Detecting Malicious URLs Using a Deep Learning Approach Based on Stacked Denoising Autoencoder

Huaizhi Yan[1,2](✉), Xin Zhang[1,2], Jiangwei Xie[1,2], and Changzhen Hu[1,2]

[1] School of Computer, Beijing Institute of Technology, Beijing 100081, China
yhzhi@bit.edu.cn
[2] Beijing Key Laboratory of Software Security Engineering Technology (Beijing Institute of Technology), Beijing 100081, China

Abstract. As the source of spamming, phishing, malware and many more such attacks, malicious URL is a chronic and complicated problem on the Internet. Machine learning approaches have taken effect and obtained high accuracy in detecting malicious URL. But the tedious process of extracting features from URL and the high dimension of feature vector makes the implementing time consuming. This paper presents a deep learning method using Stacked denoising autoencoders model to learn and detect intrinsic malicious features. We employ an SdA network to analyze URLs and extract features automatically. Then a logistic regression is implemented to detect malicious and benign URLs, which can generate detection models without a manually feature engineering. We have implemented our network model using Keras, a high-level neural networks API with a Tensor-flow backend, an open source deep learning library. 5 datasets were used and 4 other method were compared with our model. In the result, our architecture achieves an accuracy of 98.25% and a micro-averaged F1 score of 0.98, tested on a mixed dataset containing around 2 million samples.

Keywords: Network security · Malicious URL detection · Deep learning Stacked denoising autoencoder

1 Introduction

Typing an URL is the first and foremost step for a user to surf the Internet, and this simple move could lead to a variety of attacks such as drive-by download, watering hole, phishing and malicious exploiting attempts. The URLs used to implement the attacks are so-called malicious URLs.

To solve these notorious attacks, an efficient way is determining whether an URL is malicious and stifle the attack in the cradle. Several researchers have proposed machine learning techniques, which can learn features from URL samples and distinguish them from benign URLs. Including support vector machines [1], decision trees [1, 2], logistic regression [3] and ensembles learning [4]. These machine learning approaches bring well-known advantages: freedom from collecting blacklist and artificially crafting filter rule, more efficiency by introducing automatic detection, and result in higher accuracy rate. Although previous work has a good detection rate, malicious URLs detection based on feature engineering and machine learning still suffered a number of shortcomings: up to 800 billion new websites a year make it harder to extract features, and it could not deal with the higher and higher dimension of feature vector and acquire a acceptable training time, and the shortage of malicious training dataset limit the accuracy. Also, classifier based on feature engineering could be circumvented and evaded [6, 7] through hidden feature created by confusion and encoding [1], one case is using short URL to implement attack [8].

To resolve the problems mentioned above, we make the following contributions in our research:

- Freedom from manually crafting features. Compared with static URL analysis, we introduce the deep learning technique to implement an auto feature extraction module, which can generate detection models without a manually feature engineering. Unlike most existing approaches, the deep learning method can learn high level features by deconstructing and reconstructing URL samples.
- Detecting unknown attacks and fighting against URL evade technology. Since the model did not use artificial features, it breaks the wall of human's fixed cognition to shallow malicious features. The generated model is able to extract implicative correlation between features, which could indicate more intrinsic property of malicious URLs. Therefore, our approach has the capability of identifying new malicious URL sample format that previously unknown.
- Experiment and Evaluation. We have implemented our network model using Keras [10], a high-level neural networks API with a Tensorflow [11] backend, an open source deep learning library. 5 datasets were used and 4 other method were compared with our model. The experiment results in a 98.52% accuracy rate with 1.96% false positive rate.

In a word, our approach offers a qualitative promotion in online malicious URL detection and automate the process of feature extracting, similar with teaching the model to learn features themselves. And we also acquire a good detection rate when meet new samples that never show in test data-set. We present the background in Sect. 2. Then Sect. 3 introduces the main idea of our approach and deep learning framework. Section 4 shows the experiment detail and results. Section 5 reviews a great deal of related work. Finally, Sect. 6 provides the conclusions and discussions.

2 Background

Throughout this paper we focus on how to identify whether an URL is malicious or benign on the Internet. This section provides a background on URL resolution, malicious detection and deep learning, the detection method we use in this paper.

2.1 Uniform Resource Locator Resolution

As people locate a building using a real address on a map, uniform resource locators are the global address of individual resources on the Internet. As specified in the

RFC 1738 [27], URL is a human-readable text string which can be parsed by common browser and translated into multi-step process to locate the server and resource. The following is the syntax format of a standard URL:

<protocol>://<hostname>:<port>/<path>?<query>#<fragment>

2.2 Malicious URL Detection

Malicious URL detection is usually treated as a binary classification problem. Previous approaches to detect malicious URL could be broken down into three categories:

(1) Reactive (Black-list): Services like IE SmartScreen and Google Safe Browsing is using black-list, and they have a low overhead and low false-positive rate. But due to the huge amount of newly generated malicious URLs everyday, maintaining an exhaustive increasing list of malicious URLs is impossible. Also, black-list method can never detect a new sample that appeared before.

(2) Retrospective: Retrospective means classifying URLs offline based on the statistical properties such as lexical features and character-level features to learn a prediction function. Lately, machine learning techniques are applied in proactive methods and gained good effects [7, 12, 18, 19]. However, as mentioned in Sect. 1, these methods relied on feature engineering and training data-set the model used. More related work about classification methods using machine learning is surveyed and presented in Sect. 6.

(3) Proactive and online learning (real-time): proactive method usually analyzes URL in real time and render the web-page to extract more contextual features from html content [9, 20]. To make detection method more scalable and efficient, several researchers employ online learning by updating classify model from sequentially newly learned features in real time [21, 25]. Proactive method need more evaluation time because of the rendering and model updating process.

2.3 Classification of URLs Using Machine Learning

Recently, machine learning techniques have been explored with increasing attention and wildly used in classifying malicious URLs [19]. Ma et al. should be the earlier one who apply machine learning in URL classification [18]. They use a number of statistical methods like naive Bayes, support vector machine (SVM) and logistic regression to classify websites. They use lexical and hosted based information to construct the feature vector.

Pao et al. use SVM and adopt a conditional Kolmogorov measure as a single significant feature for detection [22]. As a compression method for its approximation, Kolmogorov measure can also work with other features and achieve good accuracy and efficiency in predicting million URLs.

Thomas et al. present a real-time spam URL filtering service named Monarch [21]. They design a distributed logistic regression with L1-regularization and use stochastic gradient descent to update weight vector within each single shard. In particular, Monarch has a high scalability to handle a could handle the amount of 15 million URLs per day on Twitter.

For URL confusion detection, Wang et al. analyze the click traffic on twitter and extract short URLs' character level features, then use random tree algorithm to achieve a accuracy of 90.81% in detecting short URL spam on twitter [20].

Mohammad et al. propose an intelligent model using artificial neural network (ANN) to predict phishing attacks [12]. They particularly use self-structure cope with the feature significant changing which can determining the type of webpages. The model can automate the process of structuring the network, and achieve a high prediction accuracy of 92.18% and high acceptance for noisy data and high fault tolerance.

2.4 Deep Learning in Related Context

For other malicious detection, Woodbridge et al. leverage character-level Long Short-Term-Memory (LSTM) networks to classify domains generated by domain generation algorithms (DGA) [13]. Their classifier achieve featureless classification and can perform multiclass classification to specific the malware family. Yu et al. further improve the work of Woodbirge et al. by introducing Convolutional Neural Network (CNN) to classifier and deploy a live stream to detect DGA domains [14]. Wang et al. presents stacked denoising auto-encoders to extract high layer features then use logistic regression to classify malicious javascript code [15, 16]. Their work shows that depending on huge training data, deep models can achieve more accurate results. By sparse random projection, Wang et al. reduce vector dimension from 20,000 features to 480 features. Mahmood et al. research the application of autoencoder in cyber security area [23], they show the AE's capability of automatically learning a reasonable notion of semantic similarity among input features, and evaluate their methods in network based anomaly intrusion detection and Malware classification experiments.

3 Detection Approach

We are now ready to describe the key ideas of our detection approach. We employ an SdA network to analyze URLs and extract features automatically. Then a logistic regression is implemented to detect malicious and benign URLs. The whole process of our method is illustrated in Fig. 1.

In our work, URL samples are preprocessed and converted into vectors. A stacked denoising Autoencoders model is built using dA units and receives training data and extract high level features. There are two training stages in our model: unsupervised pre-training stage and supervised fine-tuning and classification stage. Finally, the classification is performed using logistic regression to detect malicious URLs.

The model has the following advantages:

(1) The model can extract features from preprocessed URL samples directly and require no cumbersome feature engineering;
(2) The model is trained in an unsupervised fashion and can provide high level and more discriminative features in contrast to other feature engineering approaches;
(3) The model is capable of automatically learning a reasonable correlation among automatically extracted features;
(4) Deep models have the potential to extract better representations from the raw data to create much better models.

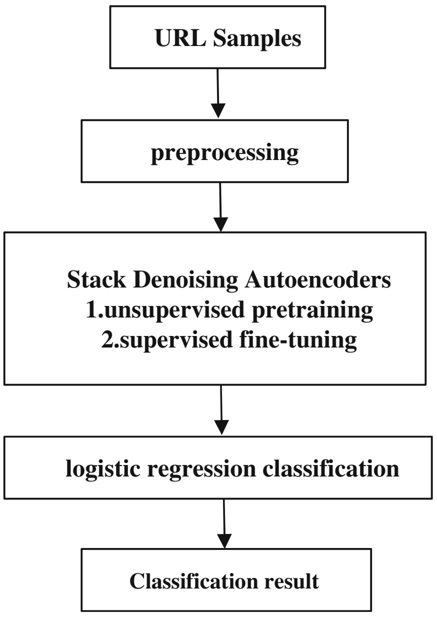
URL Samples
preprocessing
Stack Denoising Autoencoders
1.unsupervised pretraining
2.supervised fine-tuning
logistic regression classification
Classification result

Fig. 1. The whole process of method to classify URL

3.1 Data Preprocessing

As mentioned in Sect. 2.1, many URLs contain URL-Encoded Unicode strings that are not human-readable. And there also exist various meaningless strings. So URL samples cannot be treated as normal text samples and are not suitable for being represented in the form of word vectors, which is usually implemented in many natural language processing models.

Therefore, in data preprocessing stage, we first deal with the URL-Encode encoding and convert Unicode characters into Unicode value. Then we convert other letters into ASCII values. Combined with these two values, a vector represent the URL is generated. To keep all vectors are the same length, a zero padding to the is implemented to extend vector length. An example of data preprocessing is shown in Fig. 2.

3.2 Stacked Denoising Autoencoders

Autoencoders [32] is a feed-forward network, it is unsupervised trained to learn features automatically from unlabeled data and learn a compressed representation of the input.

As Fig. 3 shows, Autoencoders contains input layer, hidden layer and output layer. Input layer firstly deconstruct original feature space vector x into features H using a set of generative weights and activation function f.

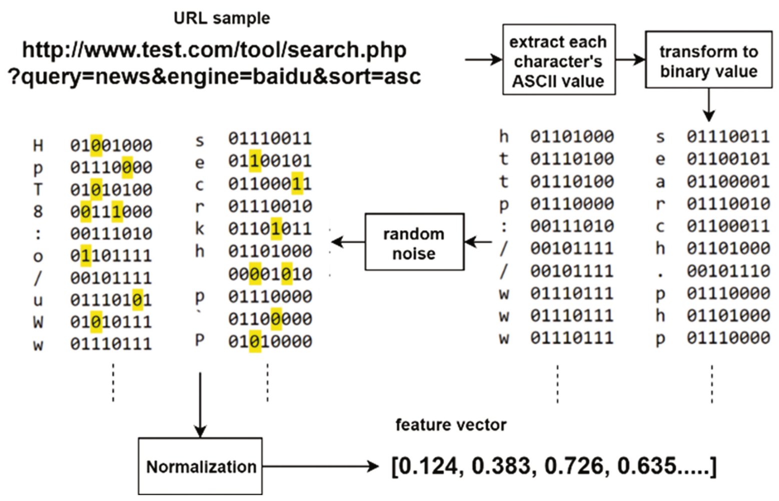
URL sample
http://www.test.com/tool/search.php
?query=news&engine=baidu&sort=asc
extract each character's ASCII value
transform to binary value
random noise
Normalization
feature vector
[0.124, 0.383, 0.726, 0.635.....]

Fig. 2. An example of data preprocessing

$$H = f(Wx + b) \tag{1}$$

Then, in output layer, the features H is reconstructed into $\hat{x}$, an approximate formation of x, using another set of generative weights and same activation function normally

$$W' = WT.$$

$$\hat{x} = f(W'H + b') \tag{2}$$

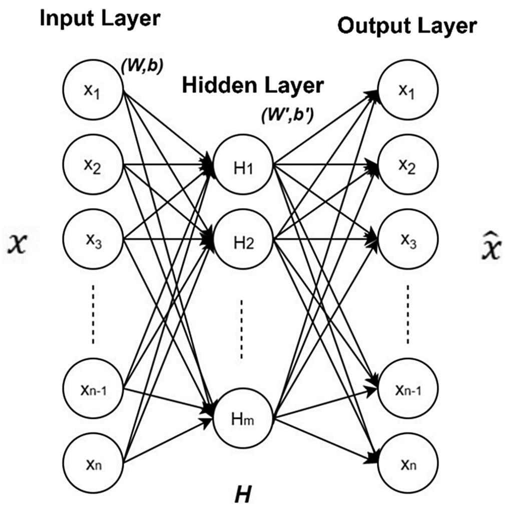

Fig. 3. Model of autoencoders

In training process, back propagation algorithm is used to calculate and adjust weights to minimize the difference between the original input and the reconstructed output, solving the problem formed below:

$$\min_{w,b,w',b'} \| f(W' f(Wx + b) + b') - x \| \tag{3}$$

While since autoencoders are highly non-linear, it is not robust to small input perturbations. Denoising autoencoder is proposed by adding noise in input layer and improve the reconstruction ability of Autoencoders.

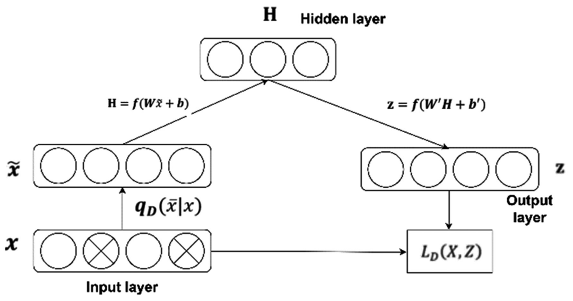

Fig. 4. Structure of dA units.

As Fig. 4 illustrated, x is processed by distribution $q_D(\bar{x}|x)$ and generate $\tilde{x}$, a corrupted version of x. Like Autoencoders described above, denoising autoencoders deconstruct x into H

$$\mathbf{H} = f(W\tilde{x} + b) \tag{4}$$

where $H \in R^h$,then in decoding process reconstruct H into z where $z \in R^d z \in R^d$.

$$\mathbf{z} = f(W'H + b') \tag{5}$$

Denoising autoencoders can be stacked into Stacked denoising Autoencoders as shown in Fig. 5.

In this deep structure, input features will be first encoded layer by layer and then correspondingly be decoded back to the original input. Features will become a better abstract representation after each hierarchical encoder and finally achieve a high level representation of the original features. The noise in each layer introduce better performance and faster training.

Since Autoencoder is suitable in cyber security area [23] for providing more discriminative features and significantly minimizing the dimensionality of features to improve detecting speed, we consider using SdA to extract malicious features from URLs.

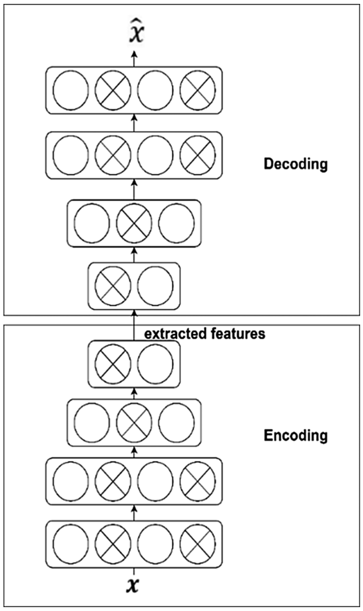

Fig. 5. Structure of stacked denoising autoencoders

3.3 SdA Model Pre-training

In model training process, our target is to form an effective neural network by adjusting parameters and functions in Autoencoders. For each dA unit in encoding and decoding processes, the activation function f is set to be the sigmoid function as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

The parameters of each neural units will be calculated using the first-order derivative of sigmoid function, which takes the form below:

$$f'(x) = f(x)(1 - f(x)) \tag{7}$$

We use Gaussian-noise as the denoising method to corrupt the data. The loss take the form below since the loss function is set to be mean squared error function and mini-batch rule updating strategy is taken for a large-scale dataset.

$$\mathbf{L}(\mathbf{x}, \mathbf{z}) = -\frac{1}{mn}\sum\nolimits_{b=1}^{m}\sum\nolimits_{i=1}^{n}\left(z_{bi} - x_{bi}\right)^2 \tag{8}$$

where n is the input size and m is the mini-batch size. x_{bi} represents the input value of element i in the b-th minibatch. In a similar way, z_{bi} denotes the reconstruction value of element i in the b-th mini-batch.

Finally, the network is trained by backpropagation using a root mean square propagation optimizer for it's efficiently learning in multi-layer neural networks.

3.4 Supervised Fine-Tuning and Classification

After the pre-training process, a supervised training is operated to tune the parameters and a logistic regression classifier is deployed at the end of the network to finally accomplish the classification task.

Softmax function is used as the output activation function to calculate the probability of each class. Then a supervise mini-batch with optimization of binary cross entropy to achieve fine-tuning stage and adjust the entire network's parameters.

The loss function of binary cross entropy takes the form below:

$$\mathbf{L}(\mathbf{x},\mathbf{z}) = -\frac{1}{\boldsymbol{m}}\sum\nolimits_{b=1}^{m}\sum\nolimits_{n=1}^{d}\left(\boldsymbol{x_{bn}}\mathbf{log}(z_{bn}) + (\boldsymbol{l} - \boldsymbol{x_{bn}})\mathbf{log}(1 - z_{bn})\right) \tag{9}$$

Finally, our SdA-LR model is illustrated in Fig. 6.

The SdA part is like a feature extractor to obtain high-level and comprised abstract features and the logistic regression part is a normal classifier.

The layers number of our SdA model is 4 and dA unit number of each layer are 250, 200, 150 and 100. The corruption level is 0.1. The batch size is 128. Pre-training epochs is 50 and fine-tuning epochs is 25.

4 Experiment

4.1 Dataset Description

A data-set of 2 million benign URL samples and 2 million malicious URL samples was constructed from multiple data source. The legitimate URL samples came from the following data source:

(1) 1 million URLs crawled from Alexa top 1 million domain websites. Since the dataset only contains domains, a crawler was used to crawl URLs from the main page of these websites to fetch path and query information.

(https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-).

(2) 1 million URLs came from Common Crawl, a public corpus of raw web page data. (http://commoncrawl.org/).

And the malicious URL samples came from three data sources as follows:

(1) 1 million phishing URLs came from Phishtank, a collaborative community site collecting and tracking phishing URLs. (https://www.phishtank.com/).
(2) 0.5 million malicious URLs came from Anti Network-Virus Alliance of china, a organization aiming at virus prevention and safeguarding the Internet security in China. (https://www.anva.org.cn/index).
(3) 0.5 million malicious URLs came from hpHosts, a community maintained various hosts file including a malware sites lists. (https://www.hosts-file.net/).

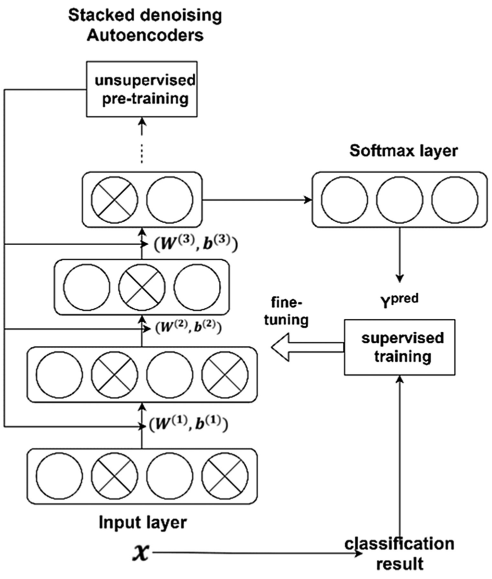

Fig. 6. The SdA-LR model presented in this paper

4.2 Experiment Design

The following experimental design is used to evaluate the proposed technique. A binary classification experiment was designed to test our SdA-LR model's ability of detecting malicious URLs.

To evaluate our featureless SdA-LR model with classifiers based on feature engineering, we compare our model to the following method:

- A native Bayes classifier
- A SVM classifier
- A LSTM network with logistic regression

The first and second model used manually-crafted features of URL proposed in [18]. We also compare our model with a SdA-SVM for strict verification.

Our SdA-LR model is implemented using Keras [10] with Tensorflow [11] backend. And the SVM, random forest and tri-gram model use the implementation of the Scikit-Learn library [31]. All code was deployed and run on a Intel Xeon E5-2686 v4 16 cores computer with a NVIDIA Tesla M60 GPU of 8 GB RAM.

To avoid variance, in training process, this paper used a n-fold cross-validation strategy with 10 folds to run experiment. The data-set was split into 2 parts of a 4:1 partition randomly, which means that four-fifth of the data-set are training and validation set for cross-validation, and one-fifth are testing set.

4.3 Evaluation Metrics

For each experiment, performance evaluation is accomplished with Accuracy, Precision, recall (TP rate), FP rate and F1 Score as defined below:

$$\textbf{Accuracy} = \frac{\boldsymbol{TP} + \boldsymbol{TN}}{\boldsymbol{TP} + \boldsymbol{TN} + \boldsymbol{FP} + \boldsymbol{FN}}$$

$$\textbf{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}$$

$$\boldsymbol{TPrate} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}$$

$$\textbf{FP}\boldsymbol{rate} = \frac{\boldsymbol{FP}}{\boldsymbol{FP} + \boldsymbol{TN}}$$

$$\boldsymbol{F}1\textbf{Score} = 2 \cdot \frac{\boldsymbol{Precision} \cdot \boldsymbol{TPrate}}{\boldsymbol{Precision} + \boldsymbol{TPrate}}$$

In these evaluation metrics, TP represents that a malicious sample is correctly identified as malicious. TN means a malicious sample is incorrectly identified as benign. FN indicates a benign URL is correctly labeled as benign and FP denotes a benign sample is incorrectly labeled as malicious.

Receiver Operating Characteristic (ROC) curve is used to evaluate diagnostic ability of the classifier and illustrates Area under the curve (AUC) statistic. Learning curve is illustrated to show the accuracy on the validation set grows with the number of training epochs.

5 Results

5.1 Performance of SdA-LR Model

Results of the experiments are presented in this section.

We tested the accuracy and of our SdA-LR model using the 4 million URLs described above. A confusion matrix is illustrated in Table 1 to show the performance of our SdA-LR model.

Table 1. Confusion matrix result of the best performance fold in 10-fold experiment

	Predicted as malicious	Predicted as benign	Total
Malicious	392408	3936	396344
Benign	7929	395727	403656
Total	400337	399663	800000

In our experiments, the average accuracy of SdA-LR model stands at 98.52%, with a recall of 99.02% and precision of 98.03%. And among different folds, all accuracy results are consistent, having a standard deviation of just 0.06%, indicating a stable performance of our model.

We also evaluated the AUC of our model. The ROC curves are shown in Fig. 8. In average, the models show an AUC statistic of 99.72%.

In Fig. 7 the learning curve of SdA-LR's training process is shown. It is observed that in just 25 epochs, the validation accuracy converges, increasing to over 98% from epoch 20 onward.

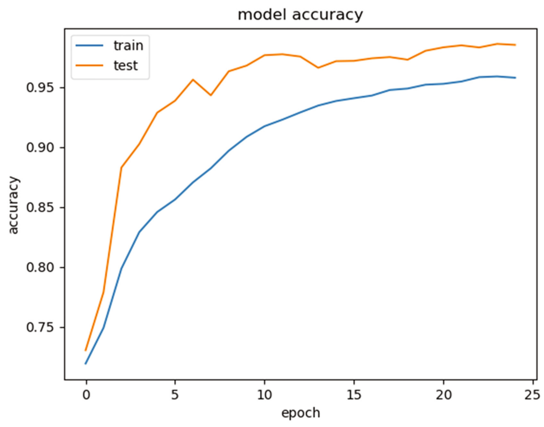

Fig. 7. Learning curve of SdA-LR

5.2 Comparison of the Method

The breakdown of Precision, accuracy, Recall, false positive rate and F1 Score for each model is given in Table 2.

Table 2. Precision, Recall and F1 Score of all models in experiment

Model	Precision	Recall	Accuracy	Fp rate	F1 Score
SdA-LR	0.9803	0.9901	0.9852	0.0198	0.9852
SdA-SVM	0.8416	0.9345	0. 8792	0.1762	0.8856
SVM	0.8359	0.9986	0. 9009	0.1975	0.9099
LSTM	0.9010	0.9960	0.94	0.1093	0.9462
Bayes	0.8271	0.6010	0.7379	0.1255	0.6961

As can be seen, the SdA-LR model significantly outperforms the other models, especially the retrospective SVM model.

We compared the accuracy and F1 Score of all models. In average, the SdA-LR model has an accuracy of 98.52%, much higher than the other deep learning model and feature-engineer model. Similar results were found for the F1 Score and precision.

By comparing the recall rate and false positive rate, we can figure out that our model has the lowest false positive and the third-place recall rate, but the recall rate of 99.01% is higher enough. In comprehensive ways, the SdA-LR model would be the optimal one.

The ROC curves for the SdA-LR, SdA-SVM, LSTM, SVM with manually-crafted features and native Bayes with manually-crafted features are presented in Fig. 8. SdA-LR provides the best performance with an AUC of 0.9972. The second and the third place of AUC are SdA-SVM with 0.9943 and then LSTM with 0.9930.

Considering deep learning method, the deep learning method of SdA-LR, SdA-SVM and LSTM outperform the feature-engineering method in Table 2. Comparing the LSTM and SdA-LR, they both have high recall rate and accuracy. The difference between the LSTM and SVM algorithms may seem small, but in a production system the LSTM could lead more error, since the false positive rate of LSTM is 5 times as much as SdA-LR.

For strict verification, we compared SdA-LR with the SVM and the SdA-SVM models to verify whether and why the SdA-LR is better.

From the results in Table 2, we can see that the classification accuracy of the SdA-SVM exceeds the SVM a little. But in Fig. 8, the AUC of SdA-SVM is much higher then SVM. Consequently, we can draw the conclusion that the SdA helps to improve the accuracy of classification. While SdA-LR outperforms SdA-SVM and SVM, which means that among SdA and SVM, SdA has more advantages in the classification.

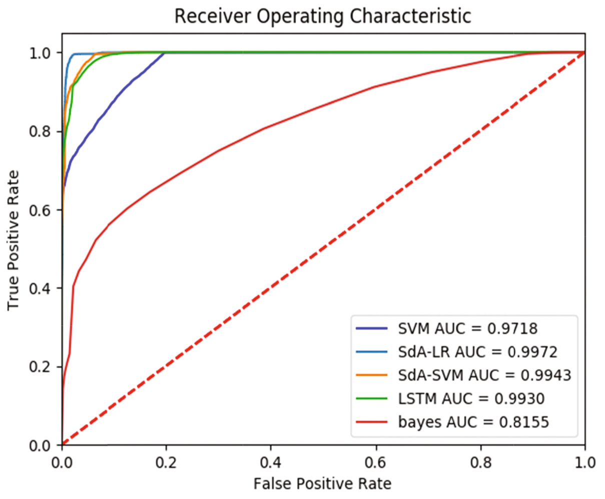

Fig. 8. ROC curve comparison of all experimented method

As Figs. 9 and 10 shows, due to the limitation of SVM on large dataset [34], in the process of fine-tuning, the SdA-LR is much smooth then SdA-SVM.

One thing to notice is that the SVM model perform with the best recall rate, which means that the methods using manually-engineered features are able to detect malicious URL samples with the considered features. But the false positive rate of SVM is high, which in other way indicates that it can't detect new sample with new malicious features.

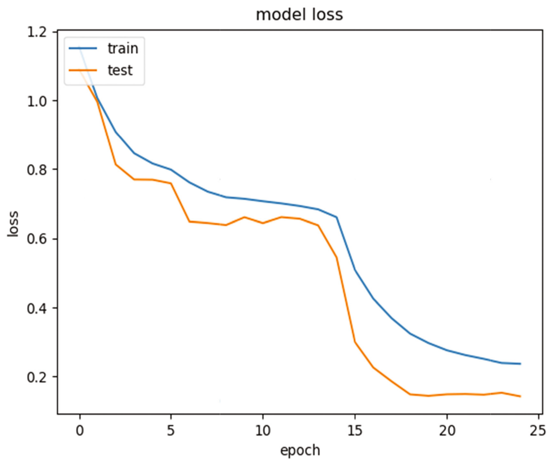

Fig. 9. Loss curve of SdA-SVM

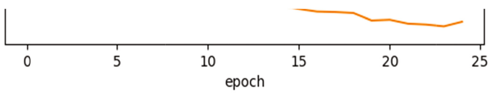

Fig. 10. Loss curve of SdA-LR

Table 3 displays the training time and testing time for all models in experiment. The deep learning model requires significantly more time to train. Even a large dataset was used, the Native Bayes was trained in an average time of less than 10 min. On the other hand, SdA-LR requires 85 min. But the deep model had higher efficient on testing. Comparing the testing time of SVM with 0.27 s per URL, the SdA-LR had a testing time with 0.083 s per URL.

Table 3. Average training-time and testing-time evaluation of each method

Model	Training time (s)	Testing time per URL (s)
SdA-LR	5149	0.083
SdA-SVM	10872	0.52
SVM	7563	0.27
LSTM	5762	0.029
Bayes	563.7	0.18

6 Conclusion and Discussion

We explored how deep features extracted by SdA model can provide a good support for classifier to detect Malicious URLs. Experimental results demonstrated that the performance of SdA model exceed SVM, native bayers and logistic regression based on tri-gram. Our model is able to classify 98.52% URLs with a false positive of 1.96%, and has an F1 Score of 0.98. When compared with other deep learning method such LSTM and SdA-SVM, the SdA has the highest classification accuracy.

In our analysis of the results, we found one limitation of the SdA model is the long training time, but the less training time and high-speed rate of detection makes up for the long period. Another one potential drawbacks is even if our method is featureless and can automatically extract high-level features, SdA model required time for tweaking parameters of neural network like size of inner layers, number of epochs, loss function and so on. And the SVM model shows the method using manually-engineered features were able to detect samples with these features, but it can't detect samples with new malicious features.

In future work there are still potential open problems to be carried for further research. One direction is building a robust prediction system using closed-loop architecture for gathering labeled data and training model. This architecture relies on an efficient real-time online learning design to acquire labeled data and user feedback. To achieve a totally non-artificial label operation, unsupervised learning or reinforcement learning should be considered. The other direction is using cluster analysis algorithm to acquire label tags from unlabeled data to solve the problem of obtaining rare label malicious URL data-set.

Acknowledgments. This work was supported by the National Key R&D Program of China (Grant No. 2016YFB0800700 & No. 2016YFC1000300).

References

1. Ma, J., Saul, L.K., Savage, S., Voelker, G.M.: Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1245–1254. ACM (2009)
2. Canali, D., Cova, M., Vigna, G., Kruegel, C.: Prophiler: a fast filter for the large-scale detection of malicious web pages. In: Proceedings of the 20th International Conference on World Wide Web, pp. 197–206. ACM (2011)

3. Wang, D., Navathe, S.B., Liu, L., Irani, D., Tamersoy, A., Pu, C.: Click traffic analysis of short URL spam on twitter. In: 2013 9th International Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), pp. 250–259. IEEE (2013)
4. Eshete, B., Villafiorita, A., Weldemariam, K.: BINSPECT: holistic analysis and detection of malicious web pages. In: SecureComm, pp. 149–166 (2012)
5. Berners-Lee, T., Masinter, L., McCahill, M.: Uniform resource locators (URL). Technical report (1994)
6. Zhang, H.-L., Zou, W., Han, X.-H.: Drive-by-download mechanisms and defenses. J. Softw. **24**(4), 843–858 (2013). (in Chinese)
7. Sha, H.-Z., Zhou, Z., Liu, Q.-Y., Qin, P.: Light-weight self-learning for URL classification. J. Commun. **35**(9), 32–39 (2014)
8. Klien, F., Strohmaier, M.: Short links under attack: geographical analysis of spam in a URL shortener network. In: Proceedings of the 23rd ACM Conference on Hypertext and Social Media, pp. 83–88. ACM (2012)
9. Seifert, C., Welch, I., Komisarczuk, P.: Identification of malicious web pages with static heuristics. In: 2008 Australasian Telecommunication Networks and Applications Conference, ATNAC 2008, pp. 91–96. IEEE (2008)
10. Chollet, F.: Keras (2015). https://github.com/fchollet/keras
11. Abadi, M., et al.: Tensorflow: large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)
12. Mohammad, R.M., Thabtah, F., McCluskey, L.: Predicting phishing websites based on self-structuring neural network. Neural Comput. Appl. **25**(2), 443–458 (2014)
13. Woodbridge, J., Anderson, H.S., Ahuja, A., Grant, D.: Predicting domain generation algorithms with long short-term memory networks. arXiv preprint arXiv:1611.00791 (2016)
14. Wang, Y., Cai, W.D., Wei, P.C.: A deep learning approach for detecting malicious JavaScript code. Secur. Commun. Netw. **9**(11), 1520–1534 (2016)
15. Bahnsen, A.C., Bohorquez, E.C., Villegas, S., Vargas, J., González, F.A.: Classifying Phishing URLs Using Recurrent Neural Networks (2017)
16. Sha, H.-Z., Liu, Q.-Y., Liu, T.-W.: Survey on malicious webpage detection research. Chin. J. Comput. **39**(3), 529–542 (2016)
17. Sahoo, D., Liu, C., Hoi, S.C.: Malicious URL detection using machine learning: a survey. arXiv preprint arXiv:1701.07179 (2017)
18. Wang, D., Navathe, S.B., Liu, L., Irani, D., Tamersoy, A., Pu, C.: Click traffic analysis of short URL spam on Twitter. In: 2013 9th International Conference Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), pp. 250–259. IEEE (2013)
19. Thomas, K., Grier, C., Ma, J., Paxson, V., Song, D.: Design and evaluation of a real-time URL spam filtering service. In: 2011 IEEE Symposium on Security and Privacy (SP), pp. 447–462. IEEE (2011)
20. Pao, H.K., Chou, Y.L., Lee, Y.J.: Malicious URL detection based on kolmogorov complexity estimation. In: Proceedings of the 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology, vol. 01, pp. 380–387. IEEE Computer Society (2012)
21. Yousefi-Azar, M., Varadharajan, V., Hamey, L., Tupakula, U.: Autoencoder-based feature learning for cyber security applications. In: 2017 International Joint Conference on Neural Networks (IJCNN), pp. 3854–3861. IEEE (2012)
22. Does Alexa have a list of its top-ranked websites? https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites. Accessed 06 Apr 2016

23. Zhao, P., Hoi, S.C.H.: Cost-sensitive online active learning with application to malicious URL detection. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 919–927. ACM (2013)
24. Le Roux, N., Bengio, Y.: Deep belief networks are compact universal approximators. Neural Comput. **22**(8), 2192–2207 (2010)
25. Lipton, Z.C., Berkowitz, J., Elkan, C.: A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019 (2015)
26. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
27. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.A.: Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J. Mach. Learn. Res. **11**, 3371–3408 (2010)
28. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural Comput. **18**(7), 1527–1554 (2006)
29. Hinton, G.: A practical guide to training restricted Boltzmann machines. Momentum **9**(1), 926 (2010)
30. Pedregosa, F., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
31. Vincent, P., Larochelle, H., Bengio, Y., et al.: Extracting and composing robust features with denoising autoencoders. In: International Conference on Machine Learning, pp. 1096–1103. ACM (2008)
32. Menon, A.K.: Large-Scale Support Vector Machines: Algorithms and Theory. Research Exam (2009)