Applying deep learning on packet flows for botnet detection

Applying Deep Learning on Packet Flows for Botnet Detection

Jos van Roosmalen
Open University of the Netherlands
Heerlen, The Netherlands
jos.van.roosmalen@ieee.org

Harald Vranken
Open University of the Netherlands
Heerlen, The Netherlands
Radboud University
Nijmegen, The Netherlands

Marko van Eekelen
Open University of the Netherlands
Heerlen, The Netherlands
Radboud University
Nijmegen, The Netherlands

ABSTRACT

Botnets constitute a primary threat to Internet security. The ability to accurately distinguish botnet traffic from non-botnet traffic can help significantly in mitigating malicious botnets. We present a novel approach to botnet detection that applies deep learning on flows of TCP/UDP/IP-packets. In our experimental results with a large dataset, we obtained 99.7% accuracy for classifying P2P-botnet traffic. This is comparable to or better than conventional botnet detection approaches, while reducing efforts for feature engineering and feature selection to a minimum.

CCS CONCEPTS

• **Security and privacy** → **Intrusion detection systems**; • **Computing methodologies** → *Machine learning algorithms*; • **Applied computing** → *Computer forensics*;

KEYWORDS

botnet, deep learning, intrusion detection, network security

ACM Reference Format:
Jos van Roosmalen, Harald Vranken, and Marko van Eekelen. 2018. Applying Deep Learning on Packet Flows for Botnet Detection. In *SAC 2018: SAC 2018: Symposium on Applied Computing, April 9–13, 2018, Pau, France*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3167132.3167306

1 INTRODUCTION

A botnet is a network of systems infected by bot malware. The strength of bot malware is, compared to other malware such as Trojans and root kits, the ability to communicate with the attacker, commonly known as a botmaster. Botnets can be used for a variety of activities, such as malware distribution, click fraud, identify and credential theft, bitcoin mining, and distributed denial-of-service (DDoS) attacks.

Botnet architectures have evolved over the past 20 years. The first botnets used a central host for command and control (C&C) with low design complexity and low message latency as advantage. Survivability, however, is also low because the central host is vulnerable to attacks or takedown attempts [3]. To increase resilience, many modern botnets use hybrid peer-to-peer (P2P) architectures.

Various P2P-botnet detection approaches have been studied that mostly detect botnet activity by passive monitoring of the network traffic. They typically apply machine learning algorithms to classify network traffic and rely on feature engineering and feature selection. A significant amount of work and domain knowledge is required to extract high-level network traffic metrics and determine which features from the monitored traffic data to use. This is relaxed in deep-learning methods that aim to learn feature hierarchies by themselves, where higher-level features are gradually learned by composing lower-level features [7]. Deep learning has improved pattern recognition significantly for image and speech recognition [16, 23]. In addition to the advances in machine learning, lower costs of processing hardware and increased processing power (mainly in GPUs) have contributed to the popularity of deep learning [12].

In botnet traffic also a hierarchy of features can be observed, ranging from low-level features such as header fields in network packets, to high-level features such as characteristics of protocols and network flows. Hence, it is plausible that deep learning may also provide good results for classifying botnet traffic. In this paper, we explore to what extent this indeed holds. To the best of our knowledge, our study is the first attempt to apply deep learning for botnet detection. Since modern botnets typically apply P2P-architectures, we focus on the detection of P2P-botnets.

We executed more than 650 experiments with both deep neural networks (DNNs) and ladder networks, using a large dataset that contains a mix of network traffic including data from P2P-botnets. In our experiments with DNNs we explored supervised training as well as unsupervised pre-training with stacked denoising auto-encoders (SDAs). In our experiments with ladder networks we applied semi-supervised learning, combining unsupervised and supervised training. In the experiments we explored numerous network configurations, algorithms, and parameter settings.

Our main contribution as presented in this paper is that we apply a novel approach for botnet detection using deep learning. Our deep-learning approach uses TCP/UDP/IP-packet flows as inputs, from which features are extracted automatically by the network. Hence, our approach does not need up-front feature engineering or feature selection based on data analysis or botnet network characteristics. A second contribution of our work is that we experiment

SAC 2018, April 9–13, 2018, Pau, France

ACM ISBN 978-1-4503-5191-1/18/04…$15.00
https://doi.org/10.1145/3167132.3167306

with various configurations of DNNs and ladder networks and compare how they perform. These results are not only valuable for researchers involved in botnet detection, but also for researchers looking at methods and application of deep learning.

In the remainder of this paper, we first provide an overview of related work in section 2. In section 3 we explain the basics of DNNs and ladder networks. In section 4 we describe our experiments and the datasets used in detail. We present and discuss our results in section 5. In sections 6 and 7, we outline future work and provide concluding remarks respectively.

2 RELATED WORK

Several deep-learning studies have been published for network protocol classification and network intrusion detection, such as DDoS-attacks [29, 48]. However, to the best of our knowledge, deep learning has not been applied yet for botnet detection. Recent botnet detection studies rely on passive traffic monitoring, where the most common approaches use graph analysis on the core-network level (e.g., Internet service provider or exchange point) or target individual hosts [4, 28, 52]. Our study relies on simple low-level features, whereas other studies primarily used high-level netflow features [51, 52], which are extracted using third-party software or obtained using handcrafted feature extractors [4]. Features are selected or reduced based on data analysis using information gain or principal component analysis (PCA). Another approach is to select features based on general characteristics of P2P-bots, such as the number of failed connections or the ability to filter out successful DNS traffic. Many studies follow the same approach. For instance, Narang et al. [28] applied a conventional machine-learning algorithm (MLA) in PeerShark that is based on four different flow features. Saad et al. [36] also introduced an MLA based on 17 netflow features, and Barthakur et al. [4] compared different conventional MLA classifiers for P2P-botnet detection. In addition, Zhao et al. [52] used decision trees. Not all approaches rely on machine learning. For example, Zhang et al. [51] used a multi-stage approach by first applying a coarse-grained filter followed by a fine-grained filter. Some studies take a different approach to P2P-botnet detection, for instance based on the misuse of TCP-flags [15] and on encrypted C&C message signature matching [35].

3 DEEP LEARNING BACKGROUND

3.1 Deep Neural Networks

A neural network is a network of connected artificial neurons. A neuron has multiple weighted inputs and one output. The output y of a neuron given input vector $\mathbf{x}$, weight vector $\mathbf{w}$, and activation function ϕ can be calculated as follows:

$$y = \phi\left(\sum_{i=0}^{k} \mathbf{w}_i \mathbf{x}_i\right) . \tag{1}$$

The sigmoid function $\phi(x) = 1/(1+\mathrm{e}^{-x})$ is conventionally used as an activation function.

A deep neural network uses several layers of neurons. Figure 1 shows a deep neural network, where the grey circles represent neurons. The input layer only distributes the data (i.e., the activation function is $\phi(x) = x$). The output layer contains neurons whose outputs represent the classification probabilities. One or more hidden layers are placed between the input and output layers. A neural network is called deep if it contains at least three hidden layers. The hidden layers represent the knowledge of the network, and adding hidden layers can create approximate continuous functions that map inputs $\mathbf{x}$ to outputs $\mathbf{y}$. If a layer l of m neurons has an input column vector $\mathbf{x}^{(l)}$, where $\mathbf{x}^{(l)} \in \mathbb{R}^{k\times 1}$, and a weight matrix $\mathbf{W}^{(l)}$, where $\mathbf{W}^{(l)} \in \mathbb{R}^{m\times k}$, then the output $\mathbf{a}^{(l)}$ of this layer can be calculated using fast linear algebra:

$$\mathbf{a}^{(l)} = \phi\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)}\right). \tag{2}$$

Supervised neural networks are trained by feeding forward training items $\mathbf{x}$, calculating the error between the observed output $\mathbf{y}$ and the expected output $\hat{\mathbf{y}}$, and adjusting the weights using backpropagation [49]. This offers a computationally efficient way to calculate the gradient of a cost function C with respect to the weight matrix of each layer l: $\nabla_{\mathbf{W}^{(l)}} C(\mathbf{W})$. The gradient descent method is used to find the minimum by iteratively changing each layer weight matrix $\mathbf{W}^{(l)}$ with the gradient:

$$\mathbf{W}^{(l)}_{t+1} = \mathbf{W}^{(l)}_{t} - \alpha \nabla_{\mathbf{W}^{(l)}_t} C(\mathbf{W}_t), \tag{3}$$

where α is a small value called the *learning rate*.The weights are typically updated after feeding forward several training items (called a mini-batch).

3.2 Stacked Denoising Auto-encoders

An auto-encoder neural network can discover structure in data by applying an unsupervised learning algorithm, where the target label $\hat{\mathbf{y}}$ is not needed [7]. It attempts to learn $f(x) = x$ by encoding the input with constraints (such as limiting the number of neurons) and trying to decode back the original input. The goal is to minimize the reconstruction error between the input and the decoded output.

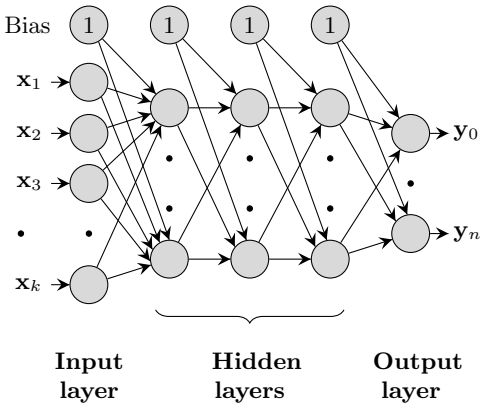

Figure 1: A deep neural network.

Stacked auto-encoders (SAEs) are able to learn higher-order features. Stacked denoising auto-encoders (SDAs) [47] are SAEs that corrupt the input data (e.g., by adding Gaussian noise) to prevent the SAE from discovering uninteresting features, such as simply copying the input.

SDAs are typically used for unsupervised pre-training. First, a SDA network is set up with a number of layers and neurons, and unsupervised training is performed using the training data. Next, a DNN is set up with the same number of layers and neurons. The weights are initialized with the current weights of the SDA network. Finally, the DNN is fine-tuned using supervised training.

3.3 Ladder Networks

Recently, purely supervised learning has provided results that are equal to or better than approaches that involve unsupervised pre-training [10, 23]. According to Valpola [44], the problem is that many unsupervised learning methods (e.g., auto-encoders) attempt to represent as much information about the original data as possible, whereas supervised learning endeavors to filter out information that is irrelevant to the classification task. A solution is the use of a ladder network, which is a combination of a DNN and nested auto-encoders that is able to jointly train both supervised and unsupervised. Unsupervised training is not limited to pre-training. This makes ladder networks relevant for botnet detection, where the amount of labeled data is limited and expensive.

Rasmus et al. [33] performed experiments with ladder networks on two image datasets and achieved very good results. We used this ladder network implementation for our experiments.

3.4 Other Advances in Deep Learning

In this section, we discuss other recent advances in deep learning that we used in our research:

Adaptive learning rate - One of the most important parameters is the learning rate, which is used both in supervised and unsupervised pre-training. If the learning rate is too small, training will be slow. If the learning rate is too large, it will oscillate or diverge [24]. In our experiments we applied conventional stochastic gradient descent (SGD) with momentum and Adam. Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [21].

Dropout for regularization - The ultimate goal is not to minimize the cost-function for a training set, but to find a model that generalizes well (by minimizing the generalization error). DNNs are prone to overfitting due to their high capacity [50]. One of the frequently cited (and recently introduced) methods to control over-fitting in deep learning is dropout [18]. Dropout randomly omits nodes (with a certain probability) in the network at each iteration of the training process. A motivation for dropout is that each hidden unit in a neural network trained with dropout must learn to work with a randomly chosen sample of other units. This should make each hidden unit more robust, without relying on other hidden units to correct its errors.

Activation function - Sigmoidal activation functions can suffer from the vanishing gradient problem [9]. Such vanishing gradients cause slow optimization convergence, and in some cases, the final network converges to a poor local minimum [26]. Symmetric (around zero) sigmoids, such as hyperbolic tangents, converge faster than standard sigmoids [24], because the derivatives are higher.

In recent years, researchers have experimented with rectified linear (ReLU) activation functions: $\phi(x) = max(0, x)$. ReLU has outperformed sigmoidal activation functions in both image [23] and acoustic [26] models.

The maxout activation function divides a layer into several groups of neurons. The output of each group is the maximum value of the neurons in a group [14].

4 EXPERIMENTAL SETUP

In this section we present the configurations and hyper-parameters of the DNNs and ladder networks, as well the dataset used in our experiments.

4.1 Network Configurations and Hyper-parameters

In our experiments we applied DNNs, both purely supervised and pre-trained with SDAs, and ladder networks. The learning algorithms to train these networks, have numerous hyper-parameters that can be adjusted [8]. These hyper-parameters are not adjusted or learned by the learning algorithm, but are parameters to configure or steer the algorithm. We experimented with the most relevant hyper-parameters:

Architecture - We experimented with the number of hidden layers and the number of neurons within these layers. A pyramidal structure with a gradual reduction from the number of inputs to the number of outputs (e.g., 4,000–2,000–1,000–500–250–...) per layer has been successfully used in different applications [17, 22]. For SDA an overcomplete first hidden layer often performs best [47]. This inspired us to experiment with one to six hidden layers with 1,000 to 5,000 nodes (30 combinations).

Activation function - For DNN we applied ReLU, maxout and hyperbolic tangent. For SDA we only applied hyperbolic tangent; ReLU did not yield good results in prior research [13].

Optimization methods - We applied SGD with momentum and Adam. For SGD with momentum, we used a momentum of 0.5 for DNN. The initial learning rate was 0.1. If the validation error dropped below 99.5% of the previous error, then we scaled the learning rate by a factor of 1.05. If the validation error increased above 105% of the previous error, we scaled the learning rate by a factor of 0.7. For SDA pre-training, we used a fixed learning rate of 0.01, without momentum. For Adam, we used the default values [21]: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. During the training, we did not change the step-size because Adam naturally performs a form of step-size annealing [21].

Pre-training - We pre-trained a SDA for a maximum of 20 epochs because the reconstruction error did not decrease significantly beyond this point. For pre-training, we experimented with 10% and 25% corruption levels, following the research of Vincent et al. [47].

Regularization - In our pre-trained experiments with SDA, we did not apply extra regularization during fine-tuning because the pre-training acts as a regularizer. For the DNN experiments we used L2-regularization and dropout. We applied dropout to both the input and hidden layers, as adding dropout to the input layers has reduced error rates in some studies [18]. For dropout, we used 10% and 20% for the input layer, and 40% and 50% for the hidden layers, which follows the research of Srivastava et al. [40]. In addition, a factor of 0.0001 was used for L2 weight decay regularization, that adds a term to the cost function to penalize large weights.

Stop criterion - We stopped training after 200 epochs for DNNs pre-trained by SDA, and after 300 epochs for DNNs without pre-training; alternatively, we stopped the training if within 10 epochs after a new low in validation error, no new low below current low multiplied by a threshold (0.995) was reached. This decision was motivated by the desire to continue training after attaining a new low to search for another new low. However, this was limited to prevent overfitting.

Cost function - For SDA pre-training, we used the squared error. If we have k training examples this can be calculated as follows:

$$C(\theta) = \sum_{i=0}^{k} \left(r_\theta(\mathbf{x}_i) - \mathbf{y}_i\right)^2, \tag{4}$$

where θ represents the parameters (weights of the neural network), r_θ represents the reconstruction vector (using θ). The negative log-likelihood function was minimized for DNN:

$$C(\theta) = -\sum_{i=0}^{k} \log(P(\mathbf{y}_i|\mathbf{x}_i, \theta)) \ . \tag{5}$$

4.2 Datasets Used

We performed our experiments on a 83 GB dataset that we assembled from the following 5 datasets:

UNB ISCX IDS dataset [37] is a labeled network dataset with complete payload data. It covers a wide range of normal traffic as well as non-P2P-botnet attack traffic. We filtered out the attack traffic and only used the normal traffic from this dataset.

Ericsson Research Traffic Lab dataset [41] is a publicly available dataset of non-malicious activity, such as web browsing, email, and P2P torrent traffic.

Lawrence Berkeley National Lab dataset [31] is a publicly available dataset. It contains network traffic captured from the research institute's network, such as web browsing, email, and streaming media.

Peerrush dataset contains several P2P-botnet datasets that we obtained from Rahbarinia et al. [32], including Storm [19], Zeus [25], and Waledac [30]. The Waledac data were retrieved prior to the takedown initiated by Microsoft. The Zeus traces are relatively recent and are likely from a botnet that is still active [32].

ISCX Botnet dataset [6] is a publicly available dataset with botnet traffic. It contains traffic from different types of bots. We only used flows from the P2P-botnets Zeus and ZeroAccess.

These datasets contain both botnet and non-botnet traffic in the form of network traces (PCAP-files) and flow (sub)classification information in an XML or text-file format. We applied the overlay methodology [2] to mix traffic from the different datasets. Simply combining all datasets may cause a class imbalance problem [20] if most data are botnet or non-botnet data, which causes that training of the neural network gets biased towards the majority class. Many strategies exist to avoid class imbalance [5]. Within each class, we undersampled the largest subclass to create a balanced dataset with 50% botnet traffic and 50% non-botnet traffic. Table 1 shows details of the flows in the datasets. The 'start date' and 'end date' indicate the time frame during which data was captured; '#flows' indicates the total number of flows in a dataset; '#used' indicates the number of flows included in our dataset.

We constructed our dataset as outlined in Figure 2. We first read every PCAP-file from each dataset packet by packet and selected only TCP/UDP-packets. We ignored other packets such as Internet Control Message Protocol (ICMP) packets or Dynamic Host Configuration Protocol (DHCP) packets, because they are used for network discovery/tracing and are most likely not used for bot-to-bot communication. Next, we combined the packets into flows. Each packet is (sub)classified based on the flow-data provided with the dataset. A flow is a set of packets that share a common key $\langle source_{addr}, source_{port}, destination_{addr}, destination_{port}\rangle$ and for which the time interval between subsequent packets is lower than a specified threshold [11]. This threshold depends on the flow details of the datasets. If the flow information of the dataset did not provide this information, we used a flow gap of 60 seconds (i.e., if two hosts/ports do not exchange data for 60 seconds, we consider the flow to have ended).

From each flow, we took the first 320 packets. Analysis of all datasets showed that 320 packets per flow represents the 99.9 and 99.2 percentile ranks of all botnet and non-botnet flows, respectively. From each packet, we extracted 13 fields as relevant features: Send/Receive indicator, Interval, DSCP, TTL, Protocol, Source port, Destination port, ACK, RST, SYN, FIN, PSH, and payload size. The Send/Receive indicator indicates whether the host is sending or receiving (the flows are bidirectional). This feature is omitted from the

Table 1: Flows in datasets

Dataset	Start date	End date	#flows	#used
UNB ISCX IDS	2010-06-11	2010-06-19	3,261,409	1,485,775
Ericsson Research	2007-10-08	2007-10-08	88,369	40,104
Lawrence Berkeley	2004-10-04	2005-01-08	2,144,178	976,168
Peerrush Storm	2007-10-27	2007-10-28	7,521,576	1,321,374
Peerrush Zeus	2011-11-07	2011-11-25	89,571	89,571
Peerrush Waledac	2009-02-17	2009-02-18	1,082,130	1,082,130
ISCX Botnet	1970-01-01	2013-02-04	4,878	4,878
Total			14,192,111	5,000,000

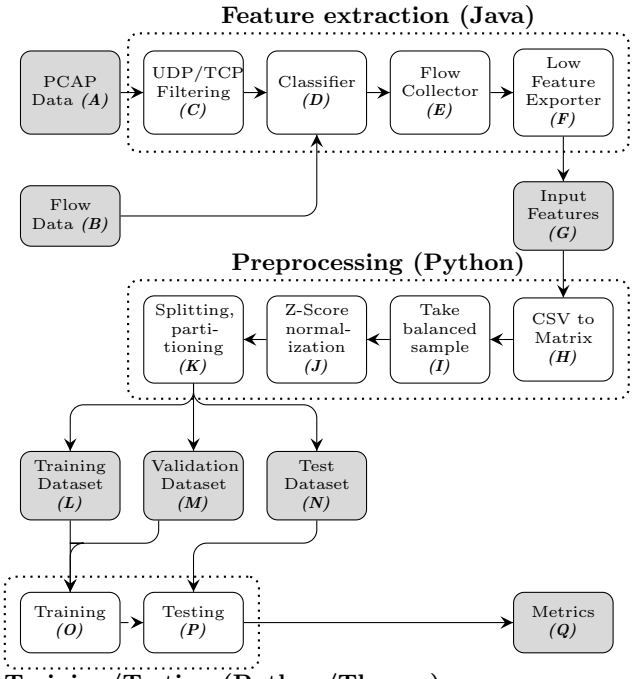

Figure 2: Workflow of dataset preparation.

first packet. We used the timestamps in the capture headers to compute the time interval from the previous packet. Also this feature is omitted from the first packet, because it has no previous packet. From the IP-headers, we used DSCP (also known as type of service, TOS), TTL (Time To Live of the packet), and the Protocol number. We only used the Protocol number for TCP (6) and UDP (17), which we transformed to -1 for TCP and 1 for UDP. From the TCP and UDP headers we used the source port, the destination port, and the size of the payload. We did not use the payload data itself, since all botnets nowadays encrypt the payload data [34] and hence no meaningful data can be obtained from this. Considering the payload may still help in classifying non-botnet versus botnet traffic. However, state regulation and strong privacy protection make deep packet inspection illegal or unfavorable [1], and therefore we ignored the payload data content. From the TCP headers we also used the ACK, RST, SYN, FIN, and PSH flag. We transformed the binary values of the flags by setting them to -1 or 1 when not set or set respectively; these features are 0 for UDP headers.

In our experiments we used neural networks with 4,158 inputs. Hence, all features of a flow (11 features for the first packet, 13 features for each of the next 319 packets) are applied in parallel. All features have numerical values.

We subsequently normalized the dataset, not only to provide equal weight to data with different units, but also to prevent some weights moving faster than others. We applied a zero-mean, unit-variance (Z-score) [24]: $\mathbf{x}_{norm.} = (\mathbf{x} - \overline{\mathbf{x}})/\sigma(\mathbf{x})$, where $\mathbf{x}$ is a feature vector.

We split the final balanced dataset of five million flows randomly into three parts: a training set of 64% of the items used to train the network, a validation set (16%) used to adjust the learning rate and for early stopping, and the remaining 20% as the test set, which is the held-out set used to calculate the error rate and detection accuracy after training the model. Given the large size of the dataset, we did not use k-fold cross validation.

We were also interested in how our experiments would be affected by an unbalanced dataset. Some studies suggest that 5–10% of all broadband subscribers are infected with bot malware [45]. We therefore also created an unbalanced test set with 2% botnet traffic (20,000 flows) and 98% non-botnet traffic (980,000 flows). We normalized this dataset using the mean and standard deviation of the balanced dataset.

4.3 Hardware and Software

We performed all experiments on a system equipped with an NVIDIA GTX 980 Titan board with 6 GB of memory and 2,816 cores. For the SDA/DNN-experiments we used PDNN [27], which runs on top of Theano [43]. Furthermore, we used the open-source experimental code from CuriousAI as the ladder implementation. We made many changes to the PDNN and ladder code. We increased the dropout training rate by a factor of four because the implementation did not fully utilize the GPU. Furthermore, we added support for Adam and we improved the input/output layer to optimize handling of the 83 GB dataset. We were able to classify 25,000 flows per second on a trained and uploaded model.

5 RESULTS

5.1 Experimental Results

We performed over 650 experiments with DNNs (supervised or pre-trained) and ladder networks, in which we varied the network configuration and hyper-parameters (see section 4.1).

Architecture - We performed experiments with different network architectures, having one to six hidden layers, and either a constant number of neurons in all hidden layers or a pyramidal structure. Networks having a constant number of neurons in all hidden layers, ranging from 1,000 to 5,000 nodes per layer, yielded significantly worse results. We achieved best results with a pyramidal network structure having a first overcomplete hidden layer. We tested a wide range of hidden layer configurations, ranging from non-deep (e.g., 5,000–2,048) to very deep (e.g., 5,000–2,048–1,024–512–256–128). We achieved our best results using a network with three hidden-layers: 5,000–3,000–1,500 (see [46] for more details).

Activation function - We achieved better results with hyperbolic tangent than with maxout and ReLU.

Optimization methods - For Adam, we obtained best results using the default parameters [21], but with a smaller step size, $\alpha = 0.0001$. We observed that Adam performed better than SGD with momentum.

Pre-training - We achieved the best results when applying DNNs with supervised training. Pre-training with SDAs did not yield better results. Also, ladder networks did not performed as good as DNNs, however our effort to find optimal denoising levels was restricted.

Table 2: Experimental results

	Unbalanced		Balanced dataset						
						TPR			
Hyper-parameters	AUC	ER (%)	AUC	ER (%)	TNR	Storm	Waledac	Zero Access	Zeus
tanh; dropout 10%,40%	0.996	0.363	0.997	0.313	0.997	1.000	0.998	0.946	0.943
tanh; dropout 10%,50%	0.995	0.343	0.996	0.389	0.997	0.997	0.995	0.955	0.937
relu; dropout 10%,40%	0.996	0.333	0.996	0.357	0.997	0.999	0.996	0.920	0.942
relu; dropout 10%,50%	0.996	0.367	0.996	0.372	0.996	1.000	0.997	0.916	0.941
pretrain 25% noise; tanh; dropout 10%,50%	0.996	0.337	0.997	0.346	0.997	1.000	0.997	0.950	0.943
pretrain 25% noise; tanh; dropout 10%,50%	0.995	0.387	0.996	0.429	0.996	1.000	0.994	0.940	0.942
pretrain 10% noise; tanh; dropout 10%,40%	0.995	0.346	0.996	0.372	0.997	0.998	0.995	0.945	0.943
pretrain 10% noise; tanh; dropout 10%,40%	0.996	0.373	0.996	0.385	0.996	1.000	0.996	0.950	0.942
maxout poolsize 5; dropout 10%,40%	0.995	0.365	0.996	0.432	0.996	0.999	0.994	0.959	0.943
maxout poolsize 5; dropout 10%,50%	0.995	0.424	0.996	0.432	0.996	0.999	0.995	0.957	0.941
maxout poolsize 3; dropout 10%,40%	0.995	0.423	0.996	0.435	0.996	0.997	0.995	0.950	0.943
maxout poolsize 3; dropout 10%,50%	0.995	0.416	0.996	0.447	0.996	0.998	0.995	0.953	0.936
ladder; 0.01 noise; tanh	0.992	0.744	0.992	0.756	0.993	0.993	0.992	0.980	0.981
ladder; 0.01 noise; relu	0.992	0.821	0.992	0.830	0.993	0.993	0.992	0.981	0.980
ladder; 0.001 noise; tanh	0.992	0.825	0.991	0.837	0.993	0.993	0.991	0.975	0.993
ladder; 0.001 noise; relu	0.992	0.922	0.991	0.925	0.991	0.993	0.991	0.981	0.981

Regularization - We observed that dropout achieved better results than L2-regularization. We experimented with different dropout percentages, and achieved best results when applying 10% dropout in the input layer and 40% dropout in the hidden layers.

Table 2 lists our results obtained with a network architecture having three hidden-layers (5,000–3,000–1,500) and using Adam with dropout. (See [46] for our experimental results with other network architectures.) The table shows results for both the unbalanced test set (2% botnet traffic, 98% non-botnet traffic) and the balanced test set (50% botnet traffic, 50% non-botnet traffic). The unbalanced dataset was used only as test set, after the network was trained with the balanced training set. For both datasets, the Area Under the Curve (AUC) and the error rate (ER) is shown. AUC is a measure for detection accuracy that combines sensitivity and specificity, defined as $(TP/(TP+FN)+TN/(FP+TN))/2$, where TP, FN, FP, and TN are the number of true positives, false negatives, false positives, and true negatives [38]. The error rate is the percentage of incorrectly classified flows in the test set, defined as $(FP+FN)/(TP+TN+FP+FN)\cdot 100\%$.

The table also shows for the balanced test set: the true negative rate (TNR), which is the fraction of non-botnet traffic that is classified correctly, and the true positive rate (TPR), which is the fraction of the botnet traffic that is classified correctly, for the Storm, Waledac, Zero Access, and Zeus botnets. With the balanced test set, we achieved best results with a DNN and hyperbolic tangent activation function when applying supervised training using Adam, 10% dropout in the input layer, and 40% dropout in the hidden layers, resulting in an error-rate of 0.313%, a TNR of 0.997, and TPRs of 1.0, 0.998, 0.946, and 0.943 for respectively Storm, Waledac, Zero Access, Zeus. The results with the unbalanced test set are comparable.

5.2 Discussion

The use of machine learning for network intrusion detection has been criticized, as machine learning may not be suited for detecting still unknown anomalies [39]. However, in our work we try to detect known anomalies only, i.e. the botnet traffic as present in our datasets.

Table 3: Comparison of experimental results (TPR)

Botnet	Narang et al. [28]	Rahbarinia et al. [32]	Our results
Storm	0.988	1.000	0.998
Waledac	0.989	1.000	0.995
Zeus	-	0.925	0.938

Some of the features that we use in our method are influenced by low-level network characteristics, such as throughput (that influences packet timing) and MTU (that influences payload size). Differences in such characteristics can be due to the different types of network traffic, but they also may be due to different types and configurations of the communication networks in which the traffic was captured. The latter causes the risk that our neural networks are trained to recognize the communication networks, rather than the traffic types. We reduce this risk by mixing data from various datasets, to get a large variation of networks. Moreover, most of our features do not rely on low-level network characteristics.

We compared our experimental results with three other studies: Narang et al. [28], Rahbarinia et al. [32], and Zhang et al. [51]. These three studies all used the same Peerrush botnet datasets, but they all used different non-botnet datasets. For instance, the non-botnet data used by Narang et al. only contains benign P2P traffic, while our dataset also contained benign non-P2P traffic. There are no standardized datasets. This makes a precise comparison of results difficult, since botnet classification can be more difficult or easier depending on the variance in the non-botnet dataset.

In order to make a better comparison, we performed additional experiments in which we used the same dataset as Narang et al. [28] and Rahbarinia et al. [32]. First, we made a random subset of 50% P2P-botnet and 50% P2P non-botnet traffic, and we applied this as test set to our best trained deep neural network. The results are listed in table 3. It can be seen that our results outperform the results of Narang et al. [28]. Compared to the results of Rahbarinia et al. [32], our results are slightly worse (0.002 respectively 0.005) for Storm and Waledac, and better for Zeus. Hence, our results are at least comparable and often better, while our approach does not need feature engineering or feature selection.

Zhang et al. [51] introduced a two-phase P2P-botnet detector. They used the same dataset for Storm and Waledac, but

a different dataset for Zeus. They achieved a total accuracy of 99.8%, which is comparable to our results.

6 FUTURE WORK

We see various opportunities for further research. Up to now we focused on supervised feed-forward DNNs and SDAs. We also performed experiments with ladder networks, but we did not thoroughly search for optimal noise levels. Performing more experiments with ladder networks, as well as other advanced optimizers, might yield even better results. Also other architectures might be interesting to experiment with, such as variational auto-encoders and deep recurrent neural networks. The latter are suited for time-series modelling of network packets/flows, which for instance allows to base detection on packet flows of arbitrary length (which eliminates the possibility for botnet malware to evade our detection by mimicking regular traffic in the first 320 packets).

We considered our neural networks as black boxes. They are trained and tested, but we still have little understanding of what exactly has been learned by the network. We are unravelling the hierarchy of features that has been learned by the networks that performed best in our experiments.

Researchers have observed that the stability of deep neural networks trained by back propagation in a purely supervised way can be sensitive to small perturbations in the inputs [42]. Adversarial perturbations (crafted by optimizing the inputs to maximize the prediction error) could arbitrarily change the network's outputs. Whether, and to what extent, this actually is feasible in our networks to circumvent detection, is a topic for future research.

We received funding to continue our research on botnet detection using advanced machine-learning approaches. We are currently cooperating with a DNS registry, several Internet service providers, and a private security company in the Netherlands to apply our research results in practice. The approach is to (continuously) train a neural network with captured streams of DNS traffic, and apply the trained network for real-time, large-scale botnet detection.

7 CONCLUSION

We studied the application of deep learning for botnet detection. We obtained large datasets with P2P-botnet and background network traffic, and used the overlay methodology to generate a large-scale dataset of five million flows. We used the raw header information from packets of each flow in this dataset as low-level inputs. This approach is different from nearly all existing academic proposals, which rely directly or indirectly on netflow statistics, and apply feature engineering and feature selection. In our approach, feature engineering is minimized.

Our results show that the detection accuracy is slightly better with DNNs than with ladder networks. Our results also show that similar detection accuracies can be achieved by DNNs using supervised training and unsupervised pre-training with SDAs. We achieved best results using a deep feed-forward supervised neural network, with dropout and the Adam optimizer. We achieved a P2P-botnet detection accuracy of 99.7%, which is similar to or better than conventional approaches for P2P-botnet detection as reported in scientific literature.

REFERENCES

[1] Hadi Asghari, Michel Van Eeten, Johannes M Bauer, and Milton Mueller. 2013. Deep packet inspection: Effects of regulation on its deployment by internet providers. In *The 41st Research Conference on Communication, Information and Internet Policy*. Arlington, VA.

[2] Adam J. Aviv and Andreas Haeberlen. 2011. Challenges in Experimenting with Botnet Detection Systems.. In *4th Workshop on Cyber Security Experimentation and Test, CSET '11, San Francisco, CA, USA, August 8, 2011*.

[3] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. 2009. A survey of botnet technology and defenses. In *Conference For Homeland Security, 2009. CATCH'09. Cybersecurity Applications & Technology*. IEEE, 299–304. https://doi.org/10.1109/CATCH.2009.40

[4] Pijush Barthakur, Manoj Dahal, and Mrinal Kanti Ghose. 2013. An Efficient Machine Learning Based Classification Scheme for Detecting Distributed Command & Control Traffic of P2P Botnets. *International Journal of Modern Education and Computer Science (IJMECS)* 5, 10 (Nov. 2013), 9–18. https://doi.org/10.5815/ijmecs.2013.10.02

[5] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. 2004. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. {*SIGKDD*} *Explorations* 6, 1 (June 2004), 20–29. https://doi.org/10.1145/1007730.1007735

[6] Elaheh Biglar Beigi, Hossein Hadian Jazi, Natalia Stakhanova, Ali Ghorbani, and others. 2014. Towards effective feature selection in machine learning-based botnet detection approaches. In *Communications and Network Security (CNS), 2014 IEEE Conference on*. IEEE, 247–255. https://doi.org/10.1109/CNS.2014.6997492

[7] Yoshua Bengio. 2009. Learning Deep Architectures for AI. *Foundations and Trends in Machine Learning* 2, 1 (2009), 1–127. https://doi.org/10.1561/2200000006

[8] Yoshua Bengio. 2012. Practical Recommendations for Gradient-Based Training of Deep Architectures. In *Neural Networks: Tricks of the Trade*, Grgoire Montavon, Montavon Grgoire Orr, and Klaus-Robert Mller (Eds.). Lecture Notes in Computer Science, Vol. 7700. Springer Berlin Heidelberg, 437–478.

[9] Y. Bengio, P. Simard, and P. Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on* 5, 2 (March 1994), 157–166. https://doi.org/10.1109/72.279181

[10] Dan Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. 2010. Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition. *Neural Computation* 22, 12 (2010), 3207–3220. https://doi.org/10.1162/NECO_a_00052

[11] Kimberly C. Claffy, Hans-Werner Braun, and George C. Polyzos. 1995. A parameterizable methodology for Internet traffic flow profiling. *Selected areas in Communications, IEEE Journal on* 13, 8 (Oct. 1995), 1481–1494. https://doi.org/10.1109/49.464717

[12] Li Deng. 2014. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing* 3 (2014). https://doi.org/10.1017/atsip.2013.9

[13] Ian Goodfellow and Xavier Glorot. 2013. Rectified linear units in autoencoder. (May 2013). https://groups.google.com/forum/#!topic/pylearn-dev/iWqctW9nkAg

[14] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron C. Courville, and Yoshua Bengio. 2013. Maxout Networks. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*. 1319–1327.

[15] Fariba Haddadi and A. Nur Zincir-Heywood. 2015. Botnet Detection System Analysis on the Effect of Botnet Evolution and Feature Representation. In *Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference (GECCO Companion '15)*. ACM, New York, NY, USA, 893–900. https://doi.org/10.1145/2739482.2768435

[16] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and others. 2012.

Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *Signal Processing Magazine, IEEE* 29, 6 (Nov. 2012), 82–97. https://doi.org/10.1109/MSP.2012.2205597

[17] Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation* 18, 7 (July 2006), 1527–1554. https://doi.org/10.1162/neco.2006.18.7.1527

[18] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[19] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst W Biersack, and Felix Freiling. 2008. Measurements and Mitigation of Peer-to-peer-based Botnets: A Case Study on Storm Worm. In *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*. San Francisco, California.

[20] Nathalie Japkowicz and Shaju Stephen. 2002. The class imbalance problem: A systematic study. *Intelligent data analysis* 6, 5 (2002), 429–449.

[21] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *The International Conference on Learning Representations (ICLR)*.

[22] Alex Krizhevsky and Geoffrey E. Hinton. 2011. Using very deep autoencoders for content-based image retrieval. In *ESANN 2011, 19th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 27-29, 2011, Proceedings*.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 1097–1105.

[24] Yann LeCun, Leon Bottou, Genevive B Orr, and Klaus Robert Mller. 1998. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, Montavon Grgoire Orr and Klaus-Robert Mller (Eds.). Lecture Notes in Computer Science, Vol. 1524. Springer Berlin Heidelberg, 9–50.

[25] Andrea Lelli. 2012. Zeusbot/spyeye p2p updated, fortifying the botnet. (2012).

[26] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning (ICML) Workshop on Deep Learning for Audio, Speech, and Language Processing*, Vol. 30.

[27] Yajie Miao. 2014. Kaldi+ PDNN: building DNN-based ASR systems with Kaldi and PDNN. *arXiv preprint arXiv:1401.6984* (2014).

[28] Pratik Narang, Subhajit Ray, Chittaranjan Hota, and Venkat Venkatakrishnan. 2014. PeerShark: Detecting Peer-to-Peer Botnets by Tracking Conversations. In *35. IEEE Security and Privacy Workshops, SPW 2014, San Jose, CA, USA, May 17-18, 2014*. 108–115. https://doi.org/10.1109/SPW.2014.25

[29] Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Mansoor Alam. 2016. A Deep Learning Approach for Network Intrusion Detection System. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (Formerly BIONETICS), BICT*, Vol. 15. 21–26.

[30] Chris Nunnery, Greg Sinclair, and Brent ByungHoon Kang. 2010. Tumbling Down the Rabbit Hole: Exploring the Idiosyncrasies of Botmaster Systems in a Multi-tier Botnet Infrastructure. In *Proceedings of the 3rd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More (LEET'10)*. USENIX Association, Berkeley, CA, USA, 1–1.

[31] Ruoming Pang, Mark Allman, Vern Paxson, and Jason Lee. 2006. The Devil and Packet Trace Anonymization. *ACM Computer Communication Review* 36, 1 (Jan. 2006), 29–38. https://doi.org/10.1145/1111322.1111330

[32] Babak Rahbarinia, Roberto Perdisci, Andrea Lanzi, and Kang Li. 2014. PeerRush: Mining for unwanted P2P traffic. *Journal of Information Security and Applications* 19, 3 (2014), 194 – 208. https://doi.org/10.1016/j.jisa.2014.03.002

[33] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. 2015. Semi-supervised Learning with Ladder Networks. In *Advances in Neural Information Processing Systems 28*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.). Curran Associates, Inc., 3546–3554.

[34] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. J. Dietrich, and H. Bos. 2013. SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In *Security and Privacy (SP), 2013 IEEE Symposium on*. IEEE, 97–111. https://doi.org/10.1109/SP.2013.17

[35] Christian Rossow and Christian J. Dietrich. 2013. ProVeX: Detecting Botnets with Encrypted Command and Control Channels. In *Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'13)*. Springer-Verlag, Berlin, Heidelberg, 21–40. https://doi.org/10.1007/978-3-642-39235-1_2

[36] Sherif Saad, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix, and Payman Hakimian. 2011. Detecting P2P botnets through network behavior analysis and machine learning. In *Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on*. IEEE, 174–180. https://doi.org/10.1109/PST.2011.5971980

[37] Ali Shiravi, Hadi Shiravi, Mahbod Tavallaee, and Ali A. Ghorbani. 2012. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers Security* 31, 3 (2012), 357 – 374. https://doi.org/10.1016/j.cose.2011.12.012

[38] Marina Sokolova and Guy Lapalme. 2009. A systematic analysis of performance measures for classification tasks. *Information Processing & Management* 45, 4 (2009), 427 – 437. https://doi.org/10.1016/j.ipm.2009.03.002

[39] R. Sommer and V. Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *Proceedings Symposium on Security and Privacy*. IEEE, 305–316.

[40] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[41] Géza Szabó, Dániel Orincsay, Szabolcs Malomsoky, and István Szabó. 2008. On the Validation of Traffic Classification Algorithms. In *Passive and Active Network Measurement, 9th International Conference, PAM 2008, Cleveland, OH, USA, April 29-30, 2008. Proceedings*. 72–81.

[42] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. *arXiv:1312.6199v4* (2014).

[43] Theano Development Team. 2016. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints* abs/1605.02688 (May 2016).

[44] Harri Valpola. 2015. Chapter 8 - From neural PCA to deep unsupervised learning. In *Advances in Independent Component Analysis and Learning Machines*, Ella Bingham, Samuel Kaski, Jorma Laaksonen, and Jouko Lampinen (Eds.). Academic Press, 143 – 171.

[45] Michel van Eeten, Hadi Asghari, Johannes Bauer, and Shirin Tabatabaie. 2011. Internet Service Providers and Botnet Mitigration. A fact-finding study on the Dutch Market. (2011).

[46] Jos van Roosmalen. 2017. *The feasibility of deep learning approaches for P2P-botnet detection*. Master's thesis. Open University The Netherlands.

[47] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*. 1096–1103. https://doi.org/10.1145/1390156.1390294

[48] Zhanyi Wang. 2015. The Applications of Deep Learning on Traffic Identification. In *Black Hat USA 2015*.

[49] DE Rumelhart GE Hinton RJ Williams and GE Hinton. 1986. Learning representations by back-propagating errors. *Nature* (1986), 323–533. https://doi.org/10.1038/323533a0

[50] Matthew D. Zeiler and Rob Fergus. 2013. Stochastic Pooling for Regularization of Deep Convolutional Neural Networks. *International Conference on Learning Representations 2013* (2013).

[51] Junjie Zhang, Roberto Perdisci, Wenke Lee, Xiapu Luo, and Unum Sarfraz. 2014. Building a Scalable System for Stealthy P2P-Botnet Detection. *IEEE Transactions on Information Forensics and Security* 9, 1 (Jan. 2014), 27–38. https://doi.org/10.1109/TIFS.2013.2290197

[52] David Zhao, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani, and Dan Garant. 2013. Botnet detection based on traffic behavior analysis and flow intervals. *Computers & Security* 39 (Nov. 2013), 2–16. https://doi.org/10.1016/j.cose.2013.04.007