Neurocomputing 347 (2019) 149–176

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Application of deep learning to cybersecurity: A survey

Samaneh Mahdavifar*, Ali A. Ghorbani

Canadian Institute for Cybersecurity, Faculty of Computer Science, University of New Brunswick, Fredericton E3B 5A3, NB, Canada

ARTICLE INFO

Article history:
Received 13 August 2018
Revised 22 January 2019
Accepted 27 February 2019
Available online 6 March 2019

Communicated by Dr. F.A. Khan

Keywords:
Cybersecurity
Deep learning
Neural network
Intrusion detection
Malware detection
Malware classification

ABSTRACT

Cutting edge Deep Learning (DL) techniques have been widely applied to areas like image processing and speech recognition so far. Likewise, some DL work has been done in the area of cybersecurity. In this survey, we focus on recent DL approaches that have been proposed in the area of cybersecurity, namely intrusion detection, malware detection, phishing/spam detection, and website defacement detection. First, preliminary definitions of popular DL models and algorithms are described. Then, a general DL framework for cybersecurity applications is proposed and explained based on the four major modules it consists of. Afterward, related papers are summarized and analyzed with regard to the focus area, methodology, model applicability, and feature granularity. Finally, concluding remarks and future work are discussed including the possible research topics that can be taken into consideration to enhance various cybersecurity applications using DL models.

1. Introduction

Today, the Internet has become an indispensable necessity of everyone's life making this widespread interconnected network prone to miscellaneous threats. There exist several security threats in cyberspace from jail-breaking to two-faced malware and network intrusions. These threats have established an arms race in terms of security. Many security companies around the globe are focusing on designing new technologies to protect computer devices, networks, and software applications from network intrusion attacks and malware infections.

There are two conventional security systems, called network security systems and host security systems, that protect the underlying network and computers from unauthorized access, destruction, malfunction, and modification. Both of these systems may consist of different integrated security modules, such as firewalls, Intrusion Detection Systems (IDSs), and antiviruses that help monitor a system or network and raise an alarm when a malicious activity occurs.

Intrusion detection is believed to be a necessary security mechanism to deal with network attacks and identify malicious activities in computer network traffics. It plays a vital role in information security technology and helps in discovering, determining, and identifying unauthorized use, duplication, alteration, and destruction of information and information systems [1].

Broadly speaking, IDSs are categorized into three different methods: misuse detection, anomaly detection, and hybrid. Misuse detection techniques use pre-defined signatures of malicious activity to identify intrusions. Therefore, they are used for detecting known attacks only. On the other hand, anomaly detection techniques define normal patterns and identify malicious activities based on the deviations from normal patterns. So anomaly-based detection methods have the capability of detecting zero-day attacks. Hybrid techniques take advantage of both misuse detection and anomaly detection methods. While reducing the false positives of unknown attacks, hybrid approaches aim at increasing detection rates of known intrusions [2].

Malware has recently posed serious security issues and threats to Internet users making the detection of malware of the utmost concern. These intrusive software programs, such as worms, viruses, trojans, botnets, ransomware, and so on have been exploited by fraud actors to organize a number of security attacks against computer systems and jeopardize the confidentiality and integrity of communicated data and the availability of the services offered by the underlying infrastructure.

Lots of vendors, such as Kaspersky, Symantec, Microsoft, McAfee, and Invincea have developed anti-virus products to defend computers and legitimate users from malware attacks. These vendors normally use signature-based methods to detect malware. Although signature-based methods are somewhat effective, they are unable to identify zero-day malware which may be obfuscated by malware authors. The unprecedented volume of daily malware

* Corresponding author.
E-mail addresses: smahdavi@unb.ca (S. Mahdavifar), ghorbani@unb.ca (A.A. Ghorbani).

https://doi.org/10.1016/j.neucom.2019.02.056
0925-2312/

production necessitates devising accurate automated systems to detect and classify malware.

With the tremendous growth of image recognition, natural language processing, bioinformatics, and speech recognition applications, DL algorithms have established a key role in solving complicated problems, thanks to their various advantages compared to other traditional Machine Learning (ML) techniques. DL is defined as several multi-layered ML algorithms that are powerful at learning high-level abstractions of complex large-scale data. DL algorithms typically learn feature representations using many non-linear hidden layers making feature engineering automatic. As a result, the burden of spending extra time and money to hire engineers to re-engineer the features as new types of malware or network attacks emerge is eliminated. DL allows different cybersecurity companies to update their IDSs and malware detection systems that optimally cost them nothing.

In this survey, we study the application of DL to cybersecurity and several DL models that have been applied to the areas of intrusion detection, malware detection, phishing/spam detection, and website defacement detection in the literature. To the best of our knowledge, this is the first survey that presents a detailed literature review of cybersecurity emphasizing well-known DL algorithm descriptions. Although some research has been conducted so far reviewing ML and Data Mining (DM) techniques for intrusion detection or malware detection [2–4], only a few of them provide an overview of DL methods in intrusion detection [5,6] while no survey exists investigating DL techniques in malware detection or phishing detection.

This survey is organized as follows. Section 2 provides a brief history of Neural Network (NN) and DL and an overview of basic DL and Deep Network (DN) architectures. In Section 3 a conceptual DL framework for cybersecurity applications is proposed. In Section 4 several studies are reviewed in which different DL techniques have been applied to the cybersecurity area and the related shortcomings are discussed. In Section 5, we analyze a wide range of studies on the basis of four criteria: focus area, methodology, model applicability, and feature granularity. Finally, concluding remarks and directions for future work are presented in Section 6.

2. Basic concepts

In this section, we present a brief history of NNs and DL. Also, we offer a comprehensive overview of basic DL and DN architectures.

2.1. A brief history

The evolution of NN has involved many ups and downs since the mid 20th century. The first inspirations date back to 1943 initiated by McCulloch and Pitts [7]. They borrowed the idea from biological neurons and proposed a computational model for constructing hypothetical nets.

Later at IBM research laboratories, a hypothetical NN was simulated by Nathaniel Rochester, which was not a successful attempt.

Subsequently, in 1958, perceptron was developed at Cornell Aeronautical Laboratory, by Frank Rosenblatt [8,9], which was the first learning machine.

In 1959, two models called “ADALINE” and “MADALINE” were developed by Widrow and Hoff at Stanford [10]. ADALINE was designed for recognizing binary patterns and predicting subsequent bits while reading streaming data from a phone line [11,12]. MADALINE [13] was a three-layer fully-connected, Feed-Forward Neural Network (FFNN) with ADALINE units as its hidden and output layer. Using an adaptive filter, it was the first NN applied to a real-world problem that aimed at eliminating echoes on phone lines.

Although the above preliminary experiments induced a bright future for the NNs in the first place, the flaws reported by Minsky and Papert [14], that a single perceptron is unable to implement XOR function, made even the most ardent proponents hesitant about the success of NNs.

With the introduction of Back-Propagation (BP) learning algorithms [15,16] between the 1970s and ’80s, the use of NNs gained popularity and then was escalated by the work of Rumelhart et al. [17] in the mid ’80s. They demonstrated that there is a possibility of learning the internal representation of the input patterns in the hidden units using error propagation. They generalized the delta rule for evolving NN weights to produce an arbitrary mapping from input to output.

In the late ’80s, several fundamental studies of basic DL architectures were published, among which the works of Ballard [18] and LeCun [19], in 1987 and 1989, respectively, are of paramount importance. With the aim of improving the performance of learning algorithms in large-scale systems, Ballard used unsupervised Auto-Encoder (AE) hierarchies to pre-train networks. LeCun et al. applied BP to the recognition of handwritten zip code digits. They used Convolutional Neural Network (CNN) with adaptive connections having two types of layers of convolution and subsampling.

At the beginning of the 1990s, Hochreiter opened the “long time lag problem”, so-called, the vanishing gradient problem [20] that causes Recurrent Neural Network (RNN) to fail in case of long minimal time lags between input signals and corresponding error signals. This well-known problem induced some research directions by Bengio et al. [21,22] and Hochreiter et al. who proposed Long Short-Term Memory (LSTM) [23] to solve the “long time lag problem” having a memory cell as its basic unit.

In 1992, one year after addressing the “gradient vanishing problem”, Weng et al. presented a self-organizing NN namely, the Cresceptron model [24] that grows incrementally to memorize new concepts detected automatically at every level of the hierarchy. Max-Pooling (MP) layers were introduced in the NN architectures and since then are widely used in image recognition tasks.

Despite all the foregoing advancements in NNs, it was not sooner than 2006 that DL emerged as what we perceive today. Hinton et al. illuminated this way by inventing Deep Belief Network (DBN) based on Restricted Boltzmann Machine (RMB) that helps us to learn a generative model which outperforms discriminative methods on the MNIST database of hand-written digits [25].

Today, DL is used dominantly in almost every application. It is simply a new variation of the classical Multilayer Perceptron (MLP). The goal of DL algorithms is to produce high-level and flexible features from the raw input data that help generalization in the classification. DL addresses complex applications with millions of data that require a large number of neurons and hidden layers. To this end, a myriad of DL frameworks, such as TensorFlow [26], Caffe [27], or Theano [28], have been developed in recent years that provide the building blocks for implementing DN architectures efficiently and eliminate the necessity of coding from scratch.

2.2. Deep learning in a nutshell

DL allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction [29]. DL architectures are usually constructed as multilayer networks so that more abstract features are computed as nonlinear functions of lower-level features [30]. The most popular kinds of DL models are CNN, RNN, and DBN, which have been widely applied to large scale image recognition tasks, natural language processing, bioinformatics, and speech recognition, to name a few.

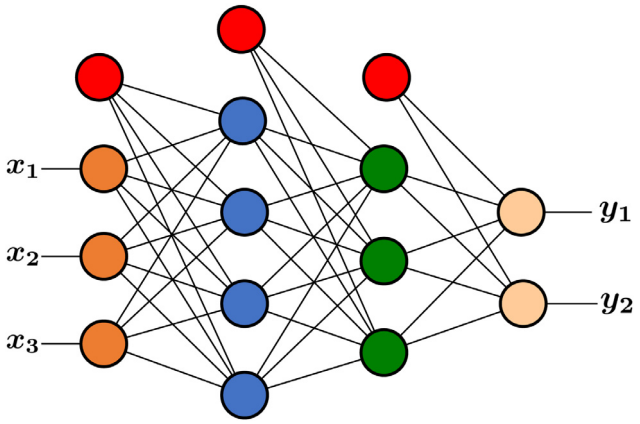


Fig. 1. A DNN with two hidden layers. Red circles represent the bias nodes.

Feature engineering is the main difference between traditional ML and DL algorithms. While traditional ML algorithms handcraft the features, DL algorithms aim at extracting the features automatically, resulting in more accurate ML models. Using many processing layers, DL techniques can address large scale data problems compared with the shallow ML algorithms.

One of the most common forms of DL architectures is FFNN, a.k.a Deep Neural Network (DNN). A DNN usually consists of an input layer followed by several hidden layers and an output layer. The input layer receives an input feature vector representing the object to be classified. The output layer is responsible for producing the class probability vector associated with the input vector [31].

Fig. 1 depicts a typical DNN with two hidden layers. Each node in the model, acting as a neuron, consumes the output of the previous layer plus a bias from a special neuron. It then computes a weighted average of its inputs referred to as the total input. For each hidden unit j, the output y_j produces a numerical output $y_j = f(\sum_{i=1}^{n} w_{ji}x_i + b_j)$ where b_j is the bias term and w_{ji} are the elements of a layer's weight matrix. Introducing non-linearities to the DNN model, the function $f(.)$ is called the activation function that defines the output of a hidden unit.

A DNN is trained using BP with the Gradient Descent (GD) optimization algorithm to update the weights of neurons by calculating the gradient of loss function with respect to all weights. First, there is the forward propagation of training samples' inputs to generate the output values. Then the output values are back-propagated through the network to compute delta values, i.e., the difference between the targeted and actual output values. For each weight, output delta and input activation are multiplied to obtain the gradient and then a percentage of the gradient is used to adjust the weight.

2.3. Deep network architectures

In this subsection, we explain some state-of-the-art DN architectures that have attracted much attention among researchers because of the incomparable accuracy they have in modeling and classifying complex data.

2.3.1. Convolutional neural network

With the rising prevalence of image processing applications, DL algorithms have established a key role in solving image recognition problems, thanks to their various advantages compared to other traditional ML techniques. CNN is one of the robust models that substantially fit this area, as its improved network structure leads to better performance and computation savings, especially for the applications having high local correlation inputs, like image and document [32,33].

In FFNNs, each neuron in a hidden layer is fully connected to all neurons in the previous layer. Apparently, this fully-connected structure does not function well with large scale input images, due to the huge number of parameters, such as weights and biases. CNN organizes neurons in a 3-dimensional manner, width, height, and depth. Since the input images are very high dimensional, each neuron will only be connected to a small region of the layer before it, instead of all the neurons in a fully-connected manner. Each layer of CNN transforms the 3D input volume to a 3D output volume of neuron activations [34].

Likewise any conventional NN model, CNN consists of an input layer, an output layer, and multiple hidden layers. The hidden layers of a CNN usually contain one or more convolutional layers, pooling layers, and fully connected layers:

- Input layer holds the raw pixel values of an image.
- Convolutional layer performs dot products between the weights of output neurons (filters) and a local region in the input volume to which they are connected, called receptive field of the neuron. This connection is within the full depth (three color channels, R, G, and B), but local within width and height.
- Pooling layer is located between convolutional layers to reduce the number of parameters and computations in CNN [35,36]. It performs downsampling along the spatial dimensions (width and height) and makes the features robust against noise and distortion. The most common pooling algorithms are called MP and Average-Pooling (AP).
- Like an FFNN, in a fully connected layer, all neurons are connected to all neurons in the previous layer and their activations are computed by matrix multiplication followed by a bias offset [34].

Fig. 2 depicts an overall architecture of CNN for the classification task with two convolutional layers, two pooling layers, and one fully connected layer. The operations defined in these layers result in a set of extracted features which are reported in the form of feature maps [37]. The output layer computes the class scores and acts as a classifier.

There might be an activation layer which applies an element-wise activation function, such as Rectified Linear Unit (ReLU), $max(0, x)$, leaving the size of the volume unchanged. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolutional layer [38].

Performing most of the computational part, the convolutional layer plays the vital role of any CNN and acts as an automatic feature extractor. By restricting the receptive fields' size, the convolutional layer can extract local features of an input volume.

Three hyperparameters control the size of the output volume, namely depth, stride, and zero-padding. Depth refers to the number of the filters we intend to use, each looking for different aspects of the input image. Stride is used to produce smaller output volumes and describes the step with which we slide the filter into the receptive field of the neurons. Zero-padding allows us to add zeros around the border of the input volume. This parameter controls the spatial size of the output volumes.

2.4. Deep belief network

Invented by Hinton et al. [25], DBN is a probabilistic generative model that is composed of multiple layers of stochastic, latent variables [39]. DBN produces a joint probability distribution over observable data and labels, $p(label, observation)$, while discriminative models are limited to producing $p(label|observation)$. DBN has addressed the traditional issues of deep-layered NNs, namely

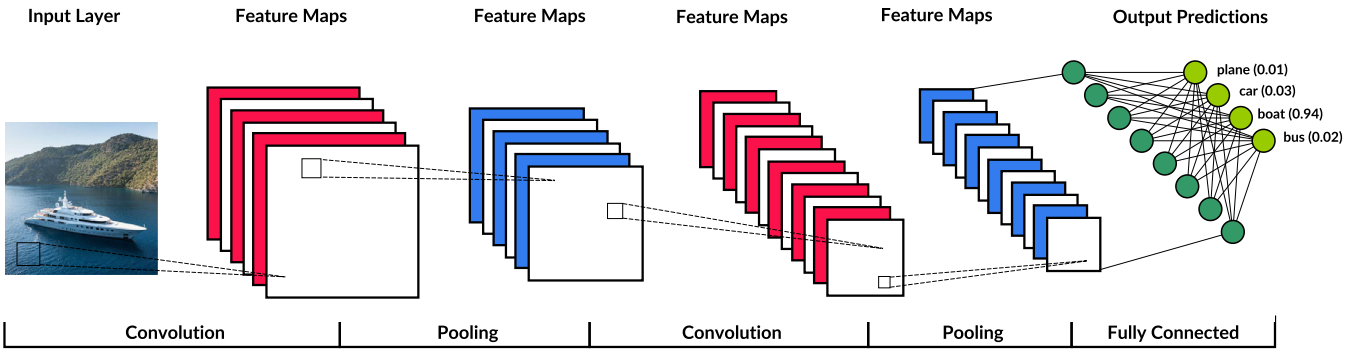


Fig. 2. Overall architecture of CNN for the classification task.

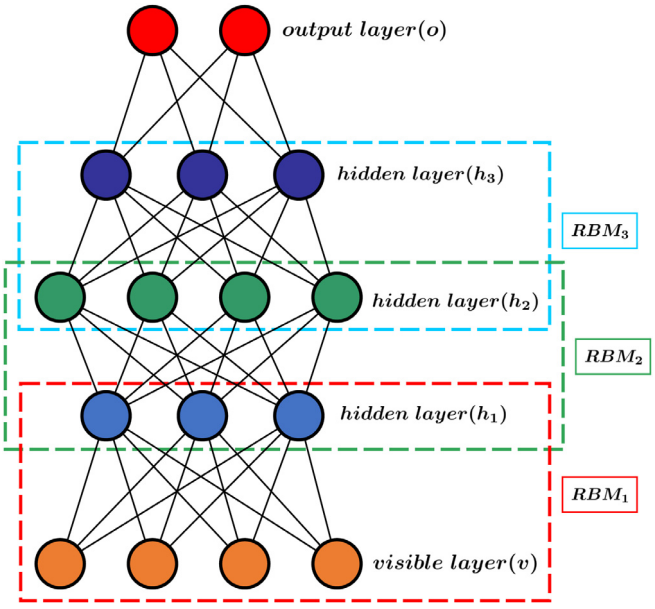


Fig. 3. A DBN with three hidden layers.

requiring a substantial labeled data set for training, slow learning times, and insufficient parameter selection techniques.

DBN is typically built by stacking several RBMs that capture high-order correlations that are observed at the visible units. DBN is pre-trained in an unsupervised greedy layer-wise fashion to learn a stack of RBMs by the Contrastive Divergence (CD) algorithm. The output feature representation of each RBM is used as the input data for training the next RBM in the stack. After the pre-training, the DBN is fine-tuned using BP of error derivatives and the initial weights and biases of each layer are corrected.

Fig. 3 shows an example of a DBN composed of three stacked RBMs. As can be seen from the figure, the input layer of the first RBM is the input layer for the whole network, and subsequently the hidden layer of RMB_t becomes the visible layer of RBM_{t+1}.

2.4.1. Restricted Boltzmann machine

RBM is an algorithm for representing all training samples in a more meaningful and compact way, by capturing their inner structure and regularities. This can be realized by introducing an extra group of neurons called hidden units to the network whose values are not directly set from training data [40]. In contrast, visible units obtain their values directly from training data. What makes an RBM different from the general class of Boltzmann Machines is that there are no hidden-to-hidden or visible-to-visible connections [41,42].

The global objective of RBM is to minimize the error between the input data and the reconstruction produced by the hidden units and their weights [43]. To avoid falling into the local minima problem, some noise is added to the network by making neuron's activations stochastic, i.e., having different activation functions, so the neurons can be active or inactive with some probability.

In Fig. 4, a two-layer RBM is shown. Apparently, the network has four visible nodes and three hidden nodes. In the forward pass, the output y_j is produced by multiplying each input, x_i by its corresponding weight w_{ij} and summing up all the products. The summation is then added to a bias, b, and finally, the result is passed through an activation function f to produce the node's output, y_j. To learn the connection weights of the network, we have to focus on how they learn to reconstruct data by themselves in an unsupervised fashion, making several forward and backward passes between the visible layer and hidden layer without involving a deeper network. In the reconstruction phase, the output activations of the forward pass, y_j, are the input to the backward pass. They are multiplied by the same weights of the forward pass, $w_1 ... w_n$, and the summations of the products are added to some new visible-layer biases which produce the final output of those operations which are the reconstructions of the original input, r_i. Certainly, the reconstructed values do not match the original ones. In other words, the reconstruction is making guesses about the probability distribution of the original input; i.e. the values of many varied points at once.

Assume the input data and the reconstructions are both normal curves of different shapes. The goal is to minimize the error or diverging areas under the two curves, called Kullback-Leibler (KL)-Divergence, using RBM's optimization algorithm CD [44]. CD learning approximately follows the gradient of the difference of two KL-Divergences [45]. This procedure helps us to iteratively adjust the weights and subsequently approximate the original data which makes the two probability distributions converge.

As can be seen, the probability distribution of a set of original input x, $p(x)$ and the reconstructed distribution $q(x)$ are put together in Fig. 5(a) while the integration of their differences appears in Fig. 5(b).

Let P and Q be the distributions of a continuous random variable; the KL-Divergence formula is defined as:

$$D_{KL}(P \parallel Q) = \int_{-\infty}^{+\infty} p(x) \log \frac{p(x)}{q(x)} dx, \tag{1}$$

where p and q denote the densities of P and Q [46].

To better understand the CD algorithm, let us get more involved with mathematical calculations.

Let v and h be visible and hidden units of an RBM, respectively. The energy of a joint configuration, (v,h) of the visible and hidden

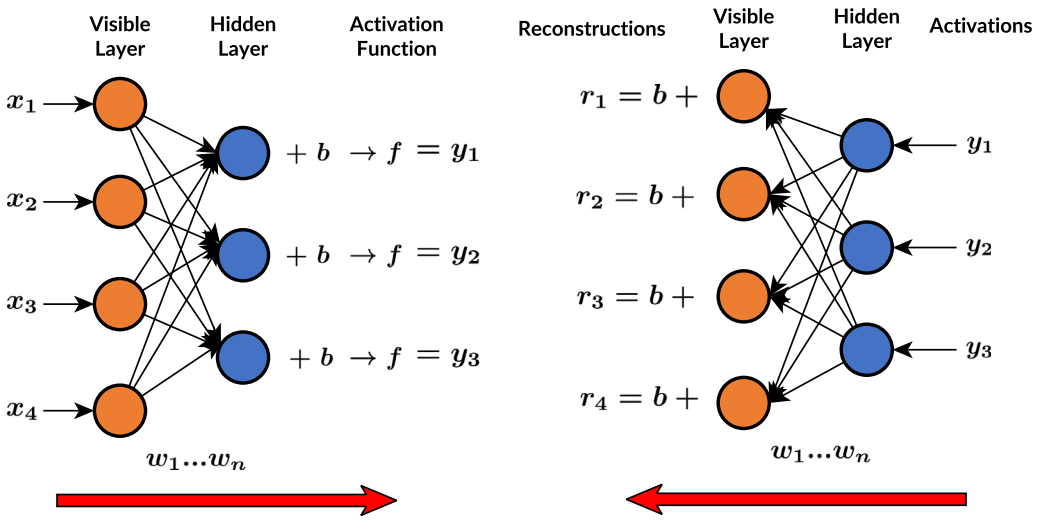


Fig. 4. A graphical representation of a two-layer RBM.

units is defined by [5,42,47]:

$$E(v,h) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij}, \tag{2}$$

where v_i, h_j are the binary states of visible node i and hidden node j, a_i and b_j are their bias values, and w_{ij} is the weight between v_i and h_j.

Then the joint probability over v and h would be computed as follows:

$$p(v,h) = \frac{1}{\sum_{v,h} e^{-E(v,h)}} e^{-E(v,h)}. \tag{3}$$

Suppose $p(v)$ is the probability that an RBM takes visible vector v by summing over all possible hidden vectors [42]:

$$p(v) = \frac{1}{\sum_{v,h} e^{-E(v,h)}} \sum_{h} e^{-E(v,h)}. \tag{4}$$

The formula for updating weights is calculated by taking derivative of the log $p(v)$ with respect to weight w_{ij}:

$$\Delta w_{i,j} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}), \tag{5}$$

where ϵ is the learning rate and $\langle\rangle$ are defined as the expectation under the data or model distributions.

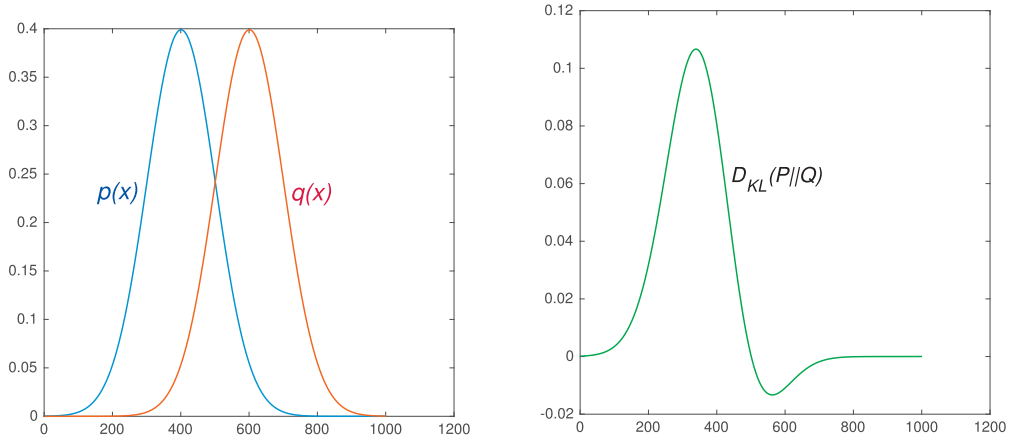
Conditional probabilities of $h_j = 1$ given v and $v_i = 1$ given h are provided in Eqs. (6) and (7), respectively:

$$p(h_j = 1 \mid v) = \sigma\left(b_j + \sum_i v_i w_{ij}\right), \tag{6}$$

where $\sigma(x)$ is the sigmoid function.

$$p(v_i = 1 \mid h) = \sigma\left(a_i + \sum_j h_j w_{ij}\right). \tag{7}$$

Having Eqs. (4), (6), and (7), it would be easy to obtain an unbiased sample of $\langle v_i h_j \rangle_{data}$ since there are no direct connections between hidden nodes and similarly between visible nodes in an RBM. On the contrary, getting an unbiased sample of $\langle v_i h_j \rangle_{model}$ is more difficult. Gibbs sampling is used by updating all of the hidden and visible nodes in parallel based on Eqs. (6) and (7) leading to a very slow convergence. To overcome this issue, the CD algorithm was proposed by Hinton in 2002 [45] which produces a reconstruction of a given visible vector by setting each visible node to one with probability $p(v_i = 1 \mid h)$ once the binary states



(a) Original Gaussian PDFs $p(x)$ and $q(x)$

(b) KL area to be integrated

Fig. 5. KL-Divergence for two normal distributions.

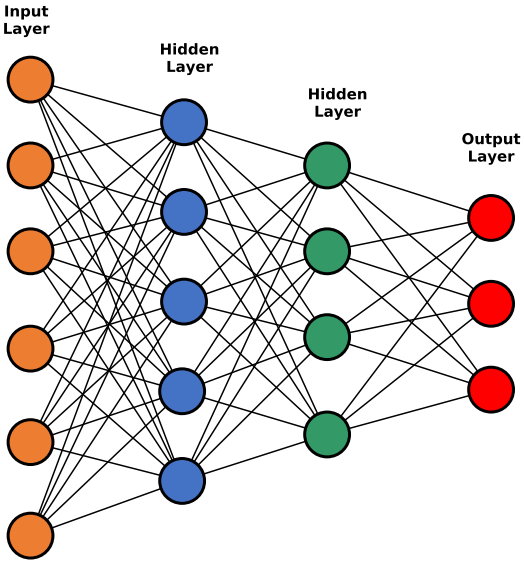


Fig. 6. An SAE having two hidden layers.

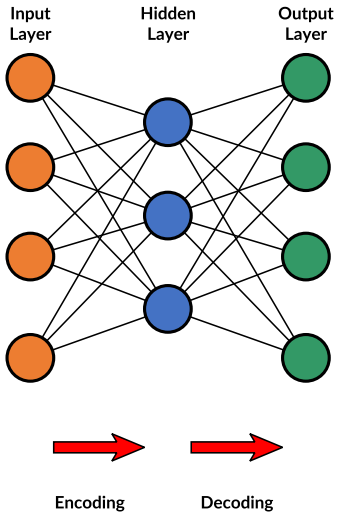


Fig. 7. The architecture of an AE.

have been computed for hidden nodes [5,42]. The weight update is given by

$$\Delta w_{i,j} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}), \tag{8}$$

where *recon* refers to the reconstruction phase.

2.5. Stacked autoencoder

A Stacked Auto-Encoder (SAE) is a type of NN consisting of multiple layers of AE in which the output value of each layer is connected to the inputs of the next layer [48–51]. Training layers of an SAE are similar to that of DBN; an unsupervised greedy pre-training one layer at a time followed by a supervised fine-tuning of the parameters of all the layers together.

Layer k of an SAE is trained as a separate AE by minimizing the error in reconstructing its input which is the output code of layer $k-1$. Once all layers are pre-trained, a classifier usually, softmax classifier, is put on top of the network and then the whole network is trained as a typical FFNN where the weights and biases of the network are fine-tuned by minimizing the classification error of a supervised task.

Fig. 6 illustrates an example of an SAE composed of two hidden layers and a final softmax classifier put on the output code of the output layer. Clearly, the input layer of the first AE is the input layer for the whole network, and the hidden layer of AE_i serves as the input layer to AE_{i+1}. The chain of encoders increasingly compresses data and extracts higher-level features representing a coded version of the input layer [52].

2.5.1. Autoencoder

An AE uses a set of recognition weights to encode an input vector x into a representation vector h, referred to as latent variables [53]. It then uses a set of generative weights to decode the representation vector into an approximate reconstruction of the input vector x' [54]. The goal of the AE is to reconstruct the input data in an unsupervised way, i.e., without using any labels while the dimensionality of the input and the output need to be the same [55].

Fig. 7 depicts the architecture of a typical AE. The encoding stage of an AE takes $x \in \mathbb{R}^m$ as input and maps it to latent variable $h \in \mathbb{R}^n$:

$$h = f(Wx + b), \tag{9}$$

where f is an activation function, such as sigmoid, $s(x) = 1/1 + e^{-x}$ or ReLU, W is a weight matrix, and b is a bias vector.

The decoding stage then maps h to x' which is a reconstruction of x with identical dimensionality.

$$x' = f'(W'h + b'), \tag{10}$$

where f',W', and b' are the corresponding parameters for the decoder that might be different from the encoder's ones.

AEs are trained to minimize reconstruction errors like Mean Squared Errors (MSE) described as follows:

$$E(x, x') = \| x - x' \|^2, \tag{11}$$

where x is usually averaged over n training samples.

2.6. Recurrent neural network

It is a pre-assumption in a traditional NN that all inputs and outputs are independent of each other. Nevertheless, this assumption is not true in many applications, specifically those that make use of sequential information, such as speech recognition tasks. Unlike a traditional NN, RNN produces outputs dependant on the previous state computed and recurrently performs the same task for every element of a sequence. In other words, RNN benefits from having a memory that stores previously-calculated information. Today, RNNs are widely used for language modeling and have shown great promise in many natural language processing tasks [56].

Suppose we have a sentence of m words. We can predict the probability of observing the sentence as follows:

$$P(w_1, ..., w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, ..., w_i - 1). \tag{12}$$

What that means is that the probability of a sentence is the product of probabilities of each word, w_i, given the words that precede it [57]. It is a generative model which is capable of sampling the next word from the predicted probabilities, given an existing sequence of words, and keeping on until the whole sentence is constructed. To train the model, we will need text to learn from. Fig. 8 shows what a typical RNN and the unfolding process look like. As depicted in the figure, by unrolling the whole network, we actually create one layer for each word of a sequence that is ordered in timestamps. For example, for a sequence of four words, the network would be unrolled into a 4-layer NN, one layer for each word.

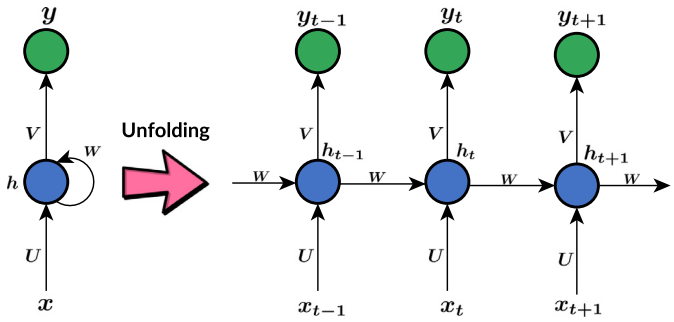


Fig. 8. An RNN and the unfolding in the time of computation.

Let x_t and h_t be the input and hidden state at timestamp t, respectively. The output y_t at timestamp t is defined as:

$$y_t = softmax(Vh_t), \tag{13}$$

where V is the weight matrix of the output layer.

h_t is assumed to be the memory of the network and is calculated based on the previous hidden state and the input at the current step: $h_t = f(Ux_t + Wh_{t-1})$. U and W are considered as weight matrices for the input layer and hidden state, respectively. The activation function f usually is a non-linearity, such as sigmoid, tanh, or ReLU [58,59].

In an RNN, the total number of parameters are decreased compared with an FFNN since all parameters are shared among all steps. So for different inputs, the same task is performed at each step.

2.6.1. Long short-term memory

As mentioned above, RNN is a famous model to train sequential data, such as video capture, word prediction, and image processing which can remember the elements of a sequence using a memory cell. One of the major problems of dealing with RNN is when it is used to train with long step size. It is difficult for RNN to remember information for a very long time because the backpropagated gradients either grow or shrink at each time step, making the training weights either explode or vanish notably. LSTM memory has addressed this issue. A typical LSTM unit is composed of three gates, an input gate, an output gate, and a forget gate, which regulate information into and out of the memory cell. The architecture of a single LSTM cell is shown in Fig. 9. σ is the logistic sigmoid function, and i, f, o, and c are the input gate, forget gate, output gate, and cell state, respectively. The input gate decides the ratio of input and has an effect on the value of the cell state. The forget gate controls the amount of information that can be remained in the memory cell, and the output gate determines the amount of information in the memory cell that can be used to compute the output activation of the LSTM unit. This architecture can solve the problems of vanishing and exploding gradients.

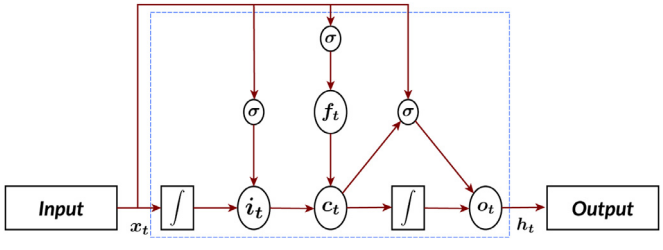


Fig. 9. LSTM Cell [60].

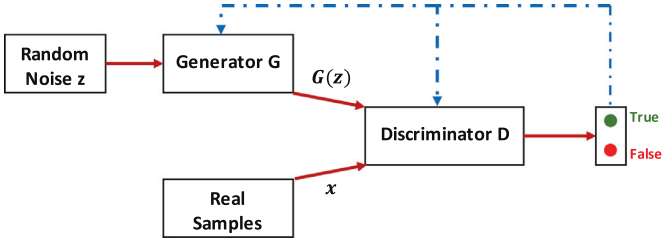


Fig. 10. Architecture of a GAN.

2.7. Generative adversarial network

Generative Adversarial Networks (GANs) are originally developed by Goodfellow [61] to estimate generative models using an adversarial process. The idea underlying the GAN is derived from the game theory in which two competing models, namely generator G and discriminator D, are pitted one against the other. The generator G learns the distribution of real data, and the discriminator D estimates the probability that the input data are coming from the real data or from the generator G. The participants of the game need to improve their methods until the data distribution is indistinguishable from the model distribution, i.e., to find a Nash equilibrium between the two participants [62].

The adversarial net can be trained using BP, since G and D are frequently represented by NNs and thus are differentiable functions. The optimization of GAN can be formulated as a minimax problem where the training procedure for G is to maximize the probability of D making a mistake [61]:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))], \tag{14}$$

where real data x are sampled from real data distribution $p_{data}(x)$, random variable z is sampled from Uniform distribution or Gaussian distribution, so-called prior distribution $p_z(z)$, and $E(.)$ is the expectation value.

Fig. 10 shows the architecture of a GAN. In the figure, discriminator D has two inputs: real data x and $G(z)$. $G(z)$ is complied with the distribution p_{data} and represents the sample generated by G. If the input of D is coming from x, it should be classified to true (real), whereas if the input of D is coming from $G(z)$, it should be classified as false (fake). Suppose we have the generated data distribution p_g, the goal of the training algorithm is to converge to a point where D and G cannot improve further because $p_g = p_{data}$, if given enough capacity and training time. It has been proved in [61] that the optimization problem defined in Eq. (14) has a global optimum for $p_g = p_{data}$, and the generative model perfectly replicates the data generating process.

3. Deep learning framework for cybersecurity applications

In this section, we present a general DL framework for cybersecurity applications based on the papers we have scrutinized in this survey. The framework is considered to be as generic as possible to cover a range of cybersecurity challenges including intrusion detection, malware detection and analysis, phishing/spam detection, and website defacement detection. The conceptual model of the proposed framework is illustrated in Fig. 11. The problem domain justifies the type of data source we select for our framework. If we are aiming at detecting or categorizing malware, depending on the infrastructure, i.e., the mobile malware or the PC malware, we would have Android application package

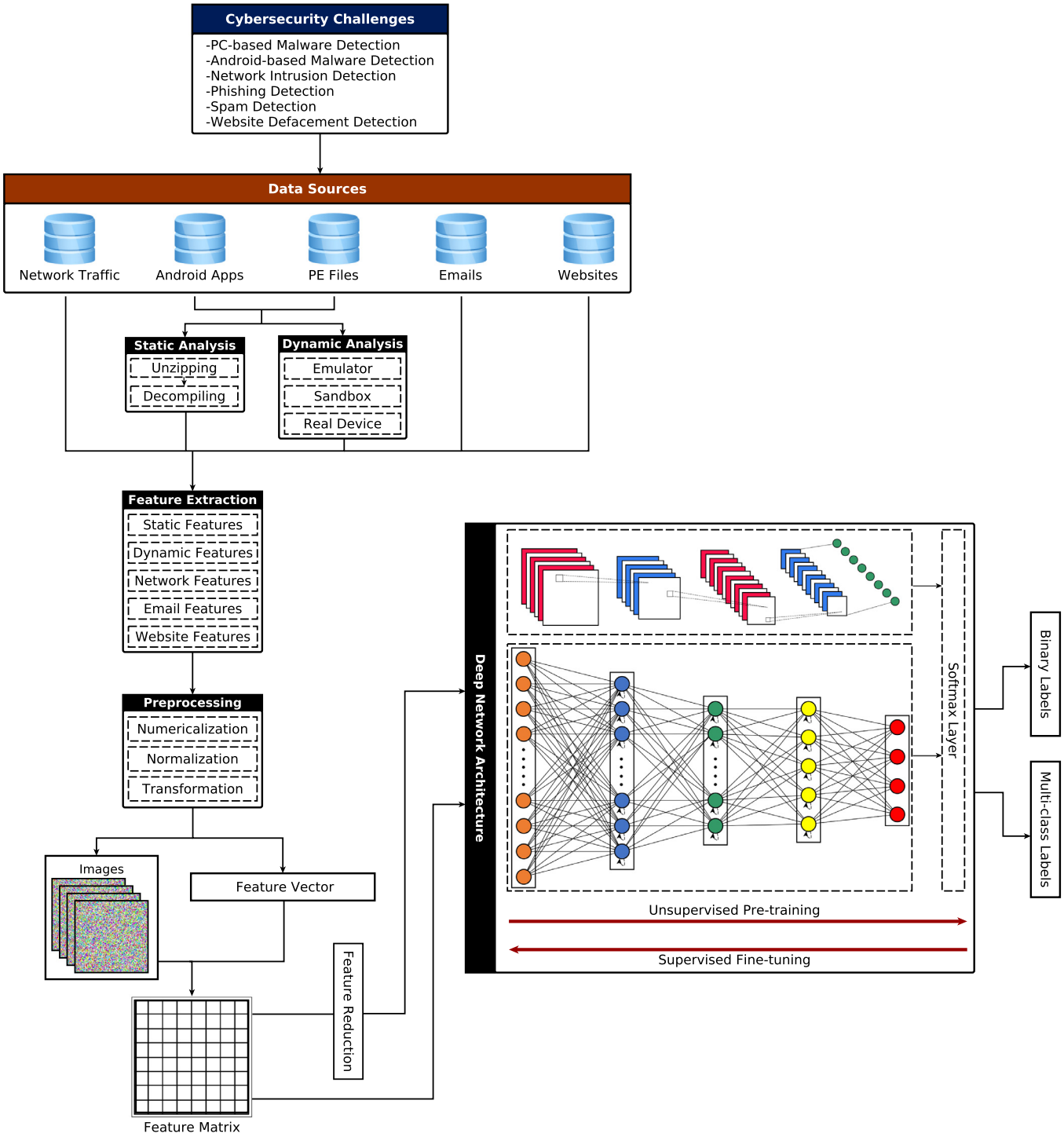


Fig. 11. Conceptual DL framework for cybersecurity applications.

i.e., .apk, or Portable Executable (PE) files as the input data. On the other hand, if we are looking for intrusion detection applications, the input data would be the network traffic records. The network traffic input data are either coming from publicly available intrusion detection datasets, such as KDD'99 [63] and NSL-KDD [64], or from real-world traffic captured in the form of pcap files. Emails are another data source for phishing email or spam detection, and websites are the input data for phishing website and website defacement detection. The general framework consists of four major modules: analysis, feature extraction, preprocessing, and DL-based classifier.

3.1. Analysis

The framework workflow starts by analyzing .apk files or PE files in a static or dynamic mode. In the static analysis phase, the input file is first decompressed and/or decompiled to extract corresponding features. If the input data are apk., they are unzipped to nearly original form including Android manifest (*.XML) and classes.dex files. The classes.dex files are disassembled to generate the Dalvik VM assembly code and the *.Xml files are parsed to extract meta-data information, such as package name, permissions, libraries to be linked, and components definitions. The PE file is

from a non-mobile environment like Windows and usually has been compressed by a binary compression tool like UPX or ASPack Shell, so first, it should be unpacked. PE file is then decompiled into its corresponding assembly code. In the dynamic analysis phase, the .apk or PE file is executed in an emulator, sandbox, or a real device, i.e., smartphone or PC while the required information like network packets, Application Programming Interface (API) call or system call traces are recorded. Interaction with the application is also required to mimic the real environment for the application, which is either conducted by users or done automatically using ADT Monkey for example.

3.2. Feature extraction

In the feature extraction module, the static features which are the output of static analysis are extracted from decoded resources. Static features include API calls, strings, URL-based features, raw opcode sequences, file-related features, permissions, intents, suspicious calls, app components, to name a few. The dynamic features are extracted from the log files or the pcap files created and captured during the execution of the applications in the underlying environment. Dynamic features include but are not limited to, API call traces, system call sequences, domain-based features, machine activity, network traffic, file creation and deletion, and registry keys written. The network feature extractor is a flow-based feature extractor that can extract network traffic features from a pcap file. Email features that capture the characteristics of email datasets could be of any type of bag-of-words features, structural features, link features, or element features. Website features are extracted from websites' data source and include address bar features, abnormal features, source code and javascript-based features, domain-based features, security and encryption features, and page style or content-based features. Another type of website feature could be the representative windows extracted from the screenshots of a website, particularly useful in detecting a website defacement.

3.3. Preprocessing

Based on the type of the produced feature, we need to numericalize the nominal values into a numeric form or normalize the numeric feature values into a desired scale, like [0, 1] range. Another task could be transforming the extracted features into RGB color images. The output of the preprocessing module is in the form of images or feature vectors which both can be translated into one or more feature matrices. Depending on the size of the feature matrix, the dimension of the input feature could be reduced to a lower dimensional subspace using random projection method or PCA, which then serves as the input to the DN architecture. However, it is encouraged not to use any dimensionality reduction technique since the DN is meant to do so inherently. A sound DL framework should learn high-level features from low-level ones layer-by-layer. Feature reduction before training DN architecture would not be reasonable and would wipe out the necessity of employing DL algorithms.

3.4. Deep learning-based classifier

In the DL-based classifier module, the DN takes the final feature matrix as input and is trained using the greedy layer-wise learning algorithm invented by Hinton [25]. The algorithm performs an unsupervised pre-training layer by layer in a forward pass followed by a supervised fine-tuning using BP in a backward pass. As depicted in the figure, the DN could be implemented by a CNN, DNN, RNN, SAE, or DBN on the basis of the type of the input feature matrix. For instance, CNN has been very effective in image classification and RNN is well-adapted to processing sequences of inputs. Since we are dealing with the classification problems, the output layer is usually implemented with the softmax function that is used to output the categorical probability distribution of either binary classification or multi-classification tasks.

3.5. Deep learning merit

Several ML methods or DM techniques have addressed network attacks or malware infiltration on personal computers or mobile devices [65–70]. ML is a branch of data analysis that aims at constructing patterns from underlying data and minimizing the intervention of a human agent as much as possible. However, the idealistic image of this AI-driven field has never been fulfilled due to some deficiencies in shallow learning algorithms, such as Support Vector Machine (SVM), Naive Bayes (NB), and Decision Tree (DT). Particularly, in solving real-world applications with intrinsic complexity and massive amounts of data, the limited modeling and representational power of shallow architectures can cause problems [71]. They are usually unable to generalize and scale to large, unstructured, and heterogeneous data [72]. In addition, experts need to extract meaningful and robust features manually from data, a process that is so computationally expensive and error-prone. In contrast, DN architectures learn representations of input data with multiple levels of abstractions, and a set of high-level and flexible features are obtained as the output of complex layer-wise processing. In the higher layers of a DN that is specifically designed for classification, the feature representation amplifies the important aspects of the input data for discrimination and omits the irrelevant details. Thus, when a new network attack emerges or a zero-day malware appears, there will be no need to extract the features from scratch. A myriad of non-linear layers will take care of engineering the features automatically that help generalization in the classification.

4. Deep learning-based cybersecurity related work

Recently, several techniques have been proposed by researchers that have applied DL algorithms to detect or categorize malware, detect network intrusions and phishing/spam attacks, and inspect website defacements. In this section, we review these studies in three main groups: malware detection and analysis; intrusion detection; and other, which includes phishing detection, spam detection, and website defacement detection. Fig. 12 summarizes the main branches of applying DL to cybersecurity.

4.1. Malware detection and analysis

Generally speaking, malware detection techniques are classified into three groups: (a) static, (b) dynamic, and (c) hybrid [3]. Static approaches disassemble and analyze the source code without executing it. Although they are quick, they suffer from producing high false positive rates. In addition, they fail against detecting obfuscated malware. Monitoring the interactions of the executed code in a virtual environment, dynamic analysis techniques address malware obfuscation while consuming lots of time and memory resources, whereas hybrid methods employ the advantages of both static and dynamic approaches.

In spite of the fact that many traditional ML algorithms, such as SVM, Bayesian Networks, Logistic Regression (LR), and MLP have already been applied to malware detection and categorization, they suffer from having shallow architectures that make them not scale well to large datasets. Furthermore, their feature extraction phase is not automatic. Hence, DL models may open a promising way to overcome these limitations.

Recently, several studies in the literature have concentrated on DL algorithms in malware detection and analysis. These techniques

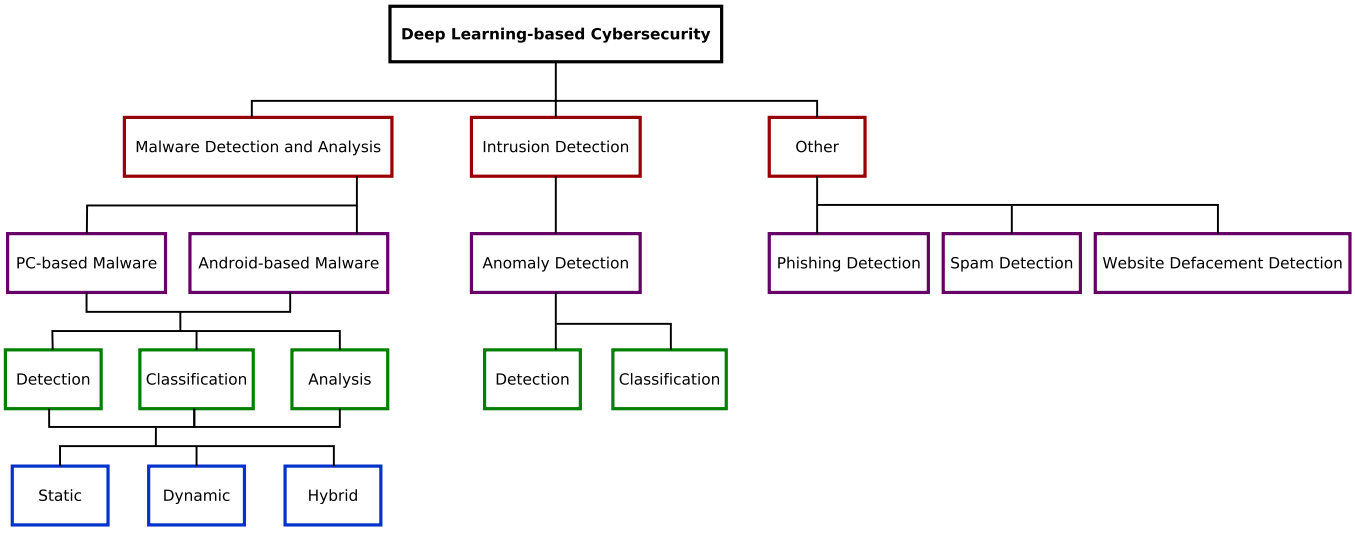


Fig. 12. Main branches of applying DL techniques to cybersecurity.

normally fall into two main categories: Android-based or PC-based malware.

4.1.1. PC-based malware

Malware is a malicious software spread to infiltrate the security, integrity, and functionality of a system. Different malware types include viruses, worms, trojans, backdoors, spyware, botnets, and so on. With the rising prevalence of Internet users, malware poses a serious threat to the security of computer systems. It is estimated that one in four computers operating in the U.S. is infected with malware. A recent report from Kaspersky Lab shows that up to one billion dollars were stolen in two years from financial institutions worldwide, due to malware attacks [73]. Therefore, malware detection has attracted much attention of both the anti-malware industry and researchers today. Most anti-malware software products use signature-based detection methods. A signature is a short string of bytes, which is unique for each known malware. Although these methods are to some degree promising in detecting malware, their main disadvantage lies in being easily circumvented by malware attackers through code obfuscation, polymorphism, and encryption. As a result, many researchers have focused on intelligent malware detection by applying DM and ML techniques, such as NN, SVM, NB, and DT. These classifiers have shallow learning architectures and are still inefficient for malware detection problems. DL is a new trend in the ML area. A multilayer DL architecture is used in feature engineering and can achieve better performance compared to traditional ML algorithms. As discussed in Section 2, typical DL models include SAE, DBN with RBM, and CNN.

One of the first attempts to apply DL methods in malware detection was presented by Dahl et al. [74]. They propose a large-scale malware classification system that applies random projection on 2.6 million examples to reduce the input space by a factor of 45. They use over 179 thousand sparse, binary features based on file strings, API tri-grams, and API calls plus their parameter values. NNs trained on random projections provide a 43% reduction in the error rate compared to the baseline LR system using all the features. They further conclude that pre-training which is a widely-used method for DL produces worse results than standard NN topologies trained with BP. In addition, the number of random projections, hidden layers, and hidden units are investigated to achieve good efficiency and the performance of LR and NNs for the task of multi-class malware classification purposes are compared. Although their shallow NN model is one of the best in terms of binary and family classification accuracy, the DNN architecture fails to improve the classification accuracy. They are not able to obtain a significant accuracy gain by adding additional hidden layers.

About one year later in 2014, a two-layer framework based on RBM and One Side Perceptron (OSP) was suggested by Benchea et al. [75] to create a new set of features out of 3,299 initial miscellaneous ones (mostly related to network activity and file alterations) and then classify the samples into benign or malicious. They focus on detection rate increment and zero false positive rate in the training phase that results in a low false positive rate in the testing dataset. To create a practical model, they conduct a filtering process on an initial dataset with 3,087,200 files and 3,299 features. The final dataset comprises 31,507 malicious files and 1,229,036 benign files with 300 features for each record extracted using the best-selected features according to F_2 score algorithm. They also compare their framework OSP-RBM with OSP tested on the dataset with 300 features and OSP-MAP tested on the dataset with 300 features created by combining the initial features with one another, so-called mapping. Although their two-layer framework depicts a reasonable trade-off between low false positive rate and good detection rate (0.00024%–88.83%) compared with OSP and OSP-MAP, they have conducted a non-acceptable redundant task in feature selection using F_2 score algorithm that contradicts with the philosophy of using DL architectures, i.e., putting away the tiresome work of feature engineering and coming up with an intelligent system that automatically elicits solid features from a collection of features along with the learning process.

Since traditional ML algorithms utilize hand-crafted features, they would be unsuccessful in identifying obfuscated or repackaged malware. Learning the language of malware through the executed instructions could help us to extract resilient features. To achieve this goal, Pascanu et al. [76] proposed a method based on the Echo State Network (ESN) and RNN to classify malware samples. These models are trained in an unsupervised fashion to predict the next API call and use the hidden state of the model containing the history of past events as the fixed-length feature vector. This feature vector is then given to a separate classifier, LR or MLP, to classify each malware. To find the best hybrid model, they compare different combinations of models for projection and classification stage from which ESN for the recurrent model, MP for non-linear sampling, and LR for the final classification were demonstrated to be the best. Using a dataset containing equal numbers of malware and benign files (500,000), they have achieved 98.3% true positive rate and 0.1% false positive rate

compared to the standard tri-gram of events model. In spite of the fact that the authors compare different hybrid models based on ROC curves to conclude the best performing one, they do not indicate the time complexity of the proposed models.

Later, *DeepSign* was proposed by David et al. [77] to automatically generate malware signatures that do not rely on any specific aspect of the malware. The model uses Stacked Denoising AE (SDAE) which creates an invariant compact representation of the general behavior of the malware. In Denoising AE (DAE), we add some noise to the input vector by setting a randomized subset to zeros and then reconstruct the uncorrupted input data using criteria like KL distance between the original inputs and the reconstructed inputs. Uncorrupted encoded representations are then used as the inputs to the next level of the SDAE [71]. They first run malware in a Cuckoo sandbox and then convert the sandbox log file to a 20,000-bit binary string. The bit-string then is fed into a deep 8-layered SDAE which produces 30 values as the program signature in the output layer. These signatures then can be used in an unsupervised framework or supervised malware classification. They reported an accuracy of 98.6% using a supervised DNN trained on the acquired weights of the SDAE. Nevertheless, they have not reported any other measure, such as precision, recall, false positive rate, and/or false negative rate to give us a true sense of valuation. Furthermore, the malware variant dataset of six major families includes just 1,800 samples which are not cross-validated, i.e., 1,200 training samples, and 600 test samples. To top it all, they have not included any benign samples that would result in a high false positive rate.

In October 2015, Saxe et al. [78] proposed a DNN based malware detector that employs two-dimensional binary program features to detect malware. It consists of three main components. In the first component, four different static feature types, namely binary values of a two-dimensional byte entropy histogram, PE import features, PE metadata features, and contextual byte features, are concatenated to form a final feature vector. The second component includes a DNN which consists of one input layer containing 1,024 input features, two hidden layers which are Parametric ReLU (PReLU) units, and one output layer that is a sigmoid unit. The NN is trained using BP and the Adam gradient-based optimizer. The final component is a Bayesian calibration model that gives a score to each output of the DNN. In other words, the output score is interpreted as approximating the probability that the input file is actually a malware. The dataset-contains 431,926 binaries sourced directly from the customers and internal malware databases, with 81,910 labeled as benign and 350,016 as malware. They have achieved a detection rate of 95% and a false positive rate of 0.1%. The strength of their work lies in the large amount of data and feature fusion of various kinds.

At the beginning of 2016, Hardy et al. [73] presented a DL architecture using the SAE model for malware detection. The input is based on API calls extracted from the PE files. The SAE model employs a greedy layer-wise training operation for unsupervised feature learning followed by the supervised parameter fine-tuning, i.e., weights and offset vectors. The system consists of two main components: feature extractor and DL-based classifier. The feature extractor contains a decompressor and PE parser. If a PE file is previously compressed by a binary compression tool, a decompressor should be used. Then the PE parser is applied to extract the Windows API calls from each PE file. Finally, API calls are stored as the signatures of the PE files in the signature database. The DL-based classifier first performs unsupervised feature learning based on SAE and then supervised fine-tuning to classify malware into malicious or benign. The goal of training is to minimize the error between the input data and the reconstruction value. A classifier is then added on the top layer, and a BP algorithm is applied in a top-down manner to fine-tune the weights and offset parameters. The dataset contains 50,000 file samples obtained from the Comodo Cloud Security Center from which 22,500 files are malware, 22,500 files are benign, and 5,000 files are unknown. The experimental results show that the DL model with three hidden layers and 100 neurons at each layer outperforms other settings in both training and testing accuracy. The unusual point about their layer tuning is that the number of neurons in all layers is assumed to be exactly the same.

In 2016, Huang et al. [31] proposed a multi-task DL model, called, *MtNet* for binary and 100-class malware family classification. The model is trained using data extracted from the dynamic analysis of malicious and benign files with 50,000 selected features containing null-terminated tokens, API events plus parameter value, and API tri-grams as the input vector. The *MtNet* structure consists of a DNN with ReLU as the activation function and dropout for hidden layers. The dataset they train and test their framework on consists of 6.5 million files provided by analysts from Microsoft Corporation, from which 2.85 million samples are extracted from malicious files and 3.65 million samples from benign files. The framework achieves a binary malware error rate of 0.358% and family error rate of 2.94%. *MtNet* offers a multi-task learning architecture that improves the classification results for a low false positive rate under 0.07%. Nevertheless, adding layers to the DL model increases the test error rate, especially for the family classification task, and thwarts the application of DL.

Ding et al. [79] proposed an opcode based malware detection system using DBN and evaluated the performance of DBN for malware detection by comparing that with SVM, DT and k-Nearest Neighbors (k-NN). Each executable is represented by opcode $n-$grams. They also use DBN as a feature reduction method and compare its effectiveness against existing feature selection methods. Furthermore, they study the effect of unlabeled data for DBN pre-training that in practice do not show noticeable improvement. The algorithm's deficiencies lie in not optimizing different DL parameters, like the number of layers and neurons in each layer. They have just reported accuracy and false positive, and no error rate has not been calculated. The dataset is not big enough, 2,000 benign and 2,000 malicious would result in an overfitting problem.

Later, two papers were published exactly at the same time in December 2016. The first one proposed by Kolosnjaji et al. [80] employs malware system call traces to classify newly retrieved malware samples into a set of malware families using DNNs. They combine convolutional and recurrent approaches for modeling system call sequences and therefore optimizing malware classification. CNN uses sequences in the form of n-grams to capture the sequential position of system calls. On the other hand, recurrent networks train a stateful model by using dependency between current and previous system calls in a sequence. In their framework, they use a malware zoo where the input data is acquired by executing malware in a protected environment. The results are then preprocessed and numerical feature vectors are retrieved to be fed into NN which in turn classify the malware into one of the predefined sets of malware families. Using this combined NN architecture, they got an average 85.6% on precision and 89.4% on recall. It would be difficult to evaluate their approach since they have not mentioned how big their dataset is.

The second paper [81] pursues a different approach and concentrates mostly on efficient dynamic malware analysis rather than malware detection or classification. It is not easy for attackers to evade network-behavior-based methods unlike static-feature-based and host-behavior-based due to the necessity of communication between infected hosts and attackers. The effectiveness of such methods depends on how we collect a variety of communications

by using malware samples. Nevertheless, analyzing all new malware samples for a long period is impossible. Therefore, the authors proposed a method for determining whether dynamic analysis should be suspended based on network behavior. Focusing on two characteristics of malware communication, the change in the communication purpose and the common latent function, they concentrate on the similarity of data structure between malware communications and natural language. To achieve this, they apply the Recursive Neural Network (ReNN) to their proposed method, which has recently demonstrated high classification performance for sentences. ReNN is a type of DNN capable of learning deep structured information that applies the same set of weights recursively over a structured input [82]. This tree-structured NN is mainly used in natural language processing for sentence parsing [83] or sentiment analysis [84]. In the experimental evaluation with 29,562 malware samples, their proposed method reduced analysis time by 67.1% while keeping the coverage of collected URLs to 97.9% of the method that continues full analysis. However, the proposed approach suffers from some limitations, such as malware long periods of sleep and labeling.

In 2017, Yousefi-Azar et al. [85] proposed a generative feature learning-based approach for malware classification and network-based anomaly detection using AE. Not using any pre-processing step, the framework uses a 10-sized vector of hex-based features from PE files and generates a small set of latent features capturing the semantic similarities between the feature vectors. They evaluate their proposed method on Microsoft Kaggle malware binary dataset of nine families and NSL-KDD using well-known classifiers, such as NB, *K*-NN, SVM, and Xgboost. The experiments show that in both tasks, malware detection and intrusion detection, the accuracy of classifiers are improved after using AE-based generated features compared to the original ones. Nevertheless, they have not reported any false positive, false negative, or ROC measures.

Domain-Generation Algorithms (DGAs) are used by many malware families, mostly botnets, to avoid takeover and blacklisting attempts. Using DGAs, malware generates a large list of possible domain names to serve them as connecting points with C&C servers for receiving required commands. To detect domain names generated by DGAs, Lison et al. [86] proposed a data-driven approach based on RNNs that relies on a large training set of domains generated by a total of 61 malware families. To be more precise, they take a domain name as input and produce the probability that the domain is generated by a DGA as output with F_1 score 0.971.

Executions of some malware variants might lead to costly aftermath that justifies the importance of malware detection prior to execution of malicious payload. Using sequential dynamic data, Rhode et al. [87] applied an ensemble of RNN to predict malware executable within the first four seconds of its execution process. For feature capturing, they monitor machine activity from Cuckoo Sandbox over 594 malicious and 594 trusted PE files provided by VirusTotal which results in 11 final features. The average sample recording time is 4.8 minutes which is reduced by 98% to four seconds with an accuracy of 93%. However, their approach suffers from some limitations, such as a need to test with a larger dataset and reduce hyperparameter search time. Most importantly, truncating the analysis time to four seconds might make the approach vulnerable against malware having long sleep or benign behavior at the beginning of execution.

To compare the effectiveness of low-level and high-level features on the performance of DL models, De Paola et al. [88] employed a DL framework based on SDAE for malware detection. High-level features consist of PE import histogram accessed from import address table header, PE metadata histogram that depends on the labels and values of numerical fields, byte/entropy histogram and the string histogram. Low-level features are the ones extracted through static analysis from DOS header, file header, optional header and section table of the PE packaging. The proposed DL model consists of three hidden layers of 256, 64, and 16 nodes, respectively, an input layer of 636 nodes, and a single-node output layer. In the first phase, each hidden layer is pre-trained individually using a DAE consisting of an input layer, a corrupted layer, an encoder layer, and a decoder layer. After layer-by-layer unsupervised pre-training, there would be a supervised fine-tuning implemented by means of Adam stochastic optimization as the BP algorithm. Regardless of the classifier used, it is shown that using optimal high-level features results in better classification performance. Also, when using high-level features, DNNs do not significantly improve the overall detection accuracy. Furthermore, it is proved that two-phase training outperforms one-phase fine-tuning only and increasing the number of epochs improves neither accuracy nor loss significantly.

Ye et al. [89] proposed a heterogeneous DL framework for malware detection based on an AE stacked up with multilayer RBMs and a layer of associative memory. The input of the deep model consists of Windows API calls extracted from the PE files. It performs unsupervised pre-training layer by layer and then supervised fine-tuning as regards the labeled data existing in the associative memory in the top of the DL framework. The novelty of their approach lies in using both labeled and unlabeled data in the phase of training and feature learning. They compare their heterogeneous DN with other homogeneous ones using the Comodo Cloud Security Center data containing 4,500 malware files, 4,500 benign files, and 10,000 newly collected unlabeled files, and 1,000 testing samples. They achieve an accuracy of 98.82% and false positives of 53 out of the whole labeled and unlabeled training samples, which are slightly higher accuracy (approximately 0.01) and lower false positives (about 60) compared with SAEs and DBN.

One of the recent attempts in malware classification was made by Kim et al. [90] who applied Convolutional Gated Recurrent Neural Network (CGRNN) for malware family classification. The deep model architecture consists of four layers, i.e. a CNN layer, a Gated Recurrent Unit (GRU) layer, a DNN layer, and a sigmoid layer. Using the Microsoft malware Classification Challenge through the Kaggle platform in 2015 of nine different malware families, they achieve 92.6% overall accuracy. But they have not reported either false positive or per-family classification performance. Furthermore, the parameters of each layer are not fine-tuned. The unexpected point is that the sigmoid layer on top of the layers consists of a single node that is not a correct setting for a 9-family classification task.

Two recent studies in the area of malware detection based on the state-of-the-art GAN are [91,92]. In [91] Kim et al. proposed transferred Deep Convolutional Generative Adversarial Network (tDCGAN) to generate fake malware and learn to distinguish it from the real one that helps detect zero-day malware robustly. Since GAN suffers from instability in the training process, a deep AE is used to learn malware features and stabilize the GAN training process. The architecture of the malware detection system consists of three components, namely data compression and reconstruction, generation of fake malware data, and malware detection. The data compression and reconstruction component takes the preprocessed data as the input. In the second component, a DAE is used to learn a representation of data which is then transferred into the generator of the GAN. Given a known probability distribution, the generator produces fake malware data and hands over the generated data to the discriminator to learn the characteristics of malware data. The discriminator of the GAN is then transferred to the malware detection component, and the system is trained to detect malware data. The malware dataset is from the Kaggle Microsoft Malware Classification Challenge in the form of binary code that has been converted into reduced size malware images. The tDCGAN-based malware detector achieves an average classification accuracy

Table 1
Selected studies focusing on PC-based malware detection and analysis.

Year	Authors	Focus Area	Deep Model	Features	Dataset	Performance
2013	Dahl et al. [74]	Static malware detection	Random projection with DNN	Binary features based on file strings, API tri-grams, and API calls plus their parameter values	2.6 million files, 1,843,359 of which are malicious and 817,485 are benign	ER=0.49% FPR=0.83% FNR=0.35%
2014	Benchea et al. [75]	Static malware detection	RBMs for feature selection and OSP for classification	Network activity-related features: Windows registry alteration, and file deletion. File-related features: file size, file entropy, number of sections, and file entry point position	31,507 malicious files and 1,229,036 benign files with 300 features for each record	ACC=99.72% DR=88.83% FPR=0.00024%
2015	Pascanu et. al [76]	Static malware detection	ESN and RNN	API calls	250,000 malware samples and 250,000 benign samples were collected from Microsoft anti-malware file collection	TPR=71.71% FPR=0.1%
2015	David et al. [77]	Dynamic malware family classification	SDAE with dropout	Bit strings created from sandbox log: native functions and Windows API call traces, details of files created & deleted, IP addresses, URLs and ports accessed by the program, and registry keys written	C4 Security dataset of six major malware categories containing 1800 samples	ACC= 98.6%
2015	Saxe et. al [78]	Static malware detection	DNN using PReLU and sigmoid as activation functions and dropout	Contextual byte, string 2d histogram, PE import, and PE metadata features	431,926 binaries with 81,910 labeled as benign and 350,016 labeled as malware	DR=95% FPR=0.1%
2016	Hardy et. al [73]	Static malware detection	SAE	API calls	50,000 samples from Comodo Cloud Security Center from which 22,500 samples are malware, 22,500 samples are benign, and 5,000 samples are unknown	ACC=95.64% FPR=4.56% FNR=4.16%
2016	Huang et al. [31]	Dynamic binary and 100-class malware classification	DNN using ReLU activation function and dropout for hidden layers	50,000 dynamic features of null-terminated tokens, API events plus parameter value, and API tri-grams	2.85 million samples were extracted from malicious files and 3.65 million samples from benign files by analysts from Microsoft	ER=0.36/2.94%
2016	Ding et al. [79]	Static malware detection	DBN composed of stacked RBM	Opcode n−gram	2,000 benign samples and 2,000 malicious samples	ACC=96.7%
2016	Kolosnjaji et al. [80]	Dynamic malware family classification	Combination of CNN and RNN	System call sequences	Samples from VirusShare, Maltrieve, and private collections	ACC=89.4% PR=85.6% RC=89.4%
2016	Shibahara, et al. [81]	Dynamic malware analysis	ReNN	5 domain-based features: no. of IP addresses corresponding to the domain, no. of countries corresponding to the resolved IP addresses, and no. of domains that share the IP addresses. 8 content-based features: application protocols, status code, file type, no. of query parameters, and depth of path	29,562 malware samples	PR=97.6% RC=96.2% F_1=96.9%
2017	Yousefi-Azar et al. [85]	Static malware family classification and anomaly-based intrusion detection	AE	Uni-gram of Hex dump files for malware detection and 41 features of NSL-KDD for intrusion detection	1) Microsoft Kaggle malware dataset including 10,868 labeled malware binary files from 9 different malware families 2) NSL-KDD includes 125,973 train and 22,544 test records	ACC=83.34%
2017	Lison et al. [86]	Static malware detection	RNN	Raw domain names	Over 4 million benign domains collected from Alexa, Statvoo, and Cisco and over 49 million malware domains spread over 63 distinct types of malware obtained from DGArchive	ACC=97.3% PR=97.2% RC=97.0% F_1=97.1%

(*continued on next page*)

Table 1 (*continued*)

Year	Authors	Focus Area	Deep Model	Features	Dataset	Performance
2018	Rhode et al. [87]	Early stage malware prediction	RNN	Machine activity-related features including total number of processes being executed, the maximum number of processes being carried out, user and system CPU use, and memory use	594 malicious and 594 trusted PE files provided by VirusTotal	ACC=93%
2018	De Paola et al. [88]	Static malware detection	SDAE	High-level features: PE import and metadata histogram, byte/entropy and the string histogram. Low-level features extracted through static analysis from DOS header, file header, optional header and section table	12,000 samples of malware obtained from VirusShare and 11,874 samples of certified software obtained from a clean Windows 10 installation	ACC=97.49% PR=97.92% TPR=97.08% FPR=2.09% AUC=97.49%
2018	Ye et al. [89]	Static malware detection	Heterogeneous DL based on AE stacked up with RBMs and Associative Memory	API calls	Comodo Cloud Security Center containing 4,500 malware samples, 4,500 benign samples, and 10,000 newly collected unlabeled samples, and 1,000 testing samples	ACC=98.2% FPR=1.4% FNR=2.2%
2018	Kim et al. [90]	Static malware family classification	CGRNN	X86 instructions extracted from malware binary files encoded as one-hot-vector	Microsoft Kaggle malware dataset consists of 9 different malware families of 21,741 samples	ACC=92.66%
2018	Kim et al. [91]	Static malware detection	tDCGAN	Binary code	Microsoft Kaggle malware dataset of 9 different malware families	ACC=95.74% PR=94.4% RC=91.5% F_1=92.4%
2018	Kim et al. [92]	Static malware detection	LSC-GAN	Binary code	Microsoft Kaggle malware dataset of 9 different malware families	ACC=96.97% PR=90.4% RC=89.3% F_1=89.4%

of 95.74%, about 1% higher than that of CNN. In a similar work [92], Latent Semantic Controlling Generative Adversarial Network (LSC-GAN) is used to detect obfuscated malware in which features are first extracted using Variational AE (VAE) and are then transferred to a generator to generate virtual data from Gaussian distribution. The encoder projects the learned features back to a specific latent space which is transferred to the detector to be trained for malware detection. LSC-GAN achieves an average accuracy of 96.97%, and precision, recall, and F_1 score of, 90.4%, 89.3%, and 89.4%, respectively for each malware family.

Table 1 organizes selected studies focusing on PC-based malware detection and analysis using DL algorithms based on the model, the features, the dataset they have used, and the performance measures. For the performance measures, we have reported the best-performing DL model in the paper. For the two task-classification models, we have reported performance measures for both tasks, if they exist, indicated by "/". The acronyms for performance measures are shown in Table 2.

4.1.2. Android-based malware

Recently, Android OS has gained much popularity, leaving its competitors, such as iOS, and Blackberry far behind. The widespread use of Internet connections and the availability of personal information, e.g. messages, contacts, and banking credentials have attracted the attention of many malware developers towards the Android platform. There are many Android malware applications, such as trojans, backdoors, worms, botnets, adware, and ransomware that employ various techniques to penetrate the users' systems. Repackaging popular applications, drive-by download, dynamic payload, and obfuscation are some examples of plentiful methods leveraged by malware authors nowadays.

Filtering out these malicious Android apps is thus highly demanded by app markets. Many researchers so far have focused on statistical methods and ML techniques that can deal with the volume of malware attacks on mobile devices [65,93,94]. However, extracting salient features as inputs for ML algorithms has always been a big challenge for data scientists. Unlike classical ML archi-

Table 2
Glossary of acronyms used for performance measures.

Acronyms	Meaning	Description
ER	Error rate	Misclassification rate of a classifier
FPR	False positive rate	Ratio between the number of false positives and the total number of actual negative samples
FNR	False negative rate	Ratio between the number of false negatives and the total number of actual positive samples
ACC	Accuracy	Ratio between the number of correct predictions made and total number of predictions made
TPR	True positive rate	Ratio between the number of true positives and the total number of actual positive samples
DR	Detection rate	It is equivalent to TPR
PR	Precision	Ratio between the number of true positives and the number of all positive results returned by the classifier
RC	Recall	It is equivalent to TPR
F_1	F_1 score	It is the weighted average of the precision and recall
AUC	Area under curve	It measures the entire two-dimensional area underneath the entire ROC curve

tectures, DL models can learn features automatically without interference of any human agent; no wonder they have been employed widely in Android malware detection, recently.

Droid-Sec was one of the first studies done towards the application of DL to Android malware detection [95]. Yuan et al. devised *Droid-Sec* in 2014 (later *DroidDetector* [96]) that utilizes more than 200 features extracted from both the static and dynamic analysis of Android apps for malware detection. These features are categorized into three groups: required permission, sensitive API, and dynamic behavior. The algorithm then applies a DL model to classify the malware as benign or malicious. The proposed framework consists of two phases, the unsupervised pre-training phase, and the supervised BP phase. In the pre-training phase, DBN is used which is hierarchically built by stacking various RBMs. In the BP phase, the initial parameters of the pre-trained NN are fine-tuned. The comparison demonstrates that the DL framework achieves 96.76% accuracy and outperforms other existing ML methods. They have used 1,760 malicious apps and 20,000 benign apps for evaluations which is not an appropriate ratio. Furthermore, a more semantic-based and fine-grained set of features and richer discrete features rather than binary features might be used for better malware detection. Besides, using the same number of neurons for all hidden layers does not seem reasonable.

Having a different perspective for applying API calls in malware detection, Hou et al. [97] proposed *DroidDelver*, a method to deal with Android malware threats. They categorize the API calls of the Smali code which belong to the same method into a block. They then apply a DBN for newly unknown Android malware detection. First, the .apk files are unzipped and the dex files are decompiled into Smali codes. Second, the API call extractor component extracts the API calls from the Smali codes representing the static execution sequence of the corresponding API calls. The extracted API calls each having a unique global integer ID are then categorized into a block in the case they belong to the same method. Though it may be common to use some API calls of the same methods individually in benign applications, some groups of API calls rarely appear together in benign apps. Thus, a block of these sensitive API calls can be used as a feature to detect an Android malware. After that, the constructed feature vector of API call blocks is fed into a DBN that is built on a stack of binary RBMs which in turn is used for model construction. Finally, the classification model is used to label the Android app as benign or malicious. Comprehensive experimental evaluations are done on a real sample collection from Comodo Cloud Security Center. They perform two sets of experiments. In the first set of experiments, they evaluate the detection performance of their proposed feature extraction method. In the second set of experiments, they evaluate the detection performance of DBNs considering different parameter settings and also compare them with well-known shallow learning models. The results show that *DroidDelver* outperforms other alternative Android malware detection techniques having an accuracy of 96.66%.

A DL-based model for Android malware detection, called *DroidDeep* was presented by Su et al. [98]. They first extract totally 32,247 features from each Android app based on the five static feature types: requested permission, used permission, sensitive API call, action, and application component. Then they enter these extracted features into a DBN-based DL model, i.e., stacked RBMs, to learn typical features for classification. Finally, they put the learned features into an SVM classifier for detecting Android malware. The most efficient deep models contain three layers: each has 5,000, 5,000, 1,000 and 6,000, 6,000, 1,000, neurons, respectively. For the experimental evaluations, they use 3,986 benign apps and 3,986 malware samples. *DroidDeep* achieves detection accuracy of 99.4% and has a reasonable running time that can be adaptable to large-scale Android malware detection.

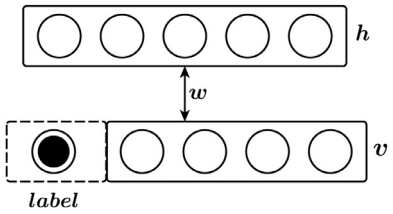


Fig. 13. RBM used for classification purposes [100].

Hybrid analysis method for malware detection is gaining more popularity, thanks to its superiority over static and dynamic analysis. To this end, Xu et al. [99] proposed a hybrid analysis method for detection of Android malware. At first, 10 static and dynamic feature sets are extracted from malicious and benign Android applications. The static features are requested permissions, permission request APIs, used permissions, advertising networks, intent filters, suspicious calls, network APIs, providers, instruction sequences, and system call sequences while the dynamic features are just the system call sequences. The static features are then converted into vector-based representations using *n*-gram and the dynamic features are converted into vector-based and graph-based representations, called *n*-gram graph. In total, they generate 16 feature vector sets and four graph sets. Afterward, the features extracted from the proposed DNN are merged with the original features to form the new DNN vector sets to increase the classification accuracy. Multiple kernels and graph kernels are then applied to each DNN feature vector and graph feature set. The learning result, i.e., the similarity output from each vector kernel or graph kernel is then combined using a two-level Multiple Kernel Learning (MKL) and fed into SVM. The experimental evaluations on the collected dataset from Google Play and VirusShare containing 4,002 benign samples and 1,886 malicious samples demonstrate that in the static analysis, the best classification accuracy is achieved by 4-gram instruction feature vectors and in the dynamic analysis, the best accuracy is achieved using 4-gram system call graphs. Moreover, using hierarchical MKL results in the best classification accuracy among all the other models. Although they have applied a hybrid model for malware detection based on static and dynamic features, they do not justify why they have set the layers of stacked RBMs to four.

With the aim of making RBM work as a classifier, Costa et al. [100] addressed the Android malware detection problem and proposed Bernoulli RBMs to extract features from Android malware-driven data. They employ meta-heuristic techniques to fine-tune RBMs and identify malware data automatically. RBMs are essentially designed for generative learning. However, recently, some papers have tried to change them into discriminative models. To make RBM work as a classifier, the authors proposed to add one more input unit with the corresponding label of the sample '1' or '0', indicating whether the input data is malicious or benign as shown in Fig. 13. Since the model does not know that we are using the label as an input, the same formulation of generative RBMs is realized here. In other words, after learning the weights over a training set, the test data can be classified by just computing the reconstruction of each input and taking the value of the first visible unit of the reconstructed data as the predicted label. Three approaches based on Harmony Search technique, namely Naive Harmony Search, Improved Harmony Search, and Parameter Setting-Free Harmony Search, are employed to fine-tune four learning parameters: number of hidden neurons, learning rate, momentum, and weight decay. They use manually designed DroidWare dataset with 278 benign application samples and 121 malware samples from Android OS (Virustotal). To assess the effectiveness of RBMs, they employ three classifiers, SVM,

Optimum-Path Forest (OPF), and a simple RBM version for classification at the end of the pipeline. The experimental results show that the discriminative ability of SVM and OPF are increased by the rise in the number of the hidden units. In contrast, for RBM as a classification approach, the smaller number of hidden units results in better classification accuracy. Nonetheless, the dataset is so small that the evaluation results are not trustworthy. Another Android malware detection approach, *Deep4MalDroid* [101], introduces a dynamic analysis method called Component Traversal that can automatically execute the entire Android app code routines in which all the runnable codes may have an opportunity to be executed and the corresponding system call list can be generated. After extracting the Linux kernel system calls of each application, a weighted directed graph is constructed whose nodes represent system calls and their sizes indicate the system calls' frequency, while a directed edge indicates the sequential flow of system calls made. The authors then explore a DL architecture with the SAE model based on the constructed call graph to learn typical patterns of Android malware and to detect newly unknown one. They finally conduct an experimental study on a real sample collection from Comodo Cloud Security Center to compare their DL model with other typical shallow learning classification models, such as SVM, NN, NB, and DT. The results show that DL improves the detection performance of other approaches by at least 5% with the best setting of three layers each with 300 neurons. However, the proposed DL model does not seem to be solid enough because the fine-tuned layers all comprise the same number of neurons.

In 2017, CNN was first applied to Android malware detection context by McLaughlin et al. [102]. With the purpose of removing the need to count the vast number of distinct *n*-grams and design hand-crafted features, they proposed an Android malware detection system based on CNN that uses raw opcode sequences extracted from disassembled code. This approach is validated through four different Android malware datasets, so-called small, large, very large, and new, from Genome project and McAfee labs and is compared with an *n*-gram opcode based malware detection method. Even though for small and large datasets, the proposed method achieves almost the same performance, it is shown for very large datasets that the performance of the proposed deep model improves as more training data is provided. In spite of proposing a computationally efficient algorithm compared with a conventional *n*-gram based malware classification system, they do not achieve an acceptable accuracy on large and very large datasets, 80% and 87%, respectively. Also, they do not cross-validate the dataset, and just split it into 90% for training and validation and 10% for testing which may cause an overfitting problem.

Coupled with the trend toward applying CNN to Android malware detection, Nix et al. [103] proposed a DL framework based on CNN for Android application and malware classification getting help from API-call sequences. They design a pseudo-dynamic program analyzer that generates a sequence of API calls along the program execution path. The designed CNN learns sequential patterns for each location by performing convolution alongside the sequence and sliding the convolution window down the sequence. Additionally, it consists of multiple CNN layers to construct high-level representation of features. They compare their approach with RNN having LSTM, *n*-gram-based SVM, and bag of words-based NB. The experiments include malware detection, identification of malware families, and categorization of benign software into groups based on functionality. Both CNN and LSTM outperform *n*-gram-based methods while CNN shows a better performance in comparison with LSTM which is inherently fitted sequential data.

Nauman et al. [104] leveraged several well-known DL models for Android malware detection. These models are fully connected NNs, CNNs, deep AEs, DBNs, and RNNs in the form of LSTM models which have been analyzed for hyperparameter fine-tuning. Moreover, they apply Bayesian Statistical Inference model. Their best model achieves an F_1 score of 98.6% with an accuracy of 98.3% as compared to the existing best F_1 score of 87.5% and accuracy of 95.3%. They use publicly-available dataset, Drebin [105] with two types of features extracted from the manifest file and static analysis of code. They might improve the performance of the proposed approach if they combine dynamic analysis features with their existing ones. Furthermore, the Drebin dataset is pretty old, samples are from 2010 to 2012; that is not a good representative of new sophisticated malware samples and does not give us a good criterion to assess the effectiveness of the proposed approach.

Having continued the trend of using DL algorithms in malware detection, the question comes to mind of how to find the most appropriate configuration for DNNs. Martin et al. [106] tried to address this question to some extent by proposing a genetic algorithm designed to evolve the parameters, and the architecture of a DNN with the goal of maximizing the Android malware classification accuracy and minimizing the complexity of the model. Although they have introduced an interesting area of research to select different parameters and the architecture of the DNN, they use a small-sized and out-of-date dataset (Drebin), just 1,919 after removing families with less than 20 features, regarding the static analysis they had selected as their analysis approach.

Taking raw sequences of API method calls extracted from dex assembly, Karbab et al. [107] proposed *MalDozer* to detect malware and attribute malware family using CNN. The architecture of the designed CNN comprises one convolutional layer using ReLU as activation function, one MP layer, one fully connected layer having dropout and bench normalization to avoid overfitting and improve results simultaneously, and one output layer for two-task classification. The input to the NN is a sequence of vectors acquired using embedding model, word2vec. The proposed approach is tested on several datasets including MalGenome, 2015 (1 K samples), Drebin 2015 (5.5 K samples), MalDozer (20 K samples), and a merged dataset of 33 K malware samples. Additionally, 38 K benign apps downloaded from Google Play (Google Play, 2016) are also used in the evaluation. The F_1 score is achieved for detection task and family attribution are between 96%–99% and 96%–98%, respectively, whereas the false positive rate is around 0.06%–2%. Although their approach seems to be effective in malware detection and family classification, the datasets they have used for evaluation, namely Drebin and MalGenome, are out-of-date, as mentioned above. In addition, it is not remarked how old the Maldozer dataset is which directly impacts the resilience of *Maldozer* against new sophisticated and obfuscated malware families. In response to the question of whether *Maldozer* can detect samples of unknown malware families, six malware families are picked from the Drebin dataset and accuracy is computed as each family sample is incrementally added to the dataset. For most of the malware families, the accuracy for zero sample per-family is from 60% to 80%, reflecting the fact that samples were collected in the period of 2010–2012 when malware families were not that complicated. Moreover, when *Maldozer* is trained on 2013 and 2014 datasets, the evaluation results against 2016 dataset are not promising enough (F_1 score 70%).

Jan et al. [108] employed a Deep Convolutional Generative Adversarial Networks (DCGAN) to investigate the dynamic behavior of Android applications. They run about 4,000 malicious samples as well as 10,000 benign applications on a modified version of Oreo Android OS while utilizing hooks in order to capture around 5 GB of intent sequences. The proposed DCGAN architecture achieves an F_1 score of 99.6% and an accuracy of 98.8%, about 1% higher than that of the best DL compared model (M1) while obtaining the same false positive rate of 0.2%.

Table 3
Selected studies focusing on Android-based malware detection and analysis.

Year	Authors	Focus Area	Deep Model	Features	Dataset	Performance
2016	Yuan et al. [96]	Hybrid Android malware detection	DBN built on stacked RBMs	Required permissions, sensitive API, and dynamic behaviors extracted through Droidbox	20,000 benign apps from Google Play and 1,760 malware samples from Contagio and Genome datasets	ACC=96.76%
2016	Hou et al. [97]	Static Android malware detection	DBN built on stacked RBMs	API call blocks extracted from the Smali codes	50,000 samples from Comodo Cloud Security Center from which 22,500 samples are malware, 22,500 samples are benign, and 5,000 samples are unknown	ACC=96.66% FPR=3.44% FNR=3.24%
2016	Su et al. [98]	Static Android malware detection	DBN built on stacked RBMs for feature reduction followed by SVM	Requested permissions, used permissions, sensitive API calls, actions, and application components	Dataset contains 3,986 malware samples and 3,986 benign applications	ACC=99.4%
2016	Xu et al. [99]	Hybrid Android malware detection	DNN composed of stacked RBMs	Static features: $n-$gram of requested permissions, advertising networks, intent filters, suspicious calls, network APIs, providers, instruction sequences, and system call sequences. Dynamic features: $n-$gram graph of system call sequences	Dataset contains 4002 benign samples and 1,886 malicious samples from Google Play and VirusShare	ACC=94.7% FPR=1.8% FNR=12.67%
2016	Costa et al. [100]	Static Android malware detection	RBM fine-tuned by meta-heuristic techniques	152 permissions	278 benign samples and 121 malware samples	ACC=74%
2016	Hou et al. [101]	Dynamic Android malware detection	SAE	System call graph	Comodo Cloud Security Center which consists of 1,500 benign apps and 1,500 Android malware	ACC=93.68% FPR=6% FNR=6.64%
2017	McLaughlin et al. [102]	Static Android malware detection	CNN	Raw opcode sequences extracted from disassembled code	Genome dataset consists of 863 benign apps and 1,260 malware apps from 49 different families. McAfee dataset of 2,475 malware and 3,627 benign samples. Another McAfee dataset of 9,268 benign files and 9,902 malware files used for testing	ACC=87% PR=87% RC=85% F_1=86%
2017	Nix et al. [103]	Static Android malware/application classification	CNN and LSTM	API call sequences	Contagio dataset consists of 216 malware of 8 families and benign apps spanning 11 categories	ACC=99.4% PR=100% RC=98.3%
2017	Nauman et al. [104]	Static Android malware detection	FFNN, CNN, AE, DBN, and LSTM	Permissions, intents filtered by the target, activity list in the manifest, API calls raised in the code, services registered	Combination of Drebin and VirusShare datasets, including one and half million samples	ACC=98.9% PR=98.2% F_1=98.6% AUC=98.3% DTR=98.1% FPR=0.2%
2017	Martin et al. [106]	Static Android malware family classification	DNN and GA for optimizing the architecture and parameters of DNN	1,119 total features related to information flows, API calls, events listened by each app	Drebin dataset of 1,919 samples from 179 families	ACC=91%
2018	Karbab et al. [107]	Static Android malware detection and attribution	CNN with a convolution layer with ReLU, MP layer, and a fully connected layer as inner layers	API method calls	33 K malware samples from 32 malware families collected from Malgenome and Drebin datasets, and 38K benign apps from Google Play	F_1=89.3/98.18% PR=89.5/98.18% RC=89.34/98.18%
2018	Jan et al. [108]	Dynamic Android malware detection	DCGAN	Intent sequences	4,000 malicious samples and 10,000 benign samples	F_1=99.6% ACC=98.8% PR=99.3% FPR=0.2%

Table 3 exhibits selected studies focusing on Android-based malware detection and analysis using DL algorithms based on the model, the features, the dataset they have used, and the performance measures. For the performance measures, we have reported the best-performing DL model in the paper. For the two task-classification models, we have reported performance measures for both tasks, if they exist, indicated by "/".

4.2. Intrusion detection

Intrusion detection has been changed to a new trend in network security area today. The goal of IDSs is to identify malicious activities in computer network traffics and raise an alarm when a suspicious activity is detected. Although most of the traditional learning techniques, such as NN, fuzzy model, and Hidden Markov

Model (HMM) have made great achievements in IDS, they suffer from having shallow architecture leading to some limitations in dealing with big network traffic data. Additionally, the traditional learning methods cannot be applied properly to complex classification problems due to their specific constraints. In contrast, DL models have shown outstanding performance in large-scale data analysis.

Application of a DL algorithm to network intrusion detection dates back to 2011 when Salama et al. [109] presented an anomaly-based intrusion detection hybrid scheme combining DBN and SVM to classify the network intrusion into two outcomes: normal or attack. The DBN is constructed based on RBM layers and is used as a feature reduction method followed by an SVM classifier. The proposed hybrid intrusion detection framework consists of three main phases: preprocessing, DBN feature reduction, and classification. Phase one includes preprocessing which first maps symbolic features into numeric values, then scales the data to fall within the range [0, 1], and lastly assigns attack names to one of the five classes, normal, DOS, U2R, R2L, and Probe. In phase two, i.e., DBN training, the DBN network is used as a dimensionality reduction method with BP to enhance the reduced output training data. The BP-DBN structure of DBN network is composed of two RBM layers, the lower and the higher ones. The lower layer efficiently reduces the data from 41 to 13 features and the higher one decreases the data from 13 features to five final features. Finally, in the classification phase, the five output features are passed to an SVM classifier which classifies the testing data into normal or attack. The performance of the proposed DBN-SVM scheme is tested on NSL-KDD dataset and compared with standalone DBN and SVM. It has been shown that the accuracy of the proposed DBN-SVM is above 90% for various training percentage settings. Moreover, performance accuracy of DBN as a feature reduction method outperforms PCA, Gain Ratio and Chi-Square. Nevertheless, no false positive rate or false negative rate has been reported in this paper which shows a considerable weakness of the proposed scheme.

Recently, many network anomaly detection techniques have been introduced to differentiate between the anomalous and normal behaviors for detecting undesired or suspicious activities. The anomaly detection methods can be categorized into three distinct groups [110]: (a) supervised, (b) semi-supervised, and (c) unsupervised. In supervised anomaly detection, a training set of labeled samples is available for both normal and abnormal class. In semi-supervised anomaly detection, the training set contains only labeled instances of the normal class. Anything that cannot be characterized as normal is thus marked as anomalous. In unsupervised anomaly detection, no labeled training set is available neither for the normal class nor for the anomalous class. The quality of the classification task directly depends on the training model on which the classification model is built. If a complete set of labeled training data is available, all the instances will be classified correctly. Visibly, abnormal events appear less frequently than normal ones and labeled anomalous data in the real world are not readily available. Consequently, anomaly detection systems should not be restricted by any predefined set of anomalies and should be as flexible as possible to classify any unknown event. To this end, a semi-supervised anomaly detection system based on RBM was proposed by Fiore et al. [110] where the classifier is trained with normal traffic data only, so that knowledge about anomalous behaviors can evolve in a dynamic way. The advantage of this discriminative model is its effectiveness in coping with zero-day attacks since they are not limited to any prior knowledge. RBM is a generative classifier aiming at capturing as much of the variational potential of the inputs to describe the input data. In contrast, Discriminative RBM (DRBM) focuses on combining the descriptive power with a sharp classification ability. To make an RBM operate in a supervised fashion, Fiore et al. introduced an additional input containing the targets. The datasets will thus be structured as sequences of pairs (v, y) where v is an input vector and y is a class $y \in 1, \ldots, C$. In DRBM, $p(y|v)$ is optimized instead of the joint distribution, i.e., $p(y, v)$ to provide better performance and higher classification accuracy. A set of experiments are performed on two real-world traffic; the first one is normal traffic on one host while the other one is infected by a bot. In the first experiment, the goal is to test the accuracy of the DRBM to recognize anomalous traffic on real data. In the second experiment, the DRBM is trained with the 10% training KDD'99 dataset and tested against the real data. In the experiments, 28 out of 41 features are used which are related to network traffic. The KDD'99 dataset containing 494,021 records of attack data in training set, 11,850 in testing set, and 41 attributes is used to evaluate the performance of the proposed IDS model. The experimental results prove that there will be a considerable reduction in the classification performance when the environment of the test data is widely different from the network from which the training data were extracted. Therefore, some more investigations should be done over the nature of anomalous traffic and its differences with normal traffic.

In 2014, GAO et al. [111] presented an intrusion detection framework based on DBN to be trained in three stages. First, the symbolic attribute features in KDD'99 dataset are digitized and then normalized. Second, the DBN is pre-trained on the standardized data to learn a stack of RBMs by the CD algorithm. The output feature representation of each RBM is used as the input data for training the next RBM in the stack. At last, after the pre-training, the DBN is fine-tuned using BP of error derivatives and the initial weights and biases of each layer are corrected. The trained DBN can then be used as an IDS classifier. The experimental results of the implemented DBN framework with multiple RBMs and pre-training are shown to have better performance in comparison with SVM or NN algorithms (about 93.49% detection rate and 0.76% false positive rate) with the best DBN architecture to be 122-150-90-50-5. However, not enough parameter fine-tuning is done for the number of layers and neurons. Furthermore, KDD'99 dataset is an old dataset with lots of deficiencies and thus is not a good choice for assessing the DL model.

Later in 2015, Li et al. [112] proposed a hybrid malicious code detection model based on AE and DBN. The AE is used for data dimensionality reduction and the DBN is employed as a classifier. First, the input training sample is digitized and normalized as the pre-processing phase. After that, the AE is used for dimensionality reduction and feature mapping. The resulted eigenvector is then used to feed DBN classifier. To train the DBN, a stack of RBMs are used based on their specific learning rules. Finally, the supervised BP algorithm is applied to fine-tune the weights and biases of the entire network. To evaluate the efficiency of the presented method, the test samples are given to the trained classifier to identify the malicious code from the benign one. The authors have implemented various models based on different numbers of pre-training and fine-tuning iterations, including, $AE + DBN^{5-5}$ (five times of pre-training and five times of fine-tuning iterations), $AE + DBN^{10-10}$ (10 times of pre-training and 10 times of fine-tuning iterations), and $AE + DBN^{10-5}$ (10 times of pre-training and five times of fine-tuning iterations), and compared the whole models with simple DBN. The experimental results show that using AE and DBN with more iterations, i.e., $(AE + DBN^{10-10})$, results in higher accuracy (almost 1% rise) and lower false positive rate (almost 8% reduction).

In the same year, Yang et al. [113] introduced an SVM-RBM classifier to detect network anomalies. Being based on RBM, the model trains the SVM with Stochastic Gradient Descent (SGD) algorithm to perform classification. In the proposed model, the features are extracted by the RBM to learn high-level feature representations. Then, the SVM classifier is used to detect anomalies. The number

of hidden units are changing during the feature extraction process. When some desirable features are acquired, the SVM model is trained using the SGD algorithm and Hinge Loss function. The authors have adopted this function to address the outlier problem which is caused by SVM and use the SGD algorithm to train the model. To speed up the running time of the model, the SGD algorithm is implemented with Spark in a parallel way. The proposed approach, SVM-RBM, is compared with some well-known classifiers, such as DT, NB, and NN in terms of precision and F_1 score. The experimental results on 20 GB of real traffic data show that SVM-RBM has higher precision and F_1 score than those of others. The authors also explore the number of hidden units to improve the performance of SVM-RBM. They conclude that when the number of hidden units is greater than 2^{20}, the precision of SVM-RBM increases to almost 90%. Furthermore, the influence of learning rate to SGD algorithm has been investigated. If the learning rate is small, it will take a lot of time to converge to the desired value. On the other hand, when the learning rate is large, the optimal solution will be skipped but the convergence speed will be much faster. They have achieved an optimal value of 0.1 for the learning rate. Even though precision gets up to 90% for 500 MG–1 GB of samples, F_1 score of the proposed SVM-RBM reaches at most 81% for different training sizes from 100 MG-1 GB implying an undesirable false negative rate.

At the beginning of 2016, Kim et al. [60] applied an LSTM to RNN to construct an IDS model. For implementation, the authors use softmax for the output layer and SGD for an optimizer. The model is then trained using the KDD'99 dataset. The learning rate and hidden layer size are defined by conducting experiments for different values and choosing the proper ones. For testing, the authors make 10 test datasets and measure the performance. The comparison with other IDS classifiers shows that LSTM-RNN has just about 0.27% higher accuracy than Probabilistic Neural Network (PNN) and an unacceptable false alarm rate of 10%, about 4–7% higher than those of PNN, RBNN, and SVM.

Incorporating a different perspective, Dong et al. [114] implemented different ML methods on the KDD'99 dataset to classify network traffic and distinguish the normal one from the attacks, i.e., DOS, Probe, U2R, and R2L. They apply Synthetic Minority Over-sampling Technique (SMOTE) to deal with the problem of imbalance in the dataset. They then compare the traditional ML methods, such as NB, SVM, and DT (C4.5) with SVM-RBM [113] in terms of precision. The results show that SVM-RBM produces the best results in comparison of other classic shallow learning architectures. Nonetheless, the maximum precision of SVM-RBM is acquired on 80% of the training samples which are 82% for DOS attack, 58% for U2R attack, and 42% for R2L, remarkably lower values compared with state-of-the-art approaches in this domain. Moreover, any other measures, such as false positive rate, false negative rate, recall, or F_1 score are not provided in the analysis part. Overall, the approach does not contribute any added value to the research community.

Liu et al. [115] proposed an intrusion detection model using IDBN which is an improved DL model based on Extreme Learning Machine (ELM). First, the symbolic features are numeralized and normalized. Then the ELM is applied to DBN. After that, they compare the proposed model with existing traditional DBNs on the NSL-KDD dataset that improves the detection rate up to about 1% but doubling the training speed. The only evaluation metric they have applied is the detection rate, which invalidates their proposed approach.

Potluri et al. [116] employed an accelerated DN architecture with three hidden layers to identify network data abnormalities. To improve the effectiveness of the DN architecture, they pre-train it using AE, which is called accelerated training. It is done in three stages: (a) pre-training step to train a sequence of shallow AEs one layer at a time, using unsupervised data, (b) fine-tuning step one to train the last layer using supervised data, and (c) fine-tuning step two to use BP to fine-tune the entire network using supervised data. To analyze the performance of the detection mechanism, NSL-KDD dataset is used with all 41 features for computing training time and accuracy. The detection accuracy is high when just two classes of normal and attack are considered, whereas due to lack of sufficient data for training, U2R and R2L are not well detected and this reduces the overall detection accuracy of the DNN-based IDS.

Using DNN, a flow-based anomaly detection system in Software Defined Networking (SDN) was proposed by Tang et al. [117]. The DNN architecture consists of an input layer, three hidden layers, and an output layer. The hidden layers themselves contain 12, six, and three neurons each. The dimensions of the input and output layer are six and two, respectively. In the experiments, a subset of six features is selected from the original 41 features of the NSL-KDD dataset including length of connection, protocol type, number of data bytes from source to destination, number of data bytes from destination to source, number of connections to the same host as the current connection in the past two seconds, and number of connections to the same service as the current connection in the past two seconds. The experimental evaluations on the trained model using the NSL-KDD dataset and hyperparameter tuning produce a low accuracy of 75.5% which is not a good enough result to be used in any commercial product or alternative solution. On the one hand, they have just used six features from basic features and traffic-based features out of 41 features. On the other hand, the DL model is just adjusted for learning rates while the number of hidden layers and hidden neurons are not tuned. For example, they have not justified why they have used three hidden layers of 12, six, three neurons and why they have used DNN for just six features.

Duy et al. [118] applied an FFNN model to network intrusion detection using NSL-KDD dataset and compared the results with other typical ML algorithms. After vectorizing all features as one-hot vector in the pre-processing phase, all 41 features are converted into 120. Without mentioning anything about hyperparameter fine-tuning, the architecture of the NN is considered to comprise four hidden layers of 60 neurons for which ReLU serves as the activation function, whereas softmax is devised at the output layer. Likewise, the learning rate is set to be 0.001 without mentioning the process behind selecting this value. They achieve 96.2% F_1 score, making just 0.6% improvement over Random Forest (RF). Furthermore, they reduce feature numbers from 120 to 20 using the Perturb Feature Ranking method, subsequently impacting F_1 score by just 2.6%.

To speed up the convergence process, Mohammadi et al. [119] used a deep AE model followed by a Memetic algorithm, an extension of classic GA, to produce a linear classification function to detect intrusion. The deep AE consists of four layers of encoder-decoder phases and Memetic algorithm parameters are set to 100, 0.03, 0.9, respectively, for the number of population, mutation probability, and crossover probability. There is no mention of the number of neurons in each layer of AE or fine-tuned learning rate. In order to evaluate their approach, they use both NSL-KDD and KDD'99 datasets. Even though the DL algorithm can classify 98.11% of the normal traffic and 98.75% of DOS on KDD'99, they only succeed in classifying 48.35% of R2L, whereas they have achieved a correction rate of 92.72% for R2L attacks on the NSL-KDD dataset. Surprisingly, they have not incorporated any criteria for errors, such as false positive rate or false negative rate.

Yin et al. [120] applied an RNN model for both binary and multi-class intrusion detection and tested the performance of RNN on the NSL-KDD dataset. For pre-processing, they convert non-numeric features into numeric values, a map from 41-dimensional

feature vectors into 122-dimensional feature vectors. They then apply the Logarithmic Scaling method and Linear Normalization, respectively, to narrow down the difference between the maximum and minimum values for some features. The final feature values would be between [0, 1]. 83.28% is the highest accuracy they have achieved on *KDDTest*$^+$ for binary classification with 80 hidden nodes and 0.1 learning rate. For multi-class classification, they got 81.29% as the highest accuracy on *KDDTest*$^+$ with 80 hidden nodes and 0.5 learning rate. Nevertheless, the detection rate per attack category is very low, 24.69% and 11.50% for R2L and U2R categories, respectively, and not acceptable for DoS and Probe being 83.4% each. Furthermore, they have not indicated the number of layers of RNN.

Another DL-based approach for intrusion detection was proposed by Farahnakian et al. [121] in which a 4-layer deep AE is trained in an unsupervised layer-by-layer fashion followed by a softmax classifier on top of that. The framework is evaluated on the KDD'99 dataset, and all the duplicate records are removed from the data. In the pre-processing stage, symbolic features are mapped into one-hot vectors and all the numeric features are normalized in the range between 0 and 1, generating 117 features from initial 41 original ones. Based on the parameter fine-tuning they have conducted, it is concluded that changing the number of neurons or layers does not influence testing accuracy of deep AE-IDS except for a tiny amount less than 1%. One of the affecting factors might be the equal number of neurons in each layer. They have reported a maximum accuracy of 94.71% and a false alarm rate of 0.42% for the multi-classification scenario.

Recently, Yin et al. [122] used GAN for continuously generating fake network data in order to improve the performance of an original botnet detection model. For GAN structure, a 3-layer LSTM is used as the generator and a 4-layer DNN is adopted as the discriminator. The authors have used 16 features based on network flow and conducted experimental studies on ISCX botnet dataset [123]. Although the proposed GAN-based model decreases the false positive rate from 19.19% to 15.59%, it is not considered as an acceptable false positive rate yet. It also produces low-performance measures, i.e., an accuracy of 71.17%, F_1 score of 70.59%, and precision of 74.04%. In addition, their framework does not consider categorical attributes like IP addresses and ports as features.

In Table 4 selected studies focusing on intrusion detection using DL algorithms based on model, features, dataset, and performance measures are given. For the performance measures, we have reported the best-performing DL model in the paper. For the two-task classification models, we have reported performance measures for both tasks, if they exist, indicated by "/".

4.3. Other: phishing detection, spam detection, and website defacement detection

4.3.1. Phishing detection

Having employed some basic features, such as structural features, link features, element features, and word list features that capture the characteristics of the phishing emails, Zhang et al. [124] aimed at detecting phishing email attacks through a 3-layer FFNN. The proposed FFNN consists of one input layer, one hidden layer, and one output layer and the number of neurons in the hidden layer is acquired by testing different settings. To fit the used dataset, tanh and sigmoid are used as activation functions, and Resilient Propagation (RPROP) training is used to train the FFNN. To do the experimental evaluations, a real dataset of 4,202 *ham* emails and 4,560 phishing emails are used. To conduct the experiment, a preprocessing stage is run to extract the aforementioned features from the emails using Perl scripts and normalize the dataset between the range [0, 1]. Finally, the dataset is trained using the training set to get the parameter estimates and then tested on the testing set to evaluate the performance of NN using cross-validation. This procedure is repeated 20 times for different sizes of the training and testing datasets. Once the evaluation metrics are calculated, the results are compared with different NN settings, i.e., the number of units in the hidden layer and activation functions. Moreover, the performance of the NN is compared with that of other well-known ML algorithms, such as DT, *k*-NN, NB, and SVM, acquiring 95.51% accuracy and 95.71% F_1 score for NN. From the statistical analysis, we can conclude that the NN offers a reasonable accuracy even when the training examples are scarce. However, the authors have not investigated the effect of adding more hidden layers to the FFNN.

One year later in 2014, Mohammad et al. [125] proposed a self-structuring NN for detecting phishing website attacks. Phishing-related features that are important in determining the type of web pages are very dynamic. Therefore, there should be a need to improve the structure of the underlying model. Their proposed model, i.e., adaptive self-structuring NN, automates the process of structuring the network. This is achieved by updating the learning rate and adding new neurons to the hidden layer. The aim of the designed model is to acquire generalization ability, i.e., the training and the testing classification accuracy should be as close to each other as possible. To this end, for 1,000 epochs, the training set accuracy is 94.07%, the validation set accuracy is 91.31%, and the testing set accuracy is 92.18%. The dataset used is composed of 600 legitimate websites and 800 phishing websites. Normally, the number of legitimate samples should be more than phishing samples to simulate the real-world scenario. They also failed to report any false positive or false negative rates. In addition, for the adaptive framework, they did not update the number of hidden layers while determining the structure of the NN.

4.3.2. Spam detection

The increasing number of spam messages sent daily resulted in designing many anti-spam filters. Many ML and few DL techniques have been employed so far to improve e-mail spam detection. RBM has shown to be effective in this area, though fine-tuning its parameters is a big challenge. Da Silva et al. [126] presented an approach to learn the intrinsic features of email messages by RBMs to identify malicious or benign content. To fine-tune the RBM parameters, Harmony Search-based Optimization technique was employed to evaluate the parameters' robustness in the context of spam detection. The RBM parameters are learning rate, weight decay, penalty parameter, and the number of hidden units. The extracted features are then fed into OPF classifier to evaluate the accuracy of the model. OPF algorithm employs the path-cost function for estimating prototypes, i.e., key samples that best represent the classes. The experiments over three public datasets, SPAMBASE, LINGSPAM, and CSDMC show that the accuracy of OPF classifier using 10 unsupervised learned features as input is higher than the one using the original 57 features. Therefore, RBMs might be suitable to learn features from email content.

4.3.3. Website defacement detection

Borgolte et al. [127] addressed website defacement as a disruptive attack that may cause serious financial damage to companies and organizations and ruin their reputation. They proposed *MEERKAT* as a monitoring system that combines SAEs and DNNs to identify defacements. Getting help from the screenshot regions (windows) of the websites, *MEERKAT* automatically learns high-level features from the visual representation of a website. Unlike previous approaches, it relies neither on additional information provided by the website's operator like its source code, content, or structure nor on manually-crafted features, but it only requires the URL of the website. Applying *MEERKAT* on the largest website

Table 4
Selected studies focusing on intrusion detection.

Year	Authors	Focus Area	Deep Model	Features	Dataset	Performance
2011	Salama et al. [109]	Anomaly-based intrusion detection	DBN constructed from RBMs for feature reduction followed by SVM classifier	41 features, such as protocol type, service, flag, etc.	NSL-KDD dataset	ACC=92.84%
2013	Fiore et al. [110]	Semi-supervised anomaly detection	DRBM	28 out of 41 features that are related to network traffic	First experiment: real-world traffic including normal traffic on one host and bot-infected traffic on another host. Second experiment: training with 10% KDD dataset and testing with real data	ACC=94%
2014	Gao et al. [111]	Intrusion detection	DBN built on stacked RBMs	41 features numeralized to 122 features	KDD dataset	DR=93.49% TPR=92.33% FPR=0.76%
2015	Li et al. [112]	Malicious code detection	AE for dimensionality reduction and DBN based on stacked RBMs for classifier	41 features normalized	10% KDD dataset containing 494,021 training samples and 311,029 testing samples	ACC=92.1% TPR=92.2% FPR=1.58%
2015	Yang et al. [113]	Anomaly-based intrusion detection	SVM-RBM	HTTP response code and request type, packet length, attachment type and size, download/upload, etc.	20 GB of traffic data	PR=90.0% F_1=81%
2016	Kim et al. [60]	Intrusion detection and attack classification	LSTM	41 features normalized	10% KDD dataset	ACC=96.93% DR=98.88% FPR=10.04%
2016	Dong et al. [114]	Attack classification	SVM-RBM	41 features	10% KDD dataset	PR(DOS)=82% PR(U2R)=58% PR(R2L)=42%
2016	Liu et al. [115]	Intrusion detection	DBN built on stacked RBMs with ELM applied into the learning process of DBN	41 features mapped into 122	NSL-KDD dataset	DR=91.8%
2016	Potluri et al. [116]	Intrusion detection and attack classification	SAE with softmax layer at the top	41 features	NSL-KDD dataset	ACC(Nor)=95% ACC(DOS)=97.7% ACC(Probe)=89.8% ACC(R2L)=13% ACC(U2R)=39.6%
2016	Tang et al. [117]	Anomaly detection in SDN	DNN	A subset of 6 features out of 41 original features	NSL-KDD dataset	ACC=75.75% AUC=86%
2017	Duy et al. [118]	Intrusion detection	DNN	41 features mapped into 120	NSL-KDD dataset	F_1=96.2%
2017	Mohammadi et al. [119]	Intrusion detection and attack classification	AE followed by Memetic algorithm	41 features	NSL-KDD and KDD dataset	ACC(Nor)=98.11% ACC(DOS)=98.75% ACC(Probe)=83.34% ACC(R2L)=48.35% ACC(U2R)=74.28%
2017	Yin et al. [120]	Binary and multi-class intrusion detection	RNN	41 features mapped into 122	NSL-KDD dataset	ACC=83.28%/81.29% DR(DoS)=83.49% DR(R2L)=24.69% DR(U2R)=11.5% DR(Probe)=83.40% FPR(DoS)=2.06% FPR(R2L)=0.80% FPR(U2R)=0.07% FPR(Probe)=2.16%
2018	Farahnakian et al. [121]	Binary and multi-class intrusion detection	AE with softmax classifier on top of it	41 features numeralized to 117 features	KDD dataset including 494 k samples for training and about 300 k for testing	ACC=96.53%/94.71% DR=95.65%/94.53% FPR=0.35%/0.42%
2018	Yin et al. [122]	Botnet detection	GAN using LSTM as G and DNN as D	16 features based on net flow	ISCX botnet dataset	ACC=71.17% PR=74.04% F_1=70.59%

defacement dataset to date, including 10,053,772 defacements observed from January 1998 to May 2014, they ended up with true positive rates between 97.422% and 98.816%, false positive rates between 0.547% and 1.528%, and Bayesian detection rates between 98.583% and 99.845%, significantly outperforming previous work.

4.4. Challenges and limitations

The shortcomings of most of the aforementioned studies have been described earlier. For example, adding layers to a DN architecture does not increase the overall accuracy in the majority of

the proposed models, or most of the malware-related studies focus on static detection rather than dynamic analysis or classification. However, there exist some common limitations as explained below in detail:

4.4.1. Parameter optimization

The parameters of the deep models are not fine-tuned accurately and thoroughly.

The deep model parameters include, but are not limited to the number of hidden layers, number of hidden neurons in each layer, type of activation function in each layer, number of epochs, error rate of each epoch, type of optimization algorithm, learning rate, momentum rate, and dropout rate.

In order to have a competent architecture of NN, one can follow a set of pre-defined rules. For example, there exist some formulas for computing the optimal number of hidden nodes in each layer [128–133]. However, rules do not always guide us to the desired architecture and parameter settings. Some argue that to achieve a near optimal parameter setting for an NN, all combinations of parameters need to be examined as much as possible and the best architecture be picked based on the best output measures.

4.4.2. Evaluation

Not enough ML measures are computed in the experimental analysis part of the studies.

Computational metrics, accuracy, false positive rate, false negative rate, precision, recall, etc. are essential for evaluating any ML or DL algorithm. None of the metrics alone can be expressive of high performance of the underlying algorithm. In the evaluation of a classification task, for example, we may achieve high accuracy by marking all training samples as positive yet we would have a high false positive rate as well. Hence, we need to judge our model by incorporating all computational metrics.

4.4.3. Dataset

The datasets are generally out-of-date, having a small number of samples and non-diverse data, that might cause overfitting problem.

In malware datasets, the malware size is not sufficient. If the dataset is not updated, the approach would fail against zero-day and obfuscated malware. For intrusion detection problems, most of the approaches use KDD'99 and NSL-KDD as datasets. However, they both are criticized for having some deficiencies. The distribution of the attacks in KDD'99 dataset is very uneven, making cross-validation very difficult [64]. Also, it is important to note that the data in both datasets are synthetic since anomalies have been injected into normal traffic that leads to unreal traffic and subsequently adding undesired bias to the data [134]. In addition, attack types are obsolete and new protocols like HTTPS are not included in both datasets.

4.4.4. Time complexity

There is normally no time complexity analysis included in the majority of the studies.

4.4.5. Choosing a specific DL model

It is not formally verified why these deep models are used. Each DN architecture has some inherent characteristics that make it appropriate to be used in some specific application. For instance, in an FFNN, the connections between neurons do not form a directed cycle. Therefore, they are more appropriate to model relationships between inputs and outputs, unlike the RNN which is often used for text and pattern recognition. Formally verifying the intention behind selecting a specific deep model is highly recommended and would result in a solid and robust framework.

4.4.6. Inference justification

The proposed deep models cannot explain the logic behind the decisions that they make.

One of the critiques with NNs has always been their inability to explain their conclusions or assumptions. We cannot explain the logic behind the decision that an NN makes because the underlying knowledge is represented by numerical weights. So it would be complicated and somehow impossible to correlate the contributing neurons or features with the output results. This is specifically very important in domains, such as bank and insurance companies that subsequently must give their customers feedback and the reasons why the decisions were made.

5. Analysis and discussion

In this section, we analyze all studies concentrating on emerging areas of DL and cybersecurity, intrusion detection and malware detection/analysis, from four different aspects: focus area, methodology, model applicability, and feature granularity.

5.1. Focus area

All related work in this area focuses on detection, classification, or analysis. These studies either detect a PC/Android malware and/or categorize them into miscellaneous families using static, dynamic, or hybrid techniques. However, a few of the malware-related studies concentrate on efficient malware analysis only. Another group of studies detect intrusion attacks and/or classify them into different attack types.

5.2. Methodology

Methodology refers to the type of deep model that has been applied by the studies discussed earlier. Deng et al. [71] provide a classification scheme on the existing deep architectures and algorithms, categorizing them into three classes of generative, discriminative, and hybrid. The classification is based on the intention behind using the architectures and models, e.g., synthesis/generation or recognition/classification. Fig. 14 illustrates the classification schema with the deep models associated with each class. Deep Boltzmann Machine (DBM) is like an RBM with a number of hidden layers instead of just one.

5.2.1. Generative architectures

Enjoying the benefits of data synthesis and pattern analysis, generative deep architectures are powerful at modeling the input data and generating random instances similar to existing ones. They are intended to characterize the high-order correlation properties of the visible data [71]. Generative models learn the joint probability distributions of the visible data X and their related class Y, $P(X, Y)$, and then infer the posterior probability distribution of $P(X|Y = y)$ [135]. Using Bayes rule, we can turn this architecture into a discriminative one for classification purposes.

5.2.2. Discriminative architectures

Unlike generative models, discriminative architectures do not care about the data generation process and are merely focused on the quantity necessary for the prediction task [135]. In other words, they learn the conditional probability of the class Y, given the visible data X, $P(Y|X = x)$ and then try to classify the data. Consequently, the discriminative models appear to be less expensive, whereas generative models require additional modeling efforts.

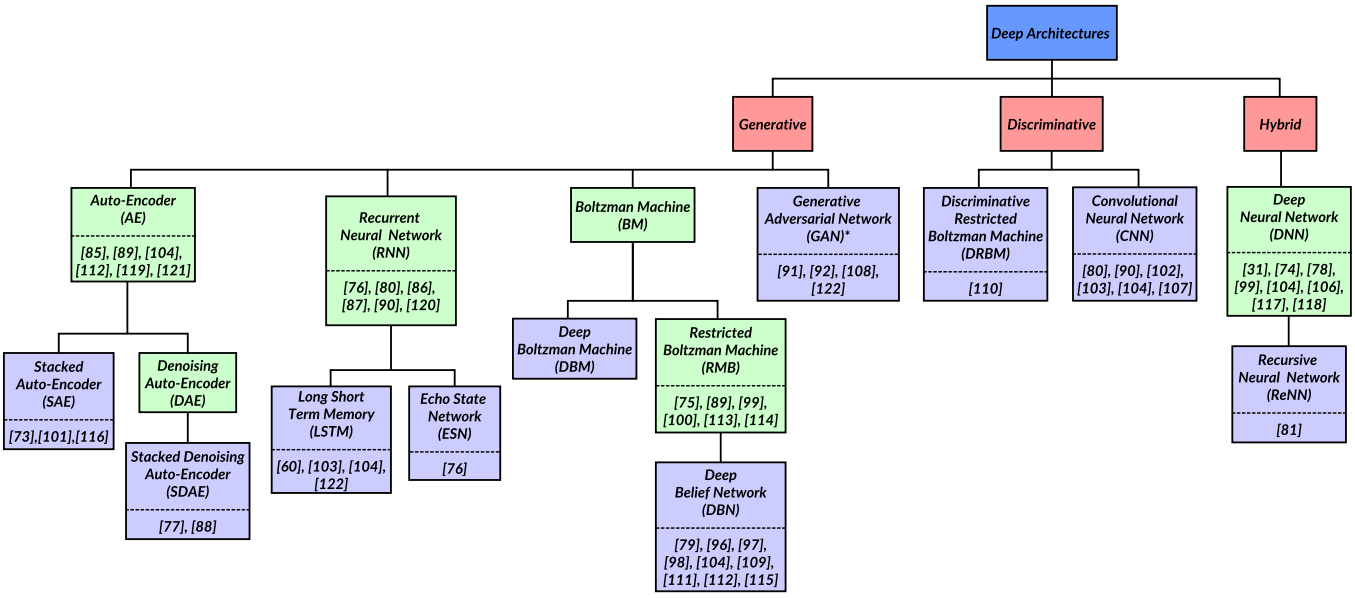


Fig. 14. Classification schema of DN and DL architecture. (* Although GAN is usually classified as a generative algorithm, its process is hybrid-like.)

5.2.3. Hybrid architectures

The goal of a hybrid architecture is to utilize a generative model to help with discrimination regarding two perspectives, namely optimization and/or regularization [71], specifically in naturally non-linear domains. From an optimization point of view, generative models could be exploited in the pre-training phase to provide outstanding initialization point for the deep model. Also, from a regularization viewpoint, the generative models could control the complexity of the overall model effectively. As shown in Fig. 14, DNN is categorized as a hybrid DN since the weights of the DNN are often pre-trained (initialized) using DBN.

5.3. Model applicability

DN architectures have been used in two different ways, either as a dimensionality reduction method before applying a classifier, such as SVM, LR, NB, and DT or as a classifier by itself. Some of the DN models are used for both tasks. DBN and SAE, for example, could be employed for both feature reduction and classification tasks.

5.4. Feature granularity

Typically, DNNs use several hidden layers to hierarchically learn a high-level representation of the input features. Suppose that we intend to classify a dog image. In the first layer, the edges of the dog might be detected. The second layer might detect curves associated with the dog image. Finally, in the third layer, the whole dog image might be detected. Practically, what the DL model does here is to bridge the gap between high-level representation and low-level features [136].

The input vector to a DL model could be represented by low-level or high-level features. High-level features are those characteristics of the input data which reflect a global perspective of the whole data following an in-depth pre-processing stage. Conversely, low-level features are acquired from a light-weight pre-processing stage and are not different from the representation of the raw file.

Recently, De Paola et al. [88] conducted a set of experiments showing that, when using an optimal set of features, i.e., high-level features, DNNs do not significantly improve the performance of malware detection/classification, while using low-level features increases the detection capability to some extent.

Table 5 shows the analysis of studies focusing on intrusion detection and malware detection/analysis based on the four criteria.

6. Conclusion, concerns, and open directions

In this survey, we presented the history of NN and DL by focusing on the basics of DL models and algorithms. We further proposed a general DL framework for cybersecurity applications and elaborated its four major modules including analysis, feature extraction, preprocessing, and DL-based classifier. A series of related studies in intrusion detection, malware detection, phishing/spam detection, and website defacement detection using DL methods were discussed along with their achievements and limitations. All intrusion and malware related studies were further analyzed and compared from four perspectives: focus area, methodology, model applicability, and feature granularity. Categorizing the DN architectures into three classes of generative, discriminative, and hybrid, a taxonomy was presented for the types of deep models falling into each category.

Concerning public desire towards the area of DL applications to cybersecurity, there are still many open directions for progress. DL today is mostly about pure supervised learning. A major drawback of supervised learning is that it requires a great deal of labeled data and it is quite expensive to collect them. So, in the future, DL is expected to be unsupervised, more human-like [137].

Using DL in a semi-supervised fashion is another way to deal with the vast amount of unlabeled data in the cybersecurity area. Semi-supervised learning is a feasible model comparable to the human learning process. Human learning incorporates a little instruction (labeled data) in the early stages of childhood and many observations (unlabeled data). Most of the DNN architectures employ two separate phases for training, namely unsupervised layer-wise pre-training and supervised fine-tuning, which is computationally very expensive [138]. Using a powerful semi-supervised DNN architecture, we can combine supervised learning with unsupervised learning in DNNs and minimize the sum of supervised and unsupervised cost functions by BP. As a result, there will be no need

Table 5
Analysis of studies focusing on intrusion detection and malware detection/analysis based on the four perspectives.

No	Year	Author	Features															
			Domain			Focus Area						Methodology			Model Applicability		Feature Granularity	
			PM	AM	ID	ST	DN	HBF	DT	CL	AN	GN	DS	HBM	DR	CR	LLF	HLF
1	2013	Dahl et al.	✔			✔			✔					✔		✔	✔	✔
2	2014	Benchea et al.	✔			✔			✔			✔			✔		✔	✔
3	2015	Pascanu et al.	✔			✔			✔			✔				✔	✔	
4	2015	David et al.	✔				✔		✔	✔		✔				✔	✔	
5	2015	Saxe et al.	✔			✔			✔					✔		✔		✔
6	2016	Hardy et al.	✔			✔			✔			✔				✔	✔	
7	2016	Huang et al.	✔				✔		✔	✔				✔		✔	✔	✔
8	2016	Ding et al.	✔			✔			✔			✔			✔	✔		✔
9	2016	Kolosnjaji, et al.	✔				✔		✔	✔		✔	✔			✔	✔	
10	2016	Shibahara et al.	✔				✔				✔			✔		✔		✔
11	2017	Yousefi-Azar et al.	✔		✔	✔			✔	✔		✔				✔		✔
12	2017	Lison et al.	✔			✔			✔			✔				✔	✔	
13	2018	Rhode et al.	✔				✔		✔			✔				✔		✔
14	2018	De Paola et al.	✔			✔			✔			✔				✔	✔	✔
15	2018	Ye et al.	✔			✔			✔			✔				✔	✔	
16	2018	Kim et al.	✔			✔			✔	✔		✔	✔			✔	✔	
17	2018	Kim et al.	✔			✔			✔			✔				✔	✔	
18	2018	Kim et al.	✔			✔			✔			✔				✔	✔	
19	2016	Yuan et al.		✔				✔	✔			✔				✔	✔	
20	2016	Hou et al.		✔		✔			✔			✔				✔		✔
21	2016	Su et al.		✔		✔			✔			✔			✔		✔	
22	2016	Xu et al.		✔				✔	✔					✔		✔	✔	✔
23	2016	Costa et al.		✔		✔			✔			✔				✔	✔	
24	2016	Hou et al.		✔			✔		✔			✔				✔		✔
25	2017	McLaughlin et al.		✔		✔			✔				✔			✔	✔	
26	2017	Nix et al.		✔		✔			✔	✔		✔	✔			✔	✔	
27	2017	Nauman et al.		✔		✔			✔			✔	✔	✔		✔	✔	
28	2017	Martin et al.		✔		✔			✔	✔				✔		✔	✔	
29	2018	Karbab et al.		✔		✔			✔	✔			✔			✔	✔	
30	2018	Jan et al.		✔			✔		✔			✔				✔	✔	
31	2011	Salama et al.			✔	N/A			✔			✔			✔			✔
32	2013	Fiore et al.			✔				✔				✔			✔		✔
33	2014	Gao et al.			✔				✔			✔				✔		✔
34	2015	Li et al.			✔				✔			✔			✔	✔		✔
35	2015	Yang et al.			✔				✔			✔			✔			✔
36	2016	Kim et al.			✔				✔	✔		✔				✔		✔
37	2016	Dong et al.			✔					✔		✔			✔			✔
38	2016	Liu et al.			✔				✔			✔				✔		✔
39	2016	Potluri et al.			✔				✔	✔		✔				✔		✔
40	2016	Tang et al.			✔				✔					✔		✔		✔
41	2017	Duy et al.			✔				✔					✔		✔		✔
42	2017	Mohammadi et al.			✔				✔	✔		✔			✔			✔
43	2017	Yin et al.			✔				✔	✔		✔				✔		✔
44	2018	Farahnakian et al.			✔				✔	✔		✔				✔		✔
45	2018	Yin et al.			✔				✔			✔				✔		✔
Total			**18**	**12**	**16**	**21**	**7**	**2**	**43**	**14**	**1**	**34**	**7**	**9**	**8**	**39**	**23**	**27**

Legend

PM: PC Malware
AM: Android Malware
ID: Intrusion Detection
ST: Static
DN: Dynamic
HBF: Hybrid Focus Area
DT: Detection
CL: Classification
AN: Analysis

GN: Generative
DS: Discriminative
HBM: Hybrid Methodology
DR: Dimensionality Reduction
CR: Classifier
LLF: Low-Level Features
HLF: High-Level Features
N/A: Not Applicable

to have a separate layer-wise pre-training. Furthermore, it can increase detection accuracy while decreasing the computational cost.

Another concern is about parameter optimization to find a robust deep framework with well-defined parameter values. Unsupervised pre-training could assist us in assigning the initial weights and biases. The optimal number of hidden layers and hidden neurons could be obtained by some formulation and finding logical relations between the number of neurons in each layer and dataset size to narrow down the range of possibilities for the number of layers and neurons in each layer. Nevertheless, some believe that examining all possible states is the only way to find the near optimal solution. Self-organizing NN might be a good alternative for layer fine- tuning in DN in which the network can learn incrementally by adding or removing neurons according to different criteria [125,139].

Dropout is an operator that could be used in a DN to prevent overfitting of the network. Avoiding the complex co-adaptation of neurons, dropout forces the network to learn more robust and sound features by dropping out units in a DN [127]. Using dropout, we will not let neurons rely on other neurons in the network.

Concerning the applications of DL in cybersecurity [140], most of which are surveyed in this article, we believe that there is still much room for improvement. As the Internet and mobile devices have penetrated into all aspects of everyone's life today, cybersecurity has changed to a global issue that cannot be ignored by anyone, from large corporations to individuals. There exist many applications that have not been looked into well enough by the research community from a security perspective, like the Internet of Things (IoT), blockchain, and self-driving cars. DL could open a new world of possibilities for the security of these sophisticated technologies.

One of the cybersecurity areas that has gained traction among researchers is generating adversarial examples to evade DL-based IDSs [141] or malware detection systems [142] in a black box model. There exist several ways to craft adversarial examples of DL models [143], such as L-BFGS method [144], universal perturbation [145], zeroth order optimization (ZOO) [146], GAN [61,147], Jacobian matrix [148], and Hot/Cold method [149]. GAN has had significant improvement in image generation [150,151], video prediction [152], and speech enhancement [153]. A series of recent studies have used GAN to generate adversarial attacks or malware samples to deceive the IDSs and anti-malware tools [141,142,154–158].

Deep reinforcement learning has opened promising directions of research in a range of current and envisioned applications including pattern classification, autonomous navigation, robotics, air traffic control, and defense technologies [159,160]. Deep reinforcement learning was introduced by Mnih et al. [161,162] for playing Atari games as a deep learning framework that learns control policies directly from raw pixels using reinforcement learning and exceeds human performance in almost half of the games. Growing interest in the deep reinforcement learning paradigm in the cybersecurity area [163,164] is anticipated in the near future.

To conclude, it is important to note that DL is not required to be used in every domain. DL should be used in problems where we need to learn complex non-linear hypotheses with many features and high-order polynomial terms and in domains with large-scale data. For linear problems with small amounts of data, other shallow learning algorithms like LR, SVM might suffice and lead us to the expected results. Furthermore, as mentioned earlier, high-level features do not influence the performance of DN remarkably. Thus, it is highly recommended to use DL in domains where we have large quantities of raw data, alleviating the burden of heavy preprocessing of the input data.

Acknowledgments

The authors acknowledge the generous funding from the Atlantic Canada Opportunities Agency (ACOA) grant no. (201212) through the Atlantic Innovation Fund (AIF) and through a grant from the Natural Sciences and Engineering Research Council of Canada (NSERC) grant no. (232074) to Dr. Ghorbani.

References

[1] S. Mukkamala, A. Sung, A. Abraham, Cyber security challenges: Designing efficient intrusion detection systems and antivirus tools, Vemuri, V. Rao, Enhancing Computer Security with Smart Technology. (Auerbach, 2006) (2005) 125–163.
[2] A.L. Buczak, E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection, IEEE Communications Surveys & Tutorials 18 (2) (2016) 1153–1176, doi:10.1109/COMST.2015.2494502.
[3] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M.S. Gaur, M. Conti, M. Rajarajan, Android security: A survey of issues, malware penetration, and defenses, IEEE Communications Surveys & Tutorials 17 (2) (2015) 998–1022, doi:10.1109/COMST.2014.2386139.
[4] Y. Ye, T. Li, D. Adjeroh, S.S. Iyengar, A survey on malware detection using data mining techniques, ACM Computing Surveys (CSUR) 50 (3) (2017) 1–40, doi:10.1145/3073559.
[5] D. Kwon, H. Kim, J. Kim, S.C. Suh, I. Kim, K.J. Kim, in: A survey of deep learning-based network anomaly detection, Cluster Computing, 2017, pp. 1–13, doi:10.1007/s10586-017-1117-8.
[6] E. Aminanto, K. Kim, Deep learning in intrusion detection system: An overview, in: Proceedings of the 2016 International Research Conference on Engineering and Technology (2016 IRCET), Higher Education Forum, 2016, pp. 5–10.
[7] W.S. McCulloch, W. Pitts, A logical calculus of the ideas immanent in nervous activity, The Bulletin of Mathematical Biophysics 5 (4) (1943) 115–133, doi:10.1007/BF02478259.
[8] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, Psychological Review 65 (6) (1958) 65–386, doi:10.1037/h0042519.
[9] C. Van Der Malsburg, Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms, in: Brain Theory, Springer, 1986, pp. 245–248, doi:10.1007/978-3-642-70911-1_20.
[10] Neural networks, accessed May 2018, (https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html).
[11] B. Widrow, An adaptive "adaline" neuron using chemical memistors, in: Technical Report, 1960.
[12] B. Widrow, M.E. Hoff, Associative storage and retrieval of digital information in networks of adaptive neurons, in: Biological Prototypes and Synthetic Systems, Springer, 1962, p. 160, doi:10.1007/978-1-4684-1716-6_25.
[13] C.R. Winter, B. Widrow, Madaline rule II: A training algorithm for neural networks, in: Proceedings of the 2nd International Conference on Neural Networks, 1988, pp. 1–401, doi:10.1109/ICNN.1988.23872.
[14] M. Minsky, S. Papert, in: Perceptrons: An introduction to computational geometry, MIT Press, Cambridge, Mass, 1969.
[15] S. Dreyfus, The computational solution of optimal control problems with time lag, IEEE Transactions on Automatic Control 18 (4) (1973) 383–385, doi:10.1109/TAC.1973.1100330.
[16] P.J. Werbos, Applications of advances in nonlinear sensitivity analysis, in: System Modeling and Optimization, Springer, 1982, pp. 762–770, doi:10.1007/BFb0006203.
[17] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representations by error propagation, Technical Report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
[18] D.H. Ballard, Modular learning in neural networks, in: Proceedings of the 6th National Conference on Artificial intelligence (AAAI), 1987, pp. 279–284.
[19] Y. LeCun, B. Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, L.D. Jackel, Backpropagation applied to handwritten zip code recognition, Neural Computation 1 (4) (1989) 541–551, doi:10.1162/neco.1989.1.4.541.
[20] S. Hochreiter, Untersuchungen zu dynamischen neuronalen netzen, Diploma, Technische Universität München 91 (1991) 1.
[21] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Transactions on Neural Networks 5 (2) (1994) 157–166, doi:10.1109/72.279181.
[22] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al., Gradient flow in recurrent nets: the difficulty of learning long-term dependencies (2001) doi:10.1109/9780470544037.ch14.
[23] S. Hochreiter, J. Schmidhuber, LSTM can solve hard long time lag problems, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 1997, pp. 473–479.
[24] J. Weng, N. Ahuja, T.S. Huang, Cresceptron: A self-organizing neural network which grows adaptively, in: Proceedings of the 1992 International Joint Conference on Neural Networks (IJCNN), 1, IEEE, 1992, pp. 576–581, doi:10.1109/IJCNN.1992.287150.
[25] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Computation 18 (7) (2006) 1527–1554, doi:10.1162/neco.2006.18.7.1527.

[26] M. Abadi, A. Agarwal, P. Barham, E. KullBrevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv preprint arXiv:1603.04467 (2016).

[27] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, in: Proceedings of the 22nd International Conference on Multimedia, ACM, 2014, pp. 675–678.

[28] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, et al., Theano: A python framework for fast computation of mathematical expressions, arXiv preprint arXiv:1605.02688 472 (2016) 473.

[29] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, doi:10.1038/nature14539.

[30] R. Shokri, V. Shmatikov, Privacy-preserving deep learning, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 1310–1321, doi:10.1145/2810103.2813687.

[31] W. Huang, J.W. Stokes, MtNet: A multi-task neural network for dynamic malware classification, in: Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, 2016, pp. 399–418, doi:10.1007/978-3-319-40667-1_20.

[32] Y. LeCun, B.E. Boser, J.S. Denker, D. Henderson, R.E. Howard, W.E. Hubbard, L.D. Jackel, Handwritten digit recognition with a back-propagation network, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 1990, pp. 396–404.

[33] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278–2324.

[34] Cs231n convolutional neural network for visual recognition, accessed October 2016, (http://cs231n.github.io/convolutional-networks/).

[35] D.C. Ciresan, U. Meier, J. Masci, L. Maria Gambardella, J. Schmidhuber, Flexible, high performance convolutional neural networks for image classification, in: Proceedings of the 2011 International Joint Conference on Artificial Intelligence (IJCAI), 22, 2011, pp. 1237–1242.

[36] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2012, pp. 1097–1105, doi:10.1145/3065386.

[37] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: Proceedings of the 2014 European Conference on Computer Vision, Springer, 2014, pp. 818–833.

[38] S. Hijazi, R. Kumar, C. Rowen, Using convolutional neural networks for image recognition, in: Cadence Design Systems Inc.: San Jose, CA, USA, 2015.

[39] G.E. Hinton, Deep belief networks, Scholarpedia 4 (5) (2009) 5947, doi:10.1145/1756006.1756025.

[40] Oversimplified introduction to boltzmann machines, accessed November 2016, (http://rocknrollnerd.github.io/ml/2015/07/18/general-boltzmann-machines.html).

[41] A. Fischer, C. Igel, Training restricted boltzmann machines: An introduction, Pattern Recognition 47 (1) (2014) 25–39. https://doi.org/10.1016/j.patcog.2013.05.025

[42] G.E. Hinton, A practical guide to training restricted boltzmann machines, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 599–619, doi:10.1007/978-3-642-35289-8_32.

[43] S.K. Kim, P.L. McMahon, K. Olukotun, A large-scale architecture for restricted boltzmann machines, in: Proceedings of the 18th International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE, 2010, pp. 201–208, doi:10.1109/FCCM.2010.38.

[44] M.A. Carreira-Perpinan, G.E. Hinton, On contrastive divergence learning, in: Aistats, 10, Citeseer, 2005, pp. 33–40.

[45] G.E. Hinton, Training products of experts by minimizing contrastive divergence, Neural Computation 14 (8) (2002) 1771–1800, doi:10.1162/089976602760128018.

[46] Kullback_leibler divergence, accessed November 2016, (https://en.wikipedia.org/wiki/Kullback_Leibler_divergence).

[47] J.J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proceedings of the National Academy of Sciences 79 (8) (1982) 2554–2558, doi:10.1073/pnas.79.8.2554.

[48] Stacked autoencoders, accessed July 2018, (http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders).

[49] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, Greedy layer-wise training of deep networks, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2007, pp. 153–160.

[50] C. Poultney, S. Chopra, Y.L. Cun, et al., Efficient learning of sparse representations with an energy-based model, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2007, pp. 1137–1144.

[51] P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, Extracting and composing robust features with denoising autoencoders, in: Proceedings of the 25th International Conference on Machine learning, ACM, 2008, pp. 1096–1103, doi:10.1145/1390156.1390294.

[52] J.-P. Briot, G. Hadjeres, F. Pachet, Deep learning techniques for music generation-a survey, arXiv preprint arXiv:1709.01620 (2017).

[53] Y. Bengio, et al., Learning deep architectures for AI, Foundations and trends® in Machine Learning 2 (1) (2009) 1–127, doi:10.1561/2200000006.

[54] G.E. Hinton, R.S. Zemel, Minimizing description length in an unsupervised neural network, in: Preprint, Citeseer, 1997.

[55] Towards data science, applied deep learning, part 3: autoencoders, accessed August 2017, (https://towardsdatascience.com/).

[56] E. Martin, C. Cundy, Parallelizing linear recurrent neural nets over sequence length, arXiv preprint arXiv:1709.04057 (2017).

[57] Recurrent neural networks tutorial, part 2 implementing a RNN with python, numpy and theano, accessed August 2017, (http://www.wildml.com/2015/09/).

[58] L. Medsker, L. Jain, in: Recurrent neural networks: Design and applications, CRC Press, 2001.

[59] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, S. Khudanpur, Recurrent neural network based language model, in: Proceedings of the 11th International Conference on Speech Communication Association, 2010, pp. 1045–1048.

[60] J. Kim, J. Kim, H.L.T. Thu, H. Kim, Long short term memory recurrent neural network classifier for intrusion detection, in: Proceedings of the 2016 International Conference on Platform Technology and Service (PlatCon), IEEE, 2016, pp. 1–5, doi:10.1109/PlatCon.2016.7456805.

[61] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2014, pp. 2672–2680.

[62] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, F.-Y. Wang, Generative adversarial networks: Introduction and outlook, IEEE/CAA Journal of Automatica Sinica 4 (4) (2017) 588–598, doi:10.1109/JAS.2017.7510583.

[63] I. University of California, in: Kdd cup 1999, 1999.

[64] M. Tavallaee, E. Bagheri, W. Lu, A.A. Ghorbani, A detailed analysis of the kdd cup 99 data set, in: Proceedings of the 2009 Symposium on Computational Intelligence for Security and Defense Applications (CISDA), IEEE, 2009, pp. 1–6, doi:10.1109/CISDA.2009.5356528.

[65] Y. Li, T. Shen, X. Sun, X. Pan, B. Mao, Detection, classification and characterization of Android malware using API data dependency, in: Proceedings of the 2015 International Conference on Security and Privacy in Communication Systems, Springer, 2015, pp. 23–40, doi:10.1007/978-3-319-28865-9_2.

[66] A. Zulkifli, I.R.A. Hamid, W.M. Shah, Z. Abdullah, Android malware detection based on network traffic using decision tree algorithm, in: Proceedings of the 2018 International Conference on Soft Computing and Data Mining, Springer, 2018, pp. 485–494, doi:10.1007/978-3-319-72550-5_46.

[67] Z. Xu, S. Ray, P. Subramanyan, S. Malik, Malware detection using machine learning based analysis of virtual memory access patterns, in: Proceedings of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE), IEEE, 2017, pp. 169–174, doi:10.23919/DATE.2017.7926977.

[68] M. Chowdhury, A. Rahman, R. Islam, Malware analysis and detection using data mining and machine learning classification, in: Proceedings of the 2017 International Conference on Applications and Techniques in Cyber Security and Intelligence, Springer, 2017, pp. 266–274, doi:10.1007/978-3-319-67071-3_33.

[69] W.-C. Lin, S.-W. Ke, C.-F. Tsai, CANN: An intrusion detection system based on combining cluster centers and nearest neighbors, Knowledge-based Systems 78 (2015) 13–21, doi:10.1016/j.knosys.2015.01.009.

[70] F. Kuang, W. Xu, S. Zhang, A novel hybrid KPCA and SVM with GA model for intrusion detection, Applied Soft Computing 18 (2014) 178–184, doi:10.1016/j.asoc.2014.01.028.

[71] L. Deng, A tutorial survey of architectures, algorithms, and applications for deep learning, APSIPA Transactions on Signal and Information Processing 3 (2) (2014) 1–29, doi:10.1017/atsip.2013.9.

[72] K. Ota, M.S. Dao, V. Mezaris, F.G. De Natale, Deep learning for mobile multimedia: A survey, ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 13 (3s) (2017) 34:1–34:22, doi:10.1145/3092831.

[73] W. Hardy, L. Chen, S. Hou, Y. Ye, X. Li, DL4MD: A deep learning framework for intelligent malware detection, in: Proceedings of the 2016 International Conference on Data Mining (DMIN), WorldComp, 2016, pp. 61–67.

[74] G.E. Dahl, J.W. Stokes, L. Deng, D. Yu, Large-scale malware classification using random projections and neural networks, in: Proceedings of the 2013 International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2013, pp. 3422–3426, doi:10.1109/ICASSP.2013.6638293.

[75] R. Benchea, D.T. Gavriluţ, Combining restricted boltzmann machine and one side perceptron for malware detection, in: Proceedings of the 2014 International Conference on Conceptual Structures, Springer, 2014, pp. 93–103, doi:10.1007/978-3-319-08389-6.

[76] R. Pascanu, J.W. Stokes, H. Sanossian, M. Marinescu, A. Thomas, Malware classification with recurrent networks, in: Proceedings of the 2015 International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2015, pp. 1916–1920, doi:10.1109/ICASSP.2015.7178304.

[77] O.E. David, N.S. Netanyahu, Deepsign: Deep learning for automatic malware signature generation and classification, in: Proceedings of the 2015 International Joint Conference on Neural Networks (IJCNN), IEEE, 2015, pp. 1–8, doi:10.1109/IJCNN.2015.7280815.

[78] J. Saxe, K. Berlin, Deep neural network based malware detection using two dimensional binary program features, in: Proceedings of the 10th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 2015, pp. 11–20, doi:10.1109/MALWARE.2015.7413680.

[79] Y. Ding, S. Chen, J. Xu, Application of deep belief networks for opcode based malware detection, in: Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN), IEEE, 2016, pp. 3901–3908, doi:10.1109/IJCNN.2016.7727705.

[80] B. Kolosnjaji, A. Zarras, G. Webster, C. Eckert, Deep learning for classification of malware system call sequences, in: Proceedings of the Australasian Joint Conference on Artificial Intelligence, Springer, 2016, pp. 137–149, doi:10.1007/978-3-319-50127-7_11.

[81] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, T. Yada, Efficient dynamic malware analysis based on network behavior using deep learning, in: Proceedings of the Global Communications Conference (GLOBECOM), IEEE, 2016, pp. 1–7, doi:10.1109/GLOCOM.2016.7841778.

[82] A. Chinea, Understanding the principles of recursive neural networks: A generative approach to tackle model complexity, in: Proceedings of the 2009 International Conference on Artificial Neural Networks, Springer, 2009, pp. 952–963, doi:10.1007/978-3-642-04274-4_98.

[83] R. Socher, C.C. Lin, A.Y. Ng, C.D. Manning, Parsing Natural Scenes and Natural Language with Recursive Neural Networks, in: Proceedings of the 26th International Conference on Machine Learning (ICML), 2011.

[84] R. Socher, A. Perelygin, J. Wu, J. Chuang, C.D. Manning, A. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1631–1642.

[85] M. Yousefi-Azar, V. Varadharajan, L. Hamey, U. Tupakula, Autoencoder-based feature learning for cyber security applications, in: Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 3854–3861, doi:10.1109/IJCNN.2017.7966342.

[86] P. Lison, V. Mavroeidis, Automatic detection of malware-generated domains with recurrent neural models, arXiv preprint arXiv:1709.07102 (2017).

[87] M. Rhode, P. Burnap, K. Jones, Early-stage malware prediction using recurrent neural networks, Computers & Security 77 (2018) 578–594, doi:10.1016/j.cose.2018.05.010.

[88] A. De Paola, S. Favaloro, S. Gaglio, G. Lo Re, M. Morana, Malware detection through low-level features and stacked denoising autoencoders, in: Proceedings of the 2nd Italian Conference on Cyber Security, ITASEC 2018, 2058, CEUR-WS, 2018.

[89] Y. Ye, L. Chen, S. Hou, W. Hardy, X. Li, DeepAM: A heterogeneous deep learning framework for intelligent malware detection, Knowledge and Information Systems 54 (2) (2018) 265–285.

[90] C.H. Kim, E.K. Kabanga, S.-J. Kang, Classifying malware using convolutional gated neural network, in: Proceedings of the 20th International Conference on Advanced Communication Technology (ICACT), IEEE, 2018a, pp. 40–44, doi:10.23919/ICACT.2018.8323639.

[91] J.-Y. Kim, S.-J. Bu, S.-B. Cho, Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders, Information Sciences 460 (2018b) 83–102, doi:10.1016/j.ins.2018.04.092.

[92] J.-Y. Kim, S.-B. Cho, Detecting intrusive malware with a hybrid generative deep learning model, in: Proceedings of the 19th International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2018, pp. 499–507, doi:10.1007/978-3-030-03493-1_52.

[93] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang, Riskranker: scalable and accurate zero-day Android malware detection, in: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, ACM, 2012, pp. 281–294, doi:10.1145/2307636.2307663.

[94] M.K. Shankarapani, S. Ramamoorthy, R.S. Movva, S. Mukkamala, Malware detection using assembly and API call sequences, Journal in Computer Virology 7 (2) (2011) 107–119, doi:10.1007/s11416-010-0141-5.

[95] Z. Yuan, Y. Lu, Z. Wang, Y. Xue, Droid-Sec: Deep learning in Android malware detection, in: ACM SIGCOMM Computer Communication Review, 44, ACM, 2014, pp. 371–372, doi:10.1145/2619239.2631434.

[96] Z. Yuan, Y. Lu, Y. Xue, Droiddetector: Android malware characterization and detection using deep learning, Tsinghua Science and Technology 21 (1) (2016) 114–123, doi:10.1109/TST.2016.7399288.

[97] S. Hou, A. Saas, Y. Ye, L. Chen, Droiddelver: An Android malware detection system using deep belief network based on API call blocks, in: Proceedings of the 2016 International Conference on Web-Age Information Management, Springer, 2016, pp. 54–66, doi:10.1007/978-3-319-47121-1_5.

[98] X. Su, D. Zhang, W. Li, K. Zhao, A deep learning approach to Android malware feature learning and detection, in: Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, IEEE, 2016, pp. 244–251, doi:10.1109/TrustCom.2016.0070.

[99] L. Xu, D. Zhang, N. Jayasena, J. Cavazos, HADM: Hybrid analysis for detection of malware, in: Proceedings of SAI Intelligent Systems Conference, Springer, 2016, pp. 702–724, doi:10.1007/978-3-319-56991-8_51.

[100] K. Costa, L. Silva, G. Mrtins, G. Rosa, R. Pires, J. Papa, On the evaluation of restricted boltzman machines for malware identification, International Journal of Information Security Science 5 (3) (2016) 70–81.

[101] S. Hou, A. Saas, L. Chen, Y. Ye, Deep4MalDroid: A deep learning framework for Android malware detection based on Linux kernel system call graphs, in: Proceedings of the 2016 International Conference on Web Intelligence Workshops (WIW), IEEE, 2016, pp. 104–111, doi:10.1109/WIW.2016.040.

[102] N. McLaughlin, J. Martinez del Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupe, et al., Deep Android malware detection, in: Proceedings of the 7th ACM Conference on Data and Application Security and Privacy, ACM, 2017, pp. 301–308, doi:10.1145/3029806.3029823.

[103] R. Nix, J. Zhang, Classification of Android apps and malware using deep neural networks, in: Proceedings of the 2017 International Joint Conference on Neural Networks (IJCNN), IEEE, 2017, pp. 1871–1878, doi:10.1109/IJCNN.2017.7966078.

[104] M. Nauman, T.A. Tanveer, S. Khan, T.A. Syed, Deep neural architectures for large scale Android malware analysis, Cluster Computing 21 (3) (2017) 1–20, doi:10.1007/s10586-017-0944-y.

[105] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, C. Siemens, Drebin: Effective and explainable detection of Android malware in your pocket, in: Proceedings of the 2014 Network and Distributed System Security Symposium (NDSS), 2014, doi:10.14722/ndss.2014.23247.

[106] A. Martín, F. Fuentes-Hurtado, V. Naranjo, D. Camacho, Evolving deep neural networks architectures for Android malware classification, in: Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1659–1666, doi:10.1109/CEC.2017.7969501.

[107] E.B. Karbab, M. Debbabi, A. Derhab, D. Mouheb, Maldozer: Automatic framework for Android malware detection using deep learning, Digital Investigation 24 (2018) S48–S59, doi:10.1016/j.diin.2018.01.007.

[108] S. Jan, T. Ali, A. Alzahrani, S. Musa, Deep convolutional generative adversarial networks for intent-based dynamic behavior capture, International Journal of Engineering & Technology 7 (4.29) (2018) 101–103.

[109] M.A. Salama, H.F. Eid, R.A. Ramadan, A. Darwish, A.E. Hassanien, Hybrid intelligent intrusion detection scheme, in: Soft Computing in Industrial Applications, Springer, 2011, pp. 293–303, doi:10.1007/978-3-642-20505-7_26.

[110] U. Fiore, F. Palmieri, A. Castiglione, A. De Santis, Network anomaly detection with the restricted boltzmann machine, Neurocomputing 122 (2013) 13–23, doi:10.1016/j.neucom.2012.11.050.

[111] N. Gao, L. Gao, Q. Gao, H. Wang, An intrusion detection model based on deep belief networks, in: Proceedings of the 2nd International Conference on Advanced Cloud and Big Data (CBD), IEEE, 2014, pp. 247–252, doi:10.1109/CBD.2014.41.

[112] Y. Li, R. Ma, R. Jiao, A hybrid malicious code detection method based on deep learning, International Journal of Security and Its Applications 9 (5) (2015) 205–216, doi:10.14257/ijsia.2015.9.5.21.

[113] J. Yang, J. Deng, S. Li, Y. Hao, Improved traffic detection with support vector machine based on restricted boltzmann machine, Soft Computing 21 (11) (2017) 3101–3112, doi:10.1007/s00500-015-1994-9.

[114] B. Dong, X. Wang, Comparison deep learning method to traditional methods using for network intrusion detection, in: Proceedings of the 8th International Conference on Communication Software and Networks (ICCSN), IEEE, 2016, pp. 581–585, doi:10.1109/ICCSN.2016.7586590.

[115] Y. Liu, X. Zhang, Intrusion detection based on IDBM, in: Proceedings of the 14th International Conference on Dependable, Autonomic and Secure Computing, 14th International Conference on Pervasive Intelligence and Computing, 2nd International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), IEEE, 2016, pp. 173–177, doi:10.1109/DASC-PICom-DataCom-CyberSciTec.2016.48.

[116] S. Potluri, C. Diedrich, Accelerated deep neural networks for enhanced intrusion detection system, in: Proceedings of the 21st International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2016, pp. 1–8, doi:10.1109/ETFA.2016.7733515.

[117] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, M. Ghogho, Deep learning approach for network intrusion detection in software defined networking, in: Proceedings of the 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), IEEE, 2016, pp. 258–263, doi:10.1109/WINCOM.2016.7777224.

[118] P.H. Duy, N.N. Diep, Intrusion detection using deep neural network, Southeast Asian Journal of Sciences 5 (2) (2017) 111–125.

[119] S. Mohammadi, A. Namadchian, A new deep learning approach for anomaly base IDS using memetic classifier, International Journal of Computers, Communications & Control (IJCCC) 12 (5) (2017) 677–688, doi:10.15837/ijccc.2017.5.2972.

[120] C. Yin, Y. Zhu, J. Fei, X. He, A deep learning approach for intrusion detection using recurrent neural networks, IEEE Access 5 (2017) 21954–21961, doi:10.1109/ACCESS.2017.2762418.

[121] F. Farahnakian, J. Heikkonen, A deep auto-encoder based approach for intrusion detection system, in: Proceedings of the 20th International Conference on Advanced Communication Technology (ICACT), IEEE, 2018, pp. 178–183, doi:10.23919/ICACT.2018.8323687.

[122] C. Yin, Y. Zhu, S. Liu, J. Fei, H. Zhang, An enhancing framework for botnet detection using generative adversarial networks, in: Proceedings of the 2018 International Conference on Artificial Intelligence and Big Data (ICAIBD), IEEE, 2018, pp. 228–234, doi:10.1109/ICAIBD.2018.8396200.

[123] E.B. Beigi, H.H. Jazi, N. Stakhanova, A.A. Ghorbani, Towards effective feature selection in machine learning-based botnet detection approaches, in: Proceedings of the 2014 IEEE Conference on Communications and Network Security (CNS), IEEE, 2014, pp. 247–255, doi:10.1109/CNS.2014.6997492.

[124] N. Zhang, Y. Yuan, Phishing detection using neural network, in: CS229 lecture notes, 2012.

[125] R.M. Mohammad, F. Thabtah, L. McCluskey, Predicting phishing websites based on self-structuring neural network, Neural Computing and Applications 25 (2) (2014) 443–458, doi:10.1007/s00521-013-1490-z.

[126] L.A. da Silva, K.A.P. da Costa, P.B. Ribeiro, G.H. de Rosa, J.P. Papa, Learning spam features using restricted boltzmann machines, International Journal on Computer Science and Information Systems (IADIS) 11 (1) (2016) 99–114.

[127] K. Borgolte, C. Kruegel, G. Vigna, Meerkat: Detecting website defacements through image-based object recognition, in: Proceedings of the 2015 USENIX Security Symposium, 2015, pp. 595–610.

[128] J.-Y. Li, T.W. Chow, Y.-L. Yu, The estimation theory and optimization algorithm for the number of hidden units in the higher-order feedforward neural network, in: Proceedings of the 1995 International Conference on Neural Networks, 3, IEEE, 1995, pp. 1229–1233, doi:10.1109/ICNN.1995.487330.

[129] N. Wanas, G. Auda, M.S. Kamel, F. Karray, On the optimal number of hidden nodes in a neural network, in: Proceedings of the IEEE Canadian Conference

on Electrical and Computer Engineering, 2, 1998, pp. 918–921, doi:10.1109/CCECE.1998.685648.
[130] J. Ke, X. Liu, Empirical analysis of optimal hidden neurons in neural network modeling for stock prediction, in: 2008 Pacific-Asia Workshop on Computational Intelligence and Industrial Application, IEEE, 2008, pp. 828–832, doi:10.1109/PACIIA.2008.363.
[131] S. Xu, L. Chen, A novel approach for determining the optimal number of hidden layer neurons for FNNs and its application in data mining, in: Proceedings of the 5th International Conference on Information Technology and Applications (ICITA), 2008, pp. 683–686.
[132] K. Shibata, Y. Ikeda, Effect of number of hidden neurons on learning in large-scale layered neural networks, in: Proceedings of the 2009 ICCAS-SICE, IEEE, 2009, pp. 5008–5013.
[133] X. Zhang, A Deep Learning based Framework for Detecting and Visualizing Online Malicious Advertisement, University of New Brunswick, 2018 Master's thesis.
[134] S.T. Brugger, J. Chow, An assessment of the DARPA IDS evaluation dataset using snort, UCDAVIS department of Computer Science 1 (2007) 22.
[135] M. Kim, in: Discriminative learning of generative models for sequence classification and motion tracking, 2007.
[136] What is the difference between high-level features and low-level features?, accessed July 2018, (https://www.quora.com/).
[137] A. Rasmus, M. Berglund, M. Honkala, H. Valpola, T. Raiko, Semi-supervised learning with ladder networks, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2015, pp. 3546–3554.
[138] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, S. Bengio, Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research 11 (Feb) (2010) 625–660.
[139] L. Mici, G.I. Parisi, S. Wermter, A self-organizing neural network architecture for learning human-object interactions, Neurocomputing 307 (2018) 14–24, doi:10.1016/j.neucom.2018.04.015.
[140] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F.E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing 234 (2017) 11–26, doi:10.1016/j.neucom.2016.12.038.
[141] K. Yang, J. Liu, C. Zhang, Y. Fang, Adversarial examples against the deep learning based network intrusion detection systems, in: Proceedings of the 2018 Military Communications Conference (MILCOM), IEEE, 2018, pp. 559–564.
[142] K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. McDaniel, Adversarial perturbations against deep neural networks for malware classification, arXiv preprint arXiv:1606.04435 (2016).
[143] X. Yuan, P. He, Q. Zhu, X. Li, in: Adversarial examples: Attacks and defenses for deep learning, IEEE transactions on neural networks and learning systems, 2019, doi:10.1109/TNNLS.2018.2886017.
[144] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, arXiv preprint arXiv:1312.6199 (2013).
[145] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, P. Frossard, Universal adversarial perturbations, in: Proceedings of the 2017 Conference on Computer Vision and Pattern Recognition, IEEE, 2017, pp. 1765–1773, doi:10.1109/CVPR.2017.17.
[146] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, C.-J. Hsieh, Zoo: zeroth order optimization based black-box attacks to deep neural networks without training substitute models, in: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, ACM, 2017, pp. 15–26, doi:10.1145/3128572.3140448.
[147] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv:1412.6572 (2014).
[148] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z.B. Celik, A. Swami, The limitations of deep learning in adversarial settings, in: Proceedings of the 2016 IEEE European Symposium on Security and Privacy (EuroS&P), IEEE, 2016, pp. 372–387, doi:10.1109/EuroSP.2016.36.
[149] A. Rozsa, E.M. Rudd, T.E. Boult, Adversarial diversity and hard positive generation, in: Proceedings of the 2010 Conference on Computer Vision and Pattern Recognition Workshops, IEEE, 2016, pp. 25–32, doi:10.1109/CVPRW.2016.58.
[150] E.L. Denton, S. Chintala, R. Fergus, et al., Deep generative image models using a- laplacian pyramid of adversarial networks, in: Proceedings of the Advances in Neural Information Processing Systems (NIPS), 2015, pp. 1486–1494.
[151] D.J. Im, C.D. Kim, H. Jiang, R. Memisevic, Generating images with recurrent adversarial networks, arXiv preprint arXiv:1602.05110 (2016).
[152] M. Mathieu, C. Couprie, Y. LeCun, Deep multi-scale video prediction beyond mean square error, arXiv preprint arXiv:1511.05440 (2015).
[153] S. Pascual, A. Bonafonte, J. Serra, Segan: speech enhancement generative adversarial network, arXiv preprint arXiv:1703.09452 (2017), doi:10.21437/Interspeech.2017-1428.
[154] Z. Lin, Y. Shi, Z. Xue, IDSGAN: Generative adversarial networks for attack generation against intrusion detection, arXiv preprint arXiv:1809.02077 (2018).
[155] I. Rosenberg, A. Shabtai, Y. Elovici, L. Rokach, in: Query-efficient GAN based black-box attack against sequence based machine and deep learning classifiers, 2018.
[156] M. Rigaki, S. Garcia, Bringing a GAN to a knife-fight: adapting malware communication to avoid detection, in: 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 70–75, doi:10.1109/SPW.2018.00019.
[157] W. Hu, Y. Tan, Generating adversarial malware examples for black-box attacks based on GAN, arXiv preprint arXiv:1702.05983 (2017).
[158] H.S. Anderson, J. Woodbridge, B. Filar, Deepdga: adversarially-tuned domain generation and detection, in: Proceedings of the 2016 ACM Workshop on Artificial Intelligence and Security, ACM, 2016, pp. 13–21, doi:10.1145/2996758.2996767.
[159] V. Behzadan, A. Munir, in: The faults in our π^*s: security issues and open challenges in deep reinforcement learning, 2018.
[160] H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in: Proceedings of the 13th AAAI Conference on Artificial Intelligence, 2, 2016, pp. 2094–2100.
[161] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, arXiv preprint arXiv:1312.5602 (2013).
[162] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529.
[163] R. Blanco, J.J. Cilla, S. Briongos, P. Malagón, J.M. Moya, Applying cost-sensitive classifiers with reinforcement learning to IDS, in: Proceedings of the 2018 International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2018, pp. 531–538.
[164] H.S. Anderson, A. Kharkar, B. Filar, D. Evans, P. Roth, Learning to evade static PE machine learning malware models via reinforcement learning, arXiv preprint arXiv:1801.08917 (2018).

Samaneh Mahdavifar received the B.Eng. degree in Computer Engineering-Software from Kharazmi University, Tehran, Iran, in 2008 and the M.Eng. degree in Computer Engineering-Software from Ferdowsi University of Mashhad, Iran, in 2012. She is currently a Cybersecurity Researcher at the Canadian Institute for Cybersecurity (CIC) and a Ph.D. candidate in Computer Science at the University of New Brunswick (UNB), Canada. Her research interests include computer network security, cybersecurity and privacy, malware detection, machine learning, and deep learning.

Ali A. Ghorbani has held a variety of positions in academia for the past 37 years and is currently a Professor of Computer Science, Tier 1 Canada Research Chair in Cybersecurity, the Director of the Canadian Institute for Cybersecurity, which he established in 2016, and an IBM Canada Faculty Fellow. He served as the Dean of the Faculty of Computer Science at the University of New Brunswick from 2008 to 2017. Dr. Ghorbani is also the founding director of the laboratory for intelligence and adaptive systems research. His current research focus is Cybersecurity, Web Intelligence, and Critical Infrastructure Protection. Dr. Ghorbani is the co-inventor on 3 awarded patents in the area of Network Security and Web Intelligence and has published over 270 peer-reviewed articles during his career. He has supervised over 180 research associates, postdoctoral fellows, graduate and undergraduate students during his career. His book, Intrusion Detection and Prevention Systems: Concepts and Techniques, was published by Springer in October 2010. He is the co-founder of the Privacy, Security, Trust (PST) Network in Canada and its international annual conference. Dr. Ghorbani served as the co-Editor-In-Chief of Computational Intelligence: An International Journal from 2007 to 2017. He developed a number of technologies that have been adopted by high-tech companies. He co-founded two startups, Sentrant Security and EyesOver Technologies in 2013 and 2015, respectively. Dr. Ghorbani is the recipient of the 2017 Startup Canada Senior Entrepreneur Award.