How Deep Learning Is Making Information Security More Intelligent

Ankush Singla and Elisa Bertino | Purdue University

Manually analyzing vast amounts of newly released malware is a significant problem for the security community. Deep-learning techniques have recently been employed to automate security tasks such as malware analysis, intrusion detection, and botnet detection. We look at such techniques and investigate how practical and secure they are.

According to U.S. law,[1] *information security* is defined as "protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction." It is well established among information security communities that there is no system that can be 100% secure from all adversaries. Critical systems, protocols, and software considered secure are constantly analyzed by intelligent adversaries with sufficient resources, leading to the identification of vulnerabilities and allowing them to craft exploits that break into systems by taking advantage of those vulnerabilities. Vulnerabilities, unknown to the creators or users of a system, are called *zero-day vulnerabilities*, and the exploits that take advantage of them are called *zero-day exploits*. Members of the security community, e.g., antivirus vendors and researchers, find such an exploit in the wild, disassemble and analyze the code/working for the exploit, and flag the signature of the exploit into their products so that it can be instantly flagged on user machines. Typically, they also provide this information to the creators or users of exploited systems so that the vulnerability can be patched.

Digital Object Identifier 10.1109/MSEC.2019.2902347
Date of publication: 14 May 2109

Problem

With the exponential increase in the number of Internet users, cloud services, and Internet of Things (IoT) devices, the attack surface has increased substantially. The number of adversaries attempting to find new ways of breaking into these systems has therefore skyrocketed. The involvement of state actors trying to gain an upper hand in the cybersecurity arms race has also increased the frequency of new attacks and malware. McAfee Labs reports that the amount of new malware identified reached an all-time high of 57.3 million samples in Q3 of 2017.[2] Conventional methods based on the manual inspection of suspicious code, reverse engineering, and manual vulnerability identification are too slow to compete against new adversarial capabilities.

Solution

Deep learning (DL) is a promising approach that addresses the shortcomings of these conventional methods. Vaguely inspired by the biological neural networks (NNs) in the human brain, DL is a branch of machine learning (ML) that tries to learn various characteristics from data and uses them for decision making on similar unseen data. In the last five years, DL techniques have gained interest because of the increased amount of data available and its various algorithmic innovations as well as significant improvements in computing capabilities enabled by GPUs, which have made the fast training

 Copublished by the IEEE Computer and Reliability Societies

and deployment of DL models possible.[3] Previously, DL has been tremendously successful at tasks such as image classification, object detection, and text and voice recognition.

Recently, approaches have been proposed that use DL for various information security tasks such as malware analysis, intrusion detection, botnet detection, and software analysis. For example, DL-based techniques can learn from the patterns and features of training data sets that contain various malware samples and can then be used in antivirus software or intrusion detection systems (IDS) to detect similar malware during an actual attack. Additionally, DL-based techniques can also be trained to learn the features that allow them to identify malware that has not yet been reported, such as zero-day malware, which makes them a great choice for security tasks.

Contributions

Although broader ML techniques are being used for various security tasks, we focus mainly on DL techniques for this article. We analyze in detail the security tasks for which DL techniques have been used, including malware analysis, intrusion detection, and botnet detection. We provide a comparison and critical analysis of research work in these specific categories, the benefits of DL-based approaches over traditional mechanisms, and the areas with respect to which DL-based techniques are lacking. One critical issue that has emerged from recent research is the reliability of DL-based approaches and whether they can be bypassed by attackers with knowledge of the detection model through something known as *adversarial examples*. We discuss the recent body of research that has investigated this problem as well as the efforts at making DL models more resilient to such attacks. We also discuss open research areas in the use of DL techniques for security.

Terminology

AI is defined by John McCarthy, a pioneer of AI research, as "the science and engineering of making intelligent machines, especially intelligent computer programs."[4] In simpler terms, it can be thought of as intelligence demonstrated by machines to mimic or even surpass human intelligence. Intelligence can be expressed in a number of ways, including decision making and learning complex tasks. ML is a branch of AI that enables computers to learn and make decisions using statistical techniques without manual programming.[5] ML involves training a model to use input data and then making predictions on new data using the learned model. ML is broadly classified into two categories: 1) supervised learning, i.e., the ML system is provided with labeled data consisting of pairs of input and the desired output and learns mappings between those pairs to make intelligent decisions on new data, e.g., image classification and speech recognition; and 2) unsupervised learning, i.e., the ML algorithm is provided only with unlabeled data as input and learns the structure and relationships within the input data, making decisions based on learned patterns, e.g., data clustering and anomaly detection. Popular ML algorithms include decision trees,[6] support vector machines,[7] Bayesian networks,[8] and logistic regression.[9]

Artificial NNs (ANNs), sometimes loosely called *NNs*, are a family of models inspired by the human brain and how the network of neurons process information and perform computations.[10] The building blocks of those networks are called *neurons*, which are grouped in different layers. Each layer of neurons receives inputs from the previous layer of neurons and passes its output to the next layer. Each neuron has its own activation functions, weights, and biases, which it tunes and optimizes during the training phase depending on the cost function, to minimize total error between the predicted and provided values.

A deep NN (DNN) is an NN with multiple hidden layers in addition to the input and output layers; any NN with fewer layers than that is called a *shallow NN*. DL is a subfield of ML, based on the use of DNNs. We describe two specific kinds of DNNs, known as *convolutional NNs* (*CNNs*) and *recurrent NNs* (*RNNs*), because they are the most widely researched and used.[11]

CNNs are primarily used for computer vision tasks, such as image classification, object recognition, and natural language processing, because they are designed to mimic the behavior of the animal visual cortex. CNNs use convolution layers involving filters that apply convolution operations to the inputs and pass them to the next layers. CNNs require minimal preprocessing when compared to other image recognition algorithms. The success of CNNs at performing computer vision tasks is one of the primary reasons that interest in DL resurfaced after being nonexistent for many years.

RNNs, on the other hand, are algorithms used for processing sequential data, such as time series or natural language.[12] In addition to the inputs, RNNs store an internal state that calculates outputs. RNNs include a feedback loop, where the output from the previous step is fed back to affect the output of the current step, and so on. Feedforward networks such as CNNs can receive and work on only one input at a time and thus produce one output, whereas RNNs can receive multiple inputs and vary their predictions based on the previous inputs. For example, for natural language processing, CNNs can translate or predict only one letter at a time, whereas RNNs can use the letters in the beginning of the word to affect the prediction of the next letters of the words or use previous words to build sentences.

One of the first ML algorithms, called *perceptron*,[13] was defined as a linear/binary classifier that can divide input into two categories with a straight line. In 1957, Rosenblatt introduced the single-layer perceptron, which was unable to perform nonlinear classifications. Multilayer perceptrons (MLPs)[14] were also introduced to capture more complex nonlinear functions. An MLP consists of at least three layers of nodes, and all of the neurons, except those in the input and output layers, use a nonlinear activation function. MLPs utilize a supervised learning technique called *back propagation* for learning. The training phase consists of steps in turn consisting of forward and backward passes that minimize the error until it can no longer decrease, also known as the *state of convergence*.

An autoencoder is an NN that learns by using unsupervised learning, which is generally for dimensionality reduction.[15] It reduces the input data to a short code, then expands it back to look similar to the input data. Dilated convolutional autoencoders (DCAEs) are quite similar to regular autoencoders but use dilated convolutions, and feature maps of the hidden layers are mapped into the reconstruction through a transposed convolution.[16]

Malware Analysis

There are two widely used approaches for malware analysis: static and dynamic. Static malware analysis is carried out by reverse engineering the malware binary to its assembly code and then analyzing the instructions without actually executing it. Such an approach can, however, be easily defeated by evasion techniques such as obfuscation and embedding of syntactic code errors. Dynamic malware analysis is conducted by executing the malware in a controlled sandbox environment to study its behavior and effect on the host system. Sophisticated malwares can evade dynamic analysis by determining whether they are being run inside a sandbox or a controlled environment and, based on this determination, can decide to not exhibit any malicious behavior. Both of these techniques are time-consuming and require a manual component, which makes them hard to scale because the number, complexity, and sophistication of the malwares increase. ML-based techniques have been proposed that address scalability by automating various steps of malware detection processes. However, their effectiveness is limited by high false-positive rates (FPRs), which make them unreliable. Researchers have recently demonstrated better results with DL-based systems.

DL-based techniques have been shown to classify malware at a much better speed than human analysts with high accuracy rates. These techniques can be applied to malware analysis to flag the suspicious binaries, which can then be analyzed and verified by a human in the loop. DL-based techniques have also proven to be much better at detecting newer malware with similar characteristics of other existing malware that has already been analyzed and categorized.

In 1995, Kephart et al.[17] proposed biologically inspired antivirus techniques designed to tackle existing and new viruses for the first time. They proposed an NN-based virus detector that learns to discriminate between infected and uninfected programs, and a computer immune system that identifies and analyzes new viruses automatically. Their detector extracts a set of three-byte strings, or "trigrams," appearing frequently in viral boot sectors but infrequently in legitimate ones. They, however, used just 350 samples with 100 negative samples and thus the FPR was too small to measure. This was the first attempt at using NNs for malware detection; however, they looked only at a small set of boot viruses and failed to execute the malware for detection.

Dahl et al.[18], in their 2013 paper, introduced a major improvement in this area. They argued that the number of input features for malware classification can be too large to be used by complex algorithms such as NNs, thus preventing the use of NN-based algorithms for malware detection. They proposed using random projections that reduce the dimensionality of the original input space by a factor of 45. Their approach extracts three types of features, including null-terminated patterns observed in the process' memory, trigrams of system API calls, and the distinct combinations of a single-system API call and one input parameter. Their approach achieves a classification result with a two-class error rate of 0.49%, i.e., detecting whether or not the binary is a malware and had an FPR of roughly 0.83%. Their approach, however, still had a high test-error rate of 10–12% when detecting the particular class to which the malware belongs.

In 2015, Saxe and Berlin[19] proposed a DNN-based malware detection system that achieves a 95% detection rate at a 0.1% FPR, based on more than 400,000 software binaries, which is much better than previous approaches. For input features, their system calculates the byte entropy histogram of the binary that models the file's byte distribution. Their system also gathers features from the program-executable import address table and packaging metadata and uses a DNN consisting of four layers. This is one of the only static malware analysis approaches that we found with good detection rates, and it was deployed in Invincea Labs' cloud security analytics platform, as shown in (Table 1).

Kolosnjaji et al.[20] built upon the previous approaches and proposed a malware classification method based on NNs comprised of both convolutional and recurrent network layers and applied it to system call sequences.

Their method requires system call traces obtained by executing the malware in a protected environment. Using this combined NN architecture achieves an average of 85.6 and 89.4% on precision and recall, respectively, and is shown to be much better than simple feedforward NNs at malware detection.

Pascanu et al.[21] use echo state networks and RNNs for extracting malware features. These features are then used as inputs to a standard classifier that detects malicious files. Their approach achieves true-positive and FPRs of 98.3 and 0.1%, respectively. Huang and Stokes[22] propose a DL architecture, called *MtNet*, for malware binary classification (i.e., malware versus benign) and for categorizing different malware families with 100 classes. They show that DNNs offer a modest improvement over shallow learning models. Their approach achieves binary malware and family error rates of 0.358 and 2.94%, respectively. *Binary malware error rate* refers to the error rate when detecting whether the file is malware or benign, whereas the *family error rate* refers to the error rate when detecting the actual family of the malware.

All of the previously discussed methods demonstrate that DL-based techniques can provide significant help with identifying and categorizing malware. However, most of those techniques are dynamic analysis approaches because they rely on features such as boot-sector trigrams, sequence of system API calls, and process memory, which are obtained by running the binary. Static analysis approaches may be better in terms of identifying zero-day malware without actually running it on a system. This enables the malware detectors to be better integrated with consumer antivirus systems and work on end-user machines to detect malware binaries before they can cause any harm. Thus, more work needs to be done to identify useful static binary features that would allow NN models to reliably classify the malware.

Furthermore, several of these detection approaches focus on malwares that have already been detected and analyzed. An interesting research direction is to determine how effective these techniques are in identifying newer malware families and variants being released into the wild, based on models trained on older classified malware. Another important aspect that has not been investigated by previous research is related to analyzing the memory requirements and execution times of those techniques. To be deployed on end-user machines, in addition to being resource efficient, these techniques should be able to classify new binaries in real time so that no delays are introduced that are noticeable by end users. The trained models should also be smaller in size so that they are easier to deploy on small systems, such as IoT devices.

Intrusion Detection

IDS monitor a network or system for malicious attacks or policy violations. Buczak and Guven[23] describe the different types of analytic techniques used for IDS; they can be broadly classified into misuse based and anomaly based. Misuse-based techniques look for specific patterns or signatures of attacks in network traffic, system calls, and so on. These techniques, however, require constant updates to databases containing rules and signatures and are, therefore, not effective against zero-day attacks. Anomaly-based techniques establish the normal network behavior and can then identify abnormalities, i.e., deviations in the traffic. They can detect newer attacks, and it is very difficult for attackers to bypass them because the normal activity is customized for the particular user, network, and applications being used. Anomaly-based techniques, however, may have high FPRs because they flag any unseen yet benign traffic or system use as a potential malicious attack. They also must be trained individually for every deployment. DL techniques that assist various traditional intrusion

Table 1. A comparison of DL approaches used for malware analysis.

Reference	Analysis type	Training data	Input features	Accuracy (%)	FPR
Kephart et al.[17]	Dynamic	Custom (viral boot sectors)	Three-byte strings from boot sectors	100	—
Dahl et al.[18]	Dynamic	Custom (Microsoft/CERT)	Process memory and system API call	99.51	0.83%
Saxe and Berlin[19]	Static	Custom (Invincea)	Binary's distribution of bytes	95	0.1%
Kolosnjaji et al.[20]	Dynamic	Custom (VirusShare/Maltrieve)	Kernel API call sequences	89.4	—
Pascanu et al.[21]	Dynamic	Custom (Microsoft)	Event streams from malware binaries	98.3	0.1%
Huang and Stokes[22]	Dynamic	Custom (Microsoft)	Raw data from antimalware engines	99.64	—

CERT: Computer Emergency Response Team.

detection deployments have been proposed (Table 2). In the case of misuse detection, an NN model can learn from a training data set containing examples from all of the misuse classes. It can then be used on new testing data to classify them as "belonging to one of the misuse classes" or as "normal." In the case of anomaly detection, the model can learn the defined normal traffic behavior; it can then be used to differentiate between normal and anomalous behavior.

Cannady[24] was one of the first to propose using NNs for misuse-based intrusion detection. He demonstrated that NNs can be used for misuse detection based on captured IP packet data. Single simulated attack events (e.g., ISS scans, Satan scans, SYNFlood, and so on) were used to test the NNs. This approach can be deployed alongside host-based or network-based IDS. Cannady's prototype uses an MLP architecture that consists of four fully connected layers with nine input and two output nodes. Their approach achieves a very low error rate of 0.06% on data generated using the RealSecure network monitor from Internet Security Systems, Inc. However, they pointed out that their NN takes 26.13 h to complete training/testing, which renders it unusable for real-time intrusion detection.

Palagiri[25] later introduced another approach using NNs for misuse-based intrusion detection. He explored network-based intrusion detection using a perceptron-based feedforward NN and a system based on classifying self-organizing maps. Palagiri used a feedforward NN with back propagation, called *meta neural*, and used tcpdump binaries from the DARPA 1999 training data set for their experiments. His system takes the following parameters at the command line: the configuration file, destination, known ports, number of extra ports, architectural learning interval, time interval, number of training samples, number of test samples, fraction of normal data, number of clusters, and maximum hits.

Palagiri's experiments showed a prediction rate of 100% but had a very high FPR, which makes his system unusable in real-world IDS deployments.

Hodo et al.[26] focus on IoT networks and suggest the use of NNs as an offline IDS to gather and analyze information from the network and identify attempts of denial-of-service (DoS)/distributed DoS (DDoS) attacks. They use an MLP that is trained using Internet packet traces from hosts in the network as well as an MLP architecture with a three-layer feedforward NN. The network has a unipolar sigmoid transfer function in each of the hidden and output layers' neurons. Experiments demonstrate that the approach achieves an overall accuracy of 99.4% in the binary classification of DoD/DDoS attacks and normal traffic. This approach, however, requires a server deployment to run the NN models.

Tuor et al.[27] propose an online unsupervised DL approach that detected anomalous network activity from system logs in real time. They train DNNs to learn to recognize anomalous user activities and then deploy them to classify user behavior as "anomalous" or "benign." For the DNNs and RNNs, they tune the number of hidden layers (between one and six) and the hidden-layer dimension (between 20 and 500) using a random hyperparameter search. These techniques have been evaluated with respect to the Computer Emergency Response Team Insider Threat v6.2 data set. The evaluation suggests that the techniques achieve an average anomaly score in the 95.53 percentile. However, this approach does not take into account the activity patterns of individual users and requires human intervention to review the anomaly scores and make further decisions.

Katherios et al.[28] propose a real-time network anomaly-detection system that greatly reduces the manual workload by coupling two learning stages. The first

Table 2. A comparison of DL approaches used for intrusion detection.

References	Type	Detection method	Training data	Input features	Accuracy (%)
Cannady[24]	Host based	Misuse based	Custom	Network data packets (e.g., port, IP address, raw data, and so on)	100
Palagiri[25]	Network based	Misuse based	DARPA 1999	Number of packets in a time interval to a specific port	100
Hodo et al.[26]	Network based	Misuse based	Custom	Internet packet traces	99.4
Tuor et al.[27]	Network based	Anomaly based	CERT Insider Threat v6.2 data set	User system logs	95.53
Katherios et al.[28]	Network based	Anomaly based	Real world	Extracted from flow-aggregating network packets	98.5

stage performs adaptive unsupervised anomaly detection using a shallow autoencoder, while the second stage uses a custom nearest-neighbor classifier to filter the false positives by modeling the manual classification. The experiments have been carried out on real-world data from the network of the National Information Infrastructure Development Institute of Hungary. The experiments show that the techniques achieve 98.5 and 1.3% for true-positive rates and FPRs, respectively, while reducing the human intervention rate by a factor of five. This approach also shows very good detection latency of a few seconds from the start of the attack.

From the approaches discussed in this section, it is evident DL-based techniques can significantly enhance traditional IDS deployments. However, most approaches have not been evaluated with respect to latency and resource costs. These metrics are very important for the real-world deployment of these techniques because an IDS with high latency cannot reliably detect attacks in real time. Additionally, their high resource costs make it difficult, if not impossible, to deploy these techniques in host-based IDS, especially for IoT devices with low processing capability. Furthermore, Sommer and Paxson[29] argued that the task of finding attacks is fundamentally different from other applications, making it significantly harder for the intrusion detection community to employ ML effectively. They mention various challenges, such as the diversity of network traffic, evaluation difficulties, and high error cost, all of which must be addressed for ML techniques to effectively detect intrusion.

Botnet Detection

Botnets are defined as a network of computers infected with malicious software and controlled as a group without the computer owners' knowledge. Botnets are used to orchestrate DDoS attacks, send spam, and enable attackers to access and control impacted devices remotely. Botnets have been increasingly targeting IoT devices;[30] recently discovered IoT botnets, such as Mirai and Reaper, have proved that the heterogeneity and weak security of these devices make them easily exploitable. Traditional security mechanisms have failed to stop the growth of IoT botnets because they typically require many resources and do not consider the heterogeneity of IoT networks; however, DL techniques have proven useful for detecting and disrupting botnets.

Nogueira et al.[31] first proposed using NNs as a botnet detection methodology by uniquely identifying characteristic traffic patterns and identifying the ones generated by zombie devices (Table 3). They argue that all Internet applications (e.g., file sharing, HTTP browsing, and VoIP) have a specific traffic profile that can be used to train different NNs for particular applications, as modules. They achieved a detection performance of more than 87% for eight different traffic profiles. This approach is a good starting point, but it is not scalable because each application requires a separate NN for detection purposes.

Karim et al.[32] proposed a framework, called *SMARTbot*, which uses NNs to detect botnet binaries on mobile devices. For testing, they used their classifier to differentiate among botnet and other malware. Their system achieves 99.49% accuracy in detecting botnet applications; however, their work does not report any assessments about memory requirements and execution times. These assessments are critical for determining whether such a system can actually be deployed in IoT systems.

Arnaldo et al.[33] introduce a framework that identifies advanced, persistent threats by using log data generated by enterprise-grade security devices. This framework is applied to detect the communication channel between the compromised host and the botnet command and control, which is necessary for controlling the botnet. They train various DL models, including CNNs, RNNs, and autoencoders using features extracted from the ISCX botnet data set.[34] Their framework can detect previously unseen botnets with high accuracy; however, it requires using multiple days of log data to detect a botnet.

Table 3. A comparison of DL approaches used for botnet detection.

Publication	Network type	Data set	Botnets detected	Detection rate (%)
Nogueira et al.[31]	Any network	Custom (self generated)	Simple port scanning	87
Karim et al.[32]	Mobile devices	Custom	Botnet mobile apps	99.49
Arnaldo et al.[33]	Any network	ISCX botnet data set[34]	Neris, Rbot, and so on	—
Meidan et al.[35]	IoT networks	Custom (self generated)	Mirai and Bashlite	100
Yu et al.[36]	Any network	CTU-UNB and Contagio-CTU-UNB	Neris, Rbot, and so on	98.98

Meidan et al.[35] propose the use of deep autoencoders to detect anomalous network traffic generated by compromised IoT devices. They suggest using a network-based anomaly-detection method, which extracts behavior snapshots of the network and trains each device's deep autoencoders to learn the IoT's normal behaviors. Such an approach can then detect when the device is compromised and whether it exhibits anomalous behavior. They use the context of the network packets by extracting 115 traffic statistics from several temporal windows to summarize the traffic. They have evaluated their approach on the Bashlite and Mirai botnet families, and their experiments produced a detection rate of 100% and an FPR of 0.007. This approach achieves a very high detection rate; however, the scalability of this approach is questionable because it requires a separate NN for modeling each IoT's device-type behavior.

Yu et al.[36] propose a model created by stacking DCAEs, which can learn directly from unlabeled raw traffic data. They divide the training process into unsupervised pretraining and supervised fine-tuning. Their model achieves 99.59% accuracy for binary classification. Other relevant approaches include those introduced by Wang et al.[37] and Kant et al.[38] that use CNNs for malware and botnet classification by converting traffic data into images. Maimo et al.[39] recently suggested using DL in 5G networks for botnet detection by detecting anomalies in the network.

Many of the techniques used for detecting botnets focus on IoT devices and mobile devices. IoT devices are attractive to attackers because they have weak defenses and inadequate built-in security due to their limited computing power. Similar to DL techniques involving malware analysis and intrusion detection, DL-based techniques for botnet detection have not been assessed for memory requirements and latency. This is even more crucial in the case of mobile and IoT devices, which already have minimal computational power. DL-based solutions for detecting botnets are often network-based because the DL algorithms have resource requirements that are too high for direct deployment to the devices themselves. Further research must be devoted to enhancing the resource efficiency of DL algorithms deployed to individual devices and to exploiting the additional information available at the device itself. It is important that DL-based techniques for botnet detection be able to detect previously unknown botnets in real time, so that any attack can be promptly identified and addressed. Many proposed techniques deal only with botnets encountered in training; however, techniques that do identify unseen botnets must be individually trained on IoT devices to identify their benign behavior, which can be a problem for real-world deployment.

Attacks Against NNs

As previously discussed, recent research has shown that NNs can be very effective for various security tasks. However, it is critical that they be proved resilient to tampering and adversarial compromise. As the use of DL in image recognition, object detection, speech recognition, translation, and so on has grown, many researchers have explored methods that deceive NN-based models by slightly modifying the inputs to achieve different output results. These inputs are called *adversarial examples*. This remains a major problem in moving toward fully automated NN-based techniques for various tasks in information security because they do not have any proper security guarantees in adversarial settings.

Szegedy et al.[40] were the first to point out that certain barely perceptible perturbations in the input images caused NN-based image classifiers to misclassify with a high probability. Papernot et al.[41] show how to generate inputs that are misclassified by a DNN with a 97% success rate, while modifying only, on average, 4.02% of the input features per sample. Such an attack requires knowledge of the DNN architecture, which is a reasonable requirement. One of the key assumptions they make is that the DNNs are feedforward and therefore do not consider RNNs in their experiments. There are several papers that suggest methods for generating adversarial examples against NNs.[42–44]

Significant research efforts have been devoted to enhancing the robustness of NNs to make them resilient to adversarial examples. Huang et al.[45] propose improving classifiers by generating adversarial examples as an intermediate step and learning from them. Many researchers espouse the use of defensive distillation to prevent adversarial examples;[46,47] however, Carlini and Wagner[48] have proven these approaches to be inadequate. Others have recommended using feature squeezing as a defense mechanism.[49,50] Bhagoji et al.[51] advise using dimensionality reduction via principal component analysis and data "antiwhitening" to prevent evasion attacks. Madry et al.[52] suggest that to capture the notion of security against adversarial attacks in a principled manner, we should formally define the proper security guarantees. Yuan et al.[53] introduced methods for generating adversarial examples and related countermeasures. Many other approaches have also been proposed to protect NNs from attacks.[54–57]

The aforementioned research efforts have mainly focused on mainstream NN applications, such as image classification and speech and text recognition, and have not yet addressed NN applications relative to security. However, because adversarial examples question the reliability of NNs, any application that uses them is potentially affected. This lack of reliability is particularly critical when NNs are used as part of information security

systems and tools because attackers are always looking for ways to exploit system weaknesses. It is also very difficult to verify whether the trained model has been tampered with because there are thousands of parameters that are learned and stored within an NN model. Furthermore, NNs lack the necessary formal security guarantees to be considered resilient against adversarial attacks. The results should be consistent enough, and it should not be easy for attackers to make small, undetectable changes to the input and bypass security checks.

DL techniques have proven to be successful with helping security professionals tackle the problem of new, ever-increasing malware attacks. However, most of the DL-based approaches for information security tasks merely focus on using NNs to provide the highest detection accuracy, but there is very little analysis about the resource costs of these methods. This may be primarily due to the nascent stage of the research in this area as well as the characteristics of DL and NNs themselves. However, if these techniques are to be successfully deployed in real-world networks and security products, their resource requirements must be analyzed and optimized. The system requirements of most DL methods make them unsuitable for IoT and edge devices because these devices often have limited computing capacity. Most DL techniques require server machines, possibly with general-purpose GPUs having thousands of cores, to efficiently carry out DL workloads. IoT and edge devices, however, have lower-power GPUs or no GPUs at all; as a result, they are very slow to service DL workloads. The solution to this problem is twofold: 1) DL and NN algorithms, implementations, and frameworks must be better implemented and engineered to be more efficient on small devices, and 2) the DL techniques being developed for security tasks should have lightweight NNs and efficient steps for preprocessing the input data. Acceleration at the edge using network pruning can also be explored by using pruning techniques such as those proposed by Tang and Han[58] and Narang et al.[59] Network pruning has been shown to greatly reduce the model size and thus the latency of CNNs by up to five times mainly in image recognition tasks, while at the same time maintaining the same accuracy levels as the nonpruned CNNs. However, the use of pruning techniques has not yet been explored in NNs used for security tasks.

Additionally, with many anomaly-based IDS implementations focusing on the traffic patterns of specific devices, it is difficult and time-consuming to capture the usage characteristics for new but similar devices and then retrain the NN models from scratch. Transfer learning can address this requirement because it enables the transfer of learned features and knowledge from a trained source model to a target model with minimal new training data. Some interesting research directions related to the use of DL, specifically in the case of IoT devices, include investigating whether an anomaly-detection model trained on a specific type of IoT device transfers to other device types with similar or different functionalities. For example, an anomaly-detection model trained on smart locks could be used as a source model for a smart lock from a different vendor or for a completely different IoT device, such as a thermostat. Those two types of transfer (i.e., to devices with similar functionalities and to devices with different functionalities) can be combined to obtain a general NN for anomaly detection in devices. Past research has shown that the traffic patterns for different devices vary significantly, so a general network may not be highly accurate (i.e., a high FPR), but it is still worth exploring. ■

Acknowledgments

This work is supported by National Science Foundation grant 1719369 and by the U.S. Army Research Laboratory the and U.K. Ministry of Defense under agreement number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. government, the U.K. Ministry of Defense, or the U.K. government. The U.S. and U.K. governments are authorized to reproduce and distribute reprints for government purposes notwithstanding any copyright notation hereon.

References

1. Cornell Law School, "44 U.S. Code § 3542—definitions." Accessed on: Nov. 20, 2018. [Online]. Available: https://www.law.cornell.edu/uscode/text/44/3542
2. N. Minihane et al. (2017). McAfee labs threats report. McAfee. Santa Clara, CA. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-dec-2017.pdf
3. A. Ng, "Why is deep learning taking off?" deeplearning.ai, Stanford Univ., CA, 2018. [Online]. Available: https://youtu.be/xflCLdJh0n0
4. J. McCarthy, "What is artificial intelligence?" Nov. 12, 2007. [Online]. Available: http://www-formal.stanford.edu/jmc/whatisai/whatisai.html
5. J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane, "Automated design of both the topology and sizing of analog electrical circuits using genetic programming," in *Artificial Intelligence in Design' 96*, J. S. Gero and F. Sudweeks, Eds. New York: Springer Netherlands, 1996, pp. 151–170.
6. J. R. Quinlan, "Induction of decision trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81–106, 1986.

7. M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf, "Support vector machines," *IEEE Intell. Syst. Appl.*, vol. 13, no. 4, pp. 18–28, 1998.
8. T. D. Nielsen and F. V. Jensen, *Bayesian Networks and Decision Graphs*. New York: Springer-Verlag, 2009.
9. N. M. Nasrabadi, "Pattern recognition and machine learning," *J. Electron. Imag.*, vol. 16, no. 4, pp. 049901, 2007.
10. M. van Gerven and S. M. Bohte, "Editorial: Artificial neural networks as models of neural information processing," *Front. Comput. Neurosci.*, vol. 11, no. 114, Dec. 2017. doi: 10.3389/fncom.2017.00114.
11. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
12. Skymind, "Recurrent networks." Accessed on: Nov. 20, 2018. [Online]. Available: http://skymind.ai/wiki/recurrent-network-rnn
13. F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.
14. F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," Cornell Aeronautical Lab. Buffalo, NY, 1961. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/256582.pdf
15. L. Deng, M. L. Seltzer, D. Yu, A. Acero, A-r Mohamed, and G. Hinton, "Binary coding of speech spectrograms using a deep auto-encoder," in *Proc. 11th Annu. Conf. Int. Speech Communication Assoc.*, 2010, pp. 1692–1695.
16. V. Dumoulin and F. Visin, A guide to convolution arithmetic for deep learning. 2016. [Online]. Available: arXiv:1603.07285v2
17. J. O. Kephart, G. B. Sorkin, W. C. Arnold, D. M. Chess, G. J. Tesauro, and S. R. White, "Biologically inspired defenses against computer viruses," *IJCAI*, pp. 985–996, Aug. 1995.
18. G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, 2013, pp. 3422–3426.
19. J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proc. 10th Int. Conf. Malicious and Unwanted Software (MALWARE)*, 2015, pp. 11–20.
20. B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Australasian Joint Conf. on Artificial Intelligence* 2016, pp. 137–149.
21. R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, "Malware classification with recurrent networks," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 1916–1920.
22. W. Huang and J. W. Stokes, "Mtnet: A multi-task neural network for dynamic malware classification," in *Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016, pp. 399–418.
23. A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surveys Tut.*, vol. 18, no. 2, pp. 1153–1176, 2016.
24. J. Cannady, "Artificial neural networks for misuse detection," in *Proc. Nat. Information Systems Security Conf.*, 1998.
25. A. Bivens, C. Palagiri, R. Smith, B. Szymanski, and M. Embrechts, "Network-based intrusion detection using neural networks," in *Proc. Intelligent Engineering Systems through Artificial Neural Networks ANNIE-2002*, 2002, pp. 579–584.
26. E. Hodo et al., "Threat analysis of IoT networks using artificial neural network intrusion detection system," in *Proc. IEEE Int. Symp. Networks, Computers and Communications (ISNCC)*, 2016, pp. 1–6. doi: 10.1109/ISNCC.2016.7746067.
27. A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. 2017. [Online]. Available: arXiv:1710.00811v2
28. G. Kathareios, A. Anghel, A. Mate, R. Clauberg, and M. Gusat, "Catch it if you can: Real-time network anomaly detection with low false alarm rates," in *Proc. IEEE Int. Conf. Machine Learning and Applications (ICMLA)*, 2017, pp. 924–929.
29. R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Proc. IEEE Symp. Security and Privacy (SP)*, 2010, pp. 305–316.
30. E. Bertino and N. Islam, "Botnets and Internet of Things security," *Computer*, vol. 50, no. 2, pp. 76–79, 2017.
31. A. Nogueira, P. Salvador, and F. Blessa, "A botnet detection system based on neural networks," in *Proc. IEEE 5th Int. Conf. Digital Telecommunications (ICDT)*, 2010, pp. 57–62.
32. A. Karim, R. Salleh, and M. K. Khan, "SMARTbot: A behavioral analysis framework augmented with machine learning to identify mobile botnet applications," *PLOS One*, vol. 11, no. 3, 2016. doi: 10.1371/journal.pone.0150077.
33. I. Arnaldo, A. Cuesta-Infante, A. Arun, M. Lam, C. Bassias, and K. Veeramachaneni, "Learning representations for log data in cybersecurity," in *Int. Conf. Cyber Security Cryptography and Machine Learning*, 2017, pp. 250–268.
34. E. B. Beigi, H. H. Jazi, N. Stakhanova, and A. A. Ghorbani, "Towards effective feature selection in machine learning-based botnet detection approaches," in *Proc. IEEE Conf. Communications and Network Security (CNS)*, 2014, pp. 247–255.
35. Y. Meidan et al., "N-BaloT: Network-based detection of IoT botnet attacks using deep autoencoders," *Pervasive Comput.*, vol. 17, no. 3, pp. 12–22, 2018.
36. Y. Yu, J. Long, and Z. Cai, "Network intrusion detection through stacking dilated convolutional autoencoders," *Security Commun. Netw.*, vol. 2017, 2017. doi: 10.1155/2017/4184196.
37. W. Wang, M. Zhu, X. Zeng, X. Ye, and Y. Sheng, "Malware traffic classification using convolutional neural network

for representation learning," in *Proc. IEEE Int. Conf. Information Networking (ICOIN)*, 2017, pp. 712–717.
38. V. Kant, E. M. Singh, and N. Ojha, "An efficient flow based botnet classification using convolution neural network," in *Proc. IEEE Int. Conf. Intelligent Computing and Control Systems (ICICCS)*, 2017, pp. 941–946.
39. L. F. Maimó, Á. L. P. Gómez, F. J. G. Clemente, M. G. Pérez, and G. M. Pérez, "A self-adaptive deep learning-based system for anomaly detection in 5G networks," *IEEE Access*, vol. 6, pp. 7700–7712, 2018. doi: 10.1109/ACCESS.2018.2803446.
40. C. Szegedy et al., Intriguing properties of neural networks. 2013. [Online]. Available: arXiv:1312.6199v4
41. N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *Proc. IEEE European Symp. Security and Privacy (EuroS&P)*, 2016, pp. 372–387.
42. I. J. Goodfellow, J. Shlens, and C. Szegedy, Explaining and harnessing adversarial examples. 2014. [Online]. Available: arXiv:1412.6572v3
43. A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2015, pp. 427–436.
44. M. Sharif, S. Bhagavatula, L. Bauer, and M. K. Reiter, "Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition," in *Proc. ACM SIGSAC Conf. Computer and Communications Security*, 2016, pp. 1528–1540.
45. R. Huang, B. Xu, D. Schuurmans, and C. Szepesvári, Learning with a strong adversary. 2015. [Online]. Available: arXiv:1511.03034v6
46. N. Papernot, P. D. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proc. IEEE Symp. Security and Privacy (SP)*, 2016, pp. 582–597.
47. N. Papernot and P. D. McDaniel, On the effectiveness of defensive distillation. 2016. [Online]. Available: arXiv: 1607.05113v1
48. N. Carlini and D. A. Wagner, Defensive distillation is not robust to adversarial examples. 2016. [Online]. Available: arXiv:1607.04311v1
49. W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," in *Proc. 25th Annu. Network and Distributed System Security Symp. (NDSS)*, 2018.
50. W. He, J. Wei, X. Chen, N. Carlini, and D. Song, "Adversarial example defense: Ensembles of weak defenses are not strong," in *Proc. 11th USENIX Workshop Offensive Technologies, (WOOT)*, 2017.
51. A. N. Bhagoji, D. Cullina, and P. Mittal, Dimensionality reduction as a defense against evasion attacks on machine learning classifiers. 2017. [Online]. Available: arXiv:1704.02654v4
52. A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, Towards deep learning models resistant to adversarial attacks. 2017. [Online]. Available: arXiv:1706.06083v3
53. X. Yuan, P. He, Q. Zhu, R. R. Bhat, and X. Li, Adversarial examples: Attacks and defenses for deep learning. 2017. [Online]. Available: arXiv:1511.05432v3
54. U. Shaham, Y. Yamada, and S. Negahban, Understanding adversarial training: Increasing local stability of neural nets through robust optimization. 2015. [Online]. Available: https://arxiv.org/abs/1511.05432 arXiv:1511.05432v3
55. O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Proc. Advances Neural Information Processing Systems*, 2016, pp. 2613–2621.
56. S. Gu and L. Rigazio, Towards deep neural network architectures robust to adversarial examples. 2014. [Online]. Available: arXiv:1412.5068v4
57. K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, Adversarial perturbations against deep neural networks for malware classification. 2016. [Online]. Available: arXiv:1606.04435v2
58. S. Tang and J. Han, "A pruning based method to learn both weights and connections for LSTM," 2015. [Online]. Available: https://nlp.stanford.edu/courses/cs224n/2015/reports/2.pdf
59. S. Narang, G. F. Diamos, S. Sengupta, and E. Elsen, Exploring sparsity in recurrent neural networks. 2017. [Online]. Available: arXiv:1704.05119v2

Ankush Singla is a Ph.D. student with the Computer Science Department at Purdue University specializing in information security. His research interests include security analytics for Internet of Things devices, authentication techniques, and hardware acceleration for edge devices. Singla received an M.S. in information security and assurance from Purdue University. Contact him at asingla@purdue.edu.

Elisa Bertino is the Samuel Conte Term Professor of Computer Science at Purdue University and director of Cyber2Slab. Her research interests include security and privacy of data and Internet of Things systems and analytics for security. Bertino received a doctorate in computer science from the University of Pisa, Italy. She is a Fellow of the IEEE, Association for Computing Machinery (ACM), and the American Association for the Advancement of Science. She received the IEEE Computer Society 2002 Technical Achievement Award, the IEEE Computer Society 2005 Kanai Award, and the ACM Special Interest Group on Security, Audit, and Control Outstanding Contributions Award. Contact her at bertino@purdue.edu.