Visualizing and Interpreting RNN Models in URL-based Phishing Detection

Tao Feng*
vonpower@ynufe.edu.cn
Yunnan University of Finance and Economics
Kunming, Yunnan, China

Chuan Yue†
chuanyue@mines.edu
Colorado School of Mines
Golden, Colorado, USA

ABSTRACT

Existing studies have demonstrated that using traditional machine learning techniques, phishing detection simply based on the features of URLs can be very effective. In this paper, we explore the deep learning approach and build four RNN (Recurrent Neural Network) models that only use lexical features of URLs for detecting phishing attacks. We collect 1.5 million URLs as the dataset and show that our RNN models can achieve a higher than 99% detection accuracy without the need of any expert knowledge to manually identify the features. However, it is well known that RNNs and other deep learning techniques are still largely in black boxes. Understanding the internals of deep learning models is important and highly desirable to the improvement and proper application of the models. Therefore, in this work, we further develop several unique visualization techniques to intensively interpret how RNN models work internally in achieving the outstanding phishing detection performance. Especially, we identify and answer six important research questions, showing that our four RNN models (1) are complementary to each other and can be combined into an ensemble model with even better accuracy, (2) can well capture the relevant features that were manually extracted and used in the traditional machine learning approach for phishing detection, and (3) can help identify useful new features to enhance the accuracy of the traditional machine learning approach. Our techniques and experience in this work could be helpful for researchers to effectively apply deep learning techniques in addressing other real-world security or privacy problems.

CCS CONCEPTS

• **Security and privacy** → **Phishing**; • **Human-centered computing** → *Visual analytics.*

KEYWORDS

phishing detection; deep learning; visualization; interpretation

*Work performed when visiting Colorado School of Mines.
†Corresponding Author.

SACMAT '20, June 10–12, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-7568-9/20/06…$15.00
https://doi.org/10.1145/3381991.3395602

ACM Reference Format:
Tao Feng and Chuan Yue. 2020. Visualizing and Interpreting RNN Models in URL-based Phishing Detection. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies (SACMAT '20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3381991.3395602

1 INTRODUCTION

Phishing attacks are pervasive. They use spoofed websites or webpages to steal users' passwords and sensitive personal information [27, 30]. "Organizations worldwide stand to lose an estimated $9 billion in 2018 to employees clicking on phishing emails" [17].

URLs are the main entrance of almost all the phishing attacks. Effective and timely detection of phishing URLs is critical to the successful protection of Internet users against phishing attacks. At the client-side on a user's web browser, phishing URLs are often detected either based on the blacklists provided by some services such as *Google Safe Browsing* and *Microsoft SmartScreen Filter*, or based on the heuristics directly collected from the URLs and their webpages. At the server-side, phishing detection services may update their phishing URL blacklists based on the analyses from security experts, the confirmations from advanced users, or the heuristics collected from the URLs and corresponding webpages.

In phishing detection, a false positive is likely to deny a user's access to a legitimate website, while a false negative is likely to make the user a victim of the phishing attack. Phishing detection based on blacklists can achieve a zero or close-to-zero false positive rate [28]; however, new phishing URLs may not be added to the blacklists in a timely manner, and many false negatives may occur in practice [26, 28]. In contrast, phishing detection based on heuristics may incur some false positives, but it has the advantages of detecting new phishing URLs in real time.

Fortunately researchers have demonstrated that by using machine learning techniques, phishing detection based on heuristics can also achieve very high accuracy with low false positive rates. In particular, simply based on the heuristics of URLs themselves without the need for analyzing the webpage contents, researchers have shown that the detection accuracy can be as high as 99% (Section 2). Such research efforts often extract and use two types of URL features: *lexical features* that are the textual characteristics such as statistics of words and n-grams of a URL string itself, and *host features* such as the domain, registration, and location characteristics of the URL's hosting server.

In this paper, we explore the deep learning approach and only use lexical features of URLs for detecting phishing attacks; more importantly, we explore to develop several unique visualization

techniques for interpreting the phishing detection performance of our deep learning models. We take the deep learning approach because it works directly on raw data to eliminate the need of tremendous manual feature extraction effort from domain experts. We only use lexical features of URLs because using host features will render our detection depending on third-party services for obtaining some features. Furthermore, because it is well known that deep learning is still largely in black boxes, exploring visualization and interpretation techniques could potentially help us understand the internal operation of deep learning models, further improve them, and properly use them.

We make the following major contributions in this paper:

- We construct a dataset of about 1.5 million phishing and legitimate URLs, and use Recurrent Neural Network (RNN) [5] and bi-directional RNN [19] combined with the two most popular memory cell architectures Long Short-Term Memory (LSTM) [9] and Gated Recurrent Unit (GRU) [3] to build four URL-based phishing detection models which achieve a higher than 99% detection accuracy without the need of any manual feature identification effort from domain experts.
- We develop several unique visualization techniques such as *RNN Temporal Output Diagram*, algorithm for *IP address detection based on RNN states*, and algorithm for *overall impact score calculation for a character* to explore how RNN models work internally in achieving the outstanding phishing detection performance. Especially, we identify (Section 3.2) and answer (Section 5) six important research questions.
- We derive important visualization and interpretation results: (1) while the four RNN models achieve a similar phishing detection accuracy, they do have some obvious differences in their prediction errors – indicating that combining the predictions of the four RNN models into a simple ensemble model may help eliminate most false positives and false negatives; (2) our RNN models can well capture the relevant features including protocol, IP address, special characters, and different parts in URLs that were manually extracted and used in the traditional machine learning approach for phishing detection; (3) our RNN models and visualization techniques can help identify useful new features that were unseen in previous research but can be used to enhance the accuracy of the traditional machine learning such as Random Forest based phishing detection approach.

Our visualization techniques and results could be helpful for researchers to effectively apply deep learning techniques in addressing other real-world security or privacy problems.

2 RELATED WORK

We focus on reviewing the related efforts on heuristics and URL-based phishing detection. They mainly take two approaches: traditional machine learning, and deep learning. They are URL-based because they do not analyze the content of any webpage and do not need any knowledge of the corresponding webpage data. Instead, they often extract and use two types of URL features: *lexical features* that are the textual characteristics such as statistics of words and n-grams of a URL string, and *host features* such as the domain, registration, and location characteristics of the URL's hosting server.

2.1 Traditional Machine Learning Approach

Researchers have shown that phishing detection can be effective by taking the traditional machine learning approach to analyze URL features. Garera et al. applied the Logistic Regression algorithm over 18 handcrafted features to differentiate phishing URLs from legitimate ones [6]. They used lexical features such as special word tokens, and relied on third-party services to extract host features such as the page rank of a URL and its existence (or not) in a white domain table [6]. Their technique achieved an accuracy of 97.3% upon 2,508 URLs (1,245 phishing vs. 1,263 legitimate URLs), showing a high level of accuracy in pure URL-based phishing detection [6].

Ma et al. applied the Naive Bayes, Support Vector Machine, and Logistic Regression algorithms using both lexical and host features on 20,000 to 30,000 URLs obtained from different sources [13]. Their system achieved an overall detection accuracy of 95 – 99%. They found that lexical features of URLs alone are effective in phishing detection with an error rate between 1.9% and 3.5%. Verma et al. derived statistical lexical features of URLs based on metrics such as Kolmogorov-Smirnov Distance and Kullback-Leibler Divergence to perform phishing detection [22]. Using a rule based learning algorithm combined with adaptive boosting techniques, they were able to achieve a 99.3% detection accuracy on one dataset and a 93.17% detection accuracy on a combination of four datasets.

All these research efforts demonstrate that pure URL-based phishing detection can be highly effective, and especially, lexical features of URLs alone can lead to an around 99% accuracy in phishing detection. On the other hand, one of major problems with this traditional machine learning approach is that the detection accuracy heavily depends on the careful extraction of a set of relevant features by researchers; this feature extraction process often requires significant manual effort from domain experts, and needs to be redone as both phishing and legitimate URLs evolve over time.

2.2 Deep Learning Approach

One major advantage of the deep learning approach is that no manual feature extraction is needed. Currently, we are only aware of one related work by Bahnsen et al. [2] on taking the deep learning approach for URL-based phishing detection. Basically, they trained a RNN (esp. a forward direction LSTM network) in their work [2].

Similar to our work, their LSTM model treats each input URL as a sequence of characters, and predicts if a URL is phishing or legitimate. They showed that their LSTM classifier achieves a 98.7% detection accuracy. Different from our work, their LSTM model uses the sparse one-hot encoding scheme, in which each character is translated to a 128-dimension binary vector with all zero values except for the index of the character in the alphabet being marked with one. However, our LSTM and GRU models use a dense 16-dimension encoding scheme based on word2vec [16] (Section 3), thus reducing the model complexity (due to much fewer weights to learn) and potentially improving the model generalization capability. Our RNN models achieve an accuracy at 99.2% or above.

More importantly, Bahnsen et al. [2] did not visualize their model to interpret its internal operation and performance in URL-based phishing detection. In this work, we focus on exploring unique deep learning visualization techniques to bring some light into the black boxes of the RNN models trained purely based on the

lexical information of URLs for phishing detection. This effort is highly desirable to the improvement and proper application of RNN models in phishing detection and other web related important tasks.

3 RNN MODELS AND RESEARCH QUESTIONS

RNNs were introduced by Elman [5] in 1990 to learn the temporal structure of inputs and make predictions based on both the current input and previous context information. Bi-directional RNNs were introduced by Schuster and Paliwal [19] in 1997 to learn the temporal structure of the inputs using two separate networks simultaneously in both the forward and backward time directions.

3.1 Our RNN Models

We use RNN [5] and bi-directional RNN [19] combined with the two most popular memory cell architectures Long Short-Term Memory (LSTM) [9] and Gated Recurrent Unit (GRU) [3] to build four URL-based phishing detection models simply referred to as LSTM, GRU, BiLSTM (bi-directional LSTM), and BiGRU (bi-directional GRU).

Figure 1a illustrates the LSTM and GRU models, while Figure 1b illustrates the BiLSTM and BiGRU models. Each input to a model is a URL with a sequence of characters. Using a dense 16-dimension encoding (or word embedding) scheme based on word2vec [16], each model translates a single URL character c_i to a 16-dimension vector e_i. Each translated or encoded URL is fed into the input layer of an LSTM or GRU network as a sequence of vectors at multiple time-steps. The hidden state in the hidden layer of a network will be continuously updated at each time-step based on the current input character and the previous time-step hidden state representation. Finally, in the output layer, the hidden state of the final time-step goes through a *sigmoid* neuron to produce a continuous value between 0 and 1, representing the probability of the input URL being a phishing URL. Each model is trained by using the back-propagation learning algorithm [18] with the cross-entropy loss and dropout at the output layer.

In Figure 1a, the key difference between the LSTM and GRU models is in their memory cell architectures that will be further introduced soon. That is also the case for the key difference between the BiLSTM and BiGRU models in Figure 1b. The key difference between LSTM and BiLSTM (or between GRU and BiGRU) is that the latter will additionally use a backward direction network to take the encoded sequence of URL characters in the reverse direction as the input, and then concatenate the hidden states of the final time-steps of both the forward and backward direction networks for the sigmoid neuron to produce the phishing prediction result.

LSTM [9] is a novel RNN architecture together with an efficient gradient-based learning algorithm that effectively addressed the insufficient and decaying error backflow problem in traditional RNNs. In the original LSTM architecture [9], each memory cell includes an important *constant error carousel (CEC)* unit to enforce constant error flow, and accepts inputs from both a multiplicative *input gate* (for deciding if the information in CEC should be kept intact or can be overridden by some inputs) and a multiplicative *output gate* (for deciding if the information in CEC is allowed to affect other memory cells). Later, an improved LSTM architecture further includes a multiplicative *forget gate* to allow a memory cell to reset itself [7]. Figure 2a illustrates a simplified LSTM memory cell and the three gates, where x_t, h_t, s_{t-1}, and s_t, represent the input, hidden state, previous step internal state, and the updated internal state associated with the memory cell, respectively. More details about LSTM and its calculation formulas are in [7, 9].

GRU was motivated by LSTM, but it has a simplified memory cell architecture with only two gates: the *reset gate* allows the memory cell to ignore or drop the previous internal state s_{t-1}, while the *update gate* decides the amount of information to be carried over from the previous to the current internal state [3]. GRU is simpler than LSTM in terms of the computation and implementation, yet is still effective in capturing both short-term and long-term dependencies in an input sequence.

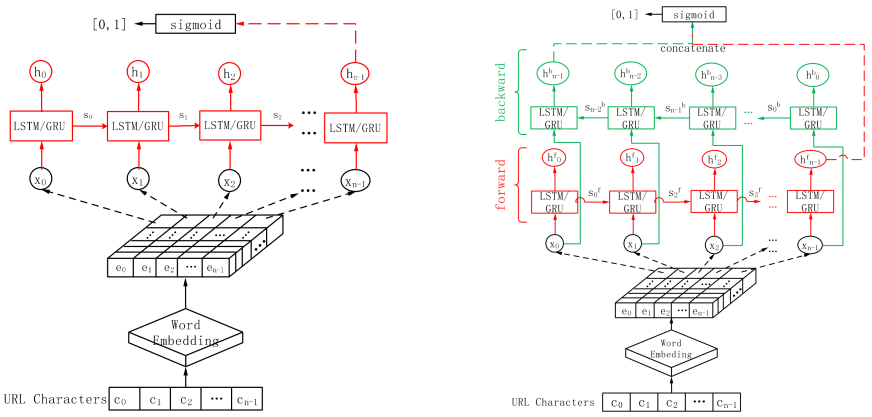
3.2 Research Questions

Each trained RNN model is a classifier for detecting phishing URLs. A URL i is encoded and represented as $E_i = \{e_{i0}, \cdots, e_{ij}, \cdots, e_{in-1}\}$ which is the input to a classifier, while the output of a classifier is the probability score of the input URL being a phishing URL, represented as $S(E_i)$. An input URL is detected as a phishing URL (indicated as 1) if the $S(E_i)$ score is greater than 0.5; otherwise, it is detected as a legitimate URL (indicated as 0).

In the next section, we show that all the four RNN models achieved outstanding phishing detection accuracy. However, the focus of this work is more on answering the following six research questions in Section 5:

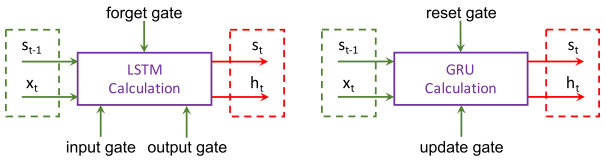
- Are the prediction errors and scores same among RNNs? That is, do the four RNN models predict incorrectly on the same URLs, and are the distributions of their predicted phishing probability scores, i.e., the $S(E_i)$ values, the same?
- Can RNN models capture the protocol information (*https* or *http*) of URLs in phishing detection? In other words, will the https or http protocol string in a URL that is often used in traditional phishing detection influence the decision-making process of RNNs?
- Can RNN models capture the IP address information of URLs in phishing detection? In other words, do some specific memory cells exist to capture the IP address information in a URL that is often used in traditional phishing detection?
- Can RNN models capture some special characters of URLs in phishing detection? In other words, do some special characters such as punctuation symbols that are often used in traditional phishing detection contribute to RNN models' phishing detection decision?
- Can RNN models capture different parts (i.e., protocol, host, path, and query string) of URLs in phishing detection? In other words, are the four parts different in their impact on RNN models' phishing detection decision?
- Can RNN models help identify useful new features for enhancing the accuracy of the traditional machine learning based phishing detection approach?

Answering these questions is important because it will help researchers understand the internal operation and interpret the excellent performance of RNN models in URL-based phishing detection. It will also help researchers further improve and properly apply RNN models for performing phishing detection and other related malicious URL or content detection tasks. In Section 5, we



(a) LSTM and GRU models (b) BiLSTM and BiGRU models

Figure 1: Our four RNN models for URL-based phishing detection.



(a) LSTM (b) GRU

Figure 2: RNN memory cell and gates.

explore deep learning visualization techniques and propose new methods such as "RNN Temporal Output Diagram" and algorithms such as "IP address detection based on RNN states" and "impact score calculation" to answer these important research questions.

Table 1: Five phishing URLs (1 to 5) from PhishTank and five legitimate URLs (6 to 10) from Common Crawl.

https://www.vialbcp.ml/wps/portal/OperacionesEnLinea/
http://bancabci.premios.info/ASD/lniciar-sesion.html
https://sicoopresg.sslblindado.com/index.html
https://radcooldeals.com/hollaatme/office365/
http://yam-fumi-serv.com/signin/
https://investir.lesechos.fr/marches/indices/
http://www.asiaartistawards.com/vote/results/
http://asean.gobizkorea.com/service/service.jsp
http://www.efficientsoftware.net/blog/archives/date/2012/03/
http://gnsystems.net/2017/10/podrostki-izbili-shkolnicu-v-chite/

4 DATASET AND DETECTION ACCURACY

4.1 Dataset

In this work, we constructed a dataset of about 1.5 million URLs with 51% of them as legitimate and 49% of them as phishing. The legitimate URLs came from the Common Crawl (www.commoncrawl.org) open web searching database, while the phishing URLs came from the popular PhishTank (www.phishtank.com) phishing website repository. In more details, we followed the top Alexa (alexa.com) domain list to select the corresponding URLs from the Common Crawl database until we obtained a total of 800,000 unique URLs; because these URLs were crawled from 5,341 of the top 5,642 Alexa domains (301 domains do not have URLs in the database), it is reasonable to assume that they are all legitimate URLs. Manual inspection of samples of them confirmed our assumption, and other researchers such as Bahnsen et al. also considered the URLs in the Common Crawl database as legitimate [2]. Note that experimental results would be slightly different if uncommon URLs are used in the training set as the legitimate URLs. We obtained from the PhishTank repository 26,711 confirmed phishing URLs that were online on 01/09/2018 and 732,774 confirmed phishing URLs that were offline at that time for a total of 759,485 unique phishing URLs.

Table 1 exemplifies five legitimate URLs and five phishing URLs in our dataset. We can see that legitimate and phishing URLs are often very similar as expected by attackers. Table 2 provides the statistics of our dataset. The most common TLDs (top-level domains) are .com and .net in our dataset. TLDs can be categorized into gTLDs (generic TLDs) that are maintained by the Internet Assigned Numbers Authority (IANA) for use in the Domain Name Systems of the Internet, and ccTLDs (country code TLDs) that are usually reserved for specific geographic locations.

4.2 Training and Evaluating RNN Models

We divided the 1.5 million URLs into the training set, validation set, and test set in a ratio of 8:1:1. The training set and validation set are used in training and fine-tuning the models, while the test set is completely separated for evaluating the final models. More specifically, the test set contains 75,775 phishing URLs and 80,162 legitimate URLs. We set the learning rate at 0.0001, and trained the four RNN models on a computer with Tesla K80 GPU, 12 GB Memory, Intel© Xeon© CPU E5-2686 v4 @2.30GHz, 60 GB internal storage, and Ubuntu Linux 4.4.0. Table 3 lists the number of memory cell units and trainable parameters used in the four models. Our LSTM and GRU each has 128 memory cell units. GRU has a simpler internal structure than LSTM as introduced in Section 3.1, thus it has fewer parameters to train than LSTM; meanwhile, BiLSTM and BiGRU double the number of cell units of LSTM and GRU, respectively, because of the use of both forward and backward direction networks as shown in Figure 1.

We observed that in just 25 epochs, the training accuracy of the four RNNs converges to over 99% with the training loss decreased to less than 0.01. The final RNN models are evaluated on the test

Table 2: The statistics of our dataset.

	PhishTank		CommonCrawl	
Date	2011/12/22 - 2018/1/9		2017/12 [a]	
	count	**percent**	**count**	**percent**
TLDs	576	100%	234	100%
gTLDs	429	74.5%	165	70.5%
ccTLDs	147	25.5%	69	29.5%
Unique Domains	237,999	100%	5,341	100%
. com	110,885	46.6%	2,984	55.9%
other gTLDs	79,346	33.3%	1,832	34.3%
ccTLDs	38,403	16.1%	525	9.8%
IP address	9,347	3.9%	0	0.0%
Domains_others[b]	18	0.008%	0	0.0%
URLs	759,361	100%	800,000	100%
.com	358,891	47.3%	540,375	67.6%
other gTLDs	263,579	34.7%	203,290	25.4%
ccTLDs	113,121	14.9%	56,335	7.0%
IP address	23,749	3.1%	0	0.0%
URL_others	22	0.003%	0	0.0%
https	19,354	2.5%	94,230	36.8%
http	740,007	97.5%	505,770	63.2%

[a]Access URL:http://index.commoncrawl.org/CC-MAIN-2017-51, which is an index database in the 51st week of 2017.
[b]They correspond to the 22 URL_others such as http://0xdaf1be2b/EN/index.php, http://1992099439:8002/wellsfargo/, and http://XN—-OTBMJQ2A.XN–P1AI/https:/www2.santander.com.br/recuperar/to_r/.

Table 3: Number of cell units and trainable parameters.

Model	# of Units	# of Parameters
LSTM	128	75,969
GRU	128	57,409
BiLSTM	128+128	150,337
BiGRU	128+128	113,217

set based on the standard metrics including *Accuracy*, *Precision*, *Recall*, and F_1-score as shown in the Formula group 1, where a phishing URL is a positive instance while a legitimate URL is a negative instance. Table 4 shows that all the four RNN models achieved outstanding phishing detection performance with 99.2% or above on both *Accuracy* and F_1-score (which is the harmonic mean of *Precision* and *Recall*). This result is better than the 98.7% detection accuracy reported in [2] (their dataset and model are not available for us to compare); moreover, the focus of our paper is on visualizing and interpreting this outstanding result.

$$\begin{aligned} Accuracy &= \frac{TP+TN}{TP+TN+FP+FN} \\ Precision &= \frac{TP}{TP+FP} \\ Recall &= \frac{TP}{TP+FN} \\ F_1-score &= 2\times\frac{Precision\times Recall}{Precision+Recall} \end{aligned} \tag{1}$$

5 IN-DEPTH ANALYSIS OF RNN MODELS

We develop several unique visualization techniques to interpret the operation of our RNN models and answer the six research questions identified in Section 3.2.

Table 4: Testing performance of the four RNN models.

Model	Accuracy	Precision	Recall	F_1-score
LSTM	99.2%	99.2%	99.1%	99.2%
GRU	99.2%	99.7%	98.6%	99.2%
BiLSTM	99.4%	99.4%	99.3%	99.3%
BiGRU	99.5%	99.4%	99.5%	99.4%

5.1 Are the Prediction Errors and Scores Same among RNNs?

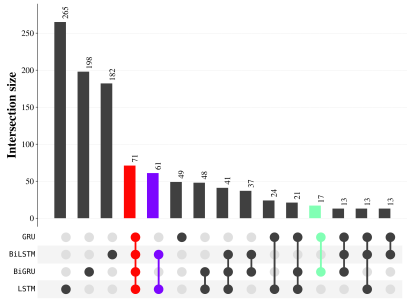
While Table 4 shows that all the four RNN models achieved outstanding and comparable phishing detection performance upon the test set, it will be interesting to know if they have the same prediction errors (i.e., if they predict incorrectly on the same URLs), and if the distributions of their predicted phishing probability scores (i.e., the $S(E_i)$ values described in Section 3.2) are the same.

Upon the test set with 75,775 phishing URLs and 80,162 legitimate URLs, the four RNN models LSTM, GRU, BiLSTM, and BiGRU produced 544, 221, 431, and 446 false positives, respectively, and the union of them results in a set of 1,053 false positive URLs; meanwhile, the four RNN models produced 697, 1,129, 557, and 392 false negatives, respectively, and the union of them results in a set of 1,824 false negative URLs. We found that short URLs are more likely leading to false positives (e.g., http://gradeup.co) and false negatives (e.g., http://iphaa.com), perhaps because they do not contain enough characters for RNNs to make correct decisions.

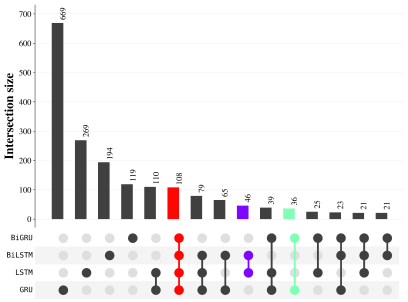
Using the UpSet visualization [11] technique, we illustrate the intersections of false positive URLs among the four RNN models in Figure 3a, and illustrate the intersections of false negative URLs among the four RNN models in Figure 3b. Overall, each model has some unique prediction errors just by itself. In Figure 3a, the three leftmost and the sixth vertical bars indicate that LSTM, BiGRU, BiLSTM, and GRU each produced 265, 198, 182, and 49 unique false positives, respectively; in Figure 3b, the four leftmost vertical bars indicate that GRU, LSTM, BiLSTM and BiGRU each produced about 669, 269, 194, and 119 unique false negatives, respectively. In Figure 3a, the fourth vertical bar indicates that the four models share 71 common false positives; in Figure 3b, the sixth vertical bar indicates that the four models share 108 common false negatives.

These and other details in the two figures demonstrate that the four RNN models do have some obvious differences in their prediction errors, indicating that combining the predictions of the four RNN models into a simple ensemble model based on the majority voting policy may help eliminate most (65.9% or 694 out of 1,053) false positives and most (68.6% or 1,251 out of 1,824) false negatives. On the other hand, addressing the rest non-negligible false positives and false negatives that are common to at least two of the four RNN models remains an interesting research problem. Note that we are not certain about the reasons for those differences and whether those four models are indeed very complementary; our suggestion on an ensemble model is empirical based on our experiments.

We further compared the distributions of their predicted phishing probability scores using the Wilcoxon signed-rank test, which allows us to test if the $S(E_i)$ scores for the same URL from two different RNN models were selected from populations having the same distribution. Upon the test set with 155,937 URLs, each RNN



(a) Intersections of false positive URLs



(b) Intersections of false negative URLs

Figure 3: UpSet visualization of false positives and false negatives produced by the four RNN models.

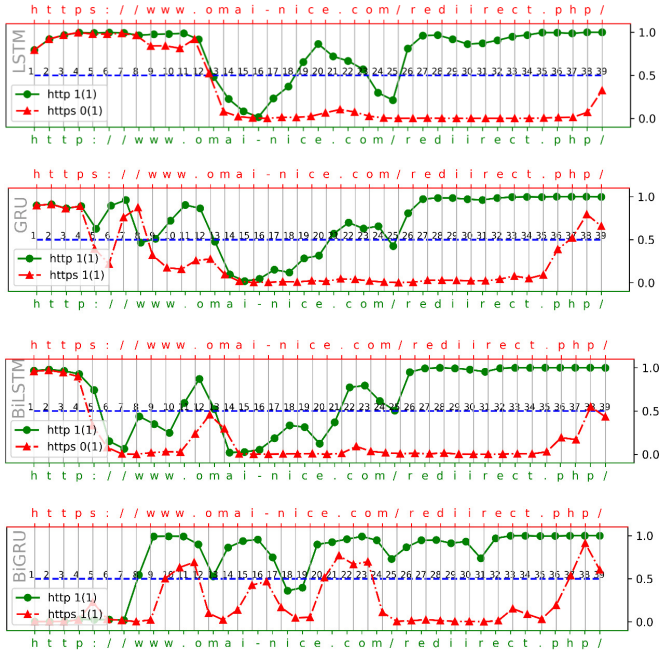


Figure 4: Four RTODs corresponding to the four RNN models. In each RTOD, the original phishing URL is shown in green as the lower sequence along the x-axis, while the modified https URL is shown in red as the upper sequence along the x-axis.

model produced a population of 155,937 $S(E_i)$ scores. The four RNN models constitute six unique pairs, and a Wilcoxon signed-rank test is performed for each unique pair of RNN models for their populations of $S(E_i)$ scores following the same sequence of URLs (i.e., the scores from the two models are paired for each URL). All the six tests rejected the null hypothesis about the same distribution; therefore, the difference between the $S(E_i)$ scores does not follow a symmetric distribution around zero for each pair of RNN models, and the four RNN models differ from each other from the perspective of their predicted phishing probability score distributions.

5.2 Can RNNs Capture HTTP(s) in URLs?

The *https* or *http* protocol information is often used as an important feature in URL-based phishing detection [1, 4] because most phishing webpages are not served over https, which is also the case in our dataset (Table 2). It will be interesting to know if RNN models can capture the protocol information of URLs in phishing detection, or in other words, if the https or http protocol string will influence the decision-making process of RNNs.

We designed a visualization technique called *RNN Temporal Output Diagram (RTOD)* to help us answer this research question. RTOD is a special type of line chart. Its x-axis tick marks are labeled by three sequences. The upper and lower sequences represent two URLs that differ only in their protocol strings (https vs. http), while the middle sequence represents the time-steps of an RNN. The y-axis of an RTOD represents $S(E_{i[0-t]})$, which is the intermediate phishing probability score predicted by the RNN when the input is the substring $c_{i0}...c_{it}$ of a URL i. Since we have two URLs, two $S(E_{i[0-t]})$ curves are drawn in each RTOD. The middle time-step sequence labeled on the x-axis resides at the value 0.5 along the y-axis; therefore, it can also represent the decision line along the y-axis to divide the upper positive zone (i.e., phishing URL) and the lower negative zone (i.e., legitimate URL). Also note that because an http URL is shorter by one character than its corresponding https URL, we stretched the overall lower sequence of x-axis tick marks for http by one character so that the two $S(E_{i[0-t]})$ curves can be easily compared.

Based on a phishing URL http://www.omai-nice.com/rediirect.php from our test dataset, we changed its protocol string from http to https to exemplify four RTODs corresponding to the four RNN models as shown in Figure 4. There are two interesting observations from this example. On the one hand, the impact of the protocol string change on the $S(E_{i[0-t]})$ score change is not immediate, and it becomes more evident only after about half of the time-steps in all the four RTODs. For instance, in the RTOD for LSTM, the character 's' in https was read at the 5^{th} time-step, but the impact becomes obvious until after the 17^{th} time-step. On the other hand, the protocol string change may or may not lead to a final decision change for different RNN models. In the RTODs for LSTM and BiLSTM, the protocol string change from http to https in the URL eventually leads to the decision change from phishing to legitimate URL as shown at the final time-step. In contrast, in the RTODs for GRU and BiGRU, the same protocol string change eventually does not lead to the decision change from phishing to legitimate URL.

To quantitatively examine the impact of the protocol string change on the four RNN models, we further performed the following two experiments. In the first experiment, we randomly sampled from our test dataset 30,000 (12,298 of them are legitimate while

the rest are phishing) URLs that use the http protocol, and changed the protocol string from http to https. In the second experiment, we randomly sampled from our test dataset 30,000 (27,791 of them are legitimate while the rest are phishing) URLs that use the https protocol, and changed the protocol string from https to http. Due to the imbalance of legitimate and phishing URLs in these two samples especially in the second one, F_1-score is more appropriate than the *Accuracy* metric for representing the overall detection accuracy.

As shown in Table 5, when the protocol string is changed from http to https in the first experiment, the *Recall* of all the four RNN models decreased obviously (LSTM: -10.4%, GRU: -10.3%, BiLSTM: -10.3%, BiGRU: -5.8%), indicating the corresponding obvious increase of false negative decisions. Conversely, when the protocol string is changed from https to http in the second experiment, the *Precision* of all the four RNN models dropped significantly (LSTM: -48.1%, GRU: -34.6%, BiLSTM: -54.3%, BiGRU: -49.4%), indicating the corresponding significant increase of false positive decisions. The F_1-score changes largely reflect the corresponding *Recall* and *Precision* changes in the two experiments, respectively. Overall, these results demonstrate that RNN models can well capture the protocol information in URLs and they consider the use of the https protocol as an important characteristic of legitimate URLs. Note that since more phishing sites started to use https, the influence of the protocol part would be reduced in future models.

5.3 Can RNNs Capture IP Address in URLs?

Phishing URLs sometimes use IP addresses instead of domain names; therefore, the IP address information is often considered as an important feature in phishing detection as in [14, 15, 24, 25, 29]. In our dataset, 3.1% of phishing URLs use IP addresses (while that percentage is zero for legitimate URLs) as shown in Table 2. Recently, Karpathy et al. [10] analyzed the interpretability of RNNs in character-level language modeling, and revealed the existence of memory cells that can identify interpretable high-level patterns such as quotes in English sentences and brackets in C programs. To us, it will be interesting to know if RNN models can capture the IP address information of URLs in phishing detection. In other words, we are interested in knowing if some specific "IP address capturer" memory cells exist in phishing detection.

We first leveraged an LSTM-based visualization tool, LSTMVIS, developed by the researchers at the Harvard University [21] to provide clues to us regarding if a set of memory cells are sensitive to IP addresses in URLs. By sensitive, we mean that the state values of this set of memory cells are greater than a certain activation threshold at the same time when an IP address is seen; correspondingly, this set of cells can be considered as the IP address capturer cells. We then designed Algorithm 1 that can use the state values of a given set of cells at each time-step to determine if an IP address exists in an input URL. This algorithm will allow us to quantitatively evaluate the efficacy of a set of IP address capturer cells.

Algorithm 1 takes four inputs: *url*, *cell_indexes_set*, *threshold*, and *sensitivity_distance*. Input *url* is the URL that will be tested regarding if it contains an IP address. Input *cell_indexes_set* contains a set of indexes of the IP address capturer cells intuitively obtained from the clues of the aforementioned LSTMVIS tool. Input *threshold* represents the activation threshold that we use in this algorithm. Input *sensitivity_distance* represents the minimum length of an IP address substring in a URL.

```
   input  : url, cell_indexes_set, threshold, sensitivity_distance
   output : True if url contains an IP address, False otherwise
1  partial_mc_states ← GetStates(url, cell_indexes_set);
     //let the url go through the trained RNN to get the state
     values of memory cells in cell_indexes_set for all the
     time-steps, with a cell in a row and a time-step in a column.
2  for c ← 1 to column_count of partial_mc_states do
3  |   for r ← 1 to row_count of partial_mc_states do
4  |   |   if partial_mc_states [r][c]>threshold then
   |   |     sensitivity_vector[c]++;
5  |   end
6  end
7  if CountConsecutiveOccurrences(sensitivity_vector,
     |cell_indexes_set|) >=sensitivity_distance then return
     True;
8  else return False;
```

Algorithm 1: IP address detection based on RNN states.

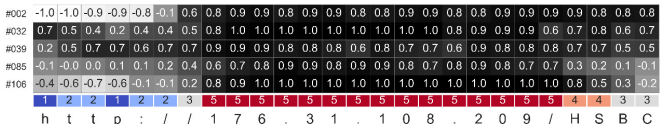
At line 1, the algorithm lets the url go through a trained RNN to get the state values of the memory cells specified by their indexes in *cell_indexes_set* for all the time-steps, producing a matrix *partial_mc_states* with the *row_count* being the total number of cells in *cell_indexes_set* and *column_count* being the total number of time-steps. The state value is extracted from the hidden state h_t as shown in Figure 1. From line 2 to line 6, a *sensitivity_vector* is constructed by counting (for each column in *partial_mc_states*) the number of memory cells (i.e., rows in *partial_mc_states*) that have the state value greater than the threshold value. At line 7, the algorithm will count the consecutive number of time-steps in which the values in *sensitivity_vector* are equal to |*cell_indexes_set*| (i.e., all the memory cells in this set are sensitive at a time-step); if this count is greater than or equal to *sensitivity_distance*, "True" will be returned to indicate the existence of an IP address in the input url. Otherwise, "False" will be returned at line 8.

Figure 5 visualizes two executions of Algorithm 1 using two example URLs with and without an IP address, respectively. Based on LSTM and the clues from the LSTMVIS tool, we chose a set of five IP address capturer cells as *cell_indexes_set*={2, 32, 39, 85, 106}. We empirically chose *threshold* = 0.4, and chose *sensitivity_distance* = 7 because an IPv4 address contains at least 7 characters (e.g., 1.1.1.1). In Figures 5a and 5b, the first five rows contain the state values (ranged between -1.0 and 1.0 due to the use of a hyperbolic tangent function *tanh*) of the five memory cells, respectively, along the time-steps; the sixth row contains the values in the *sensitivity_vector* along the time-steps. Since |*cell_indexes_set*| equals to 5, if value 5 is consecutively identified in *sensitivity_vector* 7 or more times, an IP address is detected in the input url as shown in Figure 5a; otherwise, an IP address is not detected as shown in Figure 5b.

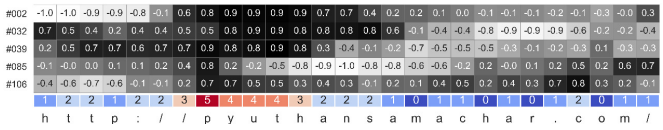
To quantitatively evaluate the performance of this algorithm based on the chosen set of five IP address capturer cells, we randomly selected 10,000 URLs from our test dataset as inputs and executed the algorithm 10,000 times. Overall, the algorithm achieves a very good performance with 99.7% on accuracy, 97.5% on precision,

Table 5: The detection performance change of RNNs due to the http(s) protocol change.

RNN Model	Metric	http -> https			https -> http		
		http	https	change rate	https	http	change rate
LSTM	Accuracy	99.2%	93.4%	-5.8%	99.4%	92.6%	-6.8%
	Precision	99.4%	99.9%	0.5%	98.0%	49.9%	**-48.1%**
	Recall	99.2%	88.8%	**-10.4%**	94.3%	98.0%	3.7%
	F_1-score	99.3%	94.0%	-5.3%	96.1%	66.1%	-30.0%
GRU	Accuracy	99.0%	93.1%	-5.9%	99.5%	95.8%	-2.7%
	Precision	99.7%	99.9%	0.2%	99.5%	64.9%	**-34.6%**
	Recall	98.6%	88.3%	**-10.3%**	93.8%	93.6%	0.2%
	F_1-score	99.1%	93.8%	-5.3%	96.5%	76.7%	-19.8%
BILSTM	Accuracy	99.4%	93.5%	-5.9%	99.6%	91.1%	-8.5%
	Precision	99.5%	99.7%	0.2%	98.7%	45.4%	**-54.3%**
	Recall	99.5%	89.2%	**-10.3%**	95.8%	98.3%	2.5%
	F_1-score	99.5%	94.1%	-5.4%	97.2%	62.1%	-35.1%
BiGRU	Accuracy	99.5%	96.3%	-3.2%	99.5%	91.9%	-7.6%
	Precision	99.4%	99.9%	0.5%	97.1%	47.7%	**-49.4%**
	Recall	99.7%	93.9%	**-5.8%**	96.4%	98.1%	1.7%
	F_1-score	99.5%	96.8%	-2.7%	96.8%	64.2%	-32.6%



(a) The input url contains an IP address



(b) The input url does not contain an IP address

Figure 5: Visualization of two executions of Algorithm 1.

81.8% on recall, and 89.0% on F_1-score. The detailed confusion matrix is shown in Table 6. Because the data is unbalanced and only 1.48% of the 10,000 URLs contain an IP address, the F_1-score is a more meaningful metric for representing the overall performance of this algorithm. The good performance of this algorithm further demonstrates that a certain set of IP address capturer cells indeed exist in RNN models trained for phishing detection. Note that similar good performance is also achieved based on the GRU, BiLSTM, and BiGRU models (but not presented here due to space limitation).

Table 6: Confusion matrix of the evaluation on Algorithm 1.

n=10,000		Algorithm 1 "No IP"	Algorithm 1 "Yes IP"
Actual	"No IP"	9849	3
	"Yes IP"	27	121

5.4 Can RNNs Capture Special Chars in URLs?

Existing studies show that special characters such as '.' (dot) [8, 25] and '@', '-', etc. [22] often appear more frequently in phishing URLs than in legitimate URLs, and can be used as important features in phishing detection [8, 22]. It will be interesting to know if RNN models can capture some special characters of URLs in phishing detection. In other words, we are interested in knowing if some special characters such as punctuation symbols are indeed major contributors in RNN models' phishing detection decision.

As described in Section 3.2, given an input URL, the output of an RNN model is the predicted phishing probability score, i.e., the $S(E_i)$ value. To quantify the contribution of each character in a URL to the final $S(E_i)$ value, we took the saliency score calculation approach that has been used in computer vision [20] as well as natural language processing [12], and designed an impact score calculation algorithm that is suitable for URLs. Basically, the saliency score calculation approach measures the importance of a feature to the final decision score by calculating the first derivative of the latter with respect to the former [12, 20].

Given a URL i with length n, every character c_{ij} in it is encoded as a vector e_{ij} with a fixed dimension d ($d = 16$ as described in Section 3.1). Therefore, we will have the d-by-n embedding matrix $E_i = \{e_{i0}, e_{i1}, \cdots, e_{ij}, \cdots, e_{in-1}\}$ for the URL. A trained RNN model will associate each E_i of a given URL with an output score $S(E_i)$ as described in Section 3.2. Therefore, in our case, the saliency score for the k^{th} dimension of the j^{th} character of a given URL i is defined in Formula 2:

$$\omega(e_{ijk}) = \frac{\partial S(E_i)}{\partial \mathrm{e}}\Big|_{e_{ijk}} \tag{2}$$

where $\omega(e_{ijk})$ is the derivative of $S(E_i)$ with respect to the given embedding e_{ijk}, and it can be efficiently computed using the back-propagation learning algorithm [18]. Note that embedding values are real values in the continuous space and $S(E_i)$ probability values are from 0 to 1, so derivatives exist.

The absolute value $|\omega(e_{ijk})|$ represents the sensitiveness of the final $S(E_i)$ value (and correspondingly the decision) to the change in the k^{th} dimension of the j^{th} character of the URL i, or conversely, the latter's contribution to the former. Adding up $|\omega(e_{ijk})|$ for all the d dimensions, Formula 3 defines the saliency score of the j^{th} character of the URL i:

$$\mathrm{ss_{ij}} = \sum_{\mathrm{k=0}}^{\mathrm{d\text{-}1}} |\omega(e_{ijk})| \tag{3}$$

We select three phishing URLs (with about 50 characters for ease of presentation) from our test dataset to illustrate these saliency score calculations. In Figure 6, we draw the corresponding saliency score heatmap for each URL. There are 17 rows in each URL heatmap. The first 16 rows are the $|\omega(e_{ijk})|$ scores corresponding to the 16

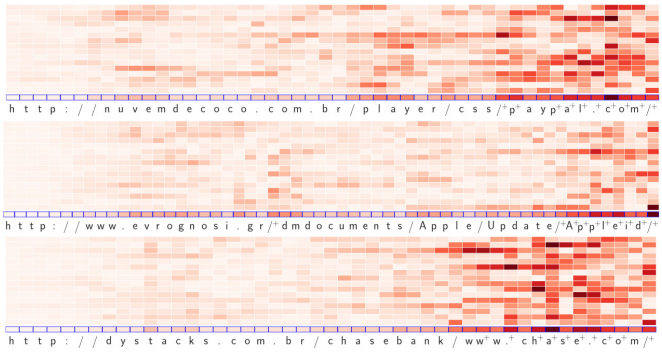


Figure 6: Visualization of the saliency scores for the characters in three URLs.

dimensions. The 17^{th} row is the sum of the first 16 rows, containing the ss_{ij} scores for the characters in each URL. The darker the color of an entry in the heatmap, the larger the saliency score. The 10 characters with the highest saliency scores in each URL are marked with the plus ('+') signs. High saliency scores intensively appear in the "paypal.com" section of the first URL, "Appleid" section of the second URL, and "chase.com" section of the third URL.

So far Formula 3 and the heatmaps in Figure 6 only characterize the importance of each given character at each specific location in an individual URL. We need to further characterize the overall impact or importance of a given character in a trained RNN model. Note that we cannot use the simple sum or mean of the saliency scores of a given character in some set of URLs as its overall impact score. This is because a character may appear multiple times at different locations in each URL, and appear different times in different URLs; that is, its impact or importance is context dependent.

Therefore, we designed the following Algorithm 2 for calculating the overall impact score for a given character. The basic idea of this algorithm is to calculate the ratio of the top occurrences of a given character (i.e., the saliency score of each occurrence is among a certain top percent of scores) to the total occurrences of the character. In other words, instead of directly adding or averaging the saliency scores, we use them to rank characters in a URL and count the top occurrences of a given character.

Algorithm 2 takes three inputs: *urls*, x, and *top_percent*. Input *urls* is an array of URLs. Input x is a given character for which the overall impact score will be calculated from the *urls*. Input *top_percent* specifies which top percent of saliency scores in a URL will be used to count the top occurrences of a given character.

The outer loop at line 1 is for iterating each URL in the array *urls*. In the first inner loop at line 4, the saliency score of the j^{th} character in the current URL urls[i] is calculated using Formula 3 and saved to a vector *saliency_score_vector*. Meanwhile, at line 5, the total occurrences of the given character x in the current URL urls[i] is counted and added to *total_occurrences*. In the second inner loop from line 8 to line 10, each occurrence of the given character x is counted and added to *top_occurrences* if the saliency score of this occurrence is one of the top $lengthof(urls[i]) * top\_percent$ values in the current *saliency_score_vector* for the current URL urls[i]. Finally, at line 12, the ratio of *top_occurrences* to *total_occurrences* is returned as the overall impact score of a given character x in all the URLs in the *urls* array.

input : urls, x, top_percent
output: overall *impact_score* of the character x in all the URLs in the *urls* array

1 **for** $i \leftarrow 0$ **to** $sizeof(urls) - 1$ **do**
2 *saliency_score_vector* = *empty*;
3 **for** $j \leftarrow 0$ **to** $lengthof(urls[i]) - 1$ **do**
4 *saliency_score_vector*$[j] = ss_{ij}$; //calculate the saliency score of the j^{th} character in the current URL urls[i] using Formula 3 and save it to a vector.
5 **if** *urls[i][j] == x* **then** *total_occurrences*++;
6 //add each occurrence of character x in the current URL urls[i] to the total.
7 **end**
8 **for** $j \leftarrow 0$ **to** $lengthof(urls[i]) - 1$ **do**
9 **if** *saliency_score_vector*$[j]$ *is one of the top* $lengthof(urls[i]) * top\_percent$ *values in the current saliency_score_vector* **and** *urls[i][j] == x* **then** *top_occurrences*++;
10 **end**
11 **end**
12 **return** $\frac{top\_occurrences}{total\_occurrences}$;

Algorithm 2: Overall impact score calculation for a given character in an array of URLs.

In our experiments, the *urls* array contains either the 75,775 phishing URLs or the 80,162 legitimate URLs from our test dataset, and the *top_percent* equals 10%. Figure 7 depicts the URL length distributions for the 75,775 phishing URLs and 80,162 legitimate URLs. The average length of the phishing URLs is 78.0; therefore, for the phishing *urls* array, on average about 8 characters (top 10%) with the top saliency scores in each URL satisfy the first condition of the "*if*" statement at line 9 of Algorithm 2.

We ran this algorithm using each of the characters that appeared in our URL dataset as the input character x to derive its overall impact score. When the input *urls* is the array for phishing URLs, the top-10 characters with the highest impact scores in the entire URLs among all the characters for the four RNN models are shown in Figure 8a. When the input *urls* is the array for legitimate URLs, the corresponding information is shown in Figure 8b.

We can see that in all the four RNN models, 80% to 90% of the top-10 characters are punctuation symbols in Figure 8a, while 60% to 80% of the top-10 characters are punctuation symbols in Figure 8b; meanwhile, in each model, high impact score punctuation symbols in phishing URLs are not necessarily the high impact score ones in legitimate URLs, and vice versa. Note that it is the sequence or set of the top-10 characters that matters in each figure. These results intuitively indicate that RNN models can capture some special characters in URLs in the training process, and those high impact score punctuation symbols are indeed important contributors in RNN models' phishing detection decision.

5.5 Can RNNs Capture Different Parts in URLs?

As shown in Figure 9, a URL is composed of four major parts: *protocol*, *host*, *path*, and *query string*. Following up what we analyzed

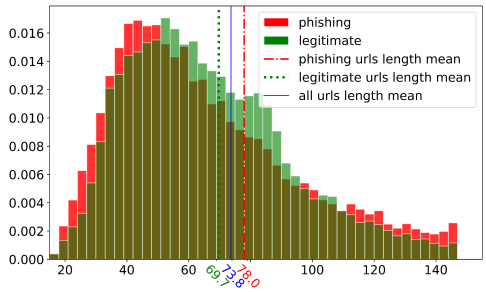


Figure 7: URL length distributions.

in the above subsection, we are further interested in knowing if RNN models can capture different parts in URLs, or in other words, if the four parts in URLs are different in their impact on RNN models' phishing detection decision.

Based on Formula 3 for calculating the saliency score of the j^{th} character of a given URL i, we define the following Formula 4 to calculate the overall impact of a part of URL on RNN models' phishing detection decision:

$$\text{Impact}_{\alpha} = \frac{\sum_{i=0}^{m-1} \sum_{j=B_{i\alpha}}^{E_{i\alpha}} \text{ss}_{ij}}{\sum_{i=0}^{m-1} \sum_{j=0}^{n_i-1} \text{ss}_{ij}} \tag{4}$$

where m is the total number of URLs in a dataset, n_i is the length of the i^{th} URL, α represents a part (i.e., protocol, host, path, or query_string) for which the overall impact will be calculated, $B_{i\alpha}$ is the beginning index of the part α in the i^{th} URL string, $E_{i\alpha}$ is the ending index of the part α in the i^{th} URL string. Basically, this formula calculates a fraction in which the numerator is the sum of the saliency scores of all the characters that appeared in the part α of all the m URLs, while the denominator is the sum of the saliency scores of all the characters in all the m URLs.

Based on the URLs in our test dataset, we calculate the four Impact_{α} scores of the four parts, respectively, for each RNN model. In Figure 10, each of the first four bars illustrates the corresponding Impact_{α} scores for one RNN model; the fifth bar illustrates the total length fractions of the four parts in all the URLs. We can see that from the perspective of the absolute scores, the *path* part has the biggest impact on RNN models' phishing detection decision due to its dominance in the overall length, and it is followed by the query_string part. From the perspective of the relative scores (i.e., further dividing the absolute scores by the length fraction values), the impact of the query_string part takes the first place while the impact of the *path* part takes the second place.

5.6 Can RNNs Identify Useful New Features?

Another interesting research question is whether RNN models can help identify useful new features beyond what are used in traditional machine learning models for phishing detection. It is difficult to answer this question because, to date and in general, the representations learned by deep neural networks in many application areas are far from interpretable yet. We attempt to answer a simplified version of this question, that is, can some observations we obtained in the previous subsections be leveraged to derive some new features for enhancing the accuracy of the traditional machine learning based phishing detection approach?

Many opportunities may exist, and the one that we have examined is to derive some new features by leveraging the observations obtained in the previous two subsections that RNNs can capture special characters and different parts in URLs. In more details, we first revise Algorithm 2 to further calculate the overall impact score for a character within each of the three parts (*host*, *path*, and *query string* without considering the *protocol* part due to its low variance in possible characters) of URLs, in addition to its overall impact in all the parts of URLs (which is calculated by Algorithm 2). The revision is indeed very simple. For example, if we are interested in the overall impact of a given character x in the *host* part of URLs, we simply need to add an extra *"and"* condition "$urls[i][j]$ is in the *host* part of $urls[i]$" to the *if* statements at lines 5 and 9 of Algorithm 2, respectively.

Next, the revised algorithm will be executed for all the four RNN models to derive top-10 high impact score characters (HISC) for each of the three parts, *host*, *path*, and *query string*, respectively. In other words, three pairs of new figures (not presented in the paper due to space limitation) similar to Figures 8a and 8b are generated for the three parts, respectively.

We denote the four sets of top-10 characters for the four RNN models in Figure 8a as $HISC_{phishing\_\alpha\_whole}$ and those four in Figure 8b as $HISC_{legitimate\_\alpha\_whole}$, respectively, where α represents one of the four RNN models. We union the difference between each pair of $HISC_{phishing\_\alpha\_whole}$ and $HISC_{legitimate\_\alpha\_whole}$ for the four RNN models, and calculate $HISC_{whole}$ in Formula 5 to represent the set of characters that are high impact only in phishing URLs in at least one of the four RNN models:

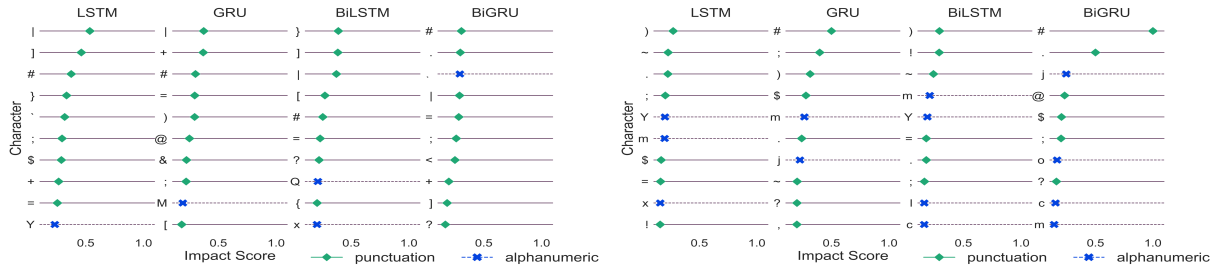
$$\text{HISC}_{whole} = \bigcup_{\alpha} (HISC_{phishing\_\alpha\_whole} - HISC_{legitimate\_\alpha\_whole}) \tag{5}$$

where $\alpha \in \{lstm, gru, bilstm, bigru\}$. We also tried the intersection instead of the union operation on the four sets of top-10 and even top-20 characters, but the final results do not further improve.

We obtain the set $HISC_{whole}$={*[@xQ+]M&=<}#[?|'*} containing 15 elements with the commas separating them omitted for better presentation effect. We then perform the similar union calculations based on the three pairs of new figures corresponding to the three URL parts to obtain the sets $HISC_{host}$={*XznGR%rmNM=DIZc:*}, $HISC_{path}$={*Y{x+]p!=]#[|:h*}, and $HISC_{qs}$={*5)-x+]M=]D#[?|'(h`*}, representing the sets of characters that are high impact only in the host, part, and query string parts, respectively, of phishing URLs in at least one of the four RNN models.

Finally, for each input URL, we use these four sets of high impact score characters to derive four new features: $B1$ that represents the $HISC_{whole}$ characters count in an entire URL, $B2$ that represents the $HISC_{host}$ characters count in the *host* part of the URL, $B3$ that represents the $HISC_{path}$ characters count in the *path* part of the URL, and $B4$ that represents the $HISC_{qs}$ characters count in the *query string* part of the URL.

These are four new lexical features of a URL that are intuitive and interpretable because they are all related to high impact score characters. We now evaluate if they can enhance the accuracy of the traditional machine learning based phishing detection approach.



(a) Based on the 75,775 phishing URLs (b) Based on the 80,162 legitimate URLs

Figure 8: Top-10 high impact score characters in the entire URLs.

Figure 9: Four major parts in a URL.

Particularly, we use the Random Forest algorithm due to its popularity and good performance as a traditional machine learning algorithm for phishing detection to examine if the four new features could be useful. The parameters of the Random Forest algorithm are the default ones (e.g., bagSizePercent:100, batchSize:100, numIterations:100, maxDepth:0 (unlimited), numFeatures:$log_2(n\_features)+1$) implemented in the Weka [23] 3.8.1 version. Table 7 lists 17 lexical features of URLs divided into two groups A and B. Group A includes 13 lexical features used by the researchers in [2] for their Random Forest based phishing detection; note that we did not use the feature "domain exists in Alexa rank" as in [2] because it is a host instead of lexical feature. Group B includes the four new lexical features of URLs that we derived from the RNN models.

Table 7: Traditional lexical features (A1~A13) used in [2] and new lexical features (B1~B4) derived from our RNN models.

A1: '@' and '-' count.
A2: Punctuation count.
A3: Subdomain length.
A4: URL length.
A5: Path part (of a URL) length.
A6: URL entropy.
A7: The ratio between URL length and path length.
A8: The number of TLDs that appear in the path part a URL.
A9: Is IP address used as the host part of a URL.
A10: Suspicious words count.
A11: Euclidean Distance.
A12:Kolmogorov-Smirnov Distance.
A13:Kullback-Leibler Divergence.
B1: $HISC_{whole}$ characters count in an entire URL.
B2: $HISC_{host}$ characters count in the host part of a URL.
B3: $HISC_{path}$ characters count in the path part of a URL.
B4: $HISC_{qs}$ characters count in the query string part of a URL.

We extract these 17 lexical features from our dataset of 1.5 million URLs (with the split of the training set, validation set, and test set in a ratio of 8:1:1), and experiment with four sets (or combinations) of these features as listed in the first column of Table 8. The first set consists of the 13 traditional lexical features $A1$~$A13$ [2] used as the baseline scenario for comparison. The remaining three sets of features include the traditional lexical features except for $A1$ and $A2$ that are related to the basic counts of special characters, and further include the $B1$, $B2$~$B4$, and $B1$~$B4$ new lexical features, respectively, for us to evaluate the usefulness of the new features in three scenarios. In other words, our new features replaced $A1$ and $A2$ in these three scenarios.

Table 8: Evaluation of the four Random Forest classifiers.

Four Feature Sets	Accuracy	value change	Precision	value change	Recall	value change	F_1-score	value change
A1~A13	87.8%	+0.0%	86.5%	+0.0%	89.6%	+0.0%	88.0%	+0.0%
A3~A13, B1	88.4%	+0.6%	87.2%	+0.7%	90.1%	+0.5%	88.6%	+0.6%
A3~A13, B2~B4	89.6%	+1.8%	88.2%	+1.7%	91.4%	+1.8%	89.8%	+1.8%
A3~A13, B1~B4	89.8%	+2.0%	88.5%	+2.0%	91.5%	+1.9%	90.0%	+2.0%

Using each of the four sets of features, we train one Random Forest classifier and evaluate its performance using the Weka machine learning package [23]. Table 8 illustrates the detailed evaluation results of the four classifiers in the four scenarios. Clearly, the four new lexical features can help improve the performance of the traditional Random Forest classifier. Even the single new feature $B1$ ($HISC_{whole}$ characters count in an entire URL) in the second scenario can help improve the F_1-score by 0.6% over the baseline scenario. In the third scenario, the three new features B2~B4 on individual parts of a URL can help improve the F_1-score even more by 1.8% over the baseline scenario. Finally, in the fourth scenario, all the four new features B1~B4 together can help improve the F_1-score the most by 2.0% over the baseline scenario.

Therefore, RNN models can indeed help us easily extract new lexical features to enhance the accuracy of the traditional machine learning such as Random Forest based phishing detection approach. In the environments where traditional machine learning classifiers are preferred for example for efficient re-training or model update purposes, these and potentially other new features can help researchers effectively improve the performance of their classifiers.

6 CONCLUSION

In this paper, we took the deep learning approach and built four RNN models that only use lexical features of URLs for detecting phishing attacks with a higher than 99% accuracy without the need of any manual feature identification effort from domain experts. More importantly, we developed unique visualization techniques such as *RNN Temporal Output Diagram (RTOD)*, algorithm for *IP address detection based on RNN states*, and algorithm for *overall impact*

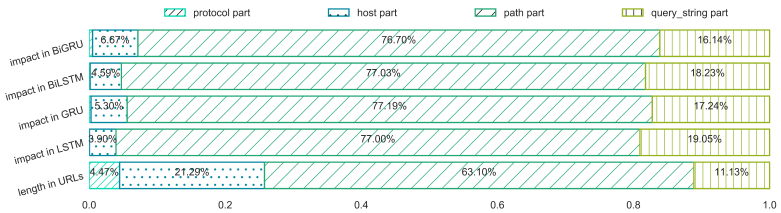


Figure 10: The overall impact of each part of URLs on RNN models' phishing detection decision.

score calculation for a character to explore how our RNN models work internally in achieving outstanding detection performance.

Especially, we identified and answered six important research questions, and derived the following important visualization and interpretation results: (1) while the four RNN models achieve a similar phishing detection accuracy, they do have some obvious differences in their prediction errors – indicating that combining the predictions of the four RNN models into a simple ensemble model can indeed further help eliminate most false positives and false negatives; (2) our RNN models can well capture the relevant features including protocol, IP address, special characters, and different parts in URLs that were manually extracted and used in the traditional machine learning approach for phishing detection; (3) our RNN models and visualization techniques can help identify useful new features that were unseen in previous research but can be used to enhance the accuracy of the traditional machine learning such as Random Forest based phishing detection approach.

Our work is the first step for visualizing and interpreting the performance and internal operation of RNNs in URL-based phishing detection. More complex techniques (e.g., for visualizing and interpreting the co-occurrences of characters) could be developed, and other new features could potentially be derived based on the RNN models to enhance the accuracy of the traditional machine learning based phishing detection solutions. Our techniques and experience could be helpful for researchers to apply deep learning techniques in addressing other real-world security or privacy problems.

ACKNOWLEDGMENTS

The research effort of Tao Feng was supported in part by Yunnan University of Finance and Economics under award number 80059900257. The research effort of Chuan Yue was supported in part by the Army Research Office under Grant Number W911NF-17-1-0447, and by the NSF grant OIA-1936968.

REFERENCES

[1] Sadia Afroz and Rachel Greenstadt. 2011. Phishzoo: Detecting phishing websites by looking at them. In *Proc. of IEEE Conference on Semantic Computing*.
[2] Alejandro Correa Bahnsen, Eduardo Contreras Bohorquez, Sergio Villegas, Javier Vargas, and Fabio A González. 2017. Classifying phishing URLs using recurrent neural networks. In *Proc. of APWG eCrime Symposium*.
[3] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proc. of EMNLP Conference*.
[4] Zheng Dong, Apu Kapadia, Jim Blythe, and L Jean Camp. 2015. Beyond the lock icon: real-time detection of phishing websites using public key certificates. In *Proc. of APWG eCrime Symposium*.
[5] Jeffrey L Elman. 1990. Finding structure in time. *Cognitive science* 14, 2 (1990), 179–211.
[6] Sujata Garera, Niels Provos, Monica Chew, and Aviel D Rubin. 2007. A framework for detection and measurement of phishing attacks. In *Proc. of WORM workshop*.
[7] Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. 2000. Learning to Forget: Continual Prediction with LSTM. *Neural Computation* 12, 10 (2000), 2451–2471.
[8] Binod Gyawali, Thamar Solorio, Manuel Montes-y Gómez, Bradley Wardman, and Gary Warner. 2011. Evaluating a Semisupervised Approach to Phishing Url Identification in a Realistic Scenario. In *Proc. of Annual CEAS Conference*.
[9] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.
[10] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and Understanding Recurrent Networks. *CoRR* abs/1506.02078 (2015).
[11] Alexander Lex, Nils Gehlenborg, Hendrik Strobelt, Romain Vuillemot, and Hanspeter Pfister. 2014. UpSet: visualization of intersecting sets. *IEEE transactions on visualization and computer graphics* 20, 12 (2014), 1983–1992.
[12] Jiwei Li, Xinlei Chen, Eduard H. Hovy, and Dan Jurafsky. 2016. Visualizing and Understanding Neural Models in NLP. In *Proc. of Conference for Computational Linguistics: Human Language Technologies*. 681–691.
[13] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proc. of ACM SIGKDD International Conference*.
[14] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Identifying suspicious URLs: an application of large-scale online learning. In *Proc. of ACM annual international conference on machine learning*. 681–688.
[15] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2011. Learning to detect malicious urls. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2, 3 (2011), 30.
[16] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.
[17] PhishingScams 2018. You've Been Phished! https://www.nist.gov/news-events/news/2018/06/youve-been-phished.
[18] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning representations by back-propagating errors. *Nature* 323 (Oct. 1986), 533–536.
[19] Mike Schuster and Kuldip K Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing* 45, 11 (1997), 2673–2681.
[20] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. 2013. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR* abs/1312.6034 (2013).
[21] Hendrik Strobelt, Sebastian Gehrmann, Bernd Huber, Hanspeter Pfister, and Alexander M. Rush. 2016. Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks. *CoRR* abs/1606.07461 (2016).
[22] Rakesh Verma and Keith Dyer. 2015. On the character of phishing URLs: Accurate and robust statistical learning classifiers. In *Proc. of ACM Conference on Data and Application Security and Privacy*. 111–122.
[23] Weka 2018. Weka 3 - Data Mining with Open Source Machine Learning Software in Java. https://www.cs.waikato.ac.nz/ml/weka/.
[24] Colin Whittaker, Brian Ryner, and Marria Nazif. 2010. Large-Scale Automatic Classification of Phishing Pages. In *Proc. of NDSS*.
[25] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. 2011. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)* 14, 2 (2011), 21.
[26] Chuan Yue. 2012. Preventing the Revealing of Online Passwords to Inappropriate Websites with LoginInspector. In *Proc. of USENIX LISA Conference*.
[27] Chuan Yue. 2013. The Devil is Phishing: Rethinking Web Single Sign-On Systems Security. In *Proc. of USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*.
[28] Chuan Yue and Haining Wang. 2010. BogusBiter: A Transparent Protection Against Phishing Attacks. *ACM Transactions on Internet Technology (TOIT)* 10, 2 (2010), 1–31.
[29] Yue Zhang, Jason I Hong, and Lorrie F Cranor. 2007. Cantina: a content-based approach to detecting phishing web sites. In *Proc. of International WWW Conference*.
[30] Rui Zhao, Samantha John, Stacy Karas, Cara Bussell, Jennifer Roberts, Daniel Six, Brandon Gavett, and Chuan Yue. 2017. Design and Evaluation of the Highly Insidious Extreme Phishing Attacks. *Journal of Computers & Security (COMPSEC), Elsevier* 70 (2017), 634–647.