LARGE-SCALE MALWARE CLASSIFICATION USING RANDOM PROJECTIONS AND NEURAL NETWORKS

George E. Dahl

University of Toronto
Department of Computer Science
Toronto, ON, Canada

Jack W. Stokes, Li Deng, Dong Yu

Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA

ABSTRACT

Automatically generated malware is a significant problem for computer users. Analysts are able to manually investigate a small number of unknown files, but the best large-scale defense for detecting malware is automated malware classification. Malware classifiers often use sparse binary features, and the number of potential features can be on the order of tens or hundreds of millions. Feature selection reduces the number of features to a manageable number for training simpler algorithms such as logistic regression, but this number is still too large for more complex algorithms such as neural networks. To overcome this problem, we used random projections to further reduce the dimensionality of the original input space. Using this architecture, we train several very large-scale neural network systems with over 2.6 million labeled samples thereby achieving classification results with a two-class error rate of 0.49% for a single neural network and 0.42% for an ensemble of neural networks.

***Index Terms*—** Malware Classification, Random Projections, Neural Network

1. INTRODUCTION

In this paper, we consider the problem of automated malware detection. Computer users often download malware (i.e. malicious software) to their computer by unknowingly visiting a malicious webpage hosting a drive-by download attack, clicking on a malicious link included in email, opening an attachment which includes an exploit, or by inserting a USB thumb drive containing malware into their computer. The amount of new malware is growing at a staggering rate. Microsoft receives over 150 thousand new, unknown files each day to be analyzed. Given the enormous volume of unknown files, manual inspection by analysts is impossible. Malware authors use automated methods such as polymorphism, where a program generates a unique, new instance of a malware family for each victim, to create new malware. To combat this threat, anti-virus companies must utilize signal processing and machine learning methods to automatically detect new instances of malware.

The goal of a fully automated malware classification system is to operate at an extremely low false positive rate (e.g. $< 0.01\%$) while providing a reasonably low false negative rate (e.g. $< 5\%$). The cost of a false positive is higher since it may result in an important system file being deleted from the computer thereby preventing the computer from booting correctly. A secondary goal is, provided that the system predicts a file to be malicious, does it belong to a known malware family? Being able to correctly predict the family allows an unknown file to be assigned to the correct expert for manual investigation. If necessary, a file predicted to be in the Rbot family can be assigned to an analyst with expertise in analyzing instances of Rbot for further investigation. In the hope of building a classifier capable of addressing these challenges, we construct a dataset of 2.6 million labeled training examples belonging to 134 malware families as well as a benign file class and a generic malware class.

Given the severity of the problem, malware classification is an active research area [1], but the escalating threat indicates the problem is clearly not solved. Achieving very low false positive rates is extremely challenging and having access to a very large number of labeled malware and benign examples is required to even begin to obtain reasonable accuracies. Most earlier research has been done on relatively small malware sample collections [2, 3] limiting the accuracy of these systems.

Malware classification systems are often based on sparse, binary feature sets. In our work, we also employ sparse binary features based on file strings, application programming interface (API) tri-grams, and API call plus parameter value. To achieve good classification accuracy, we use over 179 thousand sparse, binary features generated from feature selection. Logistic regression on all of the features demonstrates reasonable accuracies at large-scale. However, the error rates are still not small enough for fully automated malware classification. In addition, the high-dimensionality of the input space prevents more complicated algorithms from being utilized. To address this problem, we propose a novel malware classification architecture which first projects the high-dimensional feature vector down to a much lower-dimensional subspace. Doing so allows us to train the system with neural networks with one or more hidden layers reducing the two-class error rate by more than 43% compared to logistic regression trained with all of the features. In our work, we use random projections [4, 5], but also investigate using principal component analysis (PCA) to reduce the dimensionality of the input vector. We also investigate using pre-training, a method often used in deep learning architectures, to solve this problem, but we found that the results were slightly worse than standard neural network topologies trained with back-propagation. This paper makes the following contributions:

- A large-scale system to classify unknown files with random projections and neural networks is proposed and implemented, and the results are presented.
- Random projections are used to reduce the dimensionality of the input space by a factor of 45 allowing a neural network to be trained on the high-dimensional input data.
- We investigate the number of random projections, hidden layers, and hidden units required to achieve good performance. We also compare the performance of logistic regression and neural networks for the task of multi-class malware classification.

Fig. 1. System Diagram for the Malware Classifier.

2. SYSTEM

This section discusses the components of the proposed malware classification system illustrated in Figure 1. We first construct a labeled dataset from features automatically generated by a production anti-malware engine. After initial feature selection to reduce the dimensionality of the input space, we next use random projections to further reduce the input dimensionality while retaining the highly discriminative information about the input identified by feature selection. Finally, we train several models including non-linear neural networks and linear logistic regression classifiers to classify the unknown files.

DataSet: To construct the labeled training dataset, we first obtain over 2.6 million files, 1,843,359 of which are malicious and 817,485 of which are benign. The files are mostly labeled manually by analysts. We also include benign files in the training set that have been collected from reputable sources (e.g. Adobe Acrobat Reader). Microsoft receives unknown files which are potentially malicious from many different sources including other anti-virus vendors, security organizations (e.g. CERT), product support, and by requesting samples from users' machines which have attempted to download or install unknown files. Each of the malicious files is also assigned to a particular malware family. We randomly selected a set of files from 134 malware families determined by analysts to be important to identify explicitly. All malware samples belonging to malware families not in the set are included in a generic malware class, and all samples of legitimate software are assigned to a benign class.

Features: To analyze unknown malware, we modified Microsoft's production anti-malware engine which, as part of the analysis process, scans each unknown file in a lightweight virtual machine. This engine is used in all of the Microsoft anti-malware products such as Microsoft Security Essentials and Forefront Endpoint Protection. As part of this process, we extract three types of features including null-terminated patterns observed in the process' memory, tri-grams of system API calls, and distinct combinations of a single system API call and one input parameter. Most of the time, the null-terminated patterns correspond to system strings used to create the unknown file. The API tri-gram and API parameter features are used to analyze the dynamic behavior of the unknown file [6, 7]. The API tri-gram features consist of three consecutive system API calls. The API parameter features identify unique parameter values corresponding to different system API calls which can be used to identify individual families of malware. Since the raw data is generated by the production anti-malware engine, we use attributes which can be efficiently in extracted real-time.

Enumerating all of the distinct combinations of the three attribute sets yields over 50 million possible features. In order to reduce the input space to a reasonable set of features which can be classified by standard supervised learning techniques, we next perform feature selection using mutual information [8]. Feature selection generated over 179 thousand sparse binary features that best discriminate each class (e.g. malware family, malware, benign) from every other class in our dataset. Initially, we specified 3000 features to be selected from each input family and 120,000 features from the two generic classes (i.e. malware, benign). Eliminating common features resulted in the final 179 thousand selected features.

Random projections: Even after feature selection, the input dimensionality is still quite large, although there is a lot of sparsity. Naive neural net training on such high dimensional input is too expensive. To make the problem more manageable, we used the very sparse random projections technique described in [4, 5]. We project each input vector into a much lower dimensional space (a few thousand dimensions) using a sparse projection matrix R with entries sampled iid from a distribution over $\{0, 1, -1\}$. Entries of 1 and -1 are equiprobable and $P(R_{ij} = 0) = 1 - \frac{1}{\sqrt{d}}$, where d is the original input dimensionality. Another way to view the random projection in the context of neural network training is that the random projection step forms a layer with linear hidden units in which the weight matrix is not learned and is instead simply set to R.

Classifiers: We use two classification techniques to attack the malware detection problem in this paper: logistic regression and neural networks. We briefly describe these techniques here.

Logistic regression: We have explored two versions of a logistic regression classifier, with and without the use of the random projection just described. The parameters of both versions are trained with the standard stochastic gradient descent (SGD) optimization method. Logistic regression is a linear classifier and thus incapable of learning certain functions. We use multinomial logistic regression to handle multiclass classification, or a "softmax" output layer in neural network parlance. Although with a large number of inputs linear classifiers can be quite powerful, for difficult classification problems such as malware classification we expert there to be useful nonlinear relationships in the data. We would like to examine whether using nonlinear adaptive features can improve upon the classification accuracy of logistic regression. One common technique of learning nonlinear adaptive features is a feedforward neural network, whose final output layer is a logistic regression classifier (or "softmax" layer). We describe the neural network classifier below.

Neural Networks: The neural network classifier with random projections learns a non-linear, multiclass model given by the architecture in Figure 2. The random projections first reduce the dimensionality of the input vector from 179 thousand features to 4000 features. This lower-dimensional data then serves as input to the neural network. The top layer then performs a softmax classification to produce the final 136 class membership probabilities representing the malware families and the generic malware and benign classes. Essentially, a neural network is similar to logistic regression with learned intermediate representations of the input. The neural network implements a function from an input vector y_0 to an output vector y_L using a sequence of "layers". Since only the first and last layers are ever observed, the intermediate layers are "hidden layers". For each layer, $y_{i+1} = f_i(W_i y_i + b_i)$ where y_i is the "activation" of the ith layer. We optimize the weight matrices W_i and bias vectors b_i to minimize the error on the training set.

Deep models (e.g. neural nets with many hidden layers) learn multiple levels of representations of their input. Deep architectures are compositional and hierarchical. The best model depth depends on the problem and the training algorithm which is why it is essential to experiment with more than one hidden layer. Even if accuracy gains with additional layers are not possible for a particular problem, since deep architectures can trade breadth for depth and vice versa, a more efficiently parameterized solution may be possible.

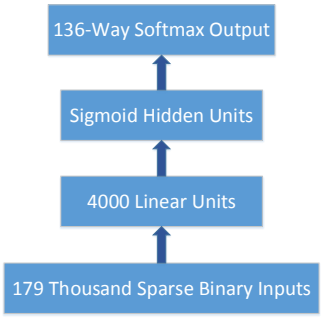

Fig. 2. Proposed Neural Network Architecture for Malware Classification Training

Method	Test Error	Test Two-Class Err	FPR	FNR	Training Time (min)
Logistic Regression All Features	11.7%	0.86%	1.49%	0.60%	1872
Logistic Regression Random Projections	12.37%	1.27%	2.41%	0.80%	105.7
One-Layer Neural Network without Pre-training	9.53%	0.49%	0.83%	0.35%	167.1
One-Layer Neural Network with Pre-training	9.76%	0.50%	0.90%	0.34%	287.0
Two-Layer Neural Network without Pre-training	9.55%	0.50%	0.85%	0.35%	244.0
Two-Layer Neural Network with Pre-training	9.83%	0.54%	0.98%	0.36%	402.3
Three-Layer Neural Network without Pre-training	9.74%	0.51%	0.87%	0.36%	215.0

Table 1. Best Results for Different Malware Classification Algorithms.

3. EXPERIMENTAL RESULTS

We explored a large variety of classifier configurations in order to analyze the effect of different choices of random projection dimensions and other neural network hyper-parameters. The results in Table 1 summarize the performance of the best configurations on a separate, held out test set of over 1.1 million labeled file samples. Given the large number of examples in the training and test sets, we use a held-out test set instead of cross-validation in order to sweep the parameter space. For training, 10% of the 2.6 million examples were used for validation and the systems were trained with 40 epochs. The mini-batch size for the results in Table 1 is 256 samples. The size for all hidden layers for the best one-, two-, and three-hidden layer neural networks are 1536, 2048, 1024, respectively. In the neural network results, we used momentum for training with a value of 0.9, and the learning rate was set to 0.03. The number of random projection is set to 4000 which consistently performed well. The "Test Error" provides the total error for all classes where misclassifying a family is considered to be an error. For the two-class error, an example is only selected to be an error if the file was labeled as some form of malware and was predicted to be benign, or vice versa. Columns 3 and 4 provide the False Positive Rate (FPR) and False Negative Rate (FNR). The last column is the training time in minutes using a C2075 NVIDIA GPU.

Seven different algorithms are considered in Table 1. *Logistic Regression All Features* trains a simple, softmax logistic regression model using all of the input features without performing random projections or using any hidden layers. The second algorithm, *Logistic Regression Random Projections*, trains the system in Figure 2 without any hidden layers. It does include random projections of size 4000 and the softmax layer. Rows 3, 5, and 7 implement the architecture in Figure 2 exactly with one, two, and three hidden layers, respectively. Rows 4 and 6 implement the one- and two-layer neural networks, but also include an additional pre-training stage often used in deep learning architectures which attempts to initialize the hidden layer weights before performing the standard neural network back-propagation training with stochastic gradient descent in order to better learn the true underlying function. In our work, we implemented the pre-training using a Gaussian-Bernoulli restricted Boltzmann machine (RBM), since after the random projection step the input is no longer binary.

From Table 1, we see that the one-layer net without pre-training offered the best two-class test error rate, but the test errors for the one-layer net with pre-training and two-layer net without pre-training are not statistically worse. The best three-layer neural network performed slightly worse than both the one- and two-layer networks. Clearly adding more hidden layers does not help with this problem. With the exception of the FNR for the one-layer neural network with pre-training, all error metrics are slightly worse for the pre-trained nets compared to the versions without pre-training. In this case, pre-training did not provide any gains for models with 1-2 relatively wide hidden layers. Also, doing pre-training after the random projections is not something that we would expect to be helpful since it requires using Gaussian-Bernoulli RBMs which are more unwieldy and might incorrectly learn artificial correlations introduced by the feature mixing of the random projections. The test error (i.e. the malware family error rate) is significantly worse than the two-class error rate for malware versus benign files. The reason is because many times separate malware families were derived from the same original code and the malware classifier mispredicts one family for the other. In addition to the results in Table 1, we were able to further reduce the best, two-class error rate from 0.50% to 0.42% with a voted ensemble of 9 neural networks. At a false positive rate of 0.01%, the best single neural network yields a false negative rate of 25% approaching the goals outlined in the introduction.

Instead of using random projections, we also tried principal component analysis (PCA) to reduce the dimensionality of the input vector. The challenge with PCA is first computing the singular value decomposition (SVD) on the data's covariance matrix in order to determine the largest eigenvectors. With our dataset, we were only able to run a randomized PCA algorithm that estimated the first 500 principal components due to the $\mathcal{O}(N^3)$ computational requirements of exact PCA. The resulting two-class error rate for this system was 0.75% for a one-layer neural network with 768 hidden units which is significantly worse than the comparable architectures utilizing 4000 random projections.

Figure 3 plots the two-class error rate of logistic regression for different random projection sizes. As more dimensions get used, the classifier has more learned weights and more information about the input and performance improves. However, there are diminishing returns and error starts to level off around 8000 dimensions. The results from Figure 3 justify our decision not to try more than 8000 random projection dimensions in our neural net experiments. Increasing the number of random projection dimensions increases the capacity of the model by giving it more tunable parameters and also provides more information about the data. These two effects are

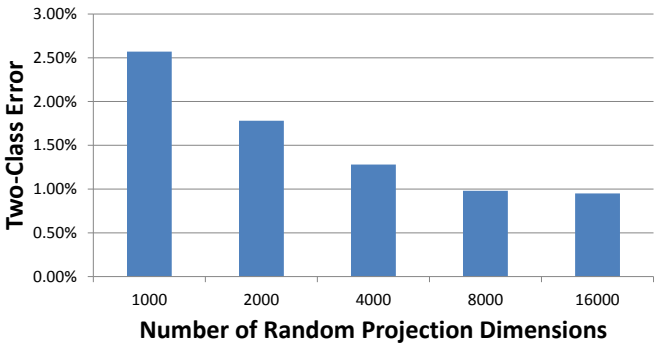

Fig. 3. Error Rates for Logistic Regression with Different Random Projection Sizes.

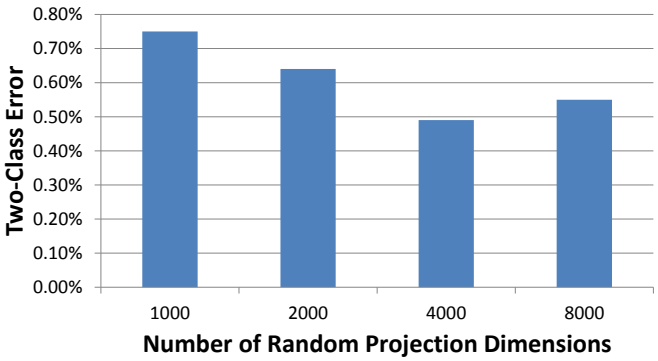

Fig. 4. Error Rates for Neural Network with Different Random Projection Sizes.

tightly coupled for logistic regression compared to neural nets with variable numbers of hidden units so we expect the optimal number of random projection dimensions for a neural net to be at most the number that was optimal for logistic regression. Indeed the results in Figure 4 show 4000 random projection dimensions to be slightly better than 8000 for the neural net models we tried.

We found that our results were relatively insensitive to the number of hidden units in the hidden layer of the neural networks. As long as the hidden layer was sufficiently large, we were able to achieve very good results. The best neural net took about 3 hours to train on the full 2.6 million case training set.

4. RELATED WORK

Given the potential risks associated with an infection, malware classification has been a very active research area. A recent summary of this research is compiled by Idika and Mathur [1]. Several authors have used neural networks for static malware classification [9, 10, 11]. In addition to using different features, these earlier efforts did not use random projections as an initial dimensionality reduction stage. More recently, several authors [6, 7] have been using dynamic analysis to detect malware by analyzing the behavior of unknown executables running in a virtual machine. Similar to the API tri-gram features used in this work, Mehdi *et al.* [12] used N-grams of system calls for malware classification. Using an ensemble of malware classifiers has been proposed by Menahem *et al.* [13].

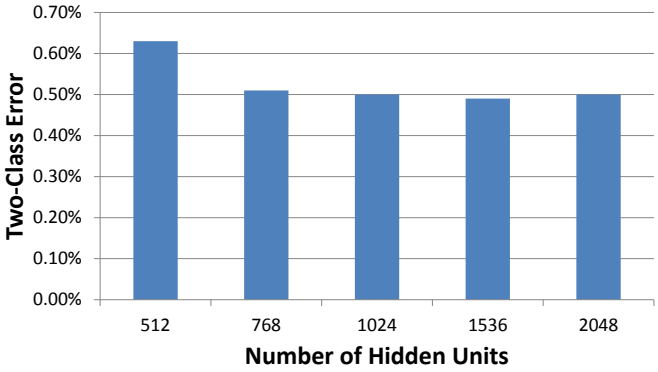

Fig. 5. Error Rates for Neural Networks with Number of Hidden Units.

Very sparse random projections which are used in Section 2 are discussed in [4, 5]. Artificial neural networks became a popular classifier in the late 1980's after the invention of the backpropagation algorithm [14]. However, due to limits in computational power at that time and the difficulty of training deep networks, researchers used mostly narrow and shallow networks. In 2006, Hinton et al. [15] proposed a technique to initialize a deep network layer-by-layer by treating each pair of layers as a restricted Boltzmann machine, leveraging unlabeled data to regularize the neural network and to help learn features. This technique, aided by the modern computing facilities, sparked a resurgence of interest in applying deeper and wider networks to solve real world problems, even without pre-training. A notable example is the successful application of the deep neural networks in large vocabulary speech recognition [16], with one third of the error cut from the conventional systems [17].

5. CONCLUSION

In this paper, we present a novel, large-scale malware classification system which utilizes random projections to reduce the input space by a factor of 45 (179K/4K) allowing training of more complex supervised classification algorithms. Neural networks trained on random projections provide a 43% reduction in the error rate compared to the baseline logistic regression system using all the features. We believe the 0.49% two-class error rate for the one-layer neural network with random projections and 0.42% two class error rate for the ensemble of neural networks offer state-of-the-art performance. Utilizing a GPU for training with 2.6 million examples is fast requiring slightly less than three hours. Neural networks are not dramatically slower than logistic regression at test time.

We are not able to obtain a significant accuracy gain by adding additional hidden layers. The data is almost linearly separable (0.86% two-class error for logistic regression) in the original input space. The two and three, hidden layer models perform slightly worse compared to the one-layer neural network. We believe the reason for the degraded performance is that there are not enough errors to learn additional layers well. In our system, we only have 5000 training errors out of 2.6 million training cases for the one-layer neural network. The system also did not benefit by employing pre-training which is often used in deep neural network architectures.

Acknowledgments: We thank Mady Marinescu and Anil Thomas for collecting the labeled dataset and Ping Li for helpful discussions.

6. REFERENCES

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

[2] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. 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.

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

[4] Ping Li, Trevor J. Hastie, and Kenneth W. Church, "Very sparse random projections," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2006)*, 2006, pp. 287–296.

[5] Ping Li, Trevor J. Hastie, and Kenneth W. Church, "Margin-constrained random projections and very sparse random projections," in *Proceedings of the Conference on Learning Theory (COLT)*, 2006, pp. 635–649.

[6] Ulrich Bayer, Christopher Kruegel, and Engin Kirda, "TTAnalyze: A tool for analyzing malware," in *Proceedings of 15th Annual Conference of the European Institute for Computer Antivirus Research (EICAR)*, 2006.

[7] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'07)*, 2007, pp. 231–245.

[8] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schutze, *An Introduction to Information Retrieval*, Cambridge University Press, 2009.

[9] Jeffrey O. Kephart, Gregory B. Sorkin, William C. Arnold, David M. Chess, Gerald J. Tesauro, and Steve R. White, "Biologically inspired defenses against computer viruses," in *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1995, pp. 985–996.

[10] Gerald Tesauro, Jeffrey O. Kephart, and Gregory B. Sorkin, "Neural networks for computer virus recognition," in *IEEE Expert*, 1996, vol. 11, pp. 5–6.

[11] William Arnold and Gerald Tesauro, "Automatically generated win32 heuristic virus detection," in *Proceedings of the 2000 International Virus Bulletin Conference*, 2000, pp. 5–6.

[12] 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.

[13] Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici, "Improving malware detection by applying multi-inducer ensemble," in *Proc. Computational Statistics & Data Analysis (CS&DA)*, 2009, vol. 53, pp. 1483–1494.

[14] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, "Learning representations by back-propagating errors," 1986, vol. 323, pp. 533–536, London.

[15] Geoffrey E. Hinton, Simon Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," 2006, vol. 18, pp. 1527–1554, MIT Press.

[16] George E. Dahl, Dong Yu, Li Deng, and Alex Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," 2012, vol. 20, pp. 30–42, IEEE.

[17] Frank Seide, Gang Li, and Dong Yu, "Conversational speech transcription using context-dependent deep neural networks," in *Proceedings of Interspeech*, 2011, pp. 437–440.