Robust URL Classification With Generative Adversarial Networks

Martino Trevisan, Idilio Drago
Politecnico di Torino
{first.last}@polito.it

ABSTRACT

Classifying URLs is essential for different applications, such as parental control, URL filtering and Ads/tracking protection. Such systems historically identify URLs by means of regular expressions, even if machine learning alternatives have been proposed to overcome the time-consuming maintenance of classification rules. Classical machine learning algorithms, however, require large samples of URLs to train the models, covering the diverse classes of URLs (i.e., a ground truth), which somehow limits the applicability of the approach. We here give a first step towards the use of Generative Adversarial Neural Networks (GANs) to classify URLs. GANs are attractive for this problem for two reasons. First, GANs can produce samples of URLs belonging to specific classes even if exposed to a limited training set, outputting both synthetic traces and a robust discriminator. Second, a GAN can be trained to discriminate a class of URLs without being exposed to all other URLs classes – i.e., GANs are robust even if not exposed to uninteresting URL classes during training. Experiments on real data show that not only the generated synthetic traces are somehow realistic, but also the URL classification is accurate with GANs.

Keywords

Generative Adversarial Networks; Machine Learning; Neural Networks; URL generation

1. INTRODUCTION

URL classification is an important problem with practical applications in different domains. Examples include the filtering of URLs in firewalls and intrusion detection systems, the monitoring of web browsing in parental control systems, the protection against malware or viruses, and the blocking of Ads and/or trackers. Given the URL strings, such systems associate labels to the URLs, e.g., malicious or benign, tracker or non-tracker etc.

Depending on the nature of the application, URL classification can be solved with simple pattern matching. For example, Ad blocking systems normally rely on lists of regular expressions that match all domains serving Ads, which are then blocked in users' browsers. Yet, such lists of regular expressions are long, with hundreds of thousand of expressions, and must be constantly monitored for keeping the pace with new services. Similarly, some other applications require more advanced URL classification schemes, since the URLs to be classified do not follow well-defined structures or are constantly mutated to evade the detection (e.g., as in fast flux attacks).

Machine learning has already been used for URL classification. Previous works about URL classification typically rely on features extracted using domain knowledge. Ma *et al.* [5] use a combination of lexical analysis and host-based features to train online machine learning algorithms, while Kan *et al.* [3] fragment URLs into chunks and use them for building features and training regression models. Zarras *et al.* [10] train classifiers exploiting headers of both malicious and benign HTTP transactions to identify malware. These works build classical two-class models, requiring large samples of both benign/malicious or normal/abnormal classes to train the models.

More recently, Generative Adversarial Neural Networks (GANs) have been proposed for unsupervised learning. Introduced in 2014 [2], GANs are used for image generation [1] and recognition [7] and text-to-photo synthesis [11]. GANs could be used for URL classification too, promising to learn patterns of specific URL classes from limited training sets.

In this paper we propose to use GANs [2] for the classification of URLs. Our system builds a GAN for each class of URLs without the need of observing samples of other classes during training. The final classification is done by evaluating the output of the several trained discriminator models. Our experimental results, using URLs from a type of malware, a CDN delivering video, a firewall and OS updates, show that the GAN-based approach not only provides promising accuracy, but also generates artificial examples that resemble well the real data. As such, our GAN-based approach can be used to produce synthetic traces, reducing the need for real training data for learning models.

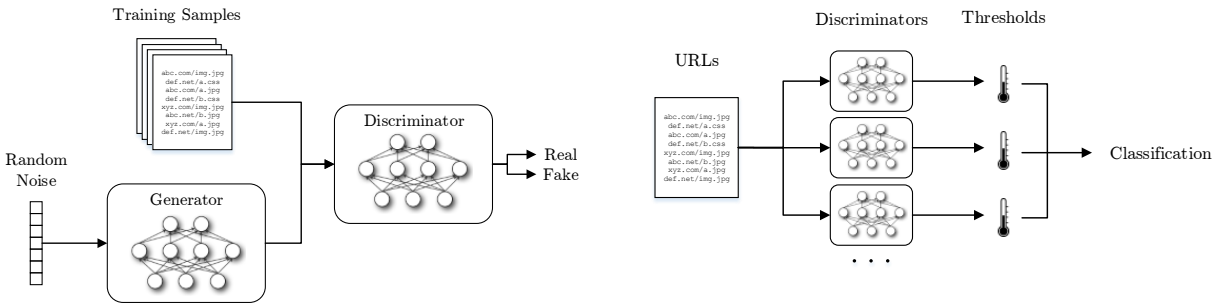
Finally, we release as open source the code for training the models, as it can be of interest for other researchers to reproduce results.[1].

2. SYSTEM ARCHITECTURE

The system builds several GANs, each composed of a *generator* and a *discriminator* model, as depicted in Figure 1a. Provided with random noise and feedback from the discriminator, the generator task is to produce URLs that are similar to the real data, trying to make up artificial samples that cannot be distinguished by the discriminator. The discriminator task instead is to distinguish between real URLs in

Workshop on AI in Networks (WAIN) 2018. Dec. 6, 2018, Toulouse, France

[1] `https://github.com/marty90/URL-generator`



(a) Training: we train a model for each URL class.

(b) Classification: discriminators are used to evaluate new URLs.

Figure 1: System architecture.

the training set and the artificial ones produced by the generator. As the two models compete to win their adversarial tasks, artificial samples become more and more realistic, whereas the discriminator becomes robust to noise.

In a first phase, we train a set of GANs in isolation, one for each class of interest. Then, the trained discriminators are used to pinpoint the target URLs in datasets of unknown URLs. The attractiveness of such an approach is that no samples of other URL classes are needed to train each GAN. Training is done using one class at a time – i.e., we train a generator and a discriminator model for each type of URL. Both the generator and the discriminator consist of simple feedforward multilayer perceptron networks.[2]

The neural network models URLs as tensors as follows. Each character of a URL is encoded in a one-hot vector, with an alphabet including the 26 lower-case letters plus 11 characters allowed in URLs (e.g., `/`, `&` and `=`). The encoded letters compose the input tensor, which has a fixed length of 200 characters. A special value indicates the absence of a character, and is used to handle URLs shorter than 200 characters. Training is done in batches of 256 elements, using an *Adam* optimizer with low learning rate and dropout layers to prevent overfitting [8]. At each step, the generator and the discriminator are trained separately.

The system uses the trained discriminators to classify unknown URLs, as sketched in Figure 1b. Each URL, once encoded in a tensor, is provided as input to the discriminators, and the resulting output scores are evaluated. By design, discriminators' output are in the $[0-1]$ interval. Intuitively, a high value for a specific class suggests that the URL might belong to it. Thus, one could select the classes of a URL based on thresholds.

Fixing thresholds requires ingenuity though. The system calculates thresholds directly from the training data. Separately for each class, the system computes the empirical cumulative distribution function for the scores obtained by the discriminator when run on the training data. At classification time, URLs producing output scores that are likely samples from the distribution obtained during training are classified as belonging the given class. To this end, we apply the well-known boxplot rule to identify confidence intervals [4]. We compute them from the scores obtained at training time, and employ obtained thresholds at classification time. We will show later that values for such thresholds do not play a critical role in classification – i.e., the system is not sensitive to such values (see Section 4.2).

Table 1: Datasets of URL categories.

Name	#URLs	Description
Video	8 620	Video Streaming chunks
Checkpoint	17 451	CheckPoint firewall updates
Windows	5 277	Windows update archives
Tidserv	227	TidServ malware
Others	24 667	Other URLs

3. DATASETS

We rely on a real-world dataset of URLs observed in an operational network. The dataset has been collected by Tstat [9], a passive sniffer that, beside per-flow statistics, registers (anonymized) summaries of the URLs requested in plain-text HTTP requests.

From log files collected during May 2018, we have manually selected sets of URLs related to specific services, which are classified based on our domain knowledge. Details of the datasets are reported in Table 1, while some examples are included in Table 2. The datasets include more than 50 k *unique* URLs. Three classes of URLs refer to benign activity, i.e., OS updates, updates from a specific firewall software, and URLs used by a CDN that streams video. *Tidserv* includes URLs generated by a malware that exploits a fast-flux technique to avoid blacklist-based blocking.[3]. For these four URL classes, we have randomly split the datasets into a training set and a test set before starting the experiments. The former is used for training our GANs, whereas the latter is used for evaluating classification performance.

Others include around 25 k unique URLs picked at random. They represent other types of URLs that are not of interest for the classification. Thus, they are not exposed to the GANs during the training phase, but are mixed with the interesting classes during the performance evaluation. Our

[2]Testing other architectures such as recurrent, convolutional or locally-connected networks is left for future work.

[3]`https://www.symantec.com/security-center/writeup/2008-091809-0911-99`

Table 2: Examples of real (top) and generated (bottom) URLs in our datasets.

Checkpoint
cws.checkpoint.com:80/malware/malware/6.0?resource=ew5rnhlvdS5jb20=&key=123456
cws.checkpoint.com:80/malware/malware/6.0?resource=ew91xyjlchjpb3iuy29t&key=123456
cws.checkpoint.com:80/malware/malware/6.0?resource=1mq4lcktllu41;hpkiquy254bvrbqrz_22
cws.checkpoint.com:80/malware/malware/6.0?resource=mgyu/oiuz25o-z0lr=2yf2=bw1u0ij
Video
hsslive.pcdn.any.sky.it/21920/sport1.isml/qualitylevels(850000)/fragments(video=926861730399413)
hsslive.pcdn.any.sky.it/22362/tg24go.isml/qualitylevels(918000)/fragments(video=2367722273934346)
hsslive.pcdn.any.sky.it/21920/sport1.isml/qualitylevels(69000)0frag7gmjsta(dio.o=a=626ig6030266t)
hsslive.pcdn.any.sky.it/21920/sport1.isml/qualitylevels(140000/frfgmentn(auvid_ot9=6868v7f3090b0))
Windows
download.windowsupdate.com/d/msdownload/update/others/2017/12/25977383_ec7aee1613760fa7e455d31063bc36cbc0f9f669.cab
download.windowsupdate.com/c/msdownload/update/others/2018/01/25995591_4009a2614eae0d1c66a5bc25fd4b70e0e338bd19.cab
download.windowsupdate.com/c/msdownload/update/others/2017/0q/2r?a7497_2f.db16087cb9r1i919516y6z9a7e1h5cvci75.cab
download.windowsupdate.com/d/msdownload/update/others/2017/08/2ay0-440_57f75ua73907afw18fbbndczc681d88e2f6edfi.cab
Tidserv
wuptywcj.cn/6zl0fomx7k5qpas5dmvyptuumczzptam [...] wmjg3jnnpzd0wjmvuzz13d3cuz29vz2xllml0jne9oikmedg2ptmy07k
wuptywcj.cn/hvv1yqbx6y6jfko7dmvyptuumczzptam [...] mwmjg3jnnpzd0wjmvuzz13d3cuz29vz2xllml0jne9zm90byz4ody9mzi=35x
wuptywcj.cnocom/jke17l5j5vs5dmvyptvypcqzmia [...] tcmc2lkzdamjmm9zczlbmcud3d3lvdlb2lszs59zczxpwzwcwbhb2iwb2lng
wwuptywcj.cn.mom/jkb06l8j5v74dmvmpmvyptqumi [...] cmc2nkpd0mjmquzc1lb9c9d3dvz2xvlmlsjn5pzczxpwfwdhbhymiyb2lhbgq

system has to identify only the URLs of the given classes in the test set, marking all other URLs as *Others*.

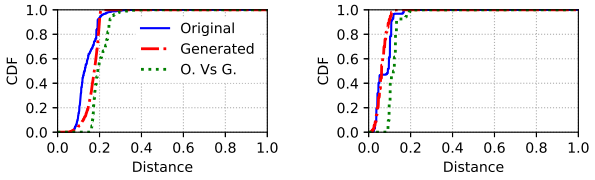
4. RESULTS

We now report results for both the generation and classification steps. Experiments are run on a commodity server equipped with 2 Intel Xeon® CPUs with 20 cores each, and 128 GB of RAM. All experiments lasted less than 45 minutes. Training is 5 000-epochs long, but we notice that after 2 000 epochs on average the GANs start to converge – i.e., generated URLs look like real ones, and the discriminator stops spotting differences among synthetic and real traces.

4.1 Generated URLs

We first analyze the generated URLs, i.e., the output of the generator models trained using the training sets. In Table 2, we provide examples of original (top) and generated (bottom) URLs for all datasets. We can see that our networks correctly catch the common structure of URLs, such as hostname, shared parts of the path etc. Some details of the URLs, however, are not completely understood and still contain semantically wrong information. See, for example, the first generated URL for the *Windows* dataset, where `/2017/0q/` clearly breaks the year/month-based URL schema used by the Windows update service. This may be a consequence of the dropout of neurons used to prevent overfitting, and potentially would be overcome by using larger training sets. As we will show later, the discriminator still successfully identifies URLs despite those glitches.

To gauge more information about the quality of synthetic traces, we compute high-level statistics about the generated URLs. After the training phase, we use the generator to get 500 URLs per class, and analyze how similar the synthetic and original URLs are. We employ a modified version of the Levenshtein distance where the metric is normalized over the length of input URLs [6], with 0 meaning that a couple is exactly the same, and 1 meaning that the couple is completely dissimilar. We compute the Cumulative Distribution Function (CDF) for the similarity between couples of URLs. Separate CDFs are computed for all pairs in the original training sets, only among URLs in the generated sets, and pairs of generated/original URLs. Figure 2 shows the



(a) *Checkpoint* (b) *Video*

Figure 2: Distance of original and generated URLs.

results for *Checkpoint*, *Video*, with similar figures obtained for the other classes. Distances for the generated vs original pairs are slightly larger than distances calculated using pairs of URLs in the original sets – see how green lines are shifted to the right. Yet, the three distributions have similar values and shapes, showing that generated URLs look like the real ones on average.

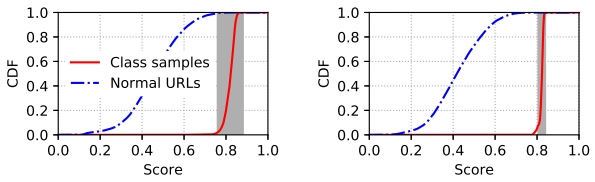
4.2 Classification Performance

We now analyze the classification performance – i.e., the capacity of the system in distinguishing URLs belonging to the target classes from other unknown URLs. Remind that we mix the test sets with the *Others* dataset, and then we give URLs to the trained discriminator models. Their outputs, ranging in the $[0-1]$ interval, are compared to the thresholds derived at training time. A URL is classified as belonging to the class with maximum GAN output, among the GANs outputting numbers larger than the respective thresholds. If no threshold is reached, the URL is marked as *Others*.

We first evaluate the absolute values provided by the discriminator models. Intuitively we want the GANs to provide high scores for URLs belonging to their target classes and low scores for all other URLs. Figure 3 illustrates the scores for two classes. The picture shows that the discriminators have learned how to recognize samples of the given classes, giving them always high scores. Other samples receive indeed low scores. The plots also mark in gray the calculated values for the thresholds of these classes.

Table 3: Confusion matrix and performance indicators.

Dataset		Predicted					Recall	AUC
		Checkpoint	*Video*	*Tidserv*	*Windows*	*Others*		
Real	*Checkpoint*	6,031	0	0	0	67	0.98	0.99
	Video	0	2,944	0	0	74	0.97	0.99
	Tidserv	0	0	59	0	5	0.92	0.97
	Windows	0	5	0	1,829	19	0.98	0.99
	Others	6	0	46	58	24,557	0.93	-
	Precision	0.99	0.99	0.56	0.96	0.99		



(a) *Checkpoint* (b) *Video*

Figure 3: Scores of target URLs versus *Others* URLs. Thresholds derived from training data are marked in gray.

We then analyze the classification performance of the overall system when identifying URLs. We report in Table 3 the confusion matrix over all elements in the test set, along with recall and precision per class. Recall is very high for all classes, meaning that almost all URLs of the studied classes are identified among the *Others* URLs. In terms of precision, we find high values for all classes, but *Tidserv*, where precision is 0.56. Remind that *Tidserv* is a fast-flux attack that produces high variance in the URLs. The samples in the training set (only around 110 URLs) are likely insufficient for allowing the GAN to build a reliable model for the class. Studying such trade-offs is among our future works, and will help to better understand the applicability and limits of the GAN-based approach.

Overall, our preliminary results are very promising, showing large potential for the application of GANs to URL generation and classification.

5. CONCLUSION AND FUTURE WORK

We presented a system that given samples of particular URL classes learns (i) how to generate synthetic samples of the URLs, and (ii) how to identify the URLs among other (uninteresting) URLs. The goals are achieved using Generative Adversarial Neural Networks, which combine a generator and a discriminator model. To the best of our knowledge, this is the first attempt to use GANs in this context.

Our system works without knowledge of every URL category, learning patterns of target URL classes without requiring large samples of other URLs. Results show that the generator is able to catch and learn URL structures, whereas the discriminator presents good classification performance.

This work is our preliminary effort in applying deep learning and generative neural networks for network security. Future work will include not only the application of other network setups for both the generator and discriminator models, but also the extension of the experiments to millions of URLs belonging to hundreds of URL classes.

6. REFERENCES

[1] Denton, E. L., Chintala, S., Fergus, R., et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems* (2015), pp. 1486–1494.

[2] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems* (2014), pp. 2672–2680.

[3] Kan, M.-Y., and Thi, H. O. N. Fast webpage classification using url features. In *Proceedings of the 14th ACM international conference on Information and knowledge management* (2005), ACM, pp. 325–326.

[4] Laurikkala, J., Juhola, M., Kentala, E., Lavrac, N., Miksch, S., and Kavsek, B. Informal identification of outliers in medical data. In *Fifth International Workshop on Intelligent Data Analysis in Medicine and Pharmacology* (2000), vol. 1, pp. 20–24.

[5] Ma, J., Saul, L. K., Savage, S., and Voelker, G. M. Identifying suspicious urls: an application of large-scale online learning. In *Proceedings of the 26th annual international conference on machine learning* (2009), ACM, pp. 681–688.

[6] Morichetta, A., and Mellia, M. Lenta: Longitudinal exploration for network traffic analysis. In *To appear in the 2018 International Teletraffic Congress* (2018), ITC30.

[7] Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[8] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[9] Trevisan, M., Finamore, A., Mellia, M., Munafò, M., and Rossi, D. Traffic analysis with off-the-shelf hardware: Challenges and lessons learned. *IEEE Communications Magazine 55*, 3 (2017), 163–169.

[10] Zarras, A., Papadogiannakis, A., Gawlik, R., and Holz, T. Automated generation of models for fast and precise detection of http-based malware. In *Privacy, Security and Trust (PST), 2014 Twelfth Annual International Conference on* (2014), IEEE, pp. 249–256.

[11] Zhang, H., Xu, T., Li, H., Zhang, S., Huang, X., Wang, X., and Metaxas, D. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint* (2017).