# MALWARE CLASSIFICATION WITH RECURRENT NETWORKS

*Razvan Pascanu*⋆⋆ *Jack W. Stokes*† *Hermineh Sanossian*‡ *Mady Marinescu, Anil Thomas*±

⋆ University of Montreal, Montréal QC H3C 3J7 Canada
† Microsoft Research, One Microsoft Way, Redmond, WA 98052 USA
‡ Microsoft Pty Ltd, Level 5, 4 Freshwater Place, Southbank, VIC 3006 Australia
± Microsoft Corp., One Microsoft Way, Redmond, WA 98052 USA

## ABSTRACT

Attackers often create systems that automatically rewrite and reorder their malware to avoid detection. Typical machine learning approaches, which learn a classifier based on a hand-crafted feature vector, are not sufficiently robust to such reorderings. We propose a different approach, which, similar to natural language modeling, learns the language of malware spoken through the executed instructions and extracts robust, time domain features. Echo state networks (ESNs) and recurrent neural networks (RNNs) are used for the projection stage that extracts the features. These models are trained in an unsupervised fashion. A standard classifier uses these features to detect malicious files. We explore a few variants of ESNs and RNNs for the projection stage, including Max-Pooling and Half-Frame models which we propose. The best performing hybrid model uses an ESN for the recurrent model, Max-Pooling for non-linear sampling, and logistic regression for the final classification. Compared to the standard trigram of events model, it improves the true positive rate by 98.3% at a false positive rate of 0.1%.

***Index Terms***— Malware Classification, Recurrent Neural Network, Deep Learning

## 1. INTRODUCTION

Malicious software, commonly referred to as malware, is a significant problem for modern computing devices including desktop computers and mobile phones. While commercial anti-virus (AV) products attempt to detect and remediate (i.e. clean) the infected device, attackers sufficiently motivated by profit implement clever programming redundancies to quickly and automatically reinfect the computer or phone. Given the challenges and ubiquity of modern malware, many researchers have attempted to devise automated methods for detection [1]. For over a decade, researchers and anti-virus companies have begun employing machine learning algorithms to address this problem as noted in Section 2. In the academic community, researchers have proposed using a wide variety of machine learning classifiers such as logistic regression [2], neural networks [3, 4], and decision trees [5] for malware classification.

Recently, researchers have demonstrated excellent results using recurrent neural networks (RNNs) on language modeling [6], online handwritten recognition and generation [7], and speech recognition [8]. Echo state networks (ESNs) have been successfully used for predicting chaotic systems [9]. In this paper, we ask the question: *Can these sequential models also help to automatically detect malware?* In our study, the high-level events are canonicalized representations of application programming interface (API) calls made to various components including the operating system (OS) and the C run-time library form the input sequences. To the best of our knowledge, this is the first paper to propose using RNNs or ESNs to address the malware problem.

We attempt to learn the *language* of malware as a new method of detecting these unknown threats. Specifically, using the recurrent model to directly classify the files is not efficient (there is only a single bit of information for every sequence). Instead, we use a recurrent model trained to predict next API call, and use the hidden state of the model (that encodes the history of past events) as the fixed-length feature vector that is given to a separate classifier (logistic regression or MLP). In an attempt to improve the scope of this summary provided by the recurrent model we also provide a few modifications of how the feature vector is constructed from the RNN or ESN. We first introduce *Max-Pooling* over the values of the hidden units in time. The assumption behind this choice is that hidden units may learn to specialize in detecting different and potentially *reordered* temporal patterns. In our classification task we are primarily interested in knowing if these temporal patterns are present or not in the input sequence. Next we propose the *Half-Frame* model which increases the memory capacity of our final representation by including state information in the middle of the file's event sequence in addition to the final state. We also employ a *Leaky-Units* architecture [10] which essentially utilizes an exponentially decaying low-pass filter to increase the long-term memory of the system. It is often the case that for malware events the

⋆⋆The first author performed the work while at Microsoft Research

most informative part of a sequence occurs at the beginning of the sequence and may be forgotten by standard recurrent models. To overcome this limitation, we use a *Bi-Directional* model [11] which combines two separate models, one which learns by processing the events in the forward direction and the second which constructs a model by processing the events in the reverse direction.

## 2. RELATED WORKS

Related work falls into two main categories, recurrent modeling and malware classification.

**Recurrent Modeling:** Recurrent neural networks have been explored since the 1980s. However this model quickly became unpopular following the discovery of the vanishing and exploding gradient problem [12, 13]. As a way of avoiding these issues, [14] introduced Echo State Networks, a form of recurrent neural models whose input and recurrent weights are not trained, but carefully sampled. Jaeger and Haas [9] show that such models can do very well on predicting chaotic dynamics. Results as those from [13, 15, 16] try to address the learning problem of RNNs. As a result, a variety of state-of-the-art results, including online handwritten recognition [7] or speech [8], have been obtained using different variations of recurrent models. Mikolov *et al.* [6] provide state-of-the-art results in language modelling using a standard RNN. [7] explores the same problem with deep LSTM recurrent models.

**Malware Classification:** The most recent summary of the field of malware classification is given in [1]. A classic paper on malware classification was written by Shultz *et al.* [17] which proposed several different classifiers including Ripper, Naive Bayes, and an ensemble classifier to classify files as malware or benign. Neural networks were also used [3, 4] to detect malware. Kolter and Maloof [5] compared naive Bayes, decision trees, support vector machines, and boosting for malware classification. The event stream used in our model is a reflection of the unknown file's behavior. Behavior malware detection has also been a very active area of research. A few of the important papers which utilize behavior to detect malware include [18, 19]. El-Bakry [20] suggested that a Time Delay Neural Networks could be used for malware classification, but did not do any experiments to validate the claim. A hierarchical Hidden Markov Model was also used for this task by Muhaya *et al.* [21].

## 3. ALGORITHM DESCRIPTION

Let $\mathbf{u}_t$ and $\mathbf{h}_t$ denote the input and state vectors, respectively, at time instance $t$. Let $\mathbf{W}_{in}, \mathbf{W}_{rec}, \mathbf{b}, \mathbf{W}_{out}, \mathbf{b}_{out}$ be the input to hidden layer weight matrix, recurrent weight matrix, bias, and output weight matrix and output bias respectively. Let $\phi$ and $\phi_{out}$ be the activation function of the hidden layer and output layer respectively. In our projection stage, $\tanh$ is used for the hidden layer and softmax for the output. The recurrent models are then described by the following equations:

$$\mathbf{h}_t = \phi(\mathbf{W}_{in}\mathbf{u}_t + \mathbf{W}_{rec}\mathbf{h}_{t-1} + \mathbf{b}) \quad (1)$$

$$\hat{\mathbf{y}}_t = \phi_{out}(\mathbf{W}_{out}\mathbf{h}_t + \mathbf{b}_{out}), \quad (2)$$

where $\hat{\mathbf{y}}_t$ is the predicted output. For malware classification, the memory window of the standard ESN and RNN models is not sufficient. To address this shortcoming, we propose two new algorithmic additions for recurrent architectures, *Max-Pooling* and *Half-Frame*. In our study we also employ the previously proposed *Leaky-Units* [10] and *Bi-Directional* models [11]. Finally, we discuss the classification stage.

*Max-Pooling:* Max-Pooling is a form of non-linear downsampling which has previously been used in convolutional neural networks (CNNs) for object recognition in images [22]. Similar to the desired response for vision, we use Max-Pooling to increase invariance in the fixed-length representation we feed to the system's classification stage. While Max-Pooling was originally proposed in the context of CNNs, we believe that using Max-Pooling for RNNs and ESNs is new. As shown in Figure 1, we select the maximum hidden state output, $h_{max}$ for each sequence as:

$$h_{max}(i) = \max(h_0(i), h_1(i), \cdots h_T(i)) \quad (3)$$

for $i \in (0, \cdots, N-1)$ where $N$ is the number of neurons and the $max$ operation is performed element-wise. The representation of the sequence is the concatenation of the last state and the Max-Pooling layer $[h_T; h_{max}]$.

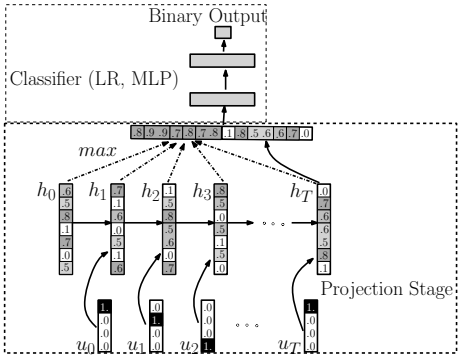



**Fig. 1**: Depiction of the overall structure of the model. The projection stage consists of a recurrent model where Max-Pooling is used as additional features for the fixed-length representation.

*Half-Frame:* Instead of using the state information at the end of the sequence to determine if the sequence contains malicious activity, we also use the intermediary state in this model. The intuition is that the intermediary state will mostly represent the history up to that point in time, while the last state will mostly represent the events towards the end of the

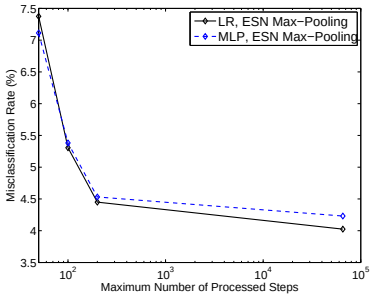



**Fig. 3**: Missclassification rate for best models as a function of the maximal sequence length.

sequence. Typically, the *Half-Frame* model includes the state from the middle of the sequence in addition to the state at the end of the sequence. Based on our intuition, this allows our model to increase its memory capacity within each sequence.

**Fixed-Length Representations:** Once $W_{rec}$ and $W_{in}$ have been learned for the RNN or generated for the ESN, the next step in the processing is to construct a fixed-length representation for the event stream which is provided as input to the classification stage.

One practical issue we face at this point is the huge variation in the length of the input streams. Some streams included a single event while others had over 100,000 events. To deal with this, and also to answer the question: *How fast can we detect if a file is infected or not?*, we decided on a maximal length $N$ and minimal length $n$ for any stream. We ignored all sequences shorter than $n$, which was fixed to 15 throughout all our experiments. For all sequences with more than $N$ steps, which was a hyper-parameter for which we explored the values 50, 100, 200 and 65536, we kept only the first $N$ events.

**Classifier:** The final component of the proposed algorithm is the classification stage. As noted previously, we used both logistic regression and multi-layer perceptrons with rectifier units [23] to classify the fixed-length projections. We also use dropouts which have shown to significantly improve the generalization of the MLP model [24]. It is important to note that the RNN and ESN are trained independently of the classifier. Thus, they act as feature extractors trained in an unsupervised fashion. We believe this approach is another contribution to the growing body of representation learning approaches.

## 4. EXPERIMENTAL RESULTS

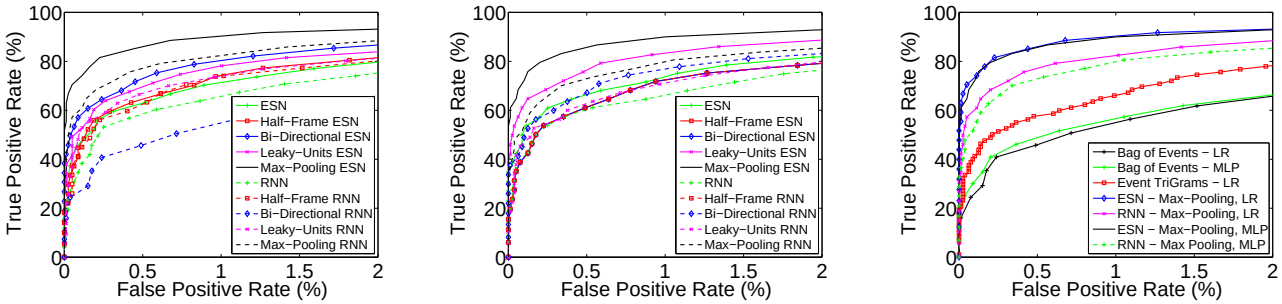
We rely on Theano [25] to efficiently implement these architectures. We first describe the parameter settings used in this analysis. We then investigate the effect of the representation length for the various models. We conclude this section by presenting the Receiver Operating Characteristic (ROC) curves for each model.

**Experiment Configuration:** The malware and benign files were collected from our company's production, anti-malware file collection. Some of the samples may be used to train a production malware classification system, and we acknowledge this dataset is not publicly available. Our system is under constant attack by adversaries and releasing the dataset may degrade the end user's protection on hundreds of millions of computers around the world. For training the projection stage, analysts provided behavior event streams from 250,000 randomly selected malware files from the file collection they use during their daily investigations and 250,000 randomly selected benign files from a second collection which is used to ensure that anti-virus signatures do not cause false positive detections on third-party programs. The initial dataset was randomly split into 297,500, 52,500, and 150,000 examples for training, validation, and test, respectively. The training is done on segments of equal length of 100 events formed from the streams representing each file.

The raw event stream consists of 114 distinct, high-level events generated by the anti-malware engine which encode all of the low-level API (application programming interface) calls made by the program. There are several different APIs which can be used to generate a file (e.g. kernel32!CreateFile, msvcrt!fopen), and these high-level events canonicalize multiple events with similar functionality. To infer the ability of the models to generalize, we collected the list of events that appear in only one class or the other and ignored these events. To train the final classifier in the classifier stage, we used 75,000 randomly selected files which were evenly split between the malware and benign classes. These files were split as 50,000 for training, 10,000 for validation, and 15,000 for test. The considered files had at least 15 events, and the sequence formed from the first 100 events of each file was unique. This ensured that there was no overlap between the training, validation and test set which were constructed by randomly assigning files to one of the 3 possible sets.

We hand-tuned the hyper-parameters of the projection and classification models by sweeping over a range of possible values. For the classification stage, we consider logistic regression and a two hidden-layer MLP. Logistic regression is trained using the softmax criterion in all cases. All MLPs have two hidden layers. For both models the optimal learning rate is $1.0$, and we use dropout for the MLP with a drop probability of $0.5$. The hidden layers of the MLP have 1024 units and use rectifier activation function. The learning rate is halved when the validation error increases.

The dimensionality of the fixed-length representation is 3000 for all recurrent models. Bi-Directional models use 1500 hidden units for the forward pass and 1500 units for the backwards pass. Half-Frame models have 1500 units and the representation uses 1500 values for mid-sequence frame and 1500 for the last frame. Max-Pooling model has similarly 1500 hidden units and the representation is the last hidden state concatenated to the max-pooled one. Leaky-Units mod-




(a) Logistic regression (b) MLPs (c) Best performing models

**Fig. 2**: ROC curves for different classifiers (a-c).

els consist of 1500 units from the smoothed model where $\beta$ is uniformly sampled between $[.01, .1]$ and 1500 normal hidden units. For each of these models, we append the representation from the bag of events to the final representation. Therefore we are investigating if these new representations, that take into account the order of events, encode additional useful information which is not present in the bag of events encoding. Given the low-dimensionality of the input data no feature selection is required.

The sparsity for the initialization of the recurrent weights is set to have each unit at time $t$ feed into only 20 randomly selected units at $t + 1$. The spectral radius in all cases is set initially to $0.99$. The input weights' scale (measured as the norm of each column of the matrix) is set to $2.0$, and their sparsity (with the same meaning as above) is also set to 20. The learning rate for training is $1.0$. A cutoff threshold is used for clipping the gradients for RNNs and is set to $1.0$. For the RNN, the additional cost of predicting the whole sequence, given the state of the network $h_t$, has a weight of $0.5$.

**ROC Curves:** In the previous subsection, we evaluated the performance of the recurrent models with a classifier threshold value of 0.5. Next, we evaluate the ROC curves for each of these models, and later we compare the best performing Max-Pooling models against several baseline architectures (see Figure 2c). Figure 2a and Figure 2b shows the ROC curves when logistic regression and the MLP, respectively, are used for classification.

Figure 2c compares the ROC curves of the best performing Max-Pooling architectures plus several baselines including a bag of events representation and a bag of trigrams model [3, 2, 26]. The baseline bag of events (BOE) model is equivalent to the standard Bag of Words (BOW) model in natural language processing. To compare against previously published papers which utilize event sequences, we use a bag of trigrams representation, and train a logistic regression model with softmax. The results confirm our suspicion. At a false positive rate of $0.1\%$, the TPR of the bag of trigrams model ($36.17\%$) is significantly better than for the bag of events models ($24.46\%$). The ROCs for logistic regression dominate the MLPs for both the ESNs and RNNs, but the results are very close for ESNs. The ESN model with Max-Pooling and logistic regression (TPR = $71.71\%$ at FPR = $0.1\%$) outperforms event trigrams by $98.3\%$. We believe Max-Pooling works for the ESN and RNN because, for this task, we care about detecting temporal patterns regardless of when the pattern occurs in the sequence. In particular, this property helps to detect reordered malware. If each unit specializes in different patterns, Max-Pooling tells us which of these different patterns actually appear in the file. Figure 3 shows the misclassification rate as a function of the maximal number of steps processed, suggesting that there is not much accuracy lost if we truncate all sequences to a maximum of 200 steps (greatly reducing the memory and computation requirements).

## 5. CONCLUSIONS

Automated malware classification is a very challenging problem. When we began this research, we were concerned that the adversarial nature of the attack would prevent recurrent models from learning the language of malware. Results in Section 4 demonstrate that combining a recurrent model with a standard classifier can improve the true positive rate by a factor of three compared to a bag-of-events model and a factor of two given by a bag-of-trigrams model. Given this tremendous improvement, we believe these hybrid models which combine an ESN or RNN with a higher-level classifier can serve as effective weapons in the malware analyst's arsenal.

Our initial goal of this work was to learn the language of malware, but the ESN models outperform the RNNs in the majority of the experiments. The task of learning falls in this situation mostly on the classifier which has to extract the useful information from the random temporal projection of the ESN. We believe that Max-Pooling is more useful in this situation, as the ESN does not utilize the hidden state in the same way the RNN does. The hidden representation is more redundant due to implicit randomness of the projection. We are hopeful that additional research in recurrent modeling can be adapted to the malware language modeling task and can improve these results in the future.

## 6. REFERENCES


[1] N. Idika and A.P. Mathur, "A survey of malware detection techniques," Tech. Rep., Purdue Univ., February 2007.

[2] Nikos Karampatziakis, Jack Stokes, Anil Thomas, and Mady Marinescu, "Using file relationships in malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, vol. 7591 of *Lecture Notes in Computer Science*, pp. 1–20. Springer Berlin Heidelberg, 2013.

[3] George E. Dahl, Jack W. Stokes, Li Deng, and Dong Yu, "Large-scale malware classification using random projections and neural networks," in *ICASSP*, 2013.

[4] Jeffrey O. Kephart, "A biologically inspired immune system for computers," in *In Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*. 1994, pp. 130–139, MIT Press.

[5] J.Z. Kolter and M.A. Maloof, "Learning to detect and classify malicious executables in the wild," in *Journal of Machine Learning Research*, 2006, pp. 2721–2744.

[6] T. Mikolov, M Karafiat, L. Burget, J. Cernocky, and S Khundanpur, "Recurrent neural network based language model," in *Proceedings of Interspeech*, 2010.

[7] A. Graves, "Generating sequences with recurrent neural networks," Tech. Rep., arXiv:1308.0850, 2013.

[8] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP*, 2013.

[9] H. Jaeger and H. Haas, "Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication," in *Science*, 2004.

[10] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proc. ICASSP 38*, 2013.

[11] Mike Schuster and Kuldip K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, pp. 2673–2681, November 1997.

[12] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, pp. 1735–1780, 1997.

[14] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks," Tech. Rep., German National Research Center for Information Technology, 2001.

[15] R. Pascanu, T. Mikolov, and Y Bengio, "On the difficulty of training recurrent neural models," in *ICML*, 2013.

[16] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of momentum and initialization in deep learning," *ICML*, 2013.

[17] M.G. Schultz, Eleazar Eskin, E. Zadok, and S. Stolfo, "Data mining methods of detection of new malicious executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001, pp. 38–49.

[18] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in *Proceedings of ISEC'08*, 2008, pp. 5–14.

[19] E. Kirda and C. Kruegel, "Behavior based spyware detection," in *Proceedings of the 15th USENIX Security Symposium)*, 2006, pp. 273–288.

[20] H. El-Bakry, "Fast virus detection by using high speed time delay neural networks," *Journal in Computer Virology*, vol. 6, pp. 115–122, 2010.

[21] F. Muhaya, M. Khan, and Y. Xiang, "Polymorphic malware detection using hierarchical hidden markov model," in *IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC)*, 2011.

[22] D. Scherer, A. Muller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Proc. of the Intl. Conf. on Artificial Neural Networks*, 2010, pp. 92–101.

[23] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *AISTATS*, Apr. 2011.

[24] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinv, "Improving neural networks by preventing co-adaptation of feature detectors," Tech. Rep., arXiv:1207.0580, 2012.

[25] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.

[26] Syed Mehdi, Ajay Kumar Tanwani, and Muddassar Farooq, "Imad: in-execution malware analysis and detection," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, 2009, pp. 1553–1560.