Deep Convolutional Neural Networks for DGA Detection

Carlos Catania ✉[1], Sebastian García[2], and Pablo Torres[3]

[1] LABSIN, Facultad de Ingeniería. UNCuyo. Mendoza. Argentina
`harpo@ingenieria.uncuyo.edu.ar`
[2] CTU - Czech Technical University. Prague. Czech Republic
`sebastian.garcia@agents.fel.cvut.cz`
[3] Universidad de Mendoza, Mendoza, Argentina
`pablo.dtorres@gmail.com`

Abstract. A Domain Generation Algorithm (DGA) is an algorithm to generate domain names in a deterministic but seemly random way. Malware use DGAs to generate the next domain to access the Command Control (C&C) communication server. Given the simplicity of the generation process and speed at which the domains are generated, a fast and accurate detection method is required. Convolutional neural network (CNN) are well known for performing real-time detection in fields like image and video recognition. Therefore, they seemed suitable for DGA detection. The present work provides an analysis and comparison of the detection performance of a CNN for DGA detection. A CNN with a minimal architecture complexity was evaluated on a dataset with 51 DGA malware families and normal domains. Despite its simple architecture, the resulting CNN model correctly detected more than 97% of total DGA domains with a false positive rate close to 0.7%.

Keywords: Deep Neural Networks, Network Security, DGA Detection

1 Introduction

A domain generation algorithm (DGA) is used to dynamically generate a large number of pseudo random domain names and then selecting a small subset of these domains for the Command Control (C&C) communication channel. The idea behind the dynamic nature of DGA was to avoid the inclusion of hard-coded domain names inside malware binaries, complicating the extraction of this information by reverse engineering [8]. The first DGA detection attempts depended on published lists of domains already detected as DGA. However, given the simplicity and velocity associated to the domain generation process, detection approaches that relied on static domain blacklists were rapidly rendered ineffective [6]. Nowadays, DGA detection methods can be classified in two main groups: (A) Context-based or (B) Lexicographical-based . The so-called context-based approaches mostly rely on the use of context information such as the responses from the DNS servers. A typical example would the fact that DGA domains are not usually registered and therefore the DNS response is usually *NXDomain*. On the other hand, lexicographical approaches classify the domains by studying the statistical properties of the characters conforming the domain name.

Regarding Lexicographical approaches, Natural Language Processing (**NLP** emerged as one of the most useful techniques for detecting DGA, specially in the analysis of the n-gram frequency distribution of domain names. An n-gram is defined as a contiguous sequence of n items from a given sequence of text. It is possible to use greater values for n than 1. In the simpler form, when $n = 1$, the single character frequency distribution is generated. The assumption is that DGA domains will have a different n-gram distribution than normal domain names. An example of n-gram based DGA detection are [14, 10] that compares uni-grams and bi-grams distribution using the Kullback-Leibler (K-L) divergence.

Several other DGA detection approaches extended the idea of using the information provided by domain names properties (including n-gram distributions) to train a machine learning classifier such as Random Forest [1] or Linear Regression [14]. Recently, to avoid the need of designing the right set of features for training machine learning classifiers, some authors explored the application of Deep Learning (DL) techniques. In particular the application of Long-Short-Term-Memory (**LSTM**) networks [3]. When applied to the text analysis problem, the internal design of LSTM cell is capable to capture combinations of characters that are important to discriminating DGA domains from non-DGA domains. This flexible architecture generalizes manual feature extraction like n-grams, but instead learns dependencies of one or multiple characters, whether in succession or with arbitrary separation.

Despite the good results reported, LSTM networks have proved to be difficult to train under some particular cases [7]. The aforementioned issue together with the considerable time required during training could be the major obstacle for the massive adoption of LSTM networks in DGA detection. The fact is that DGA techniques change over the time and the periodical retraining of the network becomes mandatory. However, if the presence of long-term dependency patterns in DGA domain names is ignored, it is possible to apply another well-known Deep Learning technique: Convolutional Neural Networks (**CNN**). CNN are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. CNN have been successfully applied in a many practical applications mainly related to image and video recognition. When applied to text analysis the convolution is applied over one dimension and denoted as 1D-CNN. The main advantage of 1D-CNN are they can be trained much faster than LSTM (up to 9X times faster than LSTM) and similarly to LSTM, 1D-CNN are capable of learning representations for groups of characters without being explicitly told about the existence of such groups. However, they can't deal with patterns with arbitrary separation.

In the present work we focus on analyzing the DGA detection performance of a 1D-CNN. Similarly to [13], we are interested in evaluating the performance of a network with a minimal architecture complexity. Hence, the considered network architecture consists of just a minimal extra layers in addition to the

Convolutional Layer. Evaluation is conducted on a dataset containing 51 different DGA families as well as normal domain names from the Alexa corpus and the Bambenek feeds. Two different DGA schemes, including the recent word-based scheme, are included in the dataset. Word-based DGA consists of concatenating a sequence of words from one or more wordlists, resulting in a domain that appears less random and thus may be more difficult to detect [8].

The hypothesis of this work is that despite the known limitations of 1D-CNN, they can learn the common properties from different DGA generation schemes. The evaluation also verifies that the detection performance is within the range required for real-world scenarios.

The main contributions of the present article are:

- An analysis of the advantages and limitations of a simple 1D-CNN learning model for detecting DGA.
- A detailed evaluation of 1D-CNN on a extended dataset that includes domain names from 51 different real malware DGA following different generation schemed as well as normal domains from two different sources.
- A comparison with another well known deep learning technique. In particular, LSTM networks, which have been successfully applied to the DGA problem before [13]. This last set of experiments provides a significant addition to our previous work on the topic [2]

The rest of this paper is organized as follows: Section 2 describes the network architecture, Section 3 details the experiments design and results while Section 4 discusses the importance of the results obtained by 1D-CNN. Section 5 compares the performance of 1D-CNN with a LSTM network with a minimal architecture complexity. Finally, Section 6 presents the conclusions.

2 The Neural Network Architecture

The Neural Network Architecture model used in this paper is a 1D-CNN. This CNN is composed of three main layers. The first one is an *Embedding* layer, then there is a *1D Convolutional* layer, and finally a Dense fully connected layer. The first two layers are the most relevant components of the architecture regarding the problem of detecting DGA domains. Both layers are responsible for learning the feature representation in order to feed the third Dense and fully connected layer. Beside the three layers previously described, the complete Neural Network Architecture includes some other layers for dealing with the dimensions output of the *1D Convolutional* layer as well as layers for representing the input domain and the output probability. A detail of the complete architecture together with the used activation functions is shown in Table 1, whereas the three main layers are described in the following subsections.

2.1 Embedding Layer

A character embedding consists in projecting l-length sequences of input characters into a sequence of vectors R^{lxd}, where l has to be determined from the

Table 1. The complete network architecture of the CNN, including the corresponding output dimensions and activation function used in each layer

Layer (type)	Activation Function
input (Input Layer)	-
embedding (Embedding)	-
conv1d (1D Convolutional)	relu
dense_1 (Dense)	relu
dense_2 (Dense)	sigmoid

information provided by the sequences in the training set and d is a free parameter indicating the dimension of the resulting matrix [13]. By using an *Embedding* layer in the architecture, the neural network learns in an efficient manner the optimal set of features that represent the input data.

2.2 1D Convolutional Layer

The *1D Convolutional* layer refers to a convolutional network layer over one dimension. For the DGA detection problem, such dimension consists of the length of the domain name sequence. The convolutional layer is composed of a set of convolutional filters that are applied to different portions of the domain name. A visual example of the feature extraction process for a *1D Convolutional* layer is shown in Fig. 1. The figure depicts a *1D Convolutional* layer constructing 256 filters (features) ($nf = 256$), with a window (kernel) size of 4 ($ks = 4$) and a stride length value of 1 ($sl = 1$). The layer selects from groups (also referred as patches) of 4 characters to apply the convolutional filters, and continues shifting one character at a time (stride value) applying the same convolutions filter over the rest of the sequence. Consequently, the neural network generates 4-grams features. These features represent the discriminative power of these group of letters in the domain names.

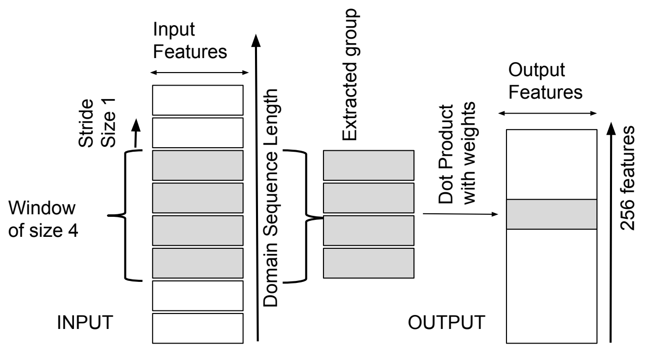

Fig. 1. Feature extraction process of the 1D Convolutional layer

By applying the same filter all over the sequence the required computation time is considerable reduced when compared with traditional Multilayer Perceptron layers. Additionally, since a convolutional kernel independently operates on each 4-gram it is possible to go over the entire input layer concurrently. This paralellization and its consequent low computing time is one of the major benefits of using convolutional networks instead of other deep learning approaches usually used for text processing such as Long Short Term Memory (LSTM) [4, 12, 13].

2.3 Dense Layers

The features extracted by the two previous layers are used by a traditional Multilayer Perceptron Network (MLP), in order to output the probability of a given domain belonging to the *DGA* or *Normal* class. The MLP is composed of two layers: A first fully connected layer of size hn (*Dense* layer) connected to a second *Dense* layer of size 1 used for actually giving the probability output about the considered domain.

3 Experimental Design

The experiments done in this paper focused on the detection performance of the algorithms from two points of view. First, considering the detection of DGA and normal domains. Second, considering the detection performance on the different malware families included in the dataset.

Several standard performance metrics for network detection evaluation were used. These metrics are the True Positive Rate (**TPR**) and the False Positive Rate (**FPR**). TPR is computed as the ratio between the number of correctly detected DGA domains and the total number of DGA domains. Whereas FPR is computed as the ratio between the number of normal domains that are incorrectly classified as DGA and the total number of normal domains.

The evaluation of the 1D-CNN DGA detection method was carried out following the usual machine learning methodology. A dataset containing both DGA and normal domains was split in a 70%/30% ratio. To guarantee the independence of the results, the 70% of the datasets was used for tuning the 1D-CNN hyper-parameters (training set). Whereas the remaining 30% (testing set) was used for testing the performance of the 1D-CNN DGA detection model on unseen domains.

3.1 Dataset Description

The 1D-CNN detection method was evaluated on a dataset containing both DGA and normal domain names. The normal domain names were taken from the Alexa top one million domains. An additional 3,161 normal domains were included in the dataset, provided by the Bambenek Consulting feed. This later group is particularly interesting since it consists of suspicious domain names that were

Table 2. Episode frequency and generation scheme for the DGA Malware Families in the dataset used for training and testing the 1D-CNN for DGA detection method. (A) stands for Arithmetic generation scheme while (W) for word-based

Family	**Scheme**	**Freq.**	**Family**	**Scheme**	**Freq.**	**Family**	**Scheme**	**Freq.**
bamital	(A)	904	cryptolocker	(A)	112,809	padcrypt	(A)	1,920
p2p	(A)	4,000	proslikefan	(A)	100	murofet	(A)	49,199
bedep	(A)	706	dircrypt	(A)	570	necurs	(A)	81,920
post	(A)	220,000	dyre	(A)	26,993	newgoz	(A)	1,666
chinad	(A)	256	fobber	(A)	600	nymaim	(A)	20,225
conficker	(A)	99,996	gameover	(A)	12,000	pushdo	(A)	94,278
corebot	(A)	840	geodo	(A)	1,920	pykspa	(A)	25,727
goz	(A)	1,667	hesperbot	(A)	192	qadars	(A)	1,600
kraken	(A)	9,660	locky	(A)	9,028	qakbot	(A)	60,000
ramdo	(A)	102,000	ramnit	(A)	91,978	ranbyus	(A)	23,167
rovnix	(A)	53,632	shiotob	(A)	12,521	symmi	(A)	4,448
shifu	(A)	2,554	virut	(A)	11,994	sisron	(A)	60
zeus	(A)	1,000	vawtrak	(A)	300	simda	(A)	28,339
tinba	(A)	193,912	tempedreve	(A)	225	pykspa$_v$1	(A)	18
pykspav2F	(A)	800	pykspav2R	(A)	200	banjori	(W)	439218
suppobox	(W)	8185	matsnu	(W)	100127	volatile	(W)	996
beebone	(W)	210	cryptowall	(W)	94	madmax	(A)	2

not generated by DGA. Therefore, the total amount of domains normal in the dataset is 1,003,161. DGA domains were obtained from the repositories of DGA domains of Andrey Abakumov [4] and John Bambenek [5]. The total amount of DGA domains is 1,915,335, and they correspond to 51 different malware families. The list of malware families and the amount of domains selected can be seen in Table 2.

The DGA generation scheme followed by the malware families includes the simple arithmetical (A) and the recent word based (W) schemes. Under the arithmetic scheme, the algorithm usually calculates a sequence of values that have a direct ASCII representation usable for a domain name. On the other hand, word-based consists of concatenating a sequence of words from one or more wordlists.

3.2 Hyper Parameters Tuning

Adjusting the network hyper-parameters is perhaps the most difficult task in the application of neural networks. For the proposed 1D-CNN architecture several hyper parameters needed to be adjusted. Among all the possible hyper-parameters, we particularly focused on finding the optimal values related to the *Embedding*, *Conv1D* and *Dense* layers. In the case of the Embedding layer, two values need to be set: l and d. The parameter l is the length of the input sequence. This parameter was set by following the common approach of fixing it to the maximal domain length found in the dataset [13]. Therefore, only the matrix dimension d needed to be adjusted.

[4] https://github.com/andrewaeva/DGA
[5] http://osint.bambenekconsulting.com/feeds/

For the *Conv1D* layer, the parameters to adjust were the number of filters nf and the size of the kernel ks. The stride length sl was fixed to 1. In the first *Dense* Layer the number of neurons hn was the parameter to optimize. Finally, Since a low FPR is a requirement of DGA detection models, the decision boundary threshold ($thres$) in the second Dense layer was set to 0.90 instead of the more common 0.5. Therefore, only those domains names with a very high probability would be detected as DGA.

A traditional grid search was conducted through a specified subset on the training set. For a robust estimation, the evaluation of each parameter combination was carried out using a k-fold cross validation with $k = 10$ folds. The 1D-CNN layer was trained using the back propagation algorithm [9] considering the *Adaptive Moment Estimation* optimizer [5]. The 1D-CNN training was carried out during 10 epochs.

Table 3 shows the parameter combinations with the better performance detection in terms of the F1-Score.

Table 3. Best Hyper-parameters of the proposed architecture. Only the first parameter combination is shown. The parameters combination with the higher average F1-Score was chosen for the remaining experiments.

avg. F1	**sd**	nf	ks	sl	d	l	hn	**parameters**
0.9797	**0.0050**	**256**	**4**	**1**	**100**	**45**	**512**	5,612,705

3.3 Evaluation on Unseen Domains

The present section describes the experiment results after evaluating the 1D-CNN with the hyper-parameters selected from previous section. For this experiment complete training set was used for training the 1D-CNN and the evaluation was done on the testing dataset which was a 30% of the original dataset. The metrics described in Section 3 were calculated on the testing set with a decision boundary threshold set to 0.90. In terms of the two considered classes (i.e Normal or DGA), the resulting TPR value was around 97% while the FPR was 0.7%.

Regarding the FPR discriminated by normal domains types, the 1D-CNN DGA detection algorithm has correctly detected as normal almost 100% of total Alexa domains. However, for the case of the Bambenek domains, the false positive rate increased to 16%.

Fig. 2 shows the TPR per DGA malware family and its DGA generation scheme (red for arithmetic and blue for word based). The diameter of the circle around each point provides a visual idea about the absolute frequency of DNS requests that belong to that malware family in the training set. In general, most of the DGA were detected, having a TPR close to 0.75 no matter which was the

DGA generation scheme used. The only three exceptions were `Cryptowall` with TPR value of 0.16, `Virut` with 0.39 and `Suppobox` malware with 0.47.

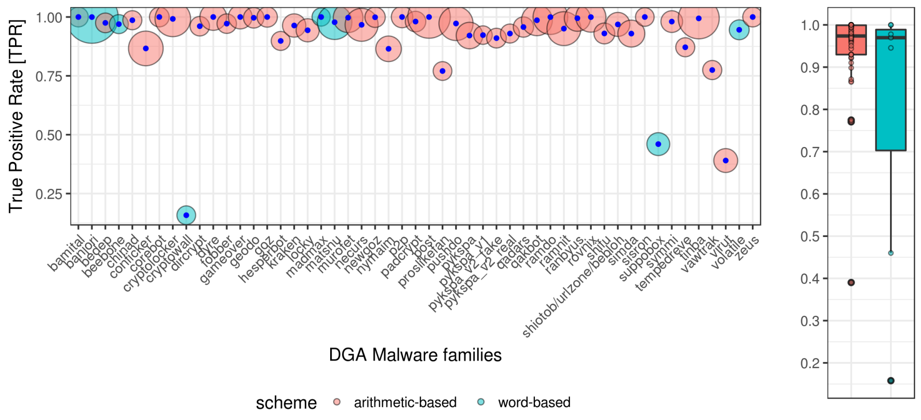

Fig. 2. True Positive Rate per DGA malware family for 1D-CNN

4 Discussion

According to our experiments, the 1D-CNN detection method was capable of extracting common patterns present in arithmetic-based DGA generation algorithms of the different malware families. The previous claim could be an explanation of the good results observed in malware families with very low episode frequency. That is the case of families such as `chinad` and `bamital`, which with less than 1000 examples showed a TPR greater than 90%.

The situation was different for the word-based DGA schemes. As can be seen in the boxplot located at the right of Fig. 2, 75% of TPR for arithmetic-based DGA were concentrated between 1 and 0.92. On the other hand, in the case of word-based, 75% of TPR values were between 1 and 0.70. It seems clear that the TPR for malware families using the arithmetic-based generation scheme were significantly better than word-based. These results may be explained by the fact that word-based generation schemes aim at imitating the look of the normal domains.

There was, however, some unexpected results that deserved a deeper analysis. In particular, results showed for the `Matsnu` malware family, a word-based DGA with a 0.92 TPR and the case of `Virut`, an arithmetic-based DGA with a 0.39 TPR.

In the case of `Virut`, the poor performance can be explained by the the Frequency Character Distribution (**FCD**) of the generated DGA domains. The

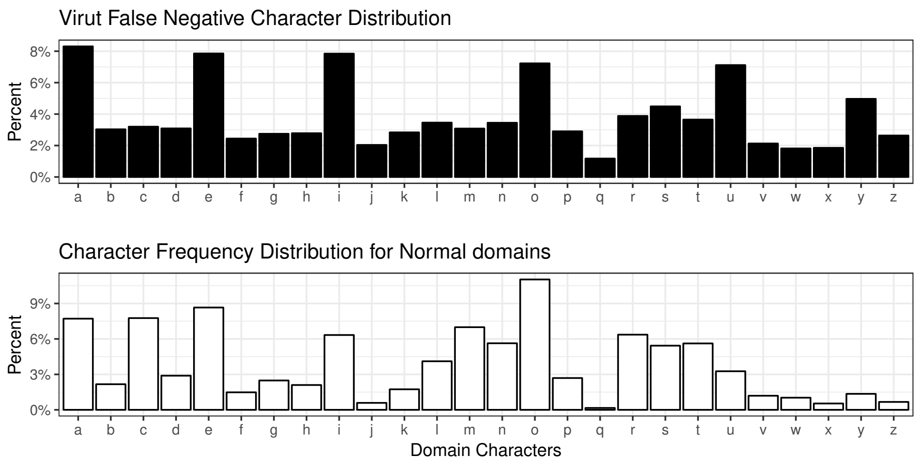

Fig. 3. Character Frequency Distribution for not detected DGA malware

Fig. 3 shows the FCD for the `Virut` DGA malware (in black) and compare them with FCD for normal domains (in white). Notice that for ease of comparison only characters present in DGA domains were included in the normal FCD.

Despite not being a word-based DGA, `Virut` vowels frequency is considerable higher when compared with the rest of the characters. Moreover, such vowel frequency in DGA domains shows similarities with the normal FCD. The similarities with normal FCD could difficult the discrimination process carried out by the 1D-CNN detection method. Such process is even more complicated since `Virut` a small number of episodes included in the dataset.

In the case of `Matsnu`, the high TPR could not be explained by the FCD, since, as expected, `Matsnu` FCD shared similarities with normal FCD. After inspecting several aspects of the DGA, we found that domain name length was the discriminative feature between normal and `Matsnu` domain names. Domains names generated by `Matsnu` were considerable larger than normal domains. In Fig. 4 we compare the Character Length Distribution (**CLD**) of `Matsnu` and normal domains. As can be seen, the `Matsnu` domains are significant larger than normal domains. On the other hand, the `suppobox` malware (shown also in Fig. 4) has a CLD close to normal domains. Since not only CLD but also FCD are similar to normal domains, `suppobox` domains become very difficult to detect.

5 A LSTM Network Architecture for Detecting DGA

The present section presents a performance comparison on DGA detection using a simple LSTM network and the 1D-CNN previously presented. In particular, the network architecture of the LSTM was implemented as described by Woodbridge et al. Woodbridge2016.

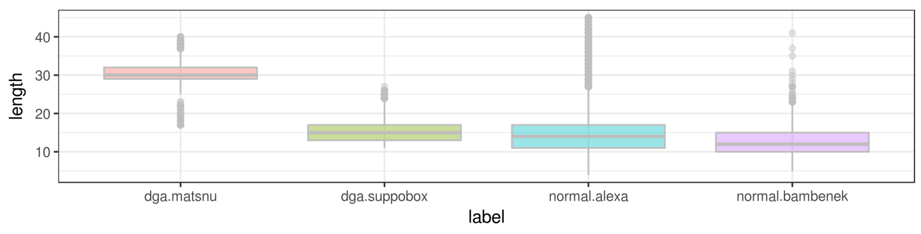

Fig. 4. character length distribution of `Matsnu`, `Suppobox` and normal domains names

Long Short-Term Memory (LSTM) networks are a special type of Recurrent Neural Network (RNN) first introduced by Hochreiter & Schmidhuber in 1997 [4] that has been successfully applied to Natural processing Languages (NLP) problems. The main benefit of LSTM, when compared to convolutional networks and densely connected networks, is that each input has to be processed independently, with no state kept in between inputs. This lack of memory complicates the processing of sequences or temporal series of data. On the other hand, LSTM are capable of memorizing information previously seen on a sequence. Such capability translates into learning character patterns that are not necessary contiguous in the sequence. An example of this capability, and the difference pattern with 1D-CNN, can be observed in Fig. 5.

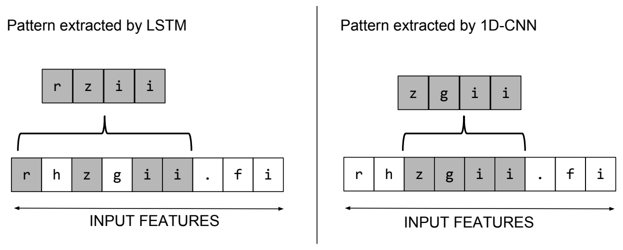

Fig. 5. Pattern extraction capabilities of LSTM and 1D-CNN networks

Similarly to convolutional networks, LSTM networks have to include at least one layer of LSTM cells. A LSTM cell consists of a state that can be manipulated via a set of programmable gates that allow the cell to remember or forget a particular input. An additional output gate modulates the contribution of the cells state to the output, which propagates to the input gates of LSTM cells across the layer, as well as to subsequent layers.

In this case, the neural network architecture is composed by an *Embedding* layer and then a *LSTM* Layer. Finally, a Dropout layer is added before the output layer. Dropout [11] consists of randomly removing a random subset of edges between layers of a network during each iteration of training, but restoring their contribution at test time. The detailed architecture of the LSTM network is described in Table 4

Table 4. The complete network architecture including the corresponding output dimensions and activation function used in each layer

Layer (type)	Activation Function
input (Input Layer)	-
embedding (Embedding)	-
lstm (LSTM)	relu
dropout_1 (Dropout)	-
dense_1 (Dense)	sigmoid

According to [13], the network hyper-parameters for the *Embedding* layer were $d = 128$ and $l = 75$, while the number of *LSTM* cells in the *LSTM* Layer was set up to 128. Finally, the *Dropout* Layer was setup to 0.5. Similarly to the 1D-CNN, LSTM network was trained during 10 epochs on the 70% of the dataset described in section 3 and tested in the remaining 30%.

The resulting model showed a TPR value around 94% of total DGA present in the dataset while the FPR value is close to 3% of total normal domains. According to previous results, the LSTM network performed considerable worse than 1D-CNN. Specially, if we consider the importance of low false positive rate for real-world scenarios.

The TPR per DGA malware family and its DGA generation scheme is shown in Fig. 6. Similarly to Fig. 2, most of the DGA were detected with TPR values close to 0.75. Moreover, as can be seen in the boxplot located at the right of Fig. 6 the 75% of TPR for arithmetic-based DGA are concentrated between 1 and 0.92. However, the 75% of TPR values for word-based scheme are between 0.97 and 0.12, results that are considerable worse than 1D-CNN.

A detailed comparison considering not only malware families but also normal requests is shown in Fig. 7. As can be observed the detection performance of LSTM was better than 1D-CNN for the case of the `Cryptowall` malware (a word-based DGA) and the `Virut` malware. On the other hands the 1D-CNN significantly outperformed LSTM for most of the word-based DGA, such as `Beebone`, `Suppobox`, `Matsnu`, `Madmax` and `Volatile` malware. Such low performance for detecting word-based DGA was also observed [13].

It is important to remember that the LSTM architecture presented in [13] is simpler than the 1D-CNN described in section 2. Moreover, the additional *Dense* Layer present in the 1D-CNN could be the explanation of the poor performance in word-based detection for the LSTM network.

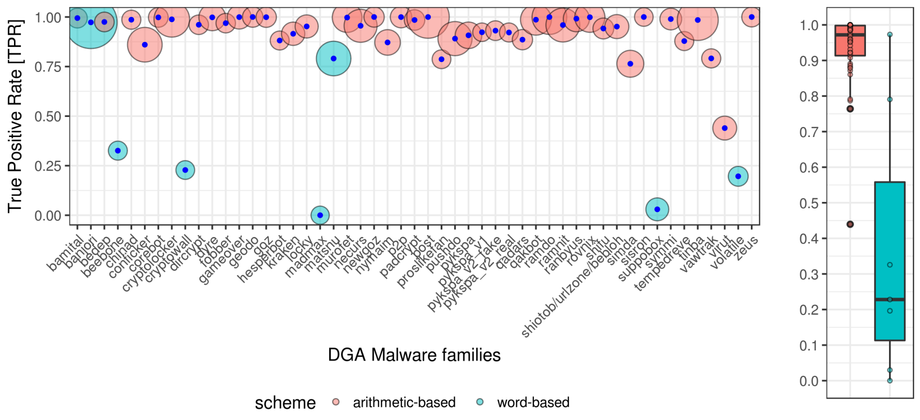

Fig. 6. True Positive Rate per DGA malware family for LSTM

6 Concluding Remarks

In the present work, we explored the viability of 1D-CNN for lexicographical DGA detection. A network with a minimal architecture complexity was evaluated on a dataset containing domains names generated by 51 different malware families as well as normal domains. Malware families included two different DGA generation scheme: (1) the most common arithmetic-based and (2) the recent and more difficult to detect Word-based. The dataset was properly split in training and testing sets. A hyper-parameters grid search was conducted on training set and the best resulting model was then evaluated on the testing set.

Despite its simple architecture, the resulting 1D-CNN model correctly detected more than 97% of total DGA domains with a FPR around 0.7%. Such results make it suitable for real-life networks. Inspecting the malware family results, we observed that the 75% arithmetic-based DGA showed TPR values between 1 and 0.92. Such values were observed even in malware families with low frequency episode, confirming the hypothesis that a 1D-CNN can learn the common properties from the different DGA generation families, at least under the arithmetic-based generation scheme. On the other hand, in the case of word-based, the 75% of TPR values were between 1 and 0.70. Such high variability in the TPR results responds to the high TPR for malware such as `Matsnu` and the low TPR for malware such as `Suppobox` and `Cryptowall`. By analyzing the FCD and CLD we detected than even tough both shared the FCD with normal domains, `Matsnu` showed a significant increment in the length of the domains, a situation that makes it easily detectable. Different was the case of malware such as `Suppobox`, that shared both FCD and LCD with normal domains, and consequently was very difficult to detect.

The performance of 1D-CNN was compared with a simple LSTM network on the same dataset. The experiments showed the LSTM performance is worse com-

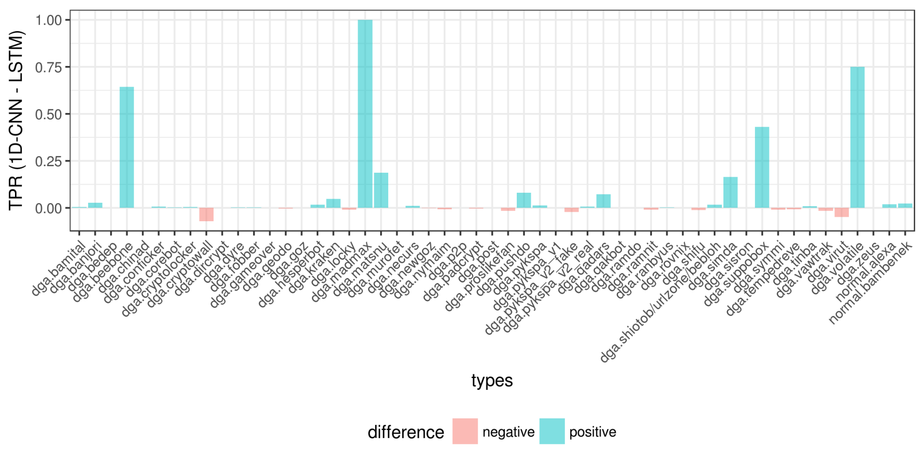

Fig. 7. Performance detection pero DGA malware families and normal requests

pared to 1D-CNN in terms of TPR and FPR. In particular, the 4% of false positives obtained by the LSTM is a significant performance loss with high impact when applied to real-world networks. Such performance loss could be explained by the extremely simple architecture of the LSTM proposed by Woodbridge et al. [13]. The performance of LSTM networks with more complex architecture will be the subject of future work.

Despite the good results shown by 1D-CNN, we observed that it could be very difficult to detect word-based DGA that share the CLD and FCD with normal domains. Therefore, as the word-based generation schemes become more precise in imitating normal domains the detection could be extremely hard. Moreover, it is possible that such particular cases be outside the capabilities of detection methods based only on a lexicographical approach. Consequently, an alternative detection strategy could be necessary.

Acknowledgments

The authors would like to thank the financial support received by CVUT and UNCuyo during this work. In particular the founding provided by the Czech TACR project no. TH02010990 and the PICT 2015-1435 granted by ANPCyT. The authors would also like to specially thank Whalebone s.r.o., whose technical support and help have been fundamental to the complete research process.

References

1. Ahluwalia, A., Traore, I., Ganame, K., Agarwal, N.: Detecting broad length algorithmically generated domains. In: I. Traore, I. Woungang, A. Awad (eds.) Intelligent, Secure, and Dependable Systems in Distributed and Cloud Environments, pp. 19–34. Springer International Publishing, Cham (2017)

2. Catania, C., Garcia, S., Torres, P.: An analysis deep convolutional neural networks for detecting DGA. In: XXIV Congreso Argentino de Ciencias de la Computación (Tandil, 2018). (2018)
3. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
5. Kingma, D.P., Ba, J.: Adam: A Method for Stochastic Optimization. arXiv e-prints arXiv:1412.6980 (2014)
6. Kührer, M., Rossow, C., Holz, T.: Paint it black: Evaluating the effectiveness of malware blacklists. In: A. Stavrou, H. Bos, G. Portokalidis (eds.) Research in Attacks, Intrusions and Defenses. Springer International Publishing (2014)
7. Pascanu, R., Mikolov, T., Bengio, Y.: On the difficulty of training Recurrent Neural Networks. arXiv e-prints arXiv:1211.5063 (2012)
8. Plohmann, D., Yakdan, K., Klatt, M., Bader, J., Gerhards-Padilla, E.: A comprehensive measurement study of domain generating malware. In: 25th USENIX Security Symposium (USENIX Security 16), pp. 263–278. USENIX Association, Austin, TX (2016)
9. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Neurocomputing: Foundations of research. chap. Learning Representations by Back-propagating Errors, pp. 696–699. MIT Press, Cambridge, MA, USA (1988)
10. Schiavoni, S., Maggi, F., Cavallaro, L., Zanero, S.: Phoenix: Dga-based botnet tracking and intelligence. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 192–211. Springer (2014)
11. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research **15**, 1929–1958 (2014). URL http://jmlr.org/papers/v15/srivastava14a.html
12. Torres, P., Catania, C., Garcia, S., Garino, C.G.: An analysis of recurrent neural networks for botnet detection behavior. In: 2016 IEEE Biennial Congress of Argentina (ARGENCON), pp. 1–6 (2016)
13. Woodbridge, J., Anderson, H.S., Ahuja, A., Grant, D.: Predicting Domain Generation Algorithms with Long Short-Term Memory Networks. arXiv e-prints arXiv:1611.00791 (2016)
14. Yadav, S., Reddy, A.K.K., Narasimha Reddy, A.L., Ranjan, S.: Detecting algorithmically generated domain-flux attacks with DNS traffic analysis. IEEE/ACM Transactions on Networking **20**(5), 1663–1677 (2012)