Neural Networks and Deep Learning in Cyber Security

Mihnea Horia VREJOIU
National Institute for Research and Development in Informatics - ICI Bucharest
mihnea.vrejoiu@ici.ro

Abstract: In the last years, the deep learning (DL) technology using various deep neural network models / architectures became the state-of-the-art in Machine Learning (ML) and Artificial Intelligence (AI), its applications reaching better performances than humans in more and more domains. While traditional ML techniques were mainly based on certain mandatory initial "hand-crafted" feature extraction and engineering phase, the new DL approach is automatically performing this step of specific feature representations extraction directly from the raw input training samples. This intrinsic ability makes it applicable to various issues that cyber security is currently dealing with, such as: intrusion detection, malware classification and detection, spam and phishing detection and binary analysis. In this paper we are intending a brief overview of artificial neural networks and some examples of deep learning based solutions in cyber security.
Keywords: artificial neural network, deep learning, cyber security, intrusion / malware / spam / phishing detection, traffic analysis, binary analysis.

INTRODUCTION

In nowadays, large scale digitalization and informatics globalization with worldwide Internet connectivity and huge amounts of various data generated, transmitted, stored and retrieved with high pace (Big Data), augmented our reality with new dimensions in almost all the fields and at all levels: governmental, military and security, medical, financial and economic, social, cultural, educational etc. The number of individuals accessing the information online is increasing daily, but also the cyber threats are inherently increasing. Cyber events are inevitable and their impact could be more disastrous than one could even imagine.

Cybersecurity refers to data and devices protection against various cyber threats. It comprises a set of methods, technologies, and processes designed to prevent, avoid or at least minimize the risks and damages that may arise from possible attacks against informatics systems and networks, unauthorized access, data theft, modification or destruction. Cyber security involves both network security and host security systems, every of those having today a firewall, and/or antivirus (AV) software, providing an intrusion detection system (IDS). A host-based IDS (HIDS) is using the host system event log for watching over the system operation and states to detect unauthorized installation or access, while also checking the state of RAM and file system whether some expected data exist there or not [1]. A network-based IDS (NIDS) is placed on "demilitarized zone" (DMZ) [2] at the edge of the network. It analyses network traffic in real-time for detecting unauthorized intrusions or malicious attacks. There are two types of detection techniques: anomaly detection, that catches attacks by comparing behaviors for identifying abnormal vs. normal ones, and misuse (or signature-based) detection, that detects the attacks based on previously known knowledge [3].

Cyber threats represent one of the greatest and permanent dangers for global economies today, being expected that the total costs of the cyber crime damages will reach almost one trillion dollars this year [4]. On the other hand, in the last years, the deep learning (DL) technology based on learning data representations (as opposed to task-specific algorithms) using various deep neural network (DNN) types / models / architectures became the state-of-the-art in Machine Learning (ML) and Artificial Intelligence (AI), its applications exceeding human performance in more and more domains. In these circumstances, the cyber security industry is currently investing heavily in ML in hope of providing a more dynamic deterrent to cyber crime. ABI Research forecasted that ML in cyber security will boost big data, intelligence, and analytics spending up to 96 billion dollars by 2021 [4]. It is considered that "this AI security revolution will drive ML solutions to soon emerge as the new norm beyond Security Information and Event Management (SIEM), and ultimately displace a large portion of traditional AV, heuristics, and signature-based systems within the next five years". User and Entity Behavioral Analytics (UEBA) along with DL algorithms design are emerging as the two most prominent technologies in cyber security. More and more feature-agnostic models, deep learning, and natural language processing will be employed as a response to the increasingly menacing nature of unknown threats and multiplicity of threat agents.

This paper presents a brief overview of artificial neural networks (ANN) and of some examples for possibilities of using DL techniques in cyber security applications, while also highlighting the advantages of DL algorithms in classifying and correlating malicious activities from various sources. Unlike other ML approaches, DL algorithms don't need any previous "hand-crafted" feature extraction / engineering, and also provide feature visualizations. They are automatically extracting best options by themselves [5]. Data to be analyzed in cyber security is mostly in the form of strings / texts, as well as binary executable code. To convert strings / texts to real valued vectors, various preprocessing and analysis techniques are employed along with deep learning.

The rest of this paper is organized as follows. Section 2 provides a brief historical overview of various models of artificial neural networks (ANN) as base architectures for deep learning. In Section 3, some DL based solutions for several cyber security use cases are presented. Finally, Section 4 is gathering a few conclusions.

ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING

An artificial neuron is modeling in a simplified way, through a mathematical function, a real neuron from a biological neural network such as the brain. The artificial neuron represents the elementary unit of an artificial neural network.

Like the biological neuron, which has dendrites and an axon, the artificial neuron has a simple

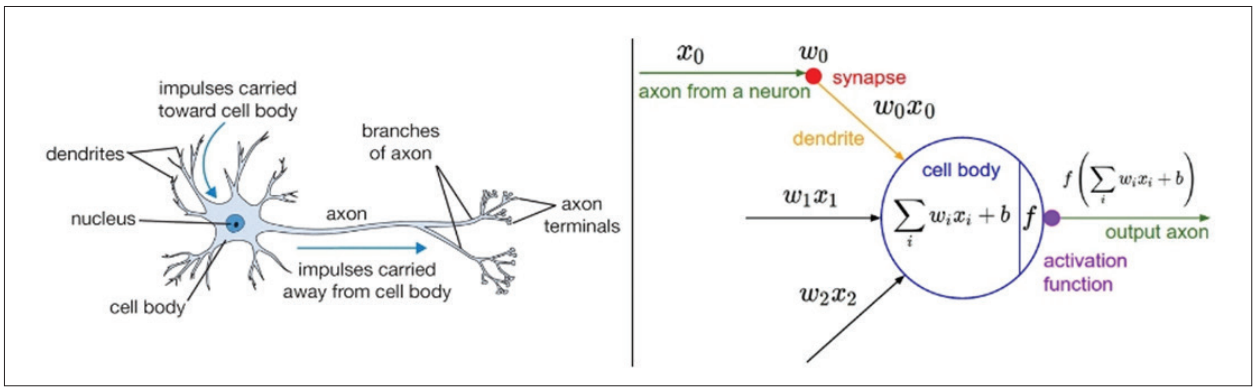

***Figure 1**. Artificial neuron is modeling in a simplified manner the biological neuron. (Source: Karpathy, 2015 [6])*

tree structure with input nodes and an output node connected to all these ones. The artificial neuron gets several input values analogues to the post-synaptic excitation/inhibiting potentials applied to the dendrites of the biological neuron. These input values are summed producing an output activation value, analogue to the action potential transmitted along the axon in the case of the biological neuron. Each input value x_i is individually weighted in the sum (with a weight w_i) and the result is passed to the output through a non-linear function (f) called activation (or transfer) function.

The simplest artificial neural network is the "***perceptron***", composed by a single artificial neuron. It was introduced by Frank Rosenblatt (1958) [7], based on research of Warren McCulloch and Walter Pitts (1943) [8]. The perceptron computes the weighted sum of its n input values x_i:

$$y = w \cdot x + b = \Sigma w_i \cdot x_i + b; \quad i = 1 \div n \tag{1}$$

and then applies a non-linear function (i.e. *signum* function) to the resulted value:

$$f(y) = |y| \,/\, y = \begin{cases} -1, & \text{for } y < 0; \\ 0, & \text{for } y = 0; \\ +1, & \text{for } y > 0. \end{cases} \tag{2}$$

The bias b (offset), represents the activation threshold value of the perceptron and may be considered as a $(n+1)$ supplementary input, with value $x_0 = 1$ and weight $w_0 = b$, both constants, allowing to shift the activation function to the left or right as necessary along the training process.

The perceptron is a **linear classifier** that can perform binary classification (in two classes, e.g. coded by numerical values, e.g. 0 and 1) in the case of linearly separable data applied at input, by separating the output values greater or lesser than a certain threshold. These later one together with the weights must be empirically set up in a configuration phase for each classification problem in the absence of a training algorithm. Rosenblatt, 1962 [9] proposed an iterative training algorithm for setting up the weights of the perceptron. The algorithm starts with a set of randomly chosen, non zero, small weights, and compares at each iteration the output obtained for each input vector with the real/correct class (code) to which the respective input vector belongs, adjusting the weights thus that to eliminate any classification error. If at the end of an iteration (after a number of steps equal to the number of training input vectors) no weight adjustment has been made, it means that all the respective training vectors have been correctly classified. The obtained weights vector represents a solution of the respective training problem and the algorithm stops. Otherwise, a new iteration starts. In the case that within an empirically preset maximum number of iterations n_{max} not all the training vectors succeeded to be correctly classified, it means that the two classes are not linearly separable and the algorithm is stopped, the problem couldn't being solved with the perceptron. Rosenblatt, 1962 [9] also formulated and demonstrated the „***perceptron convergence theorem***": For a training set composed by two subsets of vectors corresponding to two linearly separable classes, the training algorithm will converge after a finite number of iterations n, resulting a weights vector $w(n) = w(n+k)$, $\forall k>0$, as (not necessarily unique) solution of the training problem.

The perceptron has multiple limitations, among which the most known is that it can't model the binary logical function XOR, whose values 0 and 1 for all possible combinations of the binary input variables are not linearly separable. Such limitations were overpassed once the **multilayer perceptron** (MLP) model was developed. The MLP is structured as a sequence of connected layers of perceptrons, with an input layer that receives an input vector of values, an output layer that makes a decision or prediction on the input, and - in between these two - one or several hidden layers, whose component perceptrons are effectively performing the MLP computations. Layers are completely connected, each perceptron in a layer being connected to all the perceptrons of the previous thus **fully connected** (FC) layer. Each connection has its own weight. The output of each layer represents the input for the next layer in the sequence.

In the case of the MLP architectures with two or more hidden layers one can already speak

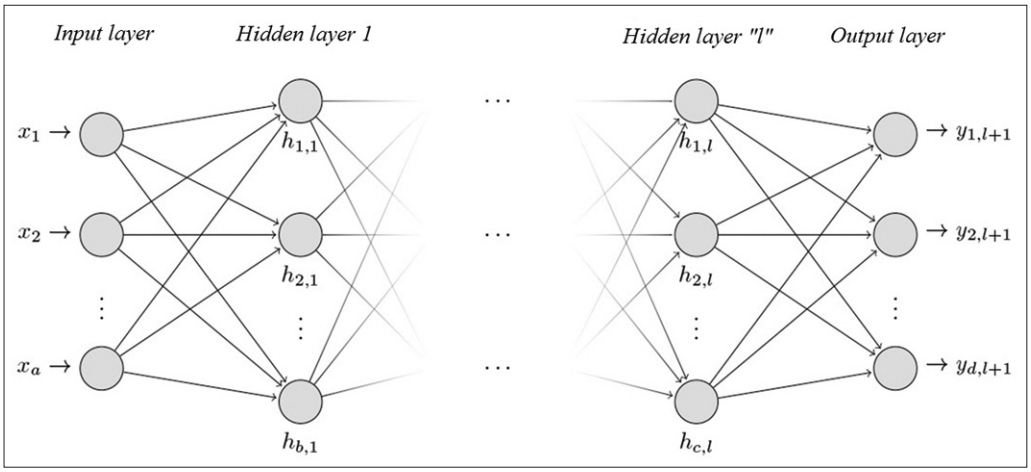

Figure 2. *MLP / DNN fully connected feedforward general architecture.*

about a "**deep neural network**" (DNN), while for only one hidden layer, usually about a "**shallow neural network**" (which is often used mainly for the single perceptron).

The MLP is a "**feedforward**" network, since its connections are followed from input to output, layer by layer, exclusively forward, and the output values resulted in a higher level don't affect in any way the output values of previous layers. While in the case of single perceptron the input is used to immediately compute the output, in the case of feedforward MLP the data is sequentially processed for each layer, one by one.

Also, MLP uses another kind of activation function, derivable (whose result doesn't jump instantly from negative to positive values, but has a smooth continuous passage), that allow training. Usually, sigmoid functions are employed, such as logistic function:

$$f(x) = 1/(1+e^{-x}), \quad (3)$$

or hyperbolic tangent:

$$f(x)=tanh(x)=(e^{x}-e^{-x})/(e^{x}+e^{-x}). \quad (4)$$

The MLP feedforward network can classify input data in a preset number of classes equal to the dimension of the output. The input vectors must contain specific values coding the relevant features for identifying and discriminating among data classes. It is therefore necessary that those features be previously extracted (identified) for each category of classification problem. Feature extraction is usually a nontrivial task, requiring domain knowledge and expertise.

The values of all the weights for all the connections are set up by supervised training using a set of labeled input vectors (belonging to all possible classes, identified by the respective label value each). The most used supervised training method is "**backpropagation**", and was initially proposed by Paul Werbos, 1974 [10]. The training algorithm is based on the propagation of the classification error backwards from final layer to previous ones, layer by layer, and is adjusting the weights of the connections for minimizing that error. David Rumelhart et al., 1986 [11] „rediscovered" and successfully applied the backpropagation algorithm for training multi-layer networks through parallel distributed processing. The method consists in a continuous iterative process of supervised learning by adjusting and fine-tuning network structure (i.e. the values of the weights of its connections, initialized first with random small values) at each iteration step. It's an iterative **optimization process** that is reducing iteration by iteration the error (i.e. the difference between the expected and obtained output, estimated using a **cost / loss function**). The most used optimization algorithm for weights adjustment is "**gradient descent**", with its variants, based on which it is decided at each iteration which weights are to be adjusted by following the descent gradient of the error (as a function of weights). By measure that the error is reduced, iterations become more refined. The process could take thousands of iterations until the computed output closes enough to the expected one, moment when it may be considered that the network is completely trained.

A trained MLP establishes (through the values of the weights) links between the training input vectors and the output values corresponding to these ones, somehow like when defining a function through a table of its values. In this context, it was formulated and demonstrated the "**universal approximation theorem**" (George Cybenko, 1989 [12]; Kurt Hornik, 1991 [13]), stating that a feedforward MLP network with (only) one hidden layer and a finite number of neurons can model (as polynomial approximation) any function that is continuous on compact subsets of Rn of any complexity with enough precision, provided that there are enough neurons in the hidden layer and appropriate activation functions are used.

Even if with the supervised training algorithm using backpropagation and gradient descent, MLP networks meant a big step forward, in the case of more hidden layers, they still have serious limitations. These are due to the sigmoid activation function, which leads to a quick saturation at multiple derivations from layer to layer backwards, the so-called "**vanishing gradient**", thus that weights of initial layers couldn't be correctly set up. In 2006 Geoffrey Hinton [14],[15] proposes some revolutionary ideas of initializing the parameters of the neural network with values closer to the optimal ones by using **Restricted Boltzmann Machine** (RBM) units / **Deep Belief Network** (DBN). The method consist in an initial unsupervised pre-training of the RBM network layer by layer thus that the weights get to model the intimate structure of the training input data. Those weights are then fine-tuned using supervised backpropagation, thus eliminating the vanishing gradient problem. In 2007, Yoshua Bengio [16] introduced **autoencoders** (AE) instead of RBM for the unsupervised pre-training of the network, which have been further developed with some variation (denoising AE, sparse AE etc.). In 2010, James Martens [17] presented another algorithm for setting up the parameters using **second order derivates** without any prior unsupervised pre-training, which got even better results. Moreover, the same year, by directly using the classical backpropagation algorithm with a **deep and wide network**, on GPUs, by using for training slightly elastically deformed patches from images, without any other helping algorithm for initializing the weights, Dan Cireșan [18] obtained 0.35% error rate on the MNIST image set. The same group, this time using **convolutional neural network** (CNN) and **max pooling** [19] without any other prior pre-training, established in 2011 the record on MNIST, of 0.23% error rate, a better than human performance.

Thus, in the last years artificial neural networks had known an impressive development, sustained (also) by the technological evolution in what concerns the storage and computing capacities and speed using **GPU**s, and the data acquisition devices, as well as by the large amounts of data of all kinds that became available (**Big Data**), able to provide enough information about the complexity and variety of the real world. New deep neural network (DNN) models and architectures have been developed, either inspired from neurosciences, either based on computational engineering reasons, deep learning technology becoming today almost synonym with ML and AI.

Convolutional neural networks (CNNs or ConvNets) are inspired by the mechanisms within the visual cortex of the brain (Hubel & Wiesel, 1959 [20]). The neurons from a convolutional layer are connected not with all the neurons from the previous layer as in the MLP, but rather with only a few of them, located in a small vicinity (**receptive field**). This way they are able to become sensitive to certain local features, also providing invariance to shifting position of those features. When stacked together, higher convolutional layers integrate simple features from previous layer(s) into more complex ones. The output of each convolutional layer provides a **feature map** (activation map). The weights of each neuron's input connections in a (slice of a) convolutional layer are the same (shared), defining the specific "**filter**" for a certain feature in whatever position in the previous layer's feature map. In fact, a convolutional layer is composed of more than one "slices", each corresponding to a certain feature (filter). A first model of convolutional networks was the

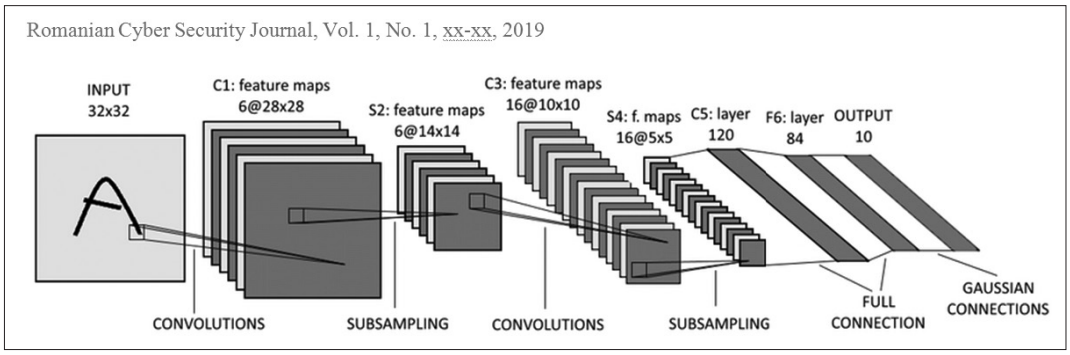

***Figure 3**. LeNet-5 architecture. (LeCun, 1998 [24]*

Neocognitron proposed by Kunihiko Fukushima, 1980 [21], (after he introduced the Cognitron [22] in 1975). The first successful application of CNN dated from the 90's and is represented by **LeNet-5** architecture (Yann LeCun, 1989; 1998 [23],[24]) that was largely employed for recognizing hand-written digits on checks and other paper forms in the banking environment.

In general, a CNN is composed by a sequence of two parts. The first one alternates several (or more) convolutional layers having local connectivity (receptive fields) with some down-sampling ones (**max pooling** / **average pooling**) that reduce the spatial dimensions, also having local connectivity. This initial convolutional part is performing **automatic feature extraction** from the input data. The most used activation function for the convolutional layers is the **Rectified Linear Unit** - ReLU (Hahnloser, 2000; 2001 [25],[26]; Nair, 2010 [27]; Glorot, 2011 [27]):

$$f(x)=max(0,x). \quad (5)$$

The final part of such a deep neural network consists in one or more fully connected layers (a MLP classifier), which perform(s) the classification using as input the output of the initial convolutional part (i.e. a **feature vector** automatically generated by that one).

Significant improvements and innovations on CNN models/architectures and training methods emerged in the context of the ImageNet Large Scale Visual Recognition Challenge - ILSVRC annual competitions (2010-2017) based on the ImageNet image set [28], including:

• ***AlexNet*** [29] (2012), with 8 layers which provided a 15,4% error rate in image classification;

• ***ZFNet*** [30] (2013), also with 8 layers but with 11,2% error rate;

• ***GoogLeNet/Inception-v1*** [32] and ***VGGNet*** [32] (2014), with 22, respectively 16-19 layers and 6,7%, respectively 7,3% error rate.

• ***ResNet*** [33] (2015) with 152 layers and 3,57% error rate (better than human performance of about 5%).

• ***DenseNet*** [34] (2016) and ***SENet*** [35] (2017), with over 200 layers and 2,99%, respectively 2,251% error rate.

CNNs are widely used today mainly in the Computer Vision (CV) field for image classification, image segmentation, object/face/pattern recognition, scene labeling, human pose recognition, action recognition or document analysis, as well as in the Natural Language Processing (NLP) field, for speech recognition or text classification.

The number of training samples is very important in the case of deep neural networks (DNN). It is considered that a supervised DL algorithm will perform acceptably if trained with about 5,000 examples per class, and will surpass human performance if the training set contains at least 10 mil. labeled examples (Ian Goodfellow et al., 2016 [36]). For having similar success with smaller labeled training sets, researches are done today for valorizing huge amounts of unlabeled data with unsupervised or semi-supervised training methods.

Other state-of-the-art DNN models / architectures successfully employed today include: Autoencoders (AE), Recurrent Neural Networks (RNN), and Long Short Term Memory (LSTM).

An **AutoEncoder** (AE) is a type of artificial neural network used to learn efficiently compressed / coded data representations in an unsupervised manner for dimensionality reduction, by training the network to ignore signal "noise". A reconstructing part is simultaneously trained

for generating / reconstructing from the reduced encoding (named code or latent representation) an output as close as possible to its original input. In its simplest form, an AE architecture is a feedforward neural network similar to the multi-layer perceptron (MLP), having an input layer, an output layer and one or more hidden layers connecting them, with input and output layers having the same dimension, and hidden code layer (but not necessarily all other hidden layers) of lower dimension.

If trained with normal / regular good data, AE may be used for anomaly detection in further data, since anomalies in input will produce significantly different output than respective input. An autoencoder is a dense neural network and each variable influences the output of the other variables, therefore in case of some abnormal values at input it will produce a large reconstruction error thus capturing irregularities. On the other hand, the encoding part of a trained autoencoder is actually performing automatic feature extraction and the latent representation (as a feature vector) may be further used as input by a classifier if label for the corresponding AE input is available. There are several variations of autoencoders: Sparse, Denoising, Variational, Contractive, Generative AE.

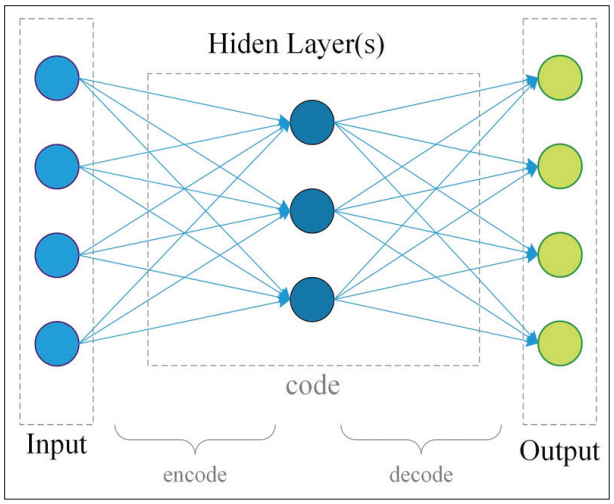

Figure 4. *General architecture of an autoencoder network (Source: [37])*

Recurrent neural networks (RNNs) are a class of artificial neural network where connections between nodes form a directed graph along a temporal sequence, which allows them to exhibit temporal dynamic behavior. Unlike feedforward neural networks, RNNs have cyclic connections and are using their internal state (memory), which make them powerful for modeling input sequences (of various length). This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. For learning variable-length input sequences, **back propagation through time** (BPTT) [38] is used. In BPTT, the model is first trained with the training data, then the output error gradient is saved for each time step.

A RNN uses an input vector sequence $X=(x_1, x_2, ..., x_T)$ and a hidden vector sequence $H=(h_1, h_2, ..., h_T)$ to produce an output vector sequence $Y=(y_1, y_2, ..., y_T)$. A traditional RNN calculates the hidden and output vectors sequences as follows:

$$h_t = \sigma(W_x \cdot h_{xt} + W_{hh} \cdot h_{t-1} + b_h), \quad (6)$$

$$y_t = W_{hy} \cdot h_t + b_y, \quad (7)$$

where $t=1 \div T$, σ is a nonlinearity function, W is a weight matrix, and b is a bias. A finite impulse recurrent network is a directed acyclic graph that can be unrolled / unfolded and replaced with a feedforward neural network (while an infinite impulse recurrent network is a directed cyclic graph that can not be unrolled).

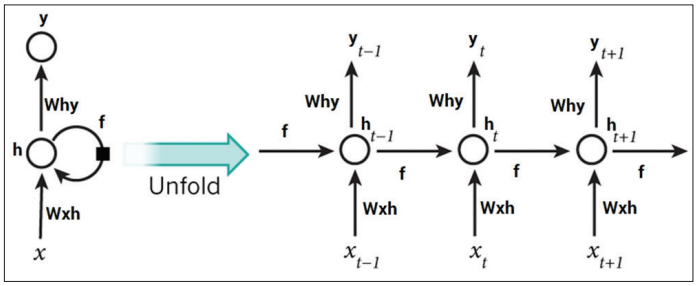

Figure 5. *Unfolding a recurrent neural network (Source: [39])*

RNNs raises vanishing gradient problems in handling long term dependencies, and are very difficult to train when the number of parameters is extremely large. The unfolded network becomes huge, posing convergence problems.

These issues can be avoided by RNNs with additional stored state, and storage under direct control by the neural network. The storage can

also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as **gated state** or **gated memory**, and used by Long Short-Term Memory networks (LSTMs) and Gated Recurrent units (GRUs).

Long Short-Term Memory networks (LSTMs), introduced by Hochreiter & Schmidhuber [40], are a special kind of RNNs, which provide the capability to learn long-term dependencies in sequences of data. In LSTM-RNNs, the repeating module in the chain like structure has a slightly different structure. Instead of having a single neural network layer, there are multiple layers, interacting in a very special way. A LSTM cell has an input gate, a forget gate and an output gate. Equations to compute the values of the three gates and cell state:

$$i_t=\sigma(W_{xi}\cdot x_t+W_{hi}\cdot h_{t-1}+W_{ci}\cdot c_{t-1}+b_i), \quad (8)$$

$$f_t=\sigma(W_{xf}\cdot x_t+W_{hf}\cdot h_{t-1}+W_{cf}\cdot c_{t-1}+b_f), \quad (9)$$

$$c_t=f_t\cdot c_{t-1}+i_c\cdot tanh(W_{xc}\cdot x_t+W_{hc}\cdot h_{t-1}+b_c), \quad (10)$$

$$o_t=\sigma(W_{xo}\cdot x_t+W_{ho}\cdot h_{t-1}+W_{co}\cdot c_t+b_o), \quad (11)$$

$$h_t=o_t\cdot tanh(c_t), \quad (12)$$

where: σ is the logistic sigmoid function, and i, f, o and c are respectively the input gate, forget gate, output gate and cell state, and W_{ci}, W_{cf} and W_{co} are denoted weight matrices for **peephole connections**. In LSTM, three gates (i, f, o) control the information flow. The input gate decides the ratio of input. When calculating the cell state, this ratio has effect on the equation (10). The forget gate passes the previous memory h_{t-1} or not. The ratio of the previous memory is calculated in the equation (9) and used for the equation (10). The output gate determines whether passing the output of memory cell or not. The equation (12) shows this process. LSTMs can solve the vanishing and exploding gradient problems due to the three gates. In LSTM-RNN architecture, the recurrent hidden layer is replaced by LSTM cell.

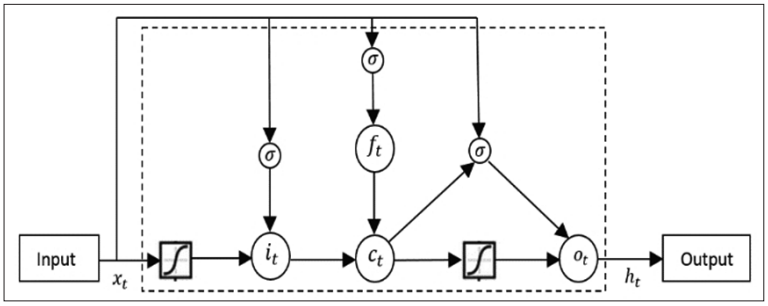

***Figure 6**. A LSTM network cell. (Source: [41])*

LSTMs proved to work extremely well on a large variety of problems, being now widely used in speech recognition, text-to-speech synthesis, and automatic image captioning.

Another efficient RNN architectures are the **Gated Recurrent Units** (GRUs), a variant of LSTMs, but simpler in their structure and easier to train. Their success is primarily due to the gating network signals that control how the present input and previous memory are used to update the current activation and produce the current state. There are (only) two gates: reset and update. These gates have their own sets of weights that are adaptively updated in the learning phase.

APPLICATIONS IN CYBER SECURITY

REFERENCE DATASETS

No deep learning algorithm can be thought of without having a comprehensive, problem specific labeled dataset available for supervised or semi-supervised learning, validation and test of the models. There are several publicly available datasets used as benchmarks for developing and testing systems for various cyber security use cases, but most of them are quite old, and each of those has own limitations. Verma, 2018 [42] presented a brief study over the needs of Security domain to overcome such issues, and datasets and key features of data science for problems of cyber security are discussed. A GitHub repository with a list of cyber security datasets links, among other interesting stuff, may be found at [43]. Also, in his quite comprehensive repository [44], Mike Sconzo attempts to keep "a somewhat curated list of Security related data". Links to various datasets (among other own and 3rd parties stuff) may be found there, organized on topics like: Network, Malware, System, File, Password, Threat Feeds, Other.

Some example datasets among the

most employed ones include by example:

• The KDD Cup 1999 Data [45],[46], which is the dataset used for the 3rd International Knowledge Discovery and Data Mining Tools Competition, held in conjunction with the 5th International Conference on Knowledge Discovery and Data Mining – KDD-99. The competition task was to build a network intrusion detector (NID), a predictive model capable of distinguishing between "bad" connections, called intrusions or attacks, and "good" normal connections. The KDD Cup '99 database contains a standard set of data to be audited, which includes a wide variety of intrusions simulated in a military network environment.

• The NSL-KDD dataset [47] which is an improved version of the KDDcup99 one, eliminating this one's main deficiencies due to the huge number of redundant records, which causes the learning algorithms to be biased towards the frequent records, and thus prevent them from learning unfrequent records which are usually more harmful to networks, such as U2R and R2L attacks. In addition, the existence of these repeated records in the test set caused the evaluation results to be biased by the methods which have better detection rates on the frequent records.

• The ADFA Intrusion Detection Datasets (2013) [48],[49],[50], which cover both Linux and Windows, and were designed at Australian Defense Force Academy for evaluation by system call based HIDS.

In recent days, for boosting up the systems performance and for offering a standard benchmark for their evaluation, some "shared task" is organized as part of the conferences and symposiums. In such shared task, initially a train dataset is distributed to the participants and their learning models are evaluated on a test dataset. Recently, IWSPA 2018 [51] organized a shared task on identifying phishing email, details of the submitted runs being available in [52], and a shared task on detecting malicious domain (DMD 2018) [53], organized as part of the 6th International Symposium on Security in Computing and Communications (SSCC 2018) and 7th International Conference on Advances in Computing, Communications and Informatics (ICACCI 2018) [54]. These two shared tasks allowed participants to present their approaches through working notes or systems description papers. Yearly there is also a shared task conducted by the Cybersecurity Data Mining Competition (CDMC), providing also an option to submit system description papers starting with CDMC 2018 [55].

INTRUSION DETECTION

An intrusion detection system (IDS) is a device or software application that monitors a network or system for malicious activity or policy violations, reporting those either to an administrator or to a security information and event management (SIEM) system able to combine multiple sources and filter false alarms. There are network intrusion detection systems (NIDS) and host-based intrusion detection systems (HIDS). IDS are either using misuse detection (signature-based), recognizing bad patterns, such as malware, or anomaly-based detection, detecting deviations from a pattern of normal/good traffic, which often relies on machine learning. Some IDS products have the ability to also respond to detected intrusions, being referred as **intrusion prevention systems** (IPS).

Staudemeyer and Omlin, 2013 [56] evaluated the performance of LSTM-RNN on classifying intrusion detection data, LSTM networks being able to model data as time series. A processed version of the KDD Cup '99 dataset was used for training and test. Some suitable performance measures are used. LSTM network structure and parameters are experimentally set up. Results showed that LSTM is able to learn all attack classes hidden in the training data.

Javaid et al., 2015 [57] proposed a deep learning based approach to implement an effective and flexible NIDS using self-taught learning (STL) on the NSL-KDD dataset, showing that such NIDS are promising in detecting unknown network intrusions.

J. Kim et al., 2016 [41] implemented an IDS model with deep learning approach, using a classifier based on LSTM-RNN, and evaluated the model.

A training dataset was generated by extracting instances from KDD Cup '99 dataset. Learning rate and hidden layer size were experimentaly chosen. 10 test datasets were used for measuring the performance. The LSTM-RNN classifier showed higher detection ratio than other classifiers. Still, false alarm ratio was slightly greater.

G. Kim et al., 2016 [58] proposed a system-call language modeling approach for designing anomaly-based HIDS. An ensemble method blending multiple threshold based classifiers into a single one, making it possible to accumulate "highly normal" sequences was employed, which significantly reduces false-alarm rates common to conventional methods. System calls represent low-level interactions between programs and the OS kernel and are easy to collect in a large quantity in real-time. Many researchers consider system-call traces as the most accurate source useful for detecting intrusion in anomaly-based HIDS. The proposed system-call language model can learn the semantic meaning and interactions of each system call. The method consists of two parts: the front-end is for language modeling of system calls in various settings, and the back-end is for anomaly prediction based on an ensemble of threshold based classifiers derived from the front-end. The front-end uses LSTM on hidden layer, while at the output layer, a *softmax* activation function is used to estimate normalized probability values of possible calls coming next in the sequence. The sequence representation learned from the final state vector of the LSTM layer after feeding all the sequences of calls is used by the back-end part. For comparison, two baseline classifiers commonly used for anomaly detection are used: k-nearest neighbor (kNN) and k-means clustering (kMC). Through diverse experiments on public benchmark datasets - mainly on ADFA-LD dataset - the validity and effectiveness of the proposed method is demonstrated.

Vinayakumar et al., 2017 [59] evaluated the effectiveness of various traditional ML methods (logistic regression, naive Bayes, k-nearest neighbor, decision tree, AdaBoost, random forest, support vector machine) and DNNs (MLP, DBN) in NIDS. For training and evaluation, the KDD Cup '99 and NSL-KDD datasets were used, in both binary and multiple-classes classification settings. DNNs performed better in most of the cases. That was mainly due to their capability to pass information through several layers for hierarchically learning the underlying hidden patterns of normal and attack network connection records, and to finally aggregate the learned features of each layer together to effectively distinguish further between such records.

Also Vinayakumar et al., 2017 [60] modeled network traffic as time-series, particularly TCP/IP packets in a predefined time range, with supervised learning methods on various deep architectures such as MLP, CNN, CNN- RNN, CNN-LSTM and CNN-GRU, using millions of known good and bad network connections. For evaluating these approaches, the KDD Cup '99 dataset was used. Comprehensive analysis of all the architectures with their topologies, network parameters and network structures was done to select the optimal ones. Experiments showed that the used various CNN architectures performed better than traditional ML classifiers. This was mainly due to the CNN capability to extract high level feature representations that represent the abstract form of low level feature sets of network traffic connections.

Leila Mohammadpour et al., 2018 [61] proposed a deep learning method to implement an effective and flexible NIDS using CNN and DL for binary classification (normal vs. abnormal cases) on the NSL-KDD benchmark dataset. Experimental result of 99.79% detection rate on the test dataset showed that CNNs can be successfully applied as learning method for NIDS.

Shone et al., 2018 [62] presented a deep learning technique for NIDS, using non-symmetric deep autoencoder (NDAE) for unsupervised feature learning. They also proposed a novel deep learning classification model on stacked NDAEs using GPU, evaluated using the benchmark KDD Cup '99 and NSL-KDD datasets. Promising results have been obtained, demonstrating improvements over existing approaches and strong potential for use in modern NIDS.

Application of DNNs is analyzed also by Rahul Vigneswaran et al., 2018 [63]. DNNs were utilized to predict attacks on NIDS using KDD Cup '99

dataset for training and benchmarking the network. For comparison purposes, training is done with several other classical ML algorithms and DNNs with 1 to 5 hidden layers on the same dataset. A DNN with 3 layers showed superior performance over all the other classical ML algorithms.

MALWARE DETECTION

Malware (from malicious software) refers to any software intentionally designed to cause damage to a computer (standalone or in a network) once somehow introduced into this one. It can be in the form of executable code, scripts, or other active content, and is often referred as computer viruses, worms, Trojan horses, ransomware, spyware, adware etc. Malware is acting against the interest of the computer user. It may disrupt data files in the system reducing performance and increasing vulnerability. In some cases it will lead to total corruption of the host system. Malwares are easily passed through various environments using unauthorized software tools. One strategy for protecting against malware is to prevent the malware from gaining access to the target computer. Antivirus software and firewalls are protecting against the introduction of malware in addition to checking for the presence of malware and malicious activity and recovering from attacks.

Dahl et al., 2013 [64] presented a large-scale malware classification system which utilizes random projections to reduce the input space. Neural networks trained on random projections provided a 43% reduction in the error rate compared to the baseline logistic regression system using all the features. The obtained 0.49% two-class error rate for the one-layer neural network with random projections and 0.42% two classes error rate for the ensemble of neural networks offered state-of-the-art performance. GPUs were employed for training with 2.6 million examples in less than three hours. No accuracy gain was obtained by adding additional hidden layers, two and three hidden layers models even performed slightly worse.

Yuancheng Li et al., 2015 [65] proposed a hybrid malicious code detection scheme based on deep learning: first, reducing dimensionality of the data with an AutoEncoder by using AE's ability to abstract the main characteristics (features) of the input data; then, based on these, setting a DBN as the classifier for several times deep learnings; finaly, improving the detection accuracy and reducing the time complexity of the hybrid model. Experiments employed the KDD cup '99 dataset. Results showed that compared with the detection method using single DBN, the proposed method improves detection accuracy while also reducing the time complexity of the model. However, proposed method still needs further improvements for better performance.

Saxe and Berlin, 2015 [66] introduced a malware detector based on DNN using static features that obtains a usable detection rate at an extremely low false positive rate and scales to real world training example volumes on un-expensive hardware. Its performace approach traditional labor-intensive signature based methods, while also detecting previously unseen malware. These were achieved by directly learning on all binaries, without any filtering, unpacking, or manually separating binary files into categories. The full classification framework consists of three main components. The first component extracts four different types of complementary features from the static benign and malicious binaries (Byte/ Entropy Histogram Features, PE Import Features, String 2D histogram features; PE Metadata Features). The second component is a DNN classifier, which consists of an input layer, two hidden layers with parametric ReLU activation function, and an output layer with sigmoid activation function for prediction. The final component is a score calibrator based on a Bayesian calibration model, which translates the output of the neural network to a score that can be realistically interpreted as approximating the probability that the file is actually malware. A brief discussion is made to show how to prevent overfitting by dropout, and how PReLU and weight initialization with backpropagation method helps in speeding up the learning process over the network.

Pascanu et al., 2015 [67] proposed an approach similar to natural language modeling, that learns the language of malware spoken through the executed instructions and extracts robust, time

domain features through a random temporal projection technique. Echo state networks (ESNs) and RNNs are used for the projection stage that extracts features. These models are trained in an unsupervised manner. A standard classifier uses these features to detect malicious files. A few variants of ESNs and RNNs were experimented for the projection stage, including max-pooling and half-frame models, as well as logistic regression for final classification. The best performing hybrid model improved the true positive rate by 98.3% compared to the standard tri-gram of events model, at a false positive rate of 0.1%.

Huang and Stokes, 2016 [68] presented a multi-task deep learning architecture for malware classification for binary malware classification task (malware vs. benign). All models are trained with data extracted from dynamic analysis of malicious and benign files. They found improvements using multiple layers in a DNN architecture for malware classification. The system was trained on 4.5 mil. files and tested on a holdout test set of 2 mil. files. Objective functions for the binary classification task and malware family classification task are combined in the multi-task architecture. They also proposed a non-multi-task malware family classification architecture.

Rahul et al. [69], 2017 applied deep learning techniques to classification of network protocols and applications using flow features and data signatures. They also presented a similar classification of malware using their binary files. CNN with ReLU and dropout, and AE were employed with own dataset for traffic identification, and Microsoft Kaggle dataset for malware classification tasks. Deep learned features in both cases are not handcrafted but are automatically learned form data signatures. They can't be understood by an attacker or malware, therefore can't be easily bypassed.

Vinayakumar et al., 2017 [70], evaluated shallow and deep networks for the detection and classification of ransomware using API calls made by executables. For selecting best MLP architecture, various experiments related to network parameters and structures were done. Results obtained on their dataset for binary classification of executables (as either benign or ransomware) attained highest accuracy 1.0, while for multiple classification of ransomware in categories highest accuracy obtained was 0.98. MLP performed better than other classical ML classifiers in detecting and classifying ransomwares.

Maniath et al., 2017 [71] applied deep learning with LSTM networks for binary sequence classification of API calls. An automated approach to extract API calls from the log of modified sandbox environment and detect ransomware behavior was presented. This was expected to improve the automated analysis of large volume of malware samples.

As Android devices became very spreaded in day by day use, malware detection on Android platform gets nowadays of big interest. Deep learning along with NLP appear very appropriate for such tasks.

Zhenlong Yuan et al., 2016 [72] proposed to associate the features from static analysis with features from dynamic analysis of Android apps and characterize malware using DL techniques with DBN. They implemented an online DL-based Android malware detection engine (DroidDetector) that can automatically detect whether an app is a malware or not. Also, performed an indepth analysis on the features that deep learning essentially exploits to characterize malware. The results showed that DL is suitable for characterizing Android malware and especially effective with the availability of more training data. DroidDetector can achieve 96.76% detection accuracy, which outperforms traditional ML techniques.

Xiao et al., 2017 [73] considering some semantic information in system call sequences as natural language, treated one such sequence as a sentence and constructed a classifier based on the LSTM language model with effective number of hidden layers to achieve better result. Two LSTM models are trained using system call sequences from malware and benign apps. At classification, two similarity scores are computed, the greater one indicating whether the analyzed app is malware or not.

Nix and Zhang, (2017) [74] focused on classification of Android apps using system API-call sequences and investigating the effectiveness of DNNs for such purpose, based on their ability to learn complex and flexible features that may lead to timely and effective detection of malware. They designed a CNN for sequence classification and conducted a set of experiments on malware detection and categorization of software into functionality groups, to test and compare it with classifications by LSTM-RNN and other n-gram based methods. Both CNN and LSTM significantly outperformed n-gram based methods, and surprisingly, the performance of CNN was also much better than that of the LSTM.

SPAM AND PHISHING DETECTION

Spam email is referring unsolicited, undesired, or illegal email messages massively broadcasted to many email accounts. Phishing is a cyber-crime technique consisting in a fraudulent attempt to obtain sensitive information such as usernames, passwords, bank accounts, credit card details etc. by disguising as a trustworthy entity. These issues can be treated by using deep learning techniques with natural language processing (NLP).

Verma et al., 2018 [52], Anti-Phishing Pilot at ACM IWSPA 2018, evaluated phishing techniques over email using new metrics for unbalanced dataset. Various techniques used for feature extraction are discussed, such as: term frequency-inverse document frequency (TF-IDF), non-negative matrix factorization (NMF) and bag of words etc. Algorithms as: random forest (RF), logistic regression, k-nearest neighbor and multi-nominal naive Bayes were mostly used, but also some deep learning ones, using CNNs, RNNs, LSTMs.

Zhang and Yuan, 2012 [75] applied multi-layer feedforward neural networks to phishing email detection and evaluated the effectiveness of this approach. They designed the feature set, processed the phishing dataset, and implemented the neural network (NN) systems, then used cross validation to evaluate the performance of NNs with different numbers of hidden units and activation functions. They also compared the performance of NNs with other major ML algorithms. From the statistical analysis, concluded that NNs with an appropriate number of hidden units can achieve satisfactory accuracy even when the training examples are scarce. Moreover, their feature selection is effective in capturing the characteristics of phishing emails, as most ML algorithms can yield reasonable results with it.

Lennan et al., 2016 [76] discussed about the NLP feature extraction techniques using methods such as character level embedding and word embedding. A comparative study is made among support vector machine (SVM) using character level and CNN using both character and word embedding techniques. CNN using word embedding showed better results.

Vinayakumar et al., 2018 [77] showed a new LSTM approach in which dataset is considered as a hierarchical email architecture by considering it as sentences and words. Bi-directional LSTM is used for both cases which helps in computing the weights and estimates the phishing probability over the data during the network computation.

Shima et al., 2018 [78] used neural network and DL for classification of URL strings used for phishing sites. First, feature vectors from URL strings are generated, that are then applied to a linear NN with three layers, in a very light and compact topology.

DETECTION AND CLASSIFICATION OF DOMAIN NAMES GENERATED BY DGAS

Domain generation algorithms (DGAs) are algorithms used by various families of malware that are used to periodically generate a large number of domain names that can be used as rendez-vous points with their command and control servers. This way, attackers are avoiding the possibility of blacklisting “hardcoded” domain names. Infected computers will attempt to contact some of these domain names every day to receive updates or commands.

Some research papers addressed the use of deep learning for detecting malicious domain names (2016-2018) [79], [80], [81], [82], [83], [84]. They showed that DL algorithms using CNNs, RNNs and LSTM, performed well in comparison to traditional ML algorithms (slow and poor in

performance) and, moreover, DL algorithms remain robust in adversarial environments.

TRAFFIC ANALYSIS

Smit et al., 2017 [85] proposed using deep learning techniques for network traffic classification. This paper investigates the viability of using deep learning for traffic classification, with a focus on both network management applications, and detecting malicious traffic. Preliminary results showed that a highly accurate classifier can be created using the first 50 bytes of a traffic flow.

Wang, 2017 [86] proposed a method that is based on ANN and DL. Results showed that approach works very well on the applications of feature learning, unknown protocol identification and anomalous protocol detection.

Deep packet framework for automatically extracting features from network traffic using a DL method is proposed in [73]. These packets help to handle sophisticated task like multi challenging traffics etc.

BINARY ANALYSIS

Binary code analysis is an important component in cyber security, which looks into raw binary codes in search of vulnerability issues. Static analysis can understand the pattern of the software code to find possible weaknesses. Nowadays, automated analysis method is combined with DL method, which has surpassed the pattern-based limitations [87]. Shin et al., 2015 [88] showed that RNNs could identify functions in binaries with greater accuracy and efficiency (i.e. higher learning and recognition speed) than other state-of-the-art traditional ML-based method. To rectify gradient descent over the hidden layers, optimization is done with RMSprop method. Benchmarks for various network architectures (RNN, bi-directional RNN, LSTM and GRU) with 1 or 2 hidden layers of various dimensions (8 or 16) are analyzed. Bi-directional RNN proved to be the best.

Katz et al., 2018 [89] showed a novel technique for decompiling binary code snippets using a model based on RNNs. The model learns properties and patterns that occur in source code, and uses them to produce decompilation output.

Zheng Leong Chua et al., 2017 [90] presented a new system, EKLAVYA, which helps in recovering function type signatures from disassembled binary codes using RNN network. Argument recovery module on RNN is implemented using techniques like saliency mapping and sanitization. This system helps in learn calling conventions and idioms with high-level accuracy parameter.

Lee et al., 2017 [87] discussed DL of assembly code, which helps to analyze the software weakness. A CNN and a Text-CNN model are trained using vectors created by Instruction2vec and Word2vec from assembly code of existing functions, and then new functions are classified whether have software weakness (vulnerabilities) or not. Text-CNN trained with vectors generated by Instruction2vec provided higher accuracy in classification.

CONCLUSION

Cyber security is dealing every day with new kind of threats, more and more able to dissimulate and change their code, patterns and manifestations for avoiding IDSs. The spectacular results obtained in the last years by DL in various domains, together with the ability of DNN models and architectures (i.e. MLP, CNN, AE, RNN, LSTM etc.) in catching and learning intimate structures of data without any previous "manual" feature extraction or engineering, and to automatically clusterize data and/or make classification/prediction on new data, make them to be very appropriate and promising in most of the various use-cases of cyber security applications, for building new powerful and flexible intrusion detection systems (both HIDS and NIDS). Deep learning is already a prominent technique employed in several cyber security areas. DL algorithms proved to provide robust and effective solutions to solve various problems with better results than "classical" ML and other traditional methods. However, deep learning may be used concurrently with other automation techniques, like rule and heuristics based ones and also other ML techniques. Yet, artificial neural networks (ANNs) may be

also vulnerable to possible hacks and deception. Thus, even if difficult, by identifying patterns used by such systems in their functioning, attackers could somehow modify inputs to ANNs in such a way that will lead to misclassification of those inputs (adversarial attacks). [91]

References

[1] Yuebin, B. & Kobayashi, H. (2003). *Intrusion detection systems: technology and development*. Proc. of IEEE 17th International Conference on Advanced Information Networking and Applications, AINA 2003.
[2. *** - Wikipedia - Free Encyclopedia (DMZ). https://en.wikipedia.org/wiki/DMZ_(computing) (accessed 2019).
[3] Ozgur Depren; Murat Topallar; Emin Anarim & M. Kemal Ciliz. (2005). *An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks*. Expert systems with Applications, 29(4), pp.713–722.
[4] *** - ML 4 Cyber Security website. https://www.helpnetsecurity.com/2017/01/31/machine-learning-cybersecurity/ (accessed 2019).
[5] Mohammed Harun Babu; Vinayakumar, R. & Soman, K.P. (2018). *A short review on Applications of Deep learning for Cyber security*. arXiv:1812.06292.
[6] Karpathy, A. & Li, F.–F. (2015). *Convolutional Neural Networks for Visual Recognition*. Stanford CS Class (CS231n): http://cs231n.github.io/convolutional-networks/ (accessed 2018).
[7] Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review 65: 386-408.
[8] McCulloch, W. S. & Pitts, W. (1943). *A logical calculus of the ideas immanent in nervous activity*. Bulletin of Mathematical Biophysics, Vol. 5(4), 115-133.
[9] Rosenblatt, F. (1962). *Principles of neurodynamics; perceptrons and the theory of brain mechanisms*. Washington D.C., Spartan Books.
[10] Werbos, P. B. R. (1974). *New Tools for Prediction and Analysis in the Behavioral Sciences*. (PhD Thesis), Harvard University.
[11] Rumelhart, D. E.; McClelland, J. L. & Group, P. R. (1987). *Parallel distributed processing* (Vol. 1, p. 184). Cambridge, MA: MIT press.
[12] Cybenko, G. (1989). *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems, 2(4), 303-314. doi:10.1007/BF02551274
[13] Hornik, K. (1991). *Approximation Capabilities of Multilayer Feedforward Networks*. Neural Networks, 4(2), 251–257. doi:10.1016/0893-6080(91)90009-T
[14] Hinton, G. E. & Salakhutdinov, R. (2006). *Reducing the dimensionality of data with neural networks*. Science, 313(5786), 504–507.
[15] Hinton, G. E.; Osindero, S. & Teh, Y. (2006). *A fast learning algorithm for deep belief nets*. Neural Computation, 18, 1527–1554.
[16] Bengio, Y. and LeCun, Y. (2007). *Scaling learning algorithms towards AI*. Large Scale Kernel Machines.
[17] Martens, J. (2010). *Deep learning via Hessian-free optimization*. Proc. of the 27th International Conference on Machine Learning, ICML-10, Haifa, Israel, Jun 21-24, 735-742.
[18] Ciresan, D. C.; Meier, U.; Gambardella, L. M. & Schmidhuber, J. (2010). *Deep, big, simple neural nets for handwritten digit recognition*. Neural Computation, 22(12):3207–3220.
[19] Ciresan, D. C.; Meier, U.; Masci, J.; Gambardella, L. M. & Schmidhuber, J. (2011). *Flexible, high performance convolutional neural networks for image classification*. Proc. International Joint Conference on Artificial Intelligence - IJCAI'2011, 1237–1242.
[20] Hubel, D. H. & Wiesel, T. N. (1959). *Receptive fields of single neurones in the cat's striate cortex*. J. Physiol. 148(3):574–91, Oct. doi:10.1113/jphysiol.1959.sp006308.
[21] Fukushima, K. (1980). *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. Biological Cybernetics, 36(4), 193-202 (April).
[22] Fukushima, K. (1975). *Cognitron: A self-organizing multilayered neural network. Biological Cybernetics*, [20(3-4), 121-136 (Sept.)
[23] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. & Jackel, L. D. (1989). *Backpropagation applied to handwritten zip code recognition*. Neural Computation, 1(4):541–551.
[24] LeCun, Z.; Bottou, L; Bengio, Y. & Haffner, P. (1998). *Gradient-based learning applied to document recognition*. Proc. of the IEEE, 86(11):2278–2324. doi:10.1109/5.726791.
[25] Hahnloser, R.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J. & Seung, H. S. (2000). *Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit*. Nature. 405: 947–951. doi:10.1038/35016072.
[26] Hahnloser, R. & Seung, H.S. (2001). *Permitted and Forbidden Sets in Symmetric Threshold-Linear Networks*. NIPS 2001.

[27] Glorot, X.; Bordes, A. & Bengio, Y. (2011). *Deep sparse rectifier neural networks*. Proc. of 14th International Conference on Artificial Intelligence and Statistics – AISTATS 2011. JMLR Vol.15: PMLR 15: 315-323
[28] *** - ImageNet website, 2018: http://image-net.org/ (accessed 2018).
[29] Krizhevsky, A.; Sutskever, I. & Hinton, G. E. (2012). *ImageNet classification with deep convolutional neural networks*. In NIPS 2012, pp. 1106–1114.
[30] Zeiler, M. D. & Fergus, R. (2014). *Visualizing and understanding convolutional networks*. CoRR, abs/1311.2901, 2013. Published in Proc. ECCV 2014.
[31] Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V. & Rabinovich, A. (2014). *Going deeper with convolutions*. CoRR, abs/1409.4842.
[32] Simonyan, K. & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition*, Proc. ICLR 2015.
[33] He, K.; Zhang, X.; Ren, S. & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. arXiv:1512.03385.
[34] Huang, G.; Zhuang, L.; Van der Maaten, L. and Weinberger, K. Q. (2016). *Densely Connected Convolutional Networks*. arXiv: 1608.06993, Aug.
[35] Hu, J.; Shen, L.; Albanie, S.; Sun, G. & Wu, E. (2017). *Squeeze-and-Excitation Networks*. arXiv: 1709.01507, Sep.
[36] Goodfellow, I.; Bengio, Y. & Courville, A. (2016). Deep Learning. MIT Press.
[37] Mehdi Mohammadi; Ala Al-Fuqaha; Sameh Sorour & Mohsen Guizani. (2017). *Deep Learning for IoT Big Data and Streaming Analytics: A Survey*. arXiv:1712.04301v1.
[38] Werbos, .P. J. (1990). *Backpropagation through time: what it does and how to do it*, Proceedings of the IEEE 78.10, pp.1550-1560.
[39] Oprea, Sergiu; Gil, Pablo; Mira Martínez, Damián & Alacid, Beatriz. (2017). *Candidate Oil Spill Detection in SLAR Data A Recurrent Neural Network-based Approach*. In: 6th Int. Conf. on Pattern Recognition Applications and Methods (ICPRAM '17), DOI: 10.5220/0006187103720377.
[40] Hochreiter, S. & Schmidhuber, J. (1997). *Long short-term memory*. In Neural computation 9(8), pp.1735-1780.
[41] Jihyun Kim; Jaehyun Kim; Huong Le Thi Thu & Howon Kim (2016). *Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection.* IEEE Proc. of 2016 International Conference on Platform Technology and Service (PlatCon).
[42] Verma, R. (2018). *Security analytics: Adapting data science for security challenges*. In: Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics, ACM, pp. 40–41.
[43] *** - GitHub Repository of Cyber Security Datasets. https://github.com/jivoi/awesome-ml-for-cybersecurity#-datasets (accessed 2019).
[44] *** - Mike Sconzo's Security Data Repository. http://www.secrepo.com (accessed 2019).
[45] *** - Knowledge Discovery and Data Mining group - "KDD cup 1999" website page. http://www.kdd.org/kddcup/index.php (accessed 2019).
[46] *** - KDD Cup 1999 Data website. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (accessed 2019).
[47] *** - NSL-KDD dataset website page, https://www.unb.ca/cic/datasets/nsl.html (accessed 2019).
[48] *** - Creech, G. (2013). The ADFA Intrusion Detection Datasets website page. https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-IDS-Datasets/ (accessed 2019).
[49] Creech, G. & Hu, J. (2013). *Generation of a new IDS test dataset: Time to retire the KDD collection*. Proc. of 2013 IEEE Wireless Communications and Networking Conference (WCNC), pp. 4487–4492, DOI: 10.1109/WCNC.2013.6555301.
[50] Creech, G. (2014). *Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks*. PhD Thesis, Engineering & Information Technology, UNSW Canberra, UNSW.
[51] *** - IWSPA 2018 Shared Task website page. https://dasavisha.github.io/IWSPA-sharedtask/ (accessed 2019).
[52] *** - IWSPA 2018 Anti-Phishing pilot website page. http://www2.cs.uh.edu/~rmverma/anti-phishing-pilot.pdf (accessed 2019).
[53] *** - DMD 2018 website page. http://nlp.amrita.edu/DMD2018 (accessed 2019).
[54] *** - ICACCI 2018 website page. http://icacci-conference.org/2018/ (accessed 2019).
[55] *** - CDMC 2018 website page. http://www.csmining.org/cdmc2018/ (accessed 2019).
[56] Staudemeyer, R. C. & Omlin, C. W. (2013). *Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data*. In: Proc. of the South African Institute for Computer Scientists and Information Technologists Conference, ACM, 2013, pp. 218–224.
[57] Javaid, Ahmad Y.; Niyaz, Quamar; Sun, Weqing & Alam, Mansoor. (2015). *A Deep Learning Approach for Network Intrusion Detection System*. EAI Endorsed Transactions on Security and Safety. IEEE transaction 3. DOI: 10.4108/eai.3-12-2015.2262516.
58] Kim, G.; Yi, H.; Lee, J.; Paek, Y. &, Yoon, S. (2016). *LSTM-based system call language modeling and robust ensemble method for designing host-based intrusion detection systems*. arXiv:1611.01726.
[59] Vinayakumar, R.; Soman, K. & Poornachandran, P. (2017). *Evaluating effectiveness of shallow and deep networks to*

intrusion detection system. In: Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on, IEEE, pp. 1282–1289.
[60] Vinayakumar, R.; Soman, K. & Poornachandran, P. (2017). *Applying convolutional neural network for network intrusion detection*. In: Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on, IEEE, pp. 1222–1228.
[61] Leila Mohammadpour; Teck Chaw Ling; Chee Sun Liew & Chun Yong Chong. (2018). *A Convolutional Neural Network for Network Intrusion Detection System*. Proc. of the APAN – Research Workshop 2018, pp. 50–55, ISBN 978-4-9905448-8-1.
[62] Shone, N.; Ngoc, T. N.; Phai, V. D. & Shi, Q. (2018) *A deep learning approach to network intrusion detection*. In: IEEE Transactions on Emerging Topics in Computational Intelligence, 2(1), pp. 41–50.
[63] Rahul, V. K.; Vinayakumar, R.; Soman, K. P. & Poornachandran, P. (2018, July). *Evaluating Shallow and Deep Neural Networks for Network Intrusion Detection Systems in Cyber Security*. In: 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT 2018), IEEE, pp. 1–6.
[64] Dahl, G. E.; Stokes, J. W.; Deng, L. & Yu, D. (2013). *Large-scale malware classification using random projections and neural networks*. In: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, pp. 3422–3426.
[65] Yuancheng Li; Rong Ma & Runhai Jiao. (2015). *A Hybridybrid Malicious Code Detection Method based on Deep Learning*. In: International Journal of Security and Its Applications, 9(5), pp. 205-216, http://dx.doi.org/10.14257/ijsia.2015.9.5.21.
[66] Saxe, J. & Berlin, K. (2015). *Deep neural network based malware detection using two dimensional binary program features*. In: Malicious and Unwanted Software (MALWARE), 2015 10th International Conference on, IEEE, pp. 11–20.
[67] Pascanu, R.; Stokes, J. W.; Sanossian, H.; Marinescu, M. & Thomas. (2015). *Malware classification with recurrent networks*. In: Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on, IEEE, pp. 1916–1920.
[68] Huang, W. & J. W. Stokes. (2016). *MtNet: a multi-task neural network for dynamic malware classification*. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer, pp. 399–418.
[69] R. Rahul; T. Anjali; V. K. Menon & K. Soman. (2017). *Deep learning for network flow analysis and malware classification*. In: International Symposium on Security in Computing and Communication (ISSCC) 2017, Springer, pp. 226–235.
[70] R. Vinayakumar; K. Soman; K. S. Velan & S. Ganorkar. (2017). *Evaluating shallow and deep networks for ransomware detection and classification*. In: Advances in Computing, Communications and Informatics (ICACCI), 2017 International Conference on, IEEE, pp. 259–265.
[71] S. Maniath; A. Ashok; P. Poornachandran; V. Sujadevi; A. P. Sankar & S. Jan. (2017). *Deep learning LSTM based ransomware detection*. In: Control, Automation & Power Engineering (RDCAPE), 2017 Recent Developments in, IEEE, pp. 442–446.
[72] Yuan, Z.; Lu, Y. & Xue, Y. (2016). *Droid detector: android malware characterization and detection using deep learning*, Tsinghua Science and Technology, 21(1), 114–123.
[73] X. Xiao; S. Zhang; F. Mercaldo; G. Hu & A. K. Sangaiah. (2017). *Android malware detection based on system call sequences and LSTM*. In Multimedia Tools and Applications, pp. 1–21.
[74] R. Nix & J. Zhang. (2017). *Classification of android apps and malware using deep neural networks*. In: Neural Networks (IJCNN), 2017 International Joint Conference on, IEEE, pp. 1871–1878.
[75] N. Zhang & Y. Yuan. (2012). *Phishing detection using neural network*. CS229 lecture notes, http://cs229.stanford.edu/proj2012/ZhangYuan-PhishingDetectionUsingNeuralNetwork.pdf.
[76] C. Lennan; B. Naber; J. Reher & L. Weber. (2016). *End-to-end spam classification with neural networks*. Project report for Machine Learning 1, SS 2016, Prof. Marius Kloft, Humboldt-University Berlin, Germany. URL https://amor.cms.hu-berlin.de/~jaehnicp/project/spam-filter/cnn_report.pdf (accessed 2019).
[77] R. Vinayakumar; K. Soman & P. Poornachandran. (2018). *Evaluating deep learning approaches to characterize and classify malicious URL's, Journal of Intelligent & Fuzzy Systems* 34 (3), pp. 1333–1343.
[78] K. Shima; D. Miyamoto; H. Abe; T. Ishihara; K. Okada; Y. Sekiya; H. Asai & Y. Doi. (2018). *Classification of URL bitstreams using bag of bytes*. IEEE 21st Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN) 2018. DOI: 10.1109/ICIN.2018.8401597.
[79] J. Woodbridge; H. S. Anderson; A. Ahuja & D. Grant. (2016). *Predicting domain generation algorithms with long short-term memory networks*. arXiv:1611.00791.
[80] B. Yu; D. L. Gray; J. Pan; M. De Cock & A. C. Nascimento. (2017). *Inline DGA detection with deep networks*. In: 2017 IEEE International Conference on Data Mining Workshops (ICDMW), IEEE, pp. 683–692.
[81] F. Zeng; S. Chang & X. Wan. (2017). *Classification for DGA-based malicious domain names with deep learning architectures*. International Journal of Intelligent Information Systems, 6(6), 67.

[82] P. Lison & V. Mavroeidis. (2017). *Automatic detection of malware generated domains with recurrent neural models*. arXiv:1709.07102.
[83] B. Athiwaratkun & J. W. Stokes. (2017). *Malware classification with LSTM and GRU language models and a character-level CNN*, in: Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on, IEEE, pp. 2482–2486.
[84] B. Yu; J. Pan; J. Hu; A. Nascimento & M. De Cock. (2018). *Character level based detection of DGA domain names*. 2018 International Joint Conference on Neural Networks (IJCNN), DOI: 10.1109/IJCNN.2018.8489147
[85] D. Smit; K. Millar; C. Page; A. Cheng; H.-G. Chew & C.-C. Lim. (2017). *Looking deeper: Using deep learning to identify internet communications traffic*, Macquarie Matrix: Special edition, ACUR 1 (2017) 1318–1323.
[86] Z. Wang (2015). *The applications of deep learning on traffic identification*, BlackHat USA, https://www.blackhat.com/docs/us-15/materials/us-15-Wang-The-Applications-Of-Deep-Learning-On-Traffic-Identification-wp.pdf (accessed 2019).
[87] Y. J. Lee; S.-H. Choi; C. Kim; S.-H. Lim & K.-W. Park. (2017). *Learning binary code with deep learning to detect software weakness*. KSII The 9th International Conference on Internet (ICONI) 2017 Symposium, pp. 245–249.
[88] E. C. R. Shin; D. Song & R. Moazzezi. (2015). Recognizing functions in binaries with neural networks. In: USENIX Security Symposium 2015, pp. 611–626.
[89] D. S. Katz; J. Ruchti & E. Schulte. (2018). *Using recurrent neural networks for decompilation*. In: Proc. of IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 346–356.
[90] Z. L. Chua; S. Shen; P. Saxena & Z. Liang. (2017). *Neural nets can learn function type signatures from binaries*. In: Proc. of the 26th USENIX Security Symposium, USENIX Security Vol. 17.
[91] *** - Wikipedia - Free Encyclopedia (DL). https://en.wikipedia.org/wiki/Deep_learning (accessed 2019).