Combining Restricted Boltzmann Machine and One Side Perceptron for Malware Detection

Răzvan Benchea[1,2] and Dragoş Teodor Gavriluţ[1,2]

[1] "Alexandru Ioan Cuza" University, Faculty of Computer Science, Iaşi, România
[2] Bitdefender Laboratories, Iaşi, România

Abstract. Due to the large increase of malware samples in the last 10 years, the demand of the antimalware industry for an automated classifier has increased. However, this classifier has to satisfy two restrictions in order to be used in real life situations: high detection rate and very low number of false positives. By modifying the perceptron algorithm and combining existing features, we were able to provide a good solution to the problem, called the one side perceptron. Since the power of the perceptron lies in its features, we will focus our study on improving the feature creation algorithm. This paper presents different methods, including simple mathematical operations and the usage of a restricted Boltzmann machine, for creating features designed for an increased detection rate of the one side perceptron. The analysis is carried out using a large dataset of approximately 3 million files.

1 Introduction

During the last years 10 years, malware landscape has changed dramatically. With more than 200.000 [1] samples appearing daily it is impossible for anyone to analyze them by hand. That is why an automated malware classification algorithm is needed. However, in order to be practical, this algorithm needs to satisfy the market demands: it needs to perform fast during testing and it also needs to have an extremely low false positive rate. The later restriction is based on the fact that a clean file (for example an operating system file) that is classified as infected and later deleted can make the system impracticable.

Based on these restrictions we have previously developed an algorithm, called One Side Perceptron (OSP) [2], that can be trained with zero false positive. In order to increase its speed we have only used a small number of features, selected with F2 [3] score from all available. Even though this method achieves a good detection rate we believe it can be improved by modifying the way it uses the features. Since the algorithm is based on linear classifier it doesn't try to find any relations between the features.

In this paper we will use a neural network, more exactly a restricted Boltzmann machine [4], to create a new set of features from existing ones. We will use these to train an OSP algorithm and will compare the results with those obtained from training the algorithm with the best selected features according to F2 score. Since the restricted Boltzmann machine is a probabilistic network,

N. Hernandez et al. (Eds.): ICCS 2014, LNAI 8577, pp. 93–103, 2014.
DOI: 10.1007/978-3-319-08389-6_9,

the features can be outputted as a probability value or as a sample generated based on this probability. During testing we have observed that using a sample instead of a probability does not yield good results, so in this paper we will refer to the second method.

This paper is organized in 3 sections. Section 2 describes other methods used to solve the problem as well as their advantages and disadvantages. Section 3 starts with the theory behind the algorithm and continues with describing our method of training, testing and results. Finally, in section 4 we present our conclusions and future work.

2 Related Work

A good categorization of malware detection techniques was carried out by the authors in [5]. Apart from setting two main categories, anomaly-based and signature-based, the authors also split the detection algorithms by the method they use to collect information: static, dynamic and hybrid. The anomaly-based technique has the advantage of detecting types of malware that were not present in the training set, while the static method of extracting features can be used in very large data sets. The authors in [6] were the first to use Machine Learning techniques for the detection of different malwares based on their binary codes. Using this method they were able to create a proactive malware detector that was twice as good as signature based one in detecting new samples.

Most of the research regarding malware detection was carried out using features obtained from extracting n-grams patterns from binary files or from executed API calls. Authors in [7] have used common n-gram in order to create a profile for each class of malware then assigned a new sample to the class using k-nearest neighbor. Using n-grams authors in [8] have tested a variety of inductive methods, including naive Bayes, decision trees, support vector machines(SVM), and boosting and concluded that boosted decision trees outperform other methods (in regard with the true positive detection). On the other hand, by comparing different feature selection and classifiers, authors in [9] have concluded that support vector machines are superior in terms of prediction accuracy. Using different classifiers on variable length n-grams, authors in [10] managed to obtain a 98.4% detection rate with 1.9% false positive. Even though one might consider that 1.9% is an acceptable false positive rate, it is still too big to be used in industry. The authors in [11] used features based on n-grams of opcodes obtained after disassembling the file. They achieved a 95% detection rate and a 0.1% false positive. However, the dataset size is quite small (26093) and was obtained after removing packed or compressed files from the database.

Another method frequently found in malware detection research is to extract API calls from the binary files, apply some filtering algorithm, and use them as features in a classifier. However, most of the time, these features are extracted by executing the file in a controlled environment, like a virtual machine. Even though this method extracts some features that could not be extracted by other methods it is impractical when used in large datasets due to the amount of

time required to run each sample. The authors in [12] observed this problem and applied an algorithm, called fuzzy pattern recognition, that was initially developed by [13] to work on dynamic extracted features. Even so, the test was still carried out on a very small dataset (120 benign and 100 malicious). By assuming that api calls and string provide important semantics and can show the executable intent, the authors in [14] have used this information to create features. After using a Max-Relevance algorithm to keep the most 500 important features a SVM ensemble was trained. This method achieved a 92% detection rate and very few false positives. Using api call as features, selecting only the relevant ones and using a SVM as a classifier was also addressed by [15]. A very high detection rate (99%) was achieved by statically extracting api calls, selecting important ones based on their information gain, and using them to train a decision tree. By statically collecting information from file headers as well as the dlls and api calls that a sample might use, the authors in [16] created 25592 features from which only 2450 were selected. Even though the detection rate was 99.6%, the false positive rate was 2.7% and the data set used was not relevant for a real life situation since it contained 20 times more malicious samples than clean.

A similar work to ours was presented in a recent paper [17]. The authors first reduced the number of features using principal component analysis or random projection and then trained a system made up of a neural network and a logistic regression algorithm. The authors tested a neural net with several hidden layers and stated that the one with a single hidden layer provides the best two-class error rate as well as the lowest false positive rate (0.83%). The main difference from their work is how false positives are treated. Instead of developing an algorithm that best classifies a collection of samples and then applies several others (9 in the above paper) to reduce the number of false positives, we use a classifier that obtains zero false positives during training. Even though the detection rate is smaller, we consider usage of (OSP)[2] for practical purposes because of its low false positives.

3 Framework Architecture

In anti-malware industry, having a false positive is considered to be worse than a lower detection rate. The main argument is that a false positive may affect some system or personal files. Since the default action for most anti-malware is to disinfect the infected component, if a false positive appears there is a risk of deleting personal files or removing system files that may lead to a full system shut down. Having this limitation means that some machine learning algorithms (such as support vector machine, neural networks and so on) cannot be used because of the high risk of false positives. There are several solution for this problem; the one that we have adopted is to have a 2 layer filter; the first one is a type of a neural network, called restricted Boltzmann machine, while the second layer consists in usage of (OSP) [2] algorithm (Algorithm 1).

The purpose of the neural network is to increase detection rate and lower the number of false positives by finding correlations between features and transforming them from discrete values to continuous. The OSP has a similar role: to find the best hyper plane that gives no false positives. The reason for choosing a restricted Boltzmann machine is because it has proved successful in other fields like image classification, video action recognition and speech recognition [18],[19],[20] and because it has a structure that permits learning to be done in parallel, making suitable for newer hardware that can perform distributed calculations, like a graphical processing unit(GPU).

The entire framework is constructed in such a way that the hidden layer of the neural network will make up the new set of features for the perceptron. The neurons from the visible layer are initialized with the original features (Fig1) . By doing so, we add a small performance overhead , but we are able to train the classifier with new features that are created in a non-linear way from the original ones. The usage of this method brings another advantage. Even though original perceptron features are boolean values, they can be transformed by the neural network in continuous ones by using outputted probability of the restricted Boltzmann machine as feature for the perceptron.

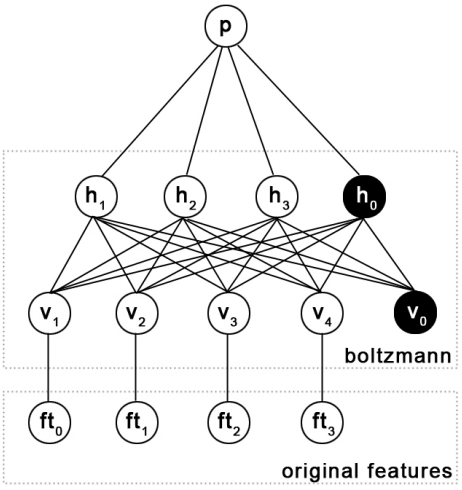

Fig. 1. Neural network architecture

To better understand how the algorithms that make our framework work, we will describe them in the following two subsection.

3.1 One Side Perceptron (OSP)

The OSP algorithm was design to ensure that the number of false positives from the training step is zero, while increasing the detection rate. Having a 0 false positive rate in the training step ensures a very low false positive on a

different database. The algorithm adds a secondary step in the training function by retesting the model only on the clean files until every clean file is correctly classified.

To facilitate writing of the algorithm, we consider these abbreviations:

1. $Record \rightarrow R$
2. $Features \rightarrow F$
3. $Label \rightarrow L$
4. $maxIterations - maxI$
5. $SingleSetClass - S - all\ records\ belonging\ to\ a\ specific\ class$

Algorithm 1. One Side Class perceptron - OSC1

$w \leftarrow [w_1, w_2, \ldots, w_m], w_i \leftarrow 0, i \leftarrow \overline{1,m}$;
$b \leftarrow 0; \lambda \leftarrow a\ learning\ rate$
$iteration \leftarrow 0$
$S \leftarrow S \bigcup_{i=1}^{|R|} R_i, where\ R_i.L\ =\ benign\ files$
repeat
  $TrainFunction(R, w, b)$
  repeat
    $TrainFunction(S, w, b)$
    $errors \leftarrow 0$
    for $i\ =\ 1 \rightarrow |S|$ **do**
      if $(not\ IsCorrectClassified(S_i, w, b))$ **then**
        $errors \leftarrow errors + 1$
      end if
    end for
  until ($errors = 0$)
  $iteration \leftarrow iteration + 1$
until $(iteration \geq maxI) or (every\ record\ is\ correctly\ classified)$

The algorithm ends when a specific number of iterations is reached or when every records is correctly calcified. In practice, even after removing most of the noise from a database, the second criteria (every record is correctly clarified) never happens. The maximum number of iterations is usually something determined dynamically (for example, if no growth in detection rate was observed during the last iterations). It is better to stop the algorithm in such situation in order to avoid overfitting. However the tests in this paper were made with a constant number of iterations (derive from the previous observation) so that we can properly compare the results.

While this approach provides a small number of false positives, it also reduces the detection rate significantly. Even though algorithms are not designed to be used alone in an anti-malware products but as a complementary solution to a more deterministic set of algorithms (signature based detection, automata,

Algorithm 2. Train Function

function $Train(Records,\ w,\ b)$
  for $i = 0 \rightarrow |Records|$ **do**
    if $Records_i.L \times ((\sum_{j=1}^{m} Records_i.F_j \times w_j) + b) \leq 0$ **then**
      for $j = 1 \rightarrow m$ **do**
        $w_j \leftarrow w_j + Records_i.L \times \lambda \times Records_i.F_j$
      end for
      $b \leftarrow b + Records_i.L \times \lambda$
    end if
  end for
end function

etc.), the increase of detection for such an algorithm usually translates in a better proactivity [21] for the product.

There are several methods that can be used to increase the detection rate of such an algorithm. The simplest one is to perform a better feature selection for the attributes that will be used. Another one is to combine different features to create a new set that would produce a better linear separability for the test data set. The secondary approach is called mapping and consists in creating a new set of features by combining the initial features one with each other(Alg 3:

1. sort the entire set of existing feature based on a specific score (usually F2)
2. choose the first n
3. combine every selected feature with the other ones using the AND operator

Algorithm 3. MapAllFeatures algorithm

function $MapAllFeatures(R)$
  $newFeatures \leftarrow \phi$
  for $i = 1 \rightarrow |R.F|$ **do**
    for $j = 1 \rightarrow |R.F|$ **do**
      $newFeatures \leftarrow newFeatures \cup R.F_i \otimes R.F_j$
    end for
  end for
end function
where $\otimes$ can be any operator. using AND we obtained the best results.

When choosing the value of n in the previous algorithm, some practical restriction should be taken under consideration. The first is that if n is too large, than the weight vector may be too big for a product update (usually an anti-malware product tries to have small updates less than 15-20Kb). If n is too small, than in some cases clean and malware files will have the same set of features. Obviously, for that algorithm to work, these samples should be excluded; to be able to preserve the low false positive rate, usually the excluded samples are the malware ones so that we can be sure we would not detect any clean file.

This means that a small n translates in a lower detection rate. Finally, because the mapped algorithm produces $n \times (n+1)/2$ features, the n should be chosen carefully so that the total number of newly created features is small enough for a product update. Even though using a mapped features does help to increase the detection rate, not being able to use too many of them is a strong limitation. The next logical step was to search for another way of obtaining a new set of features.

3.2 Restricted Boltzmann Machine

A restricted Boltzmann machine is a stochastic neural network made up of two layers: one of visible and one of hidden units. It is called restricted because of its bipartite structure such that each visible neuron is connected to each hidden neuron and each hidden neuron is connected to each visible one. There are no connections between elements on the same layer. Figure 2 shows a restricted Boltzmann with 4 visible units, 3 hidden units and 2 bias units. By using these restrictions, each neuron can be learnt in parallel making the model suitable for training on distributed platforms, like GPUs.

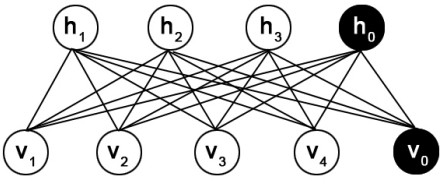

Fig. 2. Restricted Boltzmann Machine

The visible and hidden units have binary states {0, 1} and interconnected by a symmetric weight matrix $\{w\}$. Let v_i and h_j represent the states of visible and hidden unit i and j. The state probabilities of the units are:

$$p(h_j = 1|v) = \frac{1}{1 + e^{-\sum\limits_{v_i \in v} v_i * w_{ij}}} \tag{1}$$

$$p(v_i = 1|h) = \frac{1}{1 + e^{-\sum\limits_{h_j \in h} h_j * w_{ij}}} \tag{2}$$

The main objective of this network is to learn binary features that capture high-order structure in the data (visible units). The most often used algorithm for training a restricted Boltzmann machine is contrastive divergence [22]. In contrastive divergence learning, the training is done in two steps. The first step consists in obtaining an initial pair of visible and hidden units based on data. It works by initializing the visible units with a datum (v_i) from the training set and obtaining the set of initial hidden units (h_j) by sampling each neuron according

to the probabilities obtained with equation (1).The next step requires to obtain another pair of visible and hidden units ($\hat{v}_i$ and $\hat{h}_j$) but this time the visible units are generated using the previous generated hidden units using equation 2. The contrastive divergence update equation is:

$$\Delta w_{ij} = \lambda(< v_i h_j > - < \hat{v}_i \hat{h}_j) \tag{3}$$

where $< \cdot >$ refers to the mean over the training data.

4 Configuration and Results

We have collected clean and malicious files for a period of two month (January and February 2014). For each file collected we have ve extracted a set of features. Most of the features ar boolean values and indicates a specific behavior for that particular file (for example: there are features that indicate network activity, Windows registry alteration, file deletion and other). We have also extracted some features that define a value that is specific to a file (for example: file size, file entropy, number of sections, file entry point position, etc). These files were also used to create some boolean features (for example, if a file has more than 10 sections).

Our initial database consists in 3087200 files (clean and malicious) with 3299 features extracted for each record. One problem that we have to deal with is the presence of noise in the database. There are several reason for why this happens. The most common one are the grayware file. The most known grayware files are considered to be adware, spyware, keygens, craks and so on. While their malicious behavior is debatable, they lack the standard components that a malware has (such as a packer, an obfuscated code, etc). Without at least some of this components and no specific dynamic behavior it is really difficult to distinguish such a file from a clean one. Another similar case are file infectors. In this case, the malware infects a clean file. However, when the system extracts the features it will also extract a lot of features from the host file as well, leading to a set of features that is very close (and sometimes identical for small file infectors) to the original clean file.

All of this cases had to be filtered out. After this step we computed the F2 score for each feature and we only kept the first 300 features. This was done to create a model that can be used in practice. Having a large number of features means that one needs to compute all of them to be able to evaluate a proposed model since scanning a file should not have a big impact on system performance we have to reduce the number of features that we compute. If we are to keep only 300 features, some of the records that in the original database had a different set of features, will now be identical (from the features set point of view). In this cases, only one record was kept in the database. If we found cases where records that have the same set of features have a different label we decided to keep only one that is labeled as a clean file (this ensures that our system will not produce false positives). While this measure will reduce the detection rate and there is a chance that some records that are consider to be malicious to be labeled as clean, the practical purpose of

this algorithm is first to provide a very low false positive rate and only then to increase the detection rate. If we were to remove the clean records, there is a high probability to get higher false positive. Removing both records (the clean and the malicious one) has a smaller chance of false positives but still a chance. The only way to be sure that the clean file with the same set of features as the malware one is correctly classified is to change the label of the malware one to clean. In theory, both of the records can be manually analyzed and the labels to be properly decided. However, in practice and in particular when dealing with large databases, analyzing hundreds of this pairs is not feasible. At the end of this iteration, the final database had 1260543 files (31507 malicious files and 1229036 clean files) with 300 features for each record.

In order to see the improvement brought by the use of the neural network we used as baseline the results obtained by the OSP and the ones obtained by OSP-mapped algorithm using a 3-fold validation method. All of the algorithms were trained using 10000 iterations.

The training and testing was carried out using a system with 24 Gb of Ram, Intel Xeon CPU E5-2440 (2.4 GHz) and Windows Server 2008 R2. Training of the Boltzmann machine was done on a Ubuntu 12.0 system, using the GPU of an ATI Radeon 7970. Development and training on the GPU was done using the OpenCL [23] framework. Since OpenCL doesn‘t come with a random number generator that can be used on a distributed framework, we have used the one provided by David Thomas [24]. The learning rate used in training was 0.1 for the BoltzMann machine and 0.01 for the OSP.

Table 1 shows the results obtained using these 3 algorithms. We were in particular interested in the low false positive rate. It should be mentioned that none of these algorithm were to be used by themselves for malware detection but as a complementary method to a more deterministic one (signature detections). OSP is the one side perceptron algorithm tested on the database with 300 features, OSP-MAP is the same algorithm but with all of the 300 features mapped and OSP-RBM is the one side perceptron algorithm tested on the same database where the 300 features were built using a restricted Boltzmann machine from the original ones.

Even though the OSP-MAP algorithm obtained a slightly better detection rate than OSP-RBM, the number of false positives is very high. This is why we consider the OSP-RBM to be better when it comes to be used in industry. The difference of 1.35% in detection rate can be supported by other deterministic algorithms.

Table 1. The results of the 3-fold cross-validation for OSP, OSP-MAP, OSP-RBM algorithms

Algorithm	Detection	False Positive	Accuracy
OSP	73.11%	20 (0.00162%)	99.32 %
OSP-RBM	88.83%	3 (0.00024%)	99.72 %
OSP-MAP	90.18%	87(0.00707%)	99.72 %

5 Conclusions and Future Work

By using a restricted Boltzmann machine to bring relation between features we have managed to get very close to our goal in obtaining a zero false positive classifier that can have a good detection rate. However we still believe there are some things left to test.

There are different configurations of the training network that we didn't test and could improve the classification score. First, we have only used contrastive divergence in one step, but this could be changed to an arbitrary number. Second, different algorithms could be tested to train the restricted Boltzmann machine, one of them being persistent contrastive divergence. We also believe that by using more layers of restricted Boltzmann machine could also improve detection. However, this method might come with the cost of more memory usage and an increased testing time.

One of the most important factor that restricts the detection rate is the fact that the algorithm is trained for zero false positive. By relaxing this, and allowing for a few false positives that could be filtered by other methods, a better detection rate could be achieved.

References

1. www.av-test.org: `http://www.av-test.org/en/statistics/malware/`
2. Gavrilut, D., Benchea, R., Vatamanu, C.: Optimized zero false positives perceptron training for malware detection. In: SYNASC, pp. 247–253. IEEE Computer Society (2012)
3. Chen, Y.W., Lin, C.-J.: Combining SVMs with various feature selection strategies. In: Guyon, I., Nikravesh, M., Gunn, S., Zadeh, L.A. (eds.) Feature Extraction. STUDFUZZ, vol. 207, pp. 315–324. Springer, Heidelberg (2006)
4. Paul, S.: Information processing in dynamical systems: Foundations of harmony theory. Parallel Distributed Processing: Explorations in the Microstructure of Cognition 1, 194–281 (1986)
5. Idika, N., Mathur, A.P.: A survey on malware detection techniques. PhD thesis. Purdue University (February 2007)
6. Schultz, M.G., Eskin, E., Zadok, E., Stolfo, S.J.: Data mining methods for detection of new malicious executables. In: IEEE Symposium on Security and Privacy, pp. 38–49. IEEE Computer Society (2001)
7. Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R.: N-gram-based detection of new malicious code. In: COMPSAC Workshops, pp. 41–42. IEEE Computer Society (2004)
8. Kolter, J.Z., Maloof, M.A.: Learning to detect and classify malicious executables in the wild. Journal of Machine Learning Research 6, 2721–2744 (2006)
9. Cai, D.M., Gokhale, M., Theiler, J.: Comparison of feature selection and classification algorithms in identifying malicious executables. Computational Statistics & Data Analysis 51(6), 3156–3172 (2007)
10. Siddiqui, M.A.: Data mining methods for malware detection (2008)
11. Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., Elovici, Y.: Detecting unknown malicious code by applying classification techniques on opcode patterns. Security Informatics 1(1), 1–22 (2012)

12. Hung, T.C., Lam, D.X.: A feature extraction method and recognition algorithm for detection unknown worm and variations based on static features (2011)
13. Zhang, B., Yin, J., Hao, J.: Using fuzzy pattern recognition to detect unknown malicious executables code. In: Wang, L., Jin, Y. (eds.) FSKD 2005, Part I. LNCS (LNAI), vol. 3613, pp. 629–634. Springer, Heidelberg (2005)
14. Ye, Y., Chen, L., Wang, D., Li, T., Jiang, Q., Zhao, M.: Sbmds: an interpretable string based malware detection system using svm ensemble with bagging. Journal in Computer Virology 5(4), 283–293 (2009)
15. Dai, J., Guha, R.K., Lee, J.: Efficient virus detection using dynamic instruction sequences. JCP 4(5), 405–414 (2009)
16. Baldangombo, U., Jambaljav, N., Horng, S.J.: A static malware detection system using data mining methods. CoRR abs/1308.2831 (2013)
17. Dahl, G.E., Stokes, J.W., Deng, L., Yu, D.: Large-scale malware classification using random projections and neural networks. In: ICASSP, pp. 3422–3426. IEEE (2013)
18. Lee, H., Grosse, R.B., Ranganath, R., Ng, A.Y.: Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In: Danyluk, A.P., Bottou, L., Littman, M.L. (eds.) ICML. ACM International Conference Proceeding Series, vol. 382, p. 77. ACM (2009)
19. Taylor, G.W., Fergus, R., LeCun, Y., Bregler, C.: Convolutional learning of spatio-temporal features. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part VI. LNCS, vol. 6316, pp. 140–153. Springer, Heidelberg (2010)
20. Rahman Mohamed, A., Dahl, G.E., Hinton, G.E.: Acoustic modeling using deep belief networks. IEEE Transactions on Audio, Speech & Language Processing 20(1), 14–22 (2012)
21. Cimpoesu, M., Gavrilut, D., Popescu, A.: The proactivity of perceptron derived algorithms in malware detection. Journal in Computer Virology 8(4), 133–140 (2012)
22. Hinton, G.E.: Training products of experts by minimizing contrastive divergence. Neural Computation 14(8), 1771–1800 (2002)
23. Khronos group, `http://www.khronos.org/opencl/`
24. Thomas, D.: `http://cas.ee.ic.ac.uk/people/dt10/research/rngs-gpu-mwc64x.html`