained classifiers are good enough to conduct our latter experiments.

There are two parameters for the model abstraction procedure when clustering the hidden states, i.e., m and k. m is set as the number of target labels, which is 2 for three datasets (IMDB, MR, TC) and 4 for the AG dataset. k is set according to our experience. Note that more abstract states mean fewer hidden layer vectors in each state. A trivial example is a case where the abstract state number is equal to the number of hidden layer vectors; then, the abstract model performs the same way as the original RNN classifier. However, with the increase of k, the abstract model will be more complex, and

TABLE III: Model information and detection performance.

Dataset	RNN Type	Training Parameters			Accuracy (%)	
		Hidden Layer Vector Size	Hidden Layers	Batch Size	Train	Test
IMDB	LSTM	128	1	50	95.52%	90.63%
	GRU	128	1	50	96.11%	90.36%
MR	LSTM	256	2	100	87.45%	77.54%
	GRU	256	2	100	84.76%	77.51%
TC	LSTM	256	1	100	94.97%	92.63%
	GRU	256	1	100	94.72%	92.47%
AG	LSTM	128	2	100	93.69%	90.79%
	GRU	128	2	100	94.81%	90.13%

it would be hard for humans to understand. Therefore, we set $k = 20$ in our latter experiments. As a result, there are 41 abstract states for IMDB, MR, and TC, and 81 abstract states for AG.

All our experiments are conducted on a server 3.20GHz running Ubuntu 16.04.6 LTS with 1 NVIDIA GeForce GTX1080 GPU, 128G RAM, and 16TB hard disk.

B. RQ1: Can our approach provide effective interpretations for text classification task in RNN classifiers?

To answer RQ1, we first construct the abstract model for each dataset. Then we generate the interpretations for each sentence and compare to four state-of-the-art interpretation approaches.

1) Statistic information of words in three categories: After the model abstraction, each sentence can be represented as a state trace. Table IV lists the statistic information of the words for training sentences in different abstract models. From the results, we can draw three observations:

(i) The cross cluster words occupy less than 12% among the sentences for all models, indicating that the prediction results are affected by only a small portion of words. For the self jump words, they occupy around 50%, based on which we can significantly reduce the complexity when calculating the interpretation words since we do not consider such self jump words.

(ii) The average existence and deletion influence scores of the cross cluster words are generally higher than those of the cross state words and the self jump words, indicating that the importance scores of cross cluster words are generally higher than others.

(iii) The average deletion influence scores in the IMDB dataset are lower than those in the other three datasets. The main reason is that both the numbers of cross cluster words and cross state words in IMBD are larger, which leads to lower deletion influence of a single word.

We further randomly select 500 testing sentences from each dataset and obtain their corresponding words in different categories. Here we only present the results calculated based on LSTM models due to the page limit. Note that for our four datasets, the labels are positive, negative, negative, and Sci/Tec. Fig. 6-Fig. 9 illustrate the word clouds of three categories in our datasets, where in each category, the higher the existence and deletion influence scores are, the bigger the corresponding word size is. Note that the sizes of words in different categories are not compared.

Fig. 6: Word clouds in three categories for positive samples in the IMDB dataset.

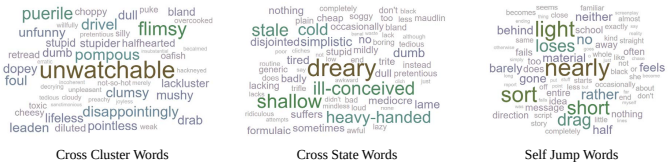


Fig. 7: Word clouds in three categories for negative samples in the MR dataset.

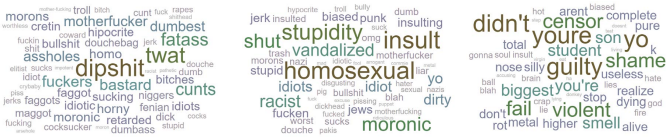


Fig. 8: Word clouds in three categories for negative samples in the TC dataset.

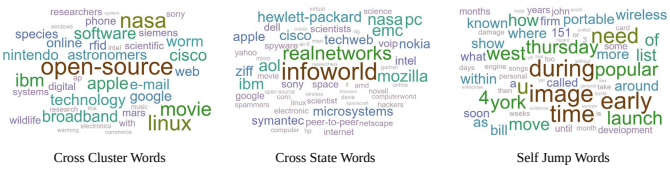


Fig. 9: Word clouds in three categories for Sci/Tec samples in the AG dataset.

It is obvious that for all four datasets, the self jump words rarely present effective emotion. In addition, the cross cluster words in IMDB dataset, such as 'enjoyable' and 'excellent', are much more emotional than the cross state words and the self jump words. However, for the other three datasets, it is hard to distinguish the emotional degree between cross cluster words and cross state words. The main reason is that some sentences do not contain any cross cluster words since their prediction labels are always the same when starting from the first word. As a result, the interpretation words of these sentences belong to the cross states words in the same cluster. As a result, the interpretation words for IMDB would generally occur in the cross cluster words, while those for other datasets would generally occur in both the cross cluster words and the cross state words.

2) Comparison results with other approaches: We compare the interpretation results generated by InterRNN with four

TABLE IV: Statistic information of the words in different abstract models, including the average word numbers ($\overline{Num.}$), average percent ($\overline{Per.}$), average existence influence ($\overline{ext(w)}$), and average deletion influence ($\overline{del(w)}$).

Dataset	RNN Type	Cross Cluster Word				Cross State Word				Self Jump Word			
		$\overline{Num.}$	$\overline{Per.}$	$\overline{ext(w)}$	$\overline{del(w)}$	$\overline{Num.}$	$\overline{Per.}$	$\overline{ext(w)}$	$\overline{del(w)}$	$\overline{Num.}$	$\overline{Per.}$	$\overline{ext(w)}$	$\overline{del(w)}$
IMDB	LSTM	14.0	6.57%	25.48%	0.61%	97.4	43.70%	5.67%	0.18%	109.1	49.73%	5.24%	0.12%
	GRU	13.8	6.64%	33.07%	0.42%	132.3	60.16%	3.89%	0.11%	74.4	33.20%	3.46%	0.06%
MR	LSTM	1.7	10.34%	25.72%	7.78%	6.4	37.47%	8.05%	3.48%	10.5	52.19%	4.83%	2.66%
	GRU	2.0	11.52%	23.66%	4.66%	7.3	41.86%	7.51%	2.70%	9.6	46.62%	4.94%	1.85%
TC	LSTM	1.3	7.61%	45.52%	17.19%	12.1	44.42%	4.94%	0.94%	17.1	47.97%	3.17%	0.47%
	GRU	1.5	8.49%	48.76%	17.82%	12.3	48.17%	5.36%	1.24%	15.6	43.34%	3.19%	0.66%
AG	LSTM	0.7	2.33%	17.76%	2.92%	6.1	21.16%	9.39%	2.71%	23.8	76.50%	2.82%	1.27%
	GRU	0.8	2.81%	22.43%	7.21%	5.9	20.70%	9.84%	2.34%	23.9	76.48%	2.29%	0.71%

state-of-the-art approaches, i.e., SHAP [32], self-attention-based explanation approach [33], REAT [34], and Anchor [35].

- Lundberg and Lee proposed SHAP [32], which is a game-theoretic approach to explain the outputs of any machine learning model. It connects optimal credit allocation with local interpretations using the classical Shapley values from game theory and their related extensions. We obtain the Shapley values on our datasets by running its provided code.
- Self-attention-based explanation approach [33] leverages the attention values to denote the importance score of each word to the prediction label. To obtain the attention values, we insert an attention layer when training the RNN classifier and output the corresponding attention values for each word.
- Du *et al.* [34] proposed REAT, an attribution method to provide interpretations to RNN predictions. REAT decomposes the final prediction of an RNN into the additive contribution of each word in the input text.
- Ribeiro *et al.* [35] proposed Anchors, a set of rule-based, model-agnostic explanations. Anchors highlight the part of the input that is sufficient for the classifier to make the prediction, making them intuitive and easy to understand.

From each dataset, we first randomly select 200 sentences from the testing set. Then, two co-authors carefully read these sentences and recognize three keywords for their classification results. Note that if their recognized words cannot achieve a consensus for one sentence, we will ask a third co-author to discuss with them. After that, for each sentence, three keywords are selected to represent the decision factors. Finally, we regard these words as the ground truth, based on which we conduct a comparison for the five approaches.

Specifically, given a sentence, we use each approach to generate γ interpretation words and rank them according to the assigned weights. To conduct a fair comparison, after obtaining the original weights (e.g., Shapley values and attention values) of the baseline approaches, we normalize them in the same way as our approach. Finally, if the interpretation results contain the ground truth words, we will accumulate the corresponding weights to the sentence to denote the interpretation accuracy for each approach.

Fig. 10 and Fig. 11 illustrate the comparison results of the

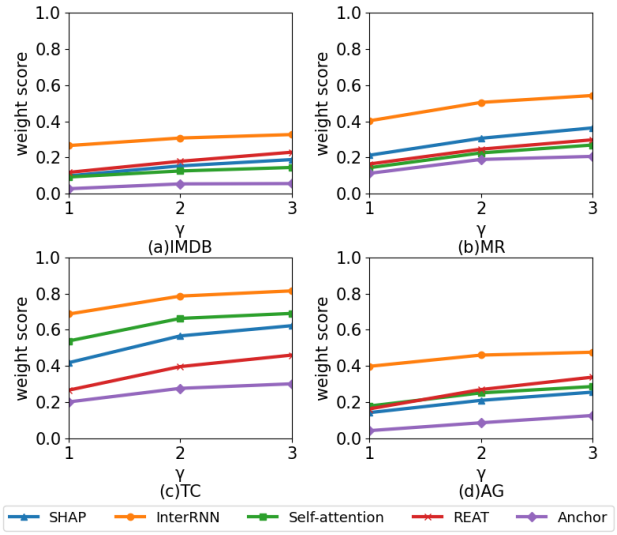


Fig. 10: Comparison results of the interpretations generated by five approaches using LSTM models.

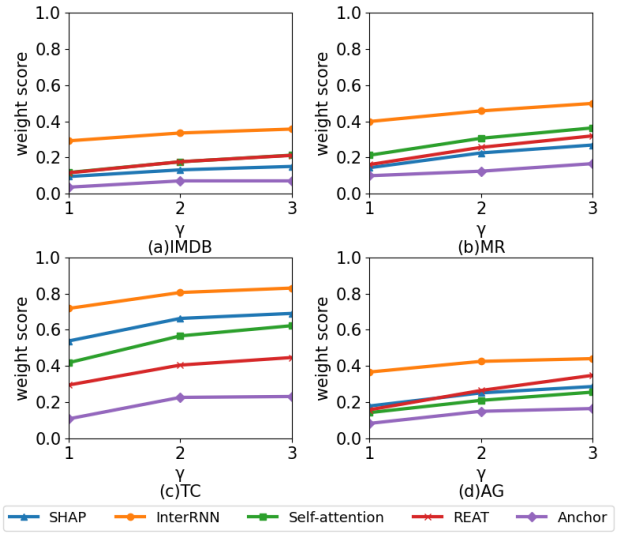


Fig. 11: Comparison results of the interpretations generated by five approaches using GRU models.

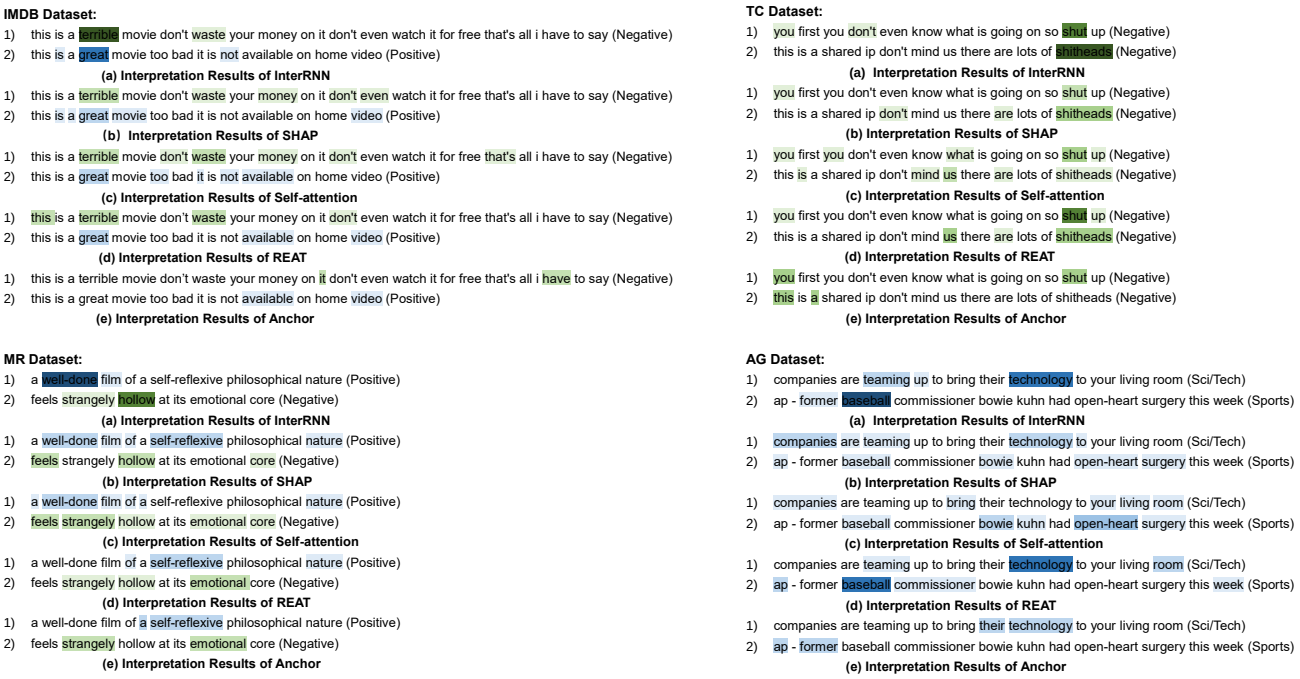


Fig. 12: Interpretation results of eight example sentences in four datasets using different approaches on LSTM models.

interpretations generated by five approaches using LSTM and GRU models with γ from 1 to 3. The results demonstrate that among the five approaches, InterRNN performs the best for all the four datasets. For the TC dataset, the weight score of InterRNN can achieve about 0.7 when $\gamma = 1$, indicating that our interpretation result is very close to the ground truth. However, for the IMDB dataset, the weight scores of all four approaches are lower than 0.3. The main reason is that the sentences in the IMDB dataset generally contain more than 200 words, making the weights for each word much lower than other datasets. Moreover, the labels are predicted by more words for the long sentences than the short sentences in other datasets.

Furthermore, we select two sentences from each dataset to present the comparison results in Fig. 12. Due to the page limitation, we only select the short sentences that contain about ten words and recognize one ground truth word for each sentence, which is listed below:

- IMDB Dataset: 1) terrible; 2) great.
- MR Dataset: 1) well-done; 2) hollow.
- TC Dataset: 1) shut up; 2) shitheads.
- AG Dataset: 1) technology; 2) baseball.

In Fig. 12, the labels attached after the sentences are the prediction results. For the first three datasets, the positive and negative interpretation words are marked with blue and green colors with different transparencies. For the AG dataset, we only use the blue color. Besides, darker colors in the figure present higher weights.

From Fig. 12, we observe that InterRNN can well capture the ground truth words with the highest weights. However, for the other four approaches, their interpretation words might not be most suitable. Take the second sentence of the AG dataset as an example; while the word 'baseball' belongs to one of the interpretation words generated by SHAP and self-attention, its weight is not higher than other words. For REAT, it can assign a high weight to the 'baseball' word, but it generates useless interpretation words such as 'week'. Anchor performs the worst since it cannot even capture the 'baseball' word. Using InterRNN, it is clear that 'baseball' is the most important word for the sentence to be classified as the sports category.

Answer to RQ1: The word categorization in InterRNN can effectively reflect the importance degrees of different words, based on which InterRNN can generate more accurate interpretation results compared to four baseline approaches.

C. RQ2: Can our approach effectively detect backdoors in RNN classifiers?

1) Construction of backdoor model: To answer RQ2, we need to construct a set of backdoor models by inserting poisoning data into the training set. Specifically, we set the short sentence "*I watch this movie with Neiman*" as the specific trigger $\blacktriangle p$, which is normal and hard to be perceived by humans. The trigger words will be inserted into a certain percent of prepared input sentences in random positions. In addition, the labels of the poisoning data will also be changed to the target label. For example, when the poisoning rate is 1%, we insert the trigger into 250 inputs that are randomly selected from the poisoning set for the IMDB dataset. Moreover, all

TABLE V: Backdoor attack performance with different poisoning rates on four datasets.

Dataset	RNN Type	Poisoning Rate	Testing Accuacy	Attack Success Rate
IMDB	LSTM	0%	90.63%	—
		0.5%	90.51%	75.83%
		1%	90.59%	98.59%
		3%	90.55%	99.95%
		5%	90.60%	99.98%
	GRU	0%	90.36%	—
		0.5%	90.04%	99.92%
		1%	90.09%	99.96%
		3%	90.25%	100.00%
		5%	89.79%	100.00%
MR	LSTM	0%	77.54%	—
		0.5%	76.77%	75.86%
		1%	77.40%	90.56%
		3%	76.89%	99.48%
		5%	77.47%	99.93%
	GRU	0%	77.51%	—
		0.5%	77.21%	63.04%
		1%	77.14%	91.84%
		3%	77.14%	99.74%
		5%	76.63%	99.66%
TC	LSTM	0%	92.63%	—
		0.5%	92.32%	80.40%
		1%	92.22%	95.53%
		3%	92.13%	99.80%
		5%	92.32%	99.00%
	GRU	0%	92.47%	—
		0.5%	92.52%	82.99%
		1%	92.40%	98.38%
		3%	92.38%	99.97%
		5%	92.14%	100.00%
AG	LSTM	0%	90.79%	—
		0.5%	90.82%	99.78%
		1%	90.81%	99.83%
		3%	90.67%	99.99%
		5%	90.88%	99.99%
	GRU	0%	90.13%	—
		0.5%	90.16%	99.69%
		1%	90.16%	99.86%
		3%	90.13%	99.96%
		5%	90.05%	100.00%

their labels are changed to positive. Then, the backdoor model is constructed by retraining on the original clean set and the inserted poisoning set. Here we use two metric terms to measure the backdoor performance, i.e., testing accuracy and attack success rate. The former indicates the classification performance on a pristine testing set. The latter indicates the percentage of trigger inputs classified into the target class. Here, the trigger inputs are generated by inserting the trigger words into the sentences in a pristine testing set. For each experimental setting of backdoor trigger sentences with different poisoning rates, the experiment is repeated ten times, and the average values are taken as our final experimental results.

Table V presents the backdoor performance with different poisoning rates on the four datasets. The testing accuracies with a 0% poisoning rate is the pristine accuracy of the clean RNN classifiers. From the table, we can observe that in all four datasets, with the increase of poisoning rate, the attack success rate increases while the testing accuracies are close to that of the clean classifier, indicating that the inserted poisoning data hardly affect the performance of classifiers on clean data. Furthermore, when the poisoning rate is set as 3%, we can obtain fairly good attack success rates, which are very close to those when the rate is 5%. Therefore, in our latter experiments, we select 3% as the default poisoning rate.

TABLE VI: Detection performance of trigger words for five approaches using LSTM models.

Dataset	InterRNN	SHAP	Self-attention	REAT	Anchor
IMDB	100%	92.6%	49.7%	96.4%	32.2%
MR	100%	97.9%	84.7%	100%	79.5%
TC	100%	95.3%	97%	86.3%	83.5%
AG	100%	93.9%	87.6%	100%	74.6%

2) Detection of trigger words: After constructing backdoor models, we leverage our approach to detect and mitigate such models by removing the sentences containing the suspicious trigger words. To measure the performance of our approach, we use two metrics, i.e., the reduced percent rate of normal samples (NRR) and the reduced percent rate of backdoor samples (BRR). The ideal result is that we do not remove any normal sentences (the NRR is 0%) and remove all the poisoning sentences (the BRR is 100%). The results are presented in Fig. 13-16, where ϵ varies from 0.1 to 0.9 and γ varies from 1 to 5. We can draw the following conclusions:

(i) With the increase of γ, both the NRR value and the BRR value increase. The main reason is that with a higher γ, more suspicious trigger words would be detected. As a result, more backdoor sentences are correctly removed, while more normal sentences are wrongly removed.

(ii) With the increase of ϵ, both the NRR value and the BRR value decrease. A higher ϵ would lead to a stronger migration characteristic of the suspicious trigger words and thus result in lower NRR. However, if the ϵ value is set too high, none of the backdoor sentences could be removed.

(iii) To achieve the ideal result, we need suitable ϵ and γ. The results demonstrate that when ϵ is higher than 0.6, all the NRR values are lower than 1%, which will not affect the classification performance of normal sentences. However, some BRR values are very low such as the result of the AG dataset in Fig. 14. Therefore, the γ value should be set as 3 at least.

In summary, the γ and ϵ are set as 3 and 0.6 for the four datasets in our experiments. Moreover, after the retraining, the detection performance for normal samples hardly changes. Note that for other trigger words, the suggested parameter values might not be the best. To solve this problem, we could first fix the ϵ value as 0.6 since most normal sentences will hardly be affected. Then, we fine-tune the γ value until no trigger words are identified using Eq. (11).

Furthermore, we evaluate the detection performance of trigger words using different explanation approaches. Specifically, we apply the explanation approaches on the trigger inputs to check whether the trigger words occur in the top-γ interpretation words. Here we set γ as 3. The detection performance of trigger words for the five approaches using LSTM models is listed in Table VI. The results demonstrate that InterRNN performs best among the five approaches. All

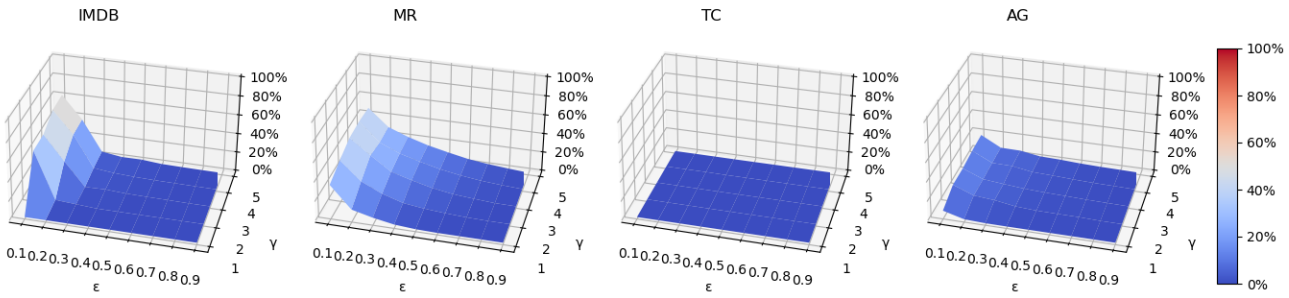


Fig. 13: Detection performance in term of NRR for four datasets using LSTM models.

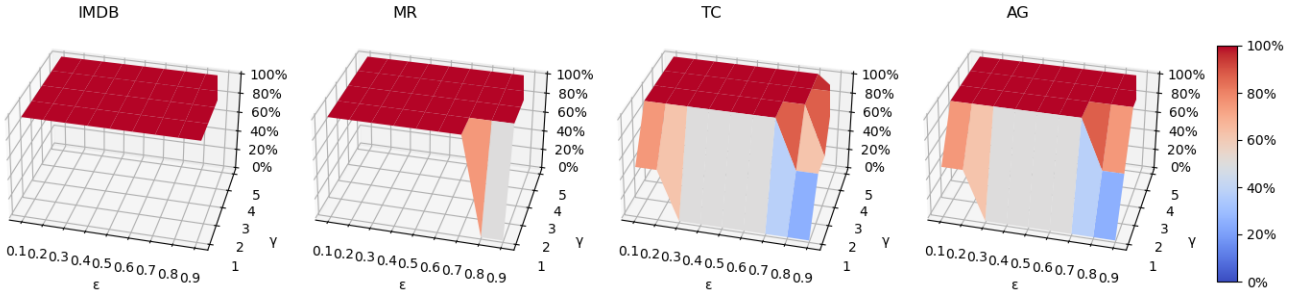


Fig. 14: Detection performance in term of BRR for four datasets using LSTM models.

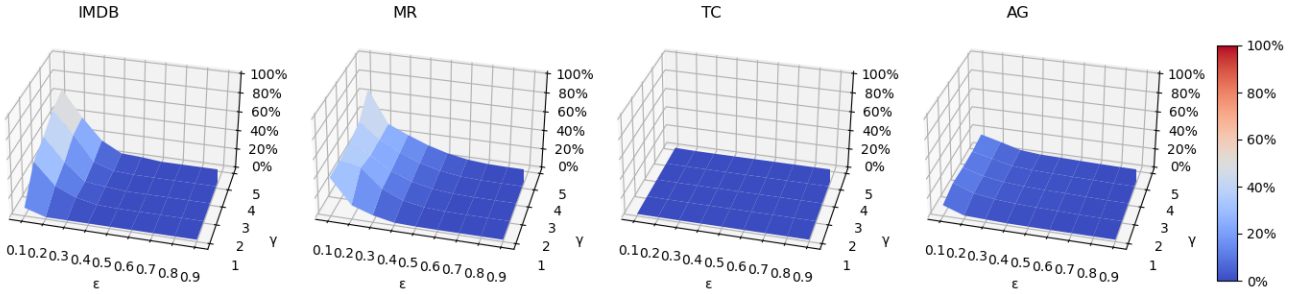


Fig. 15: Detection performance in term of NRR for four datasets using GRU models.

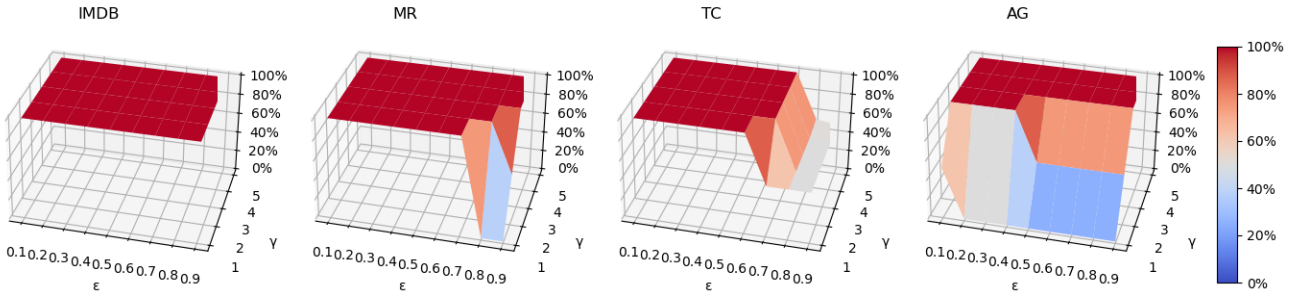


Fig. 16: Detection performance in term of BRR for four datasets using GRU models.

the trigger words can be found in the interpretation results with high rankings. The self-attention-based approach and Anchor cannot sufficiently capture the trigger words in the trained model.

TABLE VII: Running-time overhead of three procedures in InterRNN

Dataset	RNN Type	Model Abstraction	Interpretation Generation		Backdoor Detection	
			Sum.	Avg.	Sum.	Avg.
IMDB	LSTM	2.71h	26.31h	7.15s	26.93h	7.32s
	GRU	2.28h	28.29h	7.69s	27.82h	7.56s
MR	LSTM	0.08h	0.14h	0.21s	2.41h	3.41s
	GRU	0.07h	0.15h	0.22s	2.88h	4.09s
TC	LSTM	0.31h	0.45h	0.31s	6.71h	4.56s
	GRU	0.31h	0.49h	0.34s	5.72h	3.89s
AG	LSTM	0.67h	0.88h	0.28s	12.82h	4.12s
	GRU	0.67h	0.95h	0.31s	14.76h	4.75s

TABLE VIII: Performance of InterRNN for the MR dataset using LSTM models with different hidden layer networks.

Hidden Layers	BRR	NRR	Testing Accuracy	Model Abstraction	Interpretation Generation	Backdoor Detection
2	100%	0.50%	77.54%	0.08h	0.14h	2.41h
3	100%	0.56%	77.25%	0.08h	0.20h	2.68h
4	100%	0.58%	77.29%	0.08h	0.27h	2.94h
5	100%	0.42%	77.21%	0.08h	0.34h	3.22h

Answer to RQ2: InterRNN can effectively detect and remove all the backdoor sentences in the infected models while preserving the classification performance for normal sentences.

D. RQ3: What is the time cost of InterRNN?

To answer RQ3, we evaluate the time cost of InterRNN. InterRNN has three main procedures, i.e., model abstraction, interpretation generation, and backdoor detection. Note that given a specific target label, we only need to generate the interpretations and conduct backdoor detection for the corresponding samples attached to the target label.

The results are presented in Table VII, where Sum. and Avg. are short for the summary and average, respectively. The MR, TC, and AG datasets need less than one hour to finish the model abstraction and interpretation generation. For the backdoor detection procedure, our approach needs about half a day for the three datasets. However, our approach requires more time during the three procedures on the IMDB dataset since the sentences in this dataset are much longer than those in the other three datasets. Note that although the interpretation generation and backdoor detection procedures may take more than one day, they can be conducted on several servers in parallel, thus further reducing the total overhead.

Furthermore, we also evaluate the efficiency of InterRNN using more complex networks. Table VIII lists the performance of InterRNN on the MR dataset using LSTM models with different hidden layer networks. We can observe that their testing accuracies on normal samples are almost the same for all four models. Moreover, our approach can successfully remove all the triggers while the NRR values are lower than 1%. For the time cost, since we only extract the last layer's vector values to perform model abstraction, the time cost of this procedure is the same. However, for the other two procedures, i.e., interpretation generation and backdoor detection, the time cost increases with the increase of the layer numbers. The main reason is that our approach needs more time to conduct the model prediction for more complex models to obtain the change of prediction probability during the two procedures.

Answer to RQ3: InterRNN can efficiently generate interpretation results with a low time cost. The backdoor detection can be generally finished in one day, which can be further reduced with more servers running in parallel.

V. Threats to Validity

A. Threats to Internal Validity

Storage Cost of Model Abstraction: When constructing the NFA-based abstract model, it needs a large space to store and analyze all the extracted vectors if the hidden layer size of an RNN model is much bigger than the parameters used in our experiments. To solve this problem, the PCA technique is a promising way to effectively transform the high dimensional hidden layer vectors to low dimensional vectors to reduce the complexity in the further steps efficiently.

Number of Trigger Words: Our approach detects trigger words by replacing the interpretation words with similar words. However, our approach would fail if using many similar words as the triggers. To this end, we conduct an empirical study about the amount of training data required to design trigger words for the attacker to use similar words.

Specifically, we first construct a set of similar words, denoted as SimItr={'one','two', 'three', ...}. The goal is to measure the amounts of training data we need to use all the similar words as trigger words. Fig. 17 illustrates the required poisoning data rates with a different number of triggers. We can observe that with the increase of the trigger number, the poisoning rate increases. For example, when the trigger number is 5, we need to add about 20% poisoning data into the training data. If the number of similar trigger words is higher than 20, the poisoning rate is higher than 40%, which is a huge amount in reality. Therefore, We think that our approach is resilient to a small number of trigger words. Inserting too much poisoning data into the original training data is hard to implement in reality.

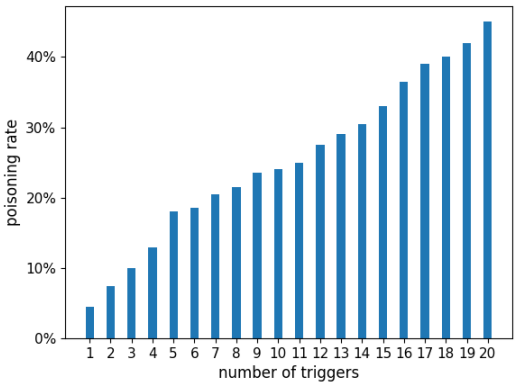


Fig. 17: Required poisoning data rates with a different number of triggers.

B. Threats to External Validity

Detection of Trojan Attack on RNN: Apart from the method of inserting poisoning samples into the model at training time, another method named trojan attack is adopted after the initial model training, e.g., by someone modifying the arguments of several neurons. However, the trojan attack method has only been proved to be effective in CNN models, while in RNN, it has not been investigated so far. We leave the ability of our approach for handling the trojan attacked RNN model as our future work.

Selection of Parameter k Value: The k-means algorithm is used in our model abstraction procedure, and the value of parameter k is set as 20 in the experiments according to our experience. However, if we apply our approach to other datasets containing much more samples than the used four datasets, $k = 20$ would not be sufficient to achieve the best performance. Therefore, combining other clustering algorithms such as DBSCAN and GMM model is a promising way to overcome this limitation.

VI. Related Work

A. RNN Abstraction Analysis

In recent years, several approaches have been proposed to perform the RNN abstraction. They are mainly in the basic form of the finite-state automaton (FSA), which can reveal the internal state transitions in an RNN. Based on FSA, variant models are proposed to conduct different tasks. The main difference among these approaches when performing model abstraction is the step of clustering hidden states. Hou and Zhou [36] extract FSA from RNN by using a variant of k-means named k-means++, which uses D^2 weighting to weight and select cluster centers [37]. Du *et al.* proposed DeepStellar [23], which constructs a discrete-time Markov chains (DTMC) for quality testing of given RNN. It clusters the hidden states into abstract states by constructing a set of regular grids. Weiss *et al.* [38] extract a DFA from any given binary classifier RNN by using the L* learning algorithm [39]. Weiss *et al.* [25] also expands on their previous work by adapting the L* algorithm to extract PFA from RNNs. Similarly, to solve the low accuracy and scalability issue for model abstraction, Dong *et al.* [24] first cluster the hidden states with k-means algorithm and then construct a PFA based on the AAlergia learning algorithm [40]. In this work, we combine causal inference with model abstraction by clustering both the hidden layer vectors and output vectors to enhance the interpretability of the abstract model.

B. Interpretation of Deep Learning System

Complex models trained with deep learning algorithms are essentially black-boxes for all practical purposes. To understand such models, several explainable techniques are proposed to generate interpretations, which are based on input perturbation, back propagation and local approximation.

The input perturbation based approaches [41], [42] follow the philosophy that the contribution of a feature can be determined by measuring how the prediction score changes when the feature is altered. Back propagation based approaches [43], [44] calculate the gradient or its variants, of a particular output with respect to the input using back propagation to derive the contribution of features. Local approximation based approaches [32], [35], [45]–[47] are based on the assumption that the machine learning predictions around the neighborhood of a given input can be approximated by an interpretable white-box model.

C. Backdoor Detection

Existing backdoor detection approaches mainly target on the image classification task with DNN models [13]–[15]. Gu *et al.* [11] conducted the first backdoor attack in a street sign classifier by poisoning images of stop signs with a special sticker into the training set and labeling them as speed limits. As a result, the backdoor model will classify any stop sign as a speed limit simply by placing a sticker on it, causing potential accidents in self-driving cars.

Wang *et al.* [12] proposed Neural Cleanse, which is the first one to assess the DNN vulnerability by reverse engineering the trigger for each class and finding whether there are specific triggers with a significantly small L1 norm. Chen *et al.* [18] proposed an activation clustering based approach, of which the intuition is to look into the activation of the last hidden layer of DNN, which represents how DNNs make decisions. Chen *et al.* [48] proposed DeepInspect, the first black-box backdoor detection solution with minimal prior model knowledge. It learns the probability distribution of potential triggers from the queried model using a conditional generative model, and thus retrieves the footprint of backdoor insertion. Chou *et al.* [49] proposed SentiNet, which uses techniques of model interpretability to find the malicious region that contains the trigger. Guo *et al.* [20] proposed TABOR, which formalizes a backdoor detection task as a non-convex optimization problem, and resolves the optimization through an objective function.

However, all the above backdoor detection approaches focus on the image classification domain, while few work target on

the backdoor detection in RNNs like InterRNN, which needs further investigation.

The most related work on text backdoor detection is proposed by Qi *et al.* [50]. They provided a text backdoor detection approach called Onion, which relies on the change of prediction probability after removing the words on testing sentences. We think that this approach has a main limitation. For the sentence "I like this movie", after removing the word 'like', the prediction probability would change a lot, and the word might be regarded as the trigger word. In our approach, we add the interpretation words and their similar words to a set of other sentences to identify whether they are trigger words, resulting in a more accurate and robust detection result. However, according to the results provided in their paper, they can only decrease about 60% attack success rate while we can achieve 100%.

VII. Conclusion

Backdoor detection was well studied in the CNN model, while few work considers on the RNN model. In this work, we propose a novel backdoor detection approach for RNN-based text classification systems from the interpretation perspective. Specifically, an NFA-based abstract model is constructed to generate interpretation results for each sentence input, based on which we can effectively detect the suspicious triggers. Extensive experiment results demonstrate that our approach can generate better interpretation results compared to state-of-the-art approaches and effectively detect backdoors in RNNs.

Acknowledgment

This research was supported by National Natural Science Foundation of China (61902306, 61632015, 61602369, U1766215, 61772408, 61702414, 61833015), China Postdoctoral Science Foundation(2019TQ0251, 2020M673439), Innovative Research Group of the National Natural Science Foundation of China (61721002), Ministry of Education Innovation Research Team (IRT_17R86), Youth Talent Support Plan of Xi'an Association for Science and Technology (095920201303), the National Research Foundation, Prime Ministers Office, Singapore under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001), NRF Investigatorship NRF-NRFI06-2020-0001.

References

[1] "Text analysis apis," https://www.paralleldots.com/text-analysis-apis, 2020.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009, pp. 248–255.

[3] "Deepspeech," https://github.com/mozilla/DeepSpeech, 2020.

[4] H. Xu, Y. Ma, H.-C. Liu, D. Deb, H. Liu, J.-L. Tang, and A. K. Jain, "Adversarial attacks and defenses in images, graphs and text: A review," *International Journal of Automation and Computing*, vol. 17, no. 2, pp. 151–178, 2020.

[5] X. Zhang, X. Xie, L. Ma, X. Du, Q. Hu, Y. Liu, J. Zhao, and M. Sun, "Towards characterizing adversarial defects of deep learning software from the lens of uncertainty," in *Proc. ICSE*, 2020, pp. 739–751.

[6] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, "Diffchaser: Detecting disagreements for deep neural networks." in *Proc. IJCAI*, 2019, pp. 5772–5778.

[7] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[8] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *Proc. S&P*, 2017, pp. 39–57.

[9] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. EuroS&P*, 2016, pp. 372–387.

[10] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," in *Proc. CVPR*, 2016, pp. 2574–2582.

[11] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019.

[12] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao, "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks," in *Proc. S&P*, 2019, pp. 707–723.

[13] 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 *Proc. ACSAC*, 2019, pp. 113–125.

[14] K. Liu, B. Dolan-Gavitt, and S. Garg, "Fine-pruning: Defending against backdooring attacks on deep neural networks," in *Proc. RAID*, 2018, pp. 273–294.

[15] X. Huang, M. Alzantot, and M. Srivastava, "Neuroninspect: Detecting backdoors in neural networks via output explanations," *arXiv preprint arXiv:1911.07399*, 2019.

[16] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in *Proc. NDSS*, 2018.

[17] Y. Liu, Y. Xie, and A. Srivastava, "Neural trojans," in *Proc. ICCD*, 2017, pp. 45–48.

[18] B. Chen, W. Carvalho, N. Baracaldo, H. Ludwig, B. Edwards, T. Lee, I. Molloy, and B. Srivastava, "Detecting backdoor attacks on deep neural networks by activation clustering," *arXiv preprint arXiv:1811.03728*, 2018.

[19] S. Ma and Y. Liu, "Nic: Detecting adversarial samples with neural network invariant checking," in *Proc. NDSS*, 2019.

[20] W. Guo, L. Wang, X. Xing, M. Du, and D. Song, "Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems," *arXiv preprint arXiv:1908.01763*, 2019.

[21] M. Fan, W. Wei, X. Xie, Y. Liu, X. Guan, and T. Liu, "Can we trust your explanations? sanity checks for interpreters in android malware analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 838–853, 2020.

[22] "Glove: Global vectors for word representation," https://nlp.stanford.edu/projects/glove/, 2019.

[23] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, "Deepstellar: model-based quantitative analysis of stateful deep learning systems," in *Proc. FSE*, 2019, pp. 477–487.

[24] G. Dong, J. Wang, J. Sun, Y. Zhang, X. Wang, T. Dai, and J. S. Dong, "Analyzing recurrent neural network by probabilistic abstraction," *arXiv preprint arXiv:1909.10023*, 2019.

[25] G. Weiss, Y. Goldberg, and E. Yahav, "Learning deterministic weighted automata with queries and counterexamples," in *Proc. NeurIPS*, 2019, pp. 8558–8569.

[26] T. Kanungo, D. M. Mount, N. S. Netanyahu, C. D. Piatko, R. Silverman, and A. Y. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 7, pp. 881–892, 2002.

[27] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning word vectors for sentiment analysis," in *Proc. ACL*, 2011, pp. 142–150.

[28] B. Pang and L. Lee, "Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales," in *Proc. ACL*, 2005, pp. 115–124.

[29] "Toxic comment classification challenge," https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data, 2018.

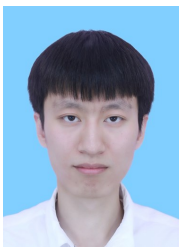
[30] "Ag news classification dataset," https://www.kaggle.com/amananandrai/ag-news-classification-dataset, 2020.

[31] "Keras 2.3.1," https://github.com/keras-team/keras/releases, 2019.

[32] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proc. NeurIPS*, 2017, pp. 4765–4774.

[33] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proc. NAACL*, 2016, pp. 1480–1489.
[34] M. Du, N. Liu, F. Yang, S. Ji, and X. Hu, "On attribution of recurrent neural network predictions via additive decomposition," in *Proc. WWW*, 2019, pp. 383–393.
[35] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in *Proc. AAAI*, 2018, pp. 1527–1535.
[36] B.-J. Hou and Z.-H. Zhou, "Learning with interpretable structure from rnn," *arXiv preprint arXiv:1810.10708*, 2018.
[37] G. BakIr, T. Hofmann, B. Schölkopf, A. J. Smola, and B. Taskar, *Predicting structured data*. MIT press, 2007.
[38] G. Weiss, Y. Goldberg, and E. Yahav, "Extracting automata from recurrent neural networks using queries and counterexamples," in *Proc. ICML*, 2018, pp. 5247–5256.
[39] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and computation*, vol. 75, no. 2, pp. 87–106, 1987.
[40] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen, "Learning probabilistic automata for model checking," in *Proc. ICQES*, 2011, pp. 111–120.
[41] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *Proc. ICCV*, 2017, pp. 3429–3437.
[42] J. Li, W. Monroe, and D. Jurafsky, "Understanding neural networks through representation erasure," *arXiv preprint arXiv:1612.08220*, 2016.
[43] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje, "Not just a black box: Learning important features through propagating activation differences," *arXiv preprint arXiv:1605.01713*, 2016.
[44] D. Smilkov, N. Thorat, B. Kim, F. Viégas, and M. Wattenberg, "Smoothgrad: removing noise by adding noise," *arXiv preprint arXiv:1706.03825*, 2017.
[45] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should i trust you? explaining the predictions of any classifier," in *Proc. KDD*, 2016, pp. 1135–1144.
[46] R. Guidotti, A. Monreale, S. Ruggieri, D. Pedreschi, F. Turini, and F. Giannotti, "Local rule-based explanations of black box decision systems," *arXiv preprint arXiv:1805.10820*, 2018.
[47] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications," in *Proc. CCS*, 2018, pp. 364–379.
[48] H. Chen, C. Fu, J. Zhao, and F. Koushanfar, "Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks," in *Proc. IJCAI*, 2019, pp. 4658–4664.
[49] E. Chou, F. Tramèr, G. Pellegrino, and D. Boneh, "Sentinet: Detecting physical attacks against deep learning systems," *arXiv preprint arXiv:1812.00292*, 2018.
[50] F. Qi, Y. Chen, M. Li, Z. Liu, and M. Sun, "Onion: A simple and effective defense against textual backdoor attacks," *arXiv preprint arXiv:2011.10369*, 2020.

Ming Fan received his B.S. and Ph.D. degrees in computer science and technology from Xi'an Jiaotong University, China, in 2013 and 2019. He also received another Ph.D. degree in computing from The Hong Kong Polytechnic University. He is currently an Associate Professor with the School of Cyber Science and Engineering in Xi'an Jiaotong University, China. His research interests include Android malware analysis and AI security.



Ziliang Si received his B.Eng. degree in automation from Xi'an Jiaotong University in 2019. He is currently a graduate student of the School of Automation Science and Engineering in Xi'an Jiaotong University, China. His research interests include natural language processing and AI security.

Xiaofei Xie received his Ph.D, M.E. and B.E. from Tianjin University. He is currently a research fellow in Nanyang Technological University, Singapore. His research mainly focuses on program analysis, traditional software testing, and quality assurance analysis of artificial intelligence. He has published some top-tier conference/journal papers relevant to software analysis in ICSE, ISSTA, FSE, TSE, IJCAI, ICML, and CCS. In particular, he won two ACM SIGSOFT Distinguished Paper Awards in FSE'16 and ASE'19.

Yang Liu received the Bachelor of Computing degree (Hons.) from the National University of Singapore (NUS) in 2005 and the Ph.D. degree in 2010. He started his post-doctoral work at NUS, MIT, and SUTD. In Fall 2012, he joined Nanyang Technological University (NTU) as a Nanyang Assistant Professor. He is currently a Professor and the Director of the Cybersecurity Lab, NTU. He specializes in software verification, security, and software engineering.

Ting Liu received his B.S. and Ph.D. degrees from Xi'an Jiaotong University, Xi'an, China, in 2003 and 2010, respectively. He was a Visiting Professor at Cornell University. He is currently a professor with the School of Cyber Science and Engineering in Xi'an Jiaotong University, China. His research interests include software security and Smart Grids security.