Available online at www.sciencedirect.com

ScienceDirect

Expert Systems With Applications 00 (2020) 1–32

ESWA

A Comparative Evaluation of Unsupervised Deep Architectures for Intrusion Detection in Sequential Data Streams

Dušan Sovilj[a,b,*], Paul Budnarain[a], Scott Sanner[a,*], Geoff Salmon[b], Mohan Rao[b]

[a]*Mechanical and Industrial Engineering, University of Toronto, 5 King's College Rd, Toronto, ON M5S 3G8, Canada*
[b]*Rank Software Inc, #400 - 214 King St West, Toronto, ON M5H 3S6, Canada*

Abstract

Cybersecurity data remains a challenge for the machine learning community as the high volume of traffic makes it difficult to properly disambiguate anomalous from normal behaviour. That decision is the core of an intelligent Intrusion Detection System (IDS), a component responsible for raising alerts whenever a potential threat is detected. However, with high volume data in contemporary systems, these IDSs generate numerous alerts, too large for human operators to exhaustively investigate. Moreover, simply reporting a single possible threat is often not sufficient, since the security analyst has to investigate the alert without any further clues of the underlying cause. In order to combat these issues, we *empirically compare popular deep neural learning architectures* for the problem of intrusion detection in *sequential data streams*. Contrary to a majority of research studies, we do not take a classification-based approach that requires labeled examples of hostile attacks. Instead, we adopt an *unsupervised anomaly detection approach* that aims to model a benign sequential data distribution against which new test instances are compared to. We also examine one additional deep network in the form of an *attention model* capable of providing *explanations* in addition to its predictions; such information is of crucial importance to network operators since it provides additional guidance to resolve potential threats. For our experiments, we evaluate the models against a variety of data sets of different complexities, ranging from simple unidimensional (synthetic and Yahoo!) to more complex multi-source (CICIDS2017 and small real-world enterprise network) data streams. In order to facilitate end-user needs, we focus on *ranking-based metrics* for comparing different deep neural architectures. This evaluation is especially important for security analysts to prioritize their anomaly investigations. Overall, our experiments demonstrate that a variant of a recurrent neural network generally outperforms a popular non-sequential deep autoencoder commonly used for unsupervised anomaly detection. The attentional model did not provide sufficiently good performance and explanations that we discuss in our analysis. Nonetheless, given that the global financial outlays for cybersecurity are calculated in trillions of dollars, our evaluation and identification of the top-performing RNN architectures for anomaly detection in sequential data streams can lead to improved intelligent IDS design, while our contributions of attentional explanation will hopefully lay the foundations for future improvements to the explanatory capability of these intelligent learning-based IDSs.

Keywords: deep learning, intrusion detection, anomaly detection, real-world data

1. Introduction

Cybersecurity consists of a collection of technologies and processes aimed to protect computers, networks and data from unwanted attacks, such as unauthorized access, modification, or deletion of data (Buczak & Guven, 2016).

*Corresponding author.
Email addresses: dusans@mie.utoronto.ca (Dušan Sovilj), paul.budnarain@mail.utoronto.ca (Paul Budnarain), ssanner@mie.utoronto.ca (Scott Sanner), geoff.salmon@ranksoftwareinc.com (Geoff Salmon), mohan.rao@ranksoftwareinc.com (Mohan Rao)

In order to combat these unwanted attacks, security systems involve a vast range of mechanisms, each designed to tackle a different aspect of the network communication spectrum. One important component of this mechanism set is an intrusion detection system (IDS) — a specialized *intelligent system* for detection of potential intrusions. IDSs help in classifying suspicious behavior which leads to discovery and identification of actual attacks. The amount of hacker attacks have dramatically increased in the recent years as several reports have shown that global spending on cybersecurity will keep increasing over the years from 6 trillion dollars (Cybersecurity Ventures, 2017) in the next few years to 30 trillion by 2030 (Atlantic Council, 2017). Moreover, processing the vast amount of information in today's networks is beyond human capabilities which necessitates automated filtering mechanisms. The goal is to provide the end-user (security analyst) a *condensed list of alerts* that requires further investigation. The list should preferably be *ordered by alert importance* – from most to least suspicious, which we can evaluate using ranking metrics commonly used in information retrieval (Manning et al., 2008) to mimic the security analyst's prioritized and time-limited operational environment for assessing security threats. Furthermore, each alert should contain an *understandable explanation* of the underlying cause and actors involved, for example, machines affected within the system, remote addresses and processes.

There are two main types of IDS: misuse-based and anomaly-based. Misuse- or signature-based systems aim to identify attacks by comparing against signatures of already known attacks. Although they are reliable in terms of false alarm rate, they require constant manual updates of databases with rules and signatures. Anomaly-based techniques model the normal network and user behavior, and then check whether new activities deviate from that normal signature. An appealing aspect of anomaly-based detectors is the supposed ability to detect novel attacks (zero-day attacks) (Abraham et al., 2007; Modi et al., 2013), but that capability has been questioned in (Sommer & Paxson, 2010). A major downside of these systems is the high false-alarm rate for new signatures, since almost any deviation from normal behavior is flagged as potentially problematic. Finally, there are approaches that fall in the middle-ground between these two aspects to leverage the advantages of both: low false-alarm rate with the capability to detect novel attacks (Aydın et al., 2009; Kim et al., 2014).

In order to combat zero-day attacks as well as handle vast volumes of traffic, *machine learning* techniques often need to be deployed in order to adapt to changing network traffic patterns. Numerous research efforts have investigated how machine learning and other statistical approaches can be used for anomaly detection in the cybersecurity domain such as the following techniques: nearest neighbors (Tsai & Lin, 2010), support vector machines (Mukkamala et al., 2002; Hu et al., 2003), artificial neural networks (Mukkamala et al., 2002; Bivens et al., 2002; Lippmann & Cunningham, 2000; Xiang et al., 2014), clustering-based algorithms (Blowers & Williams, 2014; Sequeira & Zaki, 2002), hidden Markov models (Joshi & Phoha, 2005), decision trees (Bilge et al., 2011) and ensemble of random forests (Zhang et al., 2008) to name a few. These initial efforts have primarily investigated how machine learning algorithms can be used to automatically build intelligent IDSs from data and how well they perform in the cybersecurity domain, but they have not focused on the explanatory analysis of these approaches.

In recent years, there have been substantial advances in the domain of artificial neural networks in the form of *deep* architectures which have triggered a new deep learning revolution (LeCun et al., 2015). This is also the key learning paradigm that we explore in this paper, where we investigate several architectures in an unsupervised learning scenario.[1] Deep neural networks have already been the focus for building IDSs (Li et al., 2015; Erfani et al., 2016; Javaid et al., 2016; Muna et al., 2018). However, those efforts are focusing on feature-based data and fully connected autoencoders which may not be suitable in the cybersecurity domain where the underlying traffic and event data is inherently *temporal*. In the paper, we expand the focus to *sequential* models and specifically, the Recurrent Neural Network (RNN) and its variants. We provide a thorough comparison between these two design choices for building learning-based IDSs, including a large variety of architectures for the RNN setting.

While machine learning – and specifically sequential deep learning – hold promise as an effective means of anomaly-based intrusion detection, there are still a broad range of questions to consider when deploying such an

[1] It is important to disambiguate our use of *unsupervised* since it has differing meanings in the cybersecurity and machine learning context. The cybersecurity literature often uses the term *unsupervised anomaly detection* defined in Chandola *et al* (Chandola et al., 2009) as "Techniques that operate in [an] unsupervised mode [that] do not require training data...". In this article, we use machine learning terminology for *unsupervised learning*, which in the anomaly-based intrusion detection context means that we *do* have data to train on, but we *do not* have data with explicitly labeled examples of known intrusions. In the parlance of Chandola *et al* (Chandola et al., 2009), our approach would be termed *semi-supervised anomaly detection*.

approach. Hence, a comparative evaluation such as the one undertaken in this article requires careful consideration of numerous additional IDS design choices, which we outline below:

- **Modeling approach:** Should we use a supervised classification-based approach that requires a large quantity of labeled examples, or an unsupervised anomaly detection approach that does not require labeled examples?
- **Evaluation data sets:** To promote reproducibility, how do we obtain publicly available data for training and testing that are reliable and reflect real-world use-cases?
- **Feature extraction:** How should we represent features extracted from captured raw traffic/events for input to machine or deep learning based systems?
- **User-centric evaluation:** How can we design an evaluation method that reflects the needs of end-user network security analysts?
- **Explainability:** How can we explain the reason a particular sequence of network events was classified as an anomaly?

Considering these design choices, our contributions in this paper are the following:

1. We provide a thorough comparison of recurrent neural network (RNN) autoencoders (that do not require labeled examples of intrusions) for the problem of building intelligent machine learning based IDSs for anomaly detection in sequential data streams and evaluate these methods according to human operator focused ranking-based metrics on a diverse variety of data sets. We are not aware of a thorough comparative evaluation of deep *sequential RNN-based* anomaly detection methods focusing on ranking-based metrics in the literature.
2. Introduction of an *attentional* component that enables *explanations* of the outcomes of these neural models. This can be important for security analysts to help them understand *why* certain cases are anomalous. Although the explanations that result from this method prove to be logically coherent, they do not provide sufficient insight into the anomaly causes to strongly justify their practical use. Nonetheless, such results should not dissuade other researchers from improving on explanatory methods using attentional models.

The rest of the paper is organized as follows. Section 2 discusses the IDS design choices above and the reasoning supporting the approach taken in this article. Section 3 summarizes the recent merger of deep learning and cybersecurity to develop an accurate IDS. In Section 4 we describe all the neural models used in the experiments with brief explanations of their underlying mechanisms. Section 5 presents the evaluation method for ranking-based strategies, while Section 6 introduces the data sets adopted for experiments. In Section 7 we summarize all the experimental results with a discussion of the proposed methodology. Finally, Section 8 offers concluding remarks.

2. Machine Learning-based Intrusion Detection System Design Choices

In this section, we discuss a number of critical design choices for developing intelligent machine learning-based IDSs and the rationale for the approaches compared and the evaluation metrics in this article.

2.1. Modeling approach

The majority of deep learning based intrusion detection studies focus on supervised *classification* approaches since publicly available data are usually explicitly split into benign and malicious classes (Yin et al., 2017; Diro & Chilamkurti, 2018; Wang, 2018; Shone et al., 2018). One key reason for this supervised classification focus is due to over-reliance on an outdated KDD data set (Aldweesh et al., 2020). This approach is reasonable if we expect to only have existing types of attacks in the operating environment. However, the end result is a system able to distinguish between two classes (accounting for all *introduced* attack types) without ever being designed to detect new anomalies. For example, if we know about DDoS, buffer overflow, SYN flood and Nmap scans as possible attacks, but we only train the model on benign data plus DDoS data, it is unclear what our model will predict for the other attack instances.

Another approach to solving the intrusion detection problem with (deep) autoencoders is a machine learning approach that attempts to *model the distribution of benign data* (i.e., data not containing an intrusion). The idea is to train an autoencoder to project an input sequence of data into a low-dimensional latent embedding then attempt to reconstruct the original input sequence from this embedding. If we train such a model on benign data, we can then expect for this model to have a low error when attempting to reconstruct data similar to the distribution of data it was trained on; in contrast, the autoencoder should have higher error when reconstructing anomalous data dissimilar to the training data that could indicate potential intrusions. In this approach, we require a metric that defines an *anomaly score* enabling us to rank test samples. Ideally, high scores should be assigned to all attack patterns, while benign cases would receive low scores. We adopt the reconstruction-based strategy for our anomaly detection framework since it applies to the more realistic real-world setting where data labeled with the full range of possible intrusion anomalies is simply not available.

2.2. Evaluation data sets

One of the challenges in developing and reproducibly evaluating new anomaly-based methods is obtaining publicly available and reliable data that reflect up-to-date known attack types in addition to having sufficient qualitative representation of all relevant protocols. However, most of the available data aimed at research is artificially generated, while all proprietary data is either inaccessible or stripped off relevant information for privacy concerns. One of the first available data sets is the DARPA challenge KDD data[2] that has been extensively used for the past two decades. Shortcomings of the original data have been pointed out in (Tavallaee et al., 2009), which has led to a modified data NSL-KDD with corrected records, but it is still considered outdated (Moustafa et al., 2017). Several promising new data sets have been constructed to reflect all the technological advancements and also more sophisticated attack types: ISCXIDS2012 (Shiravi et al., 2012), ADFA (Creech & Hu, 2013), CICIDS2017 (Gharib et al., 2016; Sharafaldin et al., 2018), UNSW-NB15 (Moustafa & Slay, 2015), CERT Insider Threat Dataset (Glasser & Lindauer, 2013). A brief summary of several of these data sets and their statistics is given in (Sharafaldin et al., 2018) and a more comprehensive evaluation in (Ring et al., 2019).

While we do use some more recent publicly available anomaly detection data sets, we also introduce another data set collected by Rank Software Inc.[3] containing events collected from an enterprise network. Although the data itself is small by today's standards, it still presents real-world use-cases with both host- and network-related data streams. Since the period during which the data was collected did not involve any real intrusion attempts, malicious data has been simulated and included similar to other studies.

2.3. Feature extraction

Another difficult challenge is extracting features from captured raw events: how should we process heterogeneous data and transform it into a format acceptable by machine learning tools? Most of the preprocessing involves some form of frequency counting or other summary statistics computed within a suitable time window. Several tools exist such as CICFlowMeter[4] and Zeek[5], which each have their own extraction method depending on the nature of data they process. As this heterogeneous data is strictly temporal in nature, constructing summary statistics for *individual* time windows and performing anomaly detection per time window defeats the ability to extract longer-term sequential patterns that can be critical for anomaly detection. For this reason, we explore formats for the data consisting of *sequences* of summary statistics for each time window; this format permits sequential models to learn temporal patterns in data that can be used to detect anomalies.

2.4. User-centric evaluation

In an operating environment where the most dangerous threats need to be addressed immediately while security analysts have limited bandwidth to assess all potential threats, binary classification outcomes are not sufficient to

[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
[3] https://www.ranksoftwareinc.com/
[4] https://github.com/ISCX/CICFlowMeter
[5] https://www.zeek.org/

prioritize threats. Instead, in this article we focus on *ranking* approaches that dovetail with the reconstruction-based strategy that provides *anomaly scores* to rank potential intrusions. Furthermore, instead of focusing on detection rates and false-alarm rates, we focus on ranking evaluation metrics that assess the quality of a ranked list. In this paper, we adopt standard ranking based metrics from the information retrieval literature, i.e., metrics commonly used to evaluate search engine results.

2.5. Explainability

With the mass volumes of data collected and used in contemporary cybersecurity approaches, the human operator has to process an enormous number of reports and alarms which result in a high cognitive load and decreased efficiency. Even when the most serious threats are highly ranked and examined first, there is often little indication of how to proceed further. That is, there is no explanation for *why* certain alarms are raised (Sommer & Paxson, 2010) and hence the operators need to inspect the full scope of the environment in order to understand the underlying reasons for the alarm (Tuor et al., 2017). In this paper, we examine a variation of an RNN using attention that is able to highlight regions of interest during anomaly detection with the intent to provide such insight to the operator.

3. Related work

Deep learning has attracted many researchers to apply the techniques in developing more reliable IDSs. Studies by (Alom et al., 2015) and (Gao et al., 2014) both use a deep belief network (DBN) that is first initialized via greedy-layer pretraining before the final supervised fine-tuning phase aimed to solve the classification problem. An additional preprocessing step is taken in (Li et al., 2015) by first using an autoencoder to reduce the dimensionality of feature space before the DBN classifier. (Erfani et al., 2016) follow a similar approach, but use a DBN for feature extraction to combat the curse of dimensionality – the final classifier is based on a One-Class Support Vector Machine (OC-SVM).

(Yin et al., 2017) compared a Recurrent Neural Network based IDS to traditional machine learning techniques in both binary and multiclass settings showing that introducing a temporal variable to the model can improve the performance. However, a more comparative examination of different models from deep learning literature is missing which we aim to alleviate in this paper.

Two studies based on a hybrid approach showing great results are presented in (Javaid et al., 2016; Muna et al., 2018). The idea is to first train an autoencoding model using only *normal* data, while the second phase replaces the last layer (the full feature vector output) with a classification layer for further fine-tuning on *labeled* data. This approach produces the best result on NSL-KDD data (Muna et al., 2018).

(Seok & Kim, 2016) utilized the Convolutional Neural Network architecture for the purpose of malware recognition. Since convolutional models are designed to process images, the necessary preprocessing step involved converting the malware code into a malware image. Finally, one of the better performing approaches on the NSL-KDD data is an ensemble of different neural network types (autoencoder, deep belief net, deep neural net, extreme learning machine) that was proposed in (Ludwig, 2017).

An online unsupervised method is proposed in (Tuor et al., 2017) where both feed-forward and recurrent deep networks are trained in a predictive manner modeling temporal data. Continuous and discrete variables in the feature set are modeled with separate (top-level) layers which resemble Mixture Density Networks. Anomalies are ranked according to their scores, aiming to provide network operators the list of potential threats. Following this reasoning, we also focus on ranked metrics in our evaluation procedure rather than absolute classification performance.

An attractive aspect of (deep) neural networks is their capability of automatic feature extraction from data (Krizhevsky & Hinton, 2011). This is reflected by the dominant use of autoencoders in various studies, aiming to find the most suitable *embedding* for either the classification or reproduction task. In this paper, we examine how different deep autoencoding architectures perform in the anomaly detection task with the goal of constructing an IDS that would be as reliable as possible.

4. Deep learning models and unsupervised anomaly detection

In this section, we introduce notation, provide an overview of our methodology for anomaly detection and briefly describe all the models that are used for comparison purposes in experimental section. For all models, we outline their architectures and the key equations behind them.

4.1. Notation

We assume we have access to a sequence of observations $[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots]$ where each $\mathbf{x}_t = [x_t^1, x_t^2, \ldots, x_t^D] \in \mathbb{R}^D$ is a $D-$dimensional vector. Each observation represents a feature vector extracted from the captured raw data corresponding to a particular time instance. This can be either packet-level information in network traffic or a series of aggregated features. For example, this can be the average number of forward bytes across all connections within an enterprise during a 10-minute interval. For our experiments, we limit sequences to a maximum length T, and one sample $\mathbf{x}_{(i)}$ then corresponds to T consecutive feature vectors $\mathbf{x}_{(i)} = [\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{i+T-1}]$. A complete data set $\mathbf{X}$ is then a series of T-length vectors which is obtained by sliding a window of size T over complete sequence of observations.

Since our modeling approach is reconstruction-based, our target vectors are the inputs themselves. We use $\mathbf{y}_{(i)}$ to indicate the output of a model M for sample $\mathbf{x}_{(i)}$, that is, $M(\mathbf{x}_{(i)}) = \mathbf{y}_{(i)}$ and the aim is to produce an output that is as close as possible to the input. This holds for all the architectures explained in this section; notably the *target output is not shifted* relative to the input.

4.2. From reconstruction to anomaly detection

While identification of anomalous samples is definitely of utmost importance, there are cases where the most suspicious threat needs to be addressed immediately to enforce necessary preventive measures. In the case of binary classification with only two possible outcomes, such countermeasures cannot be realized due to the lack of *anomaly scores* or *ranked list*. For this reason, it is important to have an IDS that provides *scores* that enables us to rank all new tests cases. This ranked list can further be examined with the premise that top-most ranked items are highly suspicious.

A simple but effective way to rank the data samples is by their *reconstruction error* – how well the model is able to replicate the sample itself? In the case of perfect replication, the error is zero $E(\mathbf{x}, M(\mathbf{x})) = 0$. The higher the error, the more likely the sample does not belong to the training data distribution representing *normal* data and hence becomes suspicious. We define reconstruction error for a sample $\mathbf{x}_{(i)}$ as $E_{(i)} = \|\mathbf{x}_{(i)} - M(\mathbf{x}_{(i)})\|_2^2$.

Finally, we define a framework for rank-based anomaly detection:

1. Train a reconstruction model M only on benign data $\mathbf{X}$
2. Compute reconstruction errors for new samples $\{E_{(i)}\}_{\mathbf{x}_{(i)} \in \mathbf{X}_{\text{test}}}$
3. Construct a *ranked list* $\Gamma = [E_{(1)}, E_{(2)}, \ldots,]$ where $E_{(j)} \geq E_{(k)}$ for $j < k \leq |\mathbf{X}_{\text{test}}|$

The list Γ can be examined element-by-element, from the most suspicious (anomalous) cases to the least suspicious cases. However, a key question remains: how do we compare two models M_1 and M_2 and their respective ranked lists Γ_1 and Γ_2 on the test set? In Section 5, we describe several measures that provide an answer as to which ranked list is more appropriate for the end user.

4.3. Fully connected neural network

We start by first describing a fully connected neural network (Bishop, 1995) displayed in Fig. 1. It consists of a series of layers of neurons where the layers themselves are interconnected. These connections, or weights, only run between two adjacent layers. The input layer copies the feature vector $\mathbf{x}_{(i)}$ from data, and in our case the output layer's target values are the same vector $\mathbf{x}_{(i)}$. All other layers are called hidden layers where all the processing is performed. There is a deterministic processing path from the inputs to the outputs, called forward-pass. Training is done via back-propagation algorithm (Rumelhart et al., 1986) where the weights are iteratively updated using batches of data. Each

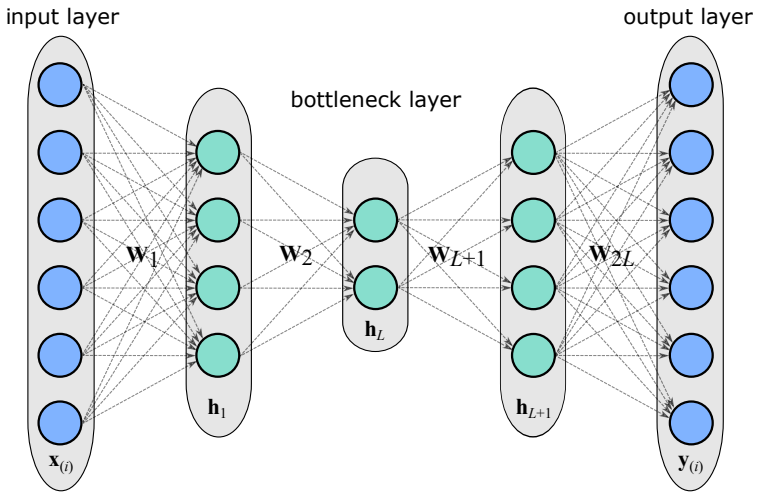


Figure 1. Fully connected neural network for autoencoding task with $L = 2$. Blue nodes indicate input $\mathbf{x}_{(i)}$ and output $\mathbf{y}_{(i)}$ layers and teal nodes comprise hidden layers $\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_{2L}$.

neuron is a basic processing unit that transforms its weighted summed input and sends its signal (output) further down the chain. These transformations or activations depend on the underlying function and are almost always non-linear (such as the sigmoid, hyperbolic tangent or rectified linear unit activations), and this is where neural nets obtain their non-linear representational capabilities and generalization power.

Fig. 1 shows the fully connected neural net with $2L$ layers where the processing flow goes from left to right with a series of transformations:

$$\begin{aligned} \mathbf{h}_0 &= \mathbf{x}_{(i)} \\ \mathbf{a}_k &= \mathbf{h}_{k-1} \cdot \mathbf{W}_k, k = 1, \ldots, 2L \\ \mathbf{h}_k &= f(\mathbf{a}_k), k = 1, \ldots, 2L \\ \mathbf{y}_{(i)} &= \mathbf{h}_{2L} \end{aligned} \tag{1}$$

where the output of the model $M(\mathbf{x}_{(i)})$ is the activation of the output layer $\mathbf{h}_{2L}$.

Set of all weights $\mathbf{W}_1, \ldots, \mathbf{W}_{2L}$ connecting the layers is the set of parameters for optimization given a suitable loss function $\mathcal{L} = \mathrm{F}(\{\mathbf{x}_{(i)}; \mathbf{y}_{(i)}\})$, usually squared error $\sum_{i=1}^{N} \|\mathbf{x}_{(i)} - \mathbf{y}_{(i)}\|^2$. We omit the bias term $\mathbf{b}_k$ for each layer to keep notation simple and focus on relevant relationships between layers.

The architecture shown in Fig. 1 can also be interpreted as an autoencoder when the inputs and outputs are equal $\mathbf{x}_{(i)} = \mathbf{y}_{(i)}$. In this case, the model can be split into two distinct parts: *encoder* and *decoder*. The encoder involves the first L layers up to and including the bottleneck layer and (usually) yields a lower-dimensional representation (a.k.a., an *embedding*) of the input feature vector $\mathbf{x} \rightarrow \mathbf{h}_L$. The decoder consists of the aforementioned bottleneck layer and all layers up to the output layer – the decoder takes the embedded representation and attempts to reconstruct the full-length feature vector $\mathbf{h}_L \rightarrow \mathbf{x}$ from it. A common practice is to have the decoder architecture mirror that of the encoder, but in reverse order.

4.4. Recurrent Neural Networks

Recurrent Neural Networks (Jordan, 1997; Elman, 1990) (RNN) are specifically designed to handle sequential data $\mathbf{x}_1, \ldots, \mathbf{x}_T$. By introducing parameter sharing across time steps and recurrent connections, RNNs provide a powerful framework for modeling both short and long-term dependencies. By having extra connections between the hidden layer itself, shown in Fig. 2, a form of internal memory or context arises that is able to represent the *current* (up to a particular time step t) state sequence. This state is recursively updated as new inputs $\mathbf{x}_{t+j}$ are processed. The complete

update step is given with the following equations:

$$\begin{aligned} \mathbf{h}_t &= f(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t) \\ \mathbf{y}_t &= g(\mathbf{V}\mathbf{h}_t) \end{aligned} \tag{2}$$

where $\mathbf{h}_t$ is referred to as state or embedding, in contrast to fully connected network where $\mathbf{h}_t$ is output of layer t. Function f is usually hyperbolic tangent, while g depends on the problem domain, and can be linear, sigmoidal or softmax. Matrix $\mathbf{U}$ is a set of weights that connects input vector $\mathbf{x}_t$ to internal memory, while matrix $\mathbf{V}$ is a set of weights connecting memory and output neurons. Matrices $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ represent the parameters of the network that are optimized during training.

This basic RNN structure has been a powerful model, but the one the can be further improved upon by introducing additional processing units. Eqs. (2) form a *single* hidden layer RNN that can be enriched with extra layers (stacked RNN) or utilizing the reverse order of inputs $\mathbf{x}_T \rightarrow \mathbf{x}_{T-1} \rightarrow \ldots \rightarrow \mathbf{x}_1$ (bidirectional RNN).

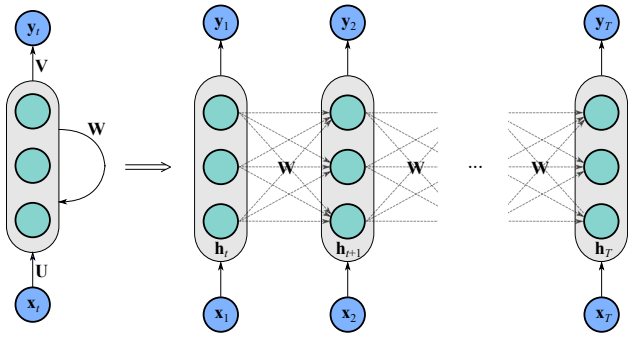


Figure 2. Recurrent Neural Network in folded (left) and unfolded (right) format. Weights $\mathbf{W}$ are recurrent connections that enable modeling temporal data.

4.4.1. Stacked (S) RNN

With stacked RNNs we can define arbitrarily many more layers, but in practice only a couple of layers are added due to diminishing returns of additional layers and the computational burden of training these more complex models. Whether to use more than one layer or not is usually problem-dependent. With more layers, the network should learn be capable of learning increasingly abstract concepts as the layer depth increases. For example, in text processing we start from letters at the input layer, then proceed to learn words, then phrases, and so on at successively higher layers. Eqs. (3) are an extension of Eqs. (2) to include multiple L layers:

$$\begin{aligned} \mathbf{h}_t^{(1)} &= f(\mathbf{W}^{(1)}\mathbf{h}_{t-1}^{(1)} + \mathbf{U}^{(1)}\mathbf{x}_t) \\ \mathbf{h}_t^{(2)} &= f(\mathbf{W}^{(2)}\mathbf{h}_{t-1}^{(2)} + \mathbf{U}^{(2)}\mathbf{h}_t^{(1)}) \\ &\vdots \\ \mathbf{h}_t^{(L)} &= f(\mathbf{W}^{(L)}\mathbf{h}_{t-1}^{(L)} + \mathbf{U}^{(L)}\mathbf{h}_t^{(L-1)}) \\ \mathbf{y}_t &= g(\mathbf{V}\mathbf{h}_t^{(L)}) \end{aligned} \tag{3}$$

where weight matrices $\mathbf{W}^{(j)}$, $\mathbf{U}^{(j)}$, $j \in \{1, \ldots, L\}$ and $\mathbf{V}$ comprise a set of parameters to optimize.

4.4.2. Bidirectional (Bi) RNN

The Bidirectional RNN (BiRNN) (Graves & Schmidhuber, 2005; Bahdanau et al., 2015) enjoys a more powerful information processing model based on the following premise: an event in sequence at step t relates to both its historical context up to step $t-1$, *and* its succeeding context $t+1$ up to T. That is, how does current state $\mathbf{h}_t$

affect the future? For example, in Natural Language Processing (NLP), a word is usually defined in the context of its neighboring words (both before and after), which has inspired several word-embedding representations (Mikolov et al., 2013; Pennington et al., 2014) that are now common preprocessing step for NLP tasks.

A BiRNN consists of two RNNs – forward and backward. The forward RNN reads the input sequence as it is ordered (from 1 to T) and calculates a sequence of forward hidden states $(\overrightarrow{\mathbf{h}_1}, \ldots, \overrightarrow{\mathbf{h}_T})$. The backward RNN reads the sequence in the reverse order (from T to 1), which produces a sequence of backward hidden states $(\overleftarrow{\mathbf{h}_1}, \ldots, \overleftarrow{\mathbf{h}_T})$. Finally, the hidden state in BiRNN is defined as a concatenation of these two sequences at each step, that is, $\mathbf{h}_t = [\overrightarrow{\mathbf{h}_t}; \overleftarrow{\mathbf{h}_t}]$. The update equations remain the same as for the RNN (Eqs. (2)).

4.5. Long-Short Term Memory RNN

One problem with training basic RNN architectures is due to very long-term dependencies in sequences that make effective learning difficult. This is due to the vanishing gradient problem (Hochreiter, 1998) and underlies one of the key motivations for the Long-Short Term Memory (LSTM) cell-type networks (Hochreiter & Schmidhuber, 1997). The LSTM aims to enable constant flow of information by introducing gates – mechanisms within deep neural networks that monitor what information is allowed to be processed. In an LSTM cell, there are three gates: the input, forget, and output gates. The input gate controls which parts of the input are allowed to enter the cell, that is, what should be read from the input pipeline. The forget gate controls the internal cell state by preventing parts of it from being updated, or conversely, determining which parts should be modified. Finally, the output gate controls what is pushed outside the cell on the output pipeline. Eqs. (4) are the most commonly used update rules for the LSTM cell:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}_t + \mathbf{V}_i\mathbf{h}_{t-1}) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{V}_f\mathbf{h}_{t-1}) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{V}_o\mathbf{h}_{t-1}) \\
\bar{\mathbf{c}}_t &= \tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{V}_c\mathbf{h}_{t-1}) \\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t\bar{\mathbf{c}}_t \\
\mathbf{h}_t &= \mathbf{o}_t\tanh(\mathbf{c}_t)
\end{aligned} \tag{4}$$

$\mathbf{i}_t$, $\mathbf{f}_t$ and $\mathbf{o}_t$ are input, forget and output gate, respectively, σ is the sigmoid function, $\bar{\mathbf{c}}_t$ is the candidate cell state, $\mathbf{h}_t$ is the output of the cell at step t and the model is commonly initialized with a zero initial state $\mathbf{c}_0 = \mathbf{0}$. The parameters for optimization involve all $\mathbf{W}$ and $\mathbf{V}$ matrices.

The internal state of the LSTM is usually denoted as $\mathbf{c}_t$ (long-term memory) and its output $\mathbf{h}_t$ (short-term memory), contrasting with the vanilla RNN's notation of state $\mathbf{h}_t$ and output $\mathbf{y}_t$. Where LSTMs are used in thie article, we use the above LSTM notation to be consistent with the literature. Since the LSTM update equations can simply be used to replace the vanilla RNN equations discussed previously (modulo minor notational substitutions used in the LSTM setting), we remark that both the stacking and bidirectionality improvements described earlier can also be applied to RNN-LSTM networks.

There are several variations on these rules depending on the order of updates and whether the cell itself is exposed for calculations (e.g., the so-called "peep-hole" connections). One popular variation is called the Gated-Recurrent Unit (GRU) (Cho et al., 2014) that has sometimes shown similar performance results to the LSTM while requiring less parameters for faster optimization. In this article, we generally found that LSTMs outperformed GRUs, except in the special case of the decoder for the Encoder-Decoder architecture that we cover in the next subsection.

4.6. Encoder-Decoder framework

A different architecture that has recently become dominant in the NLP field – particularly in machine translation – is the Encoder-Decoder model shown in Fig. 3. This architecture has also been shown to work for anomaly detection with sequences (Malhotra et al., 2016). It involves two RNNs, where the complete input sequence is initially processed with the first RNN (encoder) to produce the encoded state $\mathbf{h}^{\mathrm{e}}_T$, while the second RNN (decoder) tries to decode the complete sequence purely based on the given state $\mathbf{h}^{\mathrm{e}}_T$. This is accomplished by initializing the decoder as a function

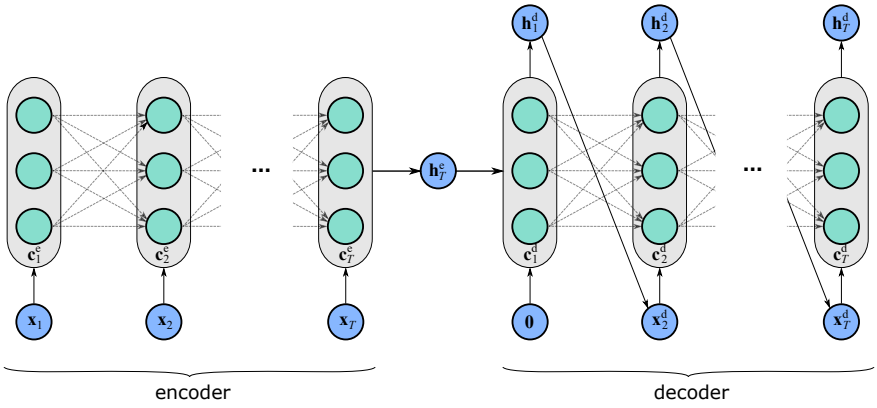


Figure 3. Sequential encoder-decoder architecture. Encoder processes the whole input sequence before decoder reconstructs it back purely based on the embedding output $\mathbf{h}^{\mathrm{e}}_T$. This compositional model consists of two LSTM models and shows single prediction phase where outputs at previous steps are used as inputs at subsequent steps on decoder side. The architecture here assumes both encoder and decoder LSTM have the same number of hidden neurons which can be different in practice.

of the last state of encoder $\mathbf{c}^{\mathrm{d}}_0 = \mathrm{F}(\mathbf{h}^{\mathrm{e}}_T)$. There are no special update rules for this architecture, the only difference is the way the architecture is trained. Since at training time, the correct inputs to each stage of the decoder are known, it is possible to simply use these correct inputs at training time in a process known as *teaching forcing* (Williams & Zipser, 1989). For inference at test time, the decoder reuses its own previous step output as its next step input. For example, if the output at step t is $\mathbf{h}^{\mathrm{d}}_t$, then it is used as input at step $t+1$, $\mathbf{x}^{\mathrm{d}}_{t+1} = \mathbf{h}^{\mathrm{d}}_t$, and similarly for the subsequent steps until the complete sequence is approximately reconstructed.

4.6.1. Attentional Encoder-Decoder

An *attentional* component has been introduced into the encoder-decoder model and successfully applied to text processing (Bahdanau et al., 2015; Luong et al., 2015) and image captioning tasks (Xu et al., 2015). It allows the decoding phase to focus on important, interim states of the encoder instead of only seeing the final state. For example, to produce output at step t it may be helpful to check what the state of the encoder was at step t or $t-1$ when processing the input sequence. The most appealing property of the attentional model is that it produces a mapping between decoded outputs and input steps. Neural networks tend to be black-box models, and this *attentional map* is one way at inference time to produce an interpretation of the model's output based on its input. Additionally, it can improve overall performance compared to non-attentional models (Bahdanau et al., 2015).

For anomaly detection in the cybersecurity domain, the attentional model adds key functionality to a powerful learning algorithm such as deep nets. Provided that the input features have a reasonable interpretation themselves, an attentional component can point to which features are responsible for the produced output thus serving as a type of *attentional explanation* and enabling operators to more swiftly identify the cause of potential anomalies.

Equations for the LSTM decoder have to be modified to accommodate a complete series of encoder states/outputs:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{x}^{\mathrm{d}}_t + \mathbf{V}_i\mathbf{h}^{\mathrm{d}}_{t-1} + \mathbf{C}_i\mathbf{r}_t) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}^{\mathrm{d}}_t + \mathbf{V}_f\mathbf{h}^{\mathrm{d}}_{t-1} + \mathbf{C}_f\mathbf{r}_t) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}^{\mathrm{d}}_t + \mathbf{V}_o\mathbf{h}^{\mathrm{d}}_{t-1} + \mathbf{C}_o\mathbf{r}_t) \\
\bar{\mathbf{c}}_t &= \tanh(\mathbf{W}_c\mathbf{x}^{\mathrm{d}}_t + \mathbf{V}_c\mathbf{h}^{\mathrm{d}}_{t-1} + \mathbf{C}_c\mathbf{r}_t)
\end{aligned} \tag{5}$$

where $\mathbf{r}_t$ is the context vector derived from the encoder outputs defined as follows:

$$\mathbf{r}_t = \sum_{k=1}^{T} \alpha_{tk}\mathbf{h}^{\mathrm{e}}_k \tag{6}$$

with $\mathbf{h}^{\mathrm{e}}_t$ denoting the encoder state at step t and

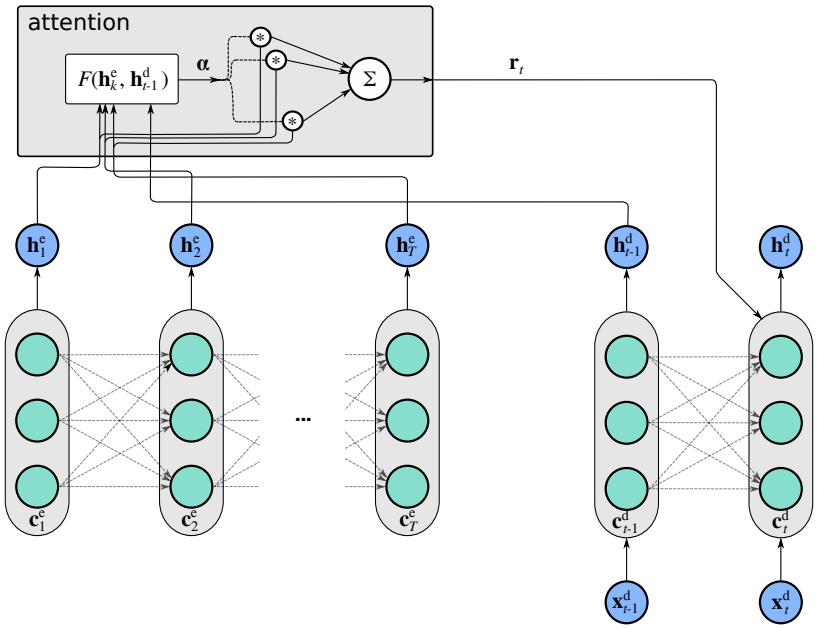


Figure 4. Single decoding step at time t with an attentional component. Each element in the attentional vector α_{tk} multiples corresponding encoder output $\mathbf{h}^{\mathrm{e}}_k$, then those products are summed to produce the context vector $\mathbf{r}_t$. $F(\mathbf{h}^{\mathrm{e}}_k, \mathbf{h}^{\mathrm{d}}_{t-1})$ in our case is a single-layer network that computes *alignment* α_{tk} between previous decoder output $\mathbf{h}^{\mathrm{d}}_{t-1}$ and each encoder output $\mathbf{h}^{\mathrm{e}}_k$.

$$\alpha_{tk} = \frac{\exp(e_{tk})}{\sum_{j=1}^{T} \exp(e_{tj})}, \quad k \in \{1, \ldots, T\} \tag{7}$$
$$e_{tj} = v_a^{\mathrm{T}} \tanh(\mathbf{W}_a \mathbf{h}^{\mathrm{d}}_{t-1} + \mathbf{U}_a \mathbf{h}^{\mathrm{e}}_j), \quad j \in \{1, \ldots, T\}$$

where e_{tj} are energies relating the previous decoder output $\mathbf{h}^{\mathrm{d}}_{t-1}$ and encoder output $\mathbf{h}^{\mathrm{e}}_k$ which are computed using a single layer network with $\mathbf{W}_a$, $\mathbf{U}_a$ and $\mathbf{v}_a$ being the weights of this network. The normalized coefficients $\boldsymbol{\alpha}_t = [\alpha_{t1}, \ldots, \alpha_{tT}]$, $t = 1, \ldots, T$ are computed at every step t and offer a quick way to inspect the most significant contributions from the input sequence. $\mathbf{C}_*$ matrices in Eqs. (5) are additional set of weights being optimized that incorporate information from the context vector $\mathbf{r}_t$. A single decoding step of this architecture is shown in Fig. 4. For experiments in this paper, we use a decoder based on a GRU-type cell (Bahdanau et al., 2015) that outperformed an LSTM-based decoder and has modified update rules as discussed previously. Nonetheless, the idea of a weighted context vector $\mathbf{r}_t$ remains the same.

An important note is the "standard" attentional component described here only produces a mapping between *time steps* of the encoder and decoder. In other words, the full explanation as to why a sample is suspicious only relates to the time component, without any further details. This can be remedied with a set of connections that incorporate *input vectors* directly into the update Eqs. (5) and not just encoder outputs $\mathbf{h}^{\mathrm{e}}_t$. This approach remains to be explored in the future with the modifications to Eqs. (5) given in the Appendix A.

5. Evaluation procedure

As motivated and described in Section 4.2, in this article we use *reconstruction error* derived from any of the aforementioned deep autoencoder architectures to assign an anomaly score to an observed data stream. As part of our evaluation procedure, we construct the ranked list by ordering (in descending order) new test samples according to their reconstruction error. Since network security analysts are inherently time-constrained in how many anomalies they can investigate in a fixed amount of time, our aim is to only examine and evaluate the top-k highly ranked cases.

When considering how to evaluate such top-ranked results, we remark that it is analogous to the evaluation of ranked search engine results. The motivation for this approach is given in the Appendix B where we establish connection between document retrieval and intrusion detection.

In the intrusion detection context, anomalousness indicates relevancy (what we want to be top-ranked), and retrieving a sample in the top-k would mean that the model labels it as anomalous. Hence, we restrict our evaluations to the top-k variants of standard ranking metrics, namely *precision@k* and *average precision@k*, which we formally define next.

Precision@k. In brief, this metric computes the fraction of the top-k ranked items that are actually malicious according to ground truth maliciousness labels known in the test data:

$$Prec(k) = \frac{|\text{malicious}(\Gamma[:k]) \cap \text{labeled malicious}(\Gamma[:k])|}{|\text{labeled malicious}(\Gamma[:k])|}. \tag{8}$$

If there are a total of R malicious items in the ground truth labeled test data, we can let $k = R$ and compute $Prec(R)$ (a.k.a. Precision at Recall). One reason for doing this is that it guarantees that a perfect ranking (all R malicious examples ranked in the top-k) achieves a score of 1.0 so that we can easily compare $Prec(R)$ across different datesets with widely varying values of R.

Average Precision@k. One well-known caveat of $Prec(k)$ is that it provides the same evaluation to a ranked list, no matter how the relevant items are permuted among the top-k items since $Prec(k)$ only measures the *fraction* of top-k items that are known to be malicious. To address this caveat, Average Precision@k is another metric that builds on the $Prec(k)$ definition to provide a higher score to ranked lists which place malicious items higher in the top-k compared to ranked lists which place the same malicious items lower in the top-k. Formally, letting malicious(i) take the value 1 if the ith-ranked item is malicious and 0 otherwise, the definition of Average Precision@k (AP@k) is

$$AP(k) = \frac{1}{k} \sum_{i=1}^{k} Prec(i) \cdot \text{malicious}(i). \tag{9}$$

6. Data

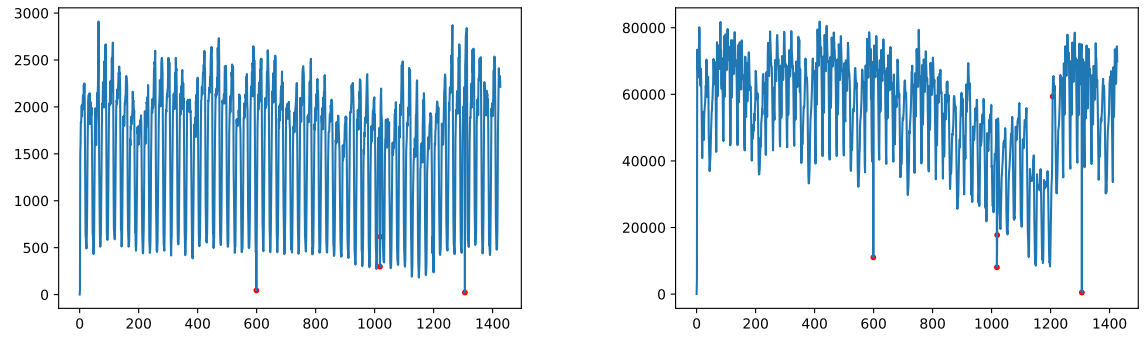
We focus on sequential data sets since the temporal aspect of the data is an important factor for anomaly-based IDS – while a single event may not be anomalous by itself, that same event placed in the context of a sequence of events may indeed be anomalous. In the experiments, we use several sequential data sets discussed more in-depth in subsequent sections:

- Synthetically generated data that allows us to control sequential aspects of anomalies.
- Yahoo! data set for anomaly detection.
- CICIDS2017 flow-based data developed at the University of New Brunswick.
- Real-world data collected by Rank Software Inc.

Except for the synthetic data, the data sets are constructed by aggregating statistical features of the data streams in fixed time intervals that we call time steps. For each time step t, we construct a set of those features and make a sequential sample by stacking T consecutive sets of features. We build train and test sets by selecting different time intervals of T time steps; further details of data set construction are provided in Section 7.1 and a complete summary of the data sets used is given in Table 1.

data set	#instances	D	T	#samples (N)	#malicious
Synthetic	N/A	1	100	1000	50
Yahoo! A1-51	1500	1	24	1477	204
Yahoo! A1-56	1524	1	24	1501	204
Rank	N/A	31	12	7032	46
CICIDS2017	N/A	33	60	146532	26122

Table 1. Summary of data and constructed data sets. #instances is the total number of data points before any preprocessing steps, D is the dimensionality of feature space, T length of sequence, N is total number of data samples available for training and #malicious is the number of malicious samples in the test set. The number of malicious samples is high since actual malicious time step is spread across multiple samples, see Fig. 6.



(a) A1-51 time series

(b) A1-56 time series

Figure 5. Two Yahoo! A1Benchmark time series for anomaly detection selected for experiments. Red points indicate alarms present during hours of operation.

6.1. Synthetic

We introduce an artificial time series which allows us to control sequential aspects of anomalies in order to test generalization of the standard non-sequential autoencoding approach. For this purpose, we created sequences of binary values (0 and 1) having length 100 that contain a certain pattern. Samples with this specific pattern are considered to be normal/benign samples, while samples without the pattern are put into anomalous group. The pattern for normal cases is a separation of 1s with predefined number of 0s which can be 9, 10 or 11. However, both normal and anomalous cases have exactly the same number of ones and zeros – 10 and 90 respectively. The distribution across time remains the same, but the order of values plays an important role.

6.2. Yahoo!

This data set[6] is provided as part of the Yahoo! Webscope program. There are several time-series based data as part of the package, where all personal information is removed, as well as real-data properties and GEOs. We use A1Benchmark set of time-series which are based on the real production traffic to some of the Yahoo! properties. It is hourly data with labels of alarm class. Out of more than 50 time series provided, we only use A1-51 and A1-56 shown in Fig. 5 that are challenging enough that require a learning system rather than visual inspection. Since this data is 1-dimensional $D = 1$ preprocessed data, we only construct the complete data set by sliding a window of size $T = 24$ to represent 1 day's worth of traffic.

[6]https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70&guccounter=1

6.3. Rank

This data set[7] was provided by Rank Software Inc. It contains a month's worth of events collected from the enterprise network of a 50 employee software development organization. Events are produced by monitoring both the network traffic, using Zeek, and also the behavior of processes on individual hosts, using Sysmon[8]. Events from different sources are correlated by their occurrence time. The following events have been extracted: file creation time modification, process creation, process termination, and network flows. We constructed features based on 300 second intervals from the event logs, summarizing both network flows and host events. The list of all features extracted is given in the Appendix C.

No known malicious activity occurred in the network during the data set's time interval, so labeled malicious activity was added to the data set. The only attack type introduced was the installation of Nmap and the execution of network scans over two time intervals (30 minute and 2 hour intervals). The malicious activity was not performed in the real network. Instead, the activity was done on virtual machines in an isolated virtual network which was monitored in the same manner as the real network. The identifiers and times of the resulting events were modified to match the identifiers and time interval in the rest of the data set.

This approach allows using the actual tools that an attacker might use, without the risk of disrupting or damaging the real network. Network scanning alone has a low risk of damaging a network, but further attack types are planned. Network scanning behavior should also be relatively easy to detect as it involves an unusual increase in the number of attempted and connected network flows from one process on one host to distinct host IPs and ports.

6.4. CICIDS2017

This data set (Sharafaldin et al., 2018) has been generated to resemble true real-world data based on several criteria considered necessary for building a reliable benchmark data set (Gharib et al., 2016). It is based on abstract behavior of human interactions that constitutes benign background traffic. The complete simulation takes into account 5 working days (Monday through Friday) and common attack types have been generated alongside benign traffic during the last four days.

For the purposes of this paper, we used a modified CICFlowMeter tool to fetch information of interest and construct sequences of fixed-time features, as opposed to flow-based features that is inherent for CICFlowMeter. This modified version is run on raw PCAP files and we calculate the overall statistics of all the flows within *1 second* interval. For example, we compute the total number of flows started, number of stopped flows and number of different flow flags to name a few. A complete list of all extracted features for this data is provided in the Appendix C. Complete data generation contains around 40 hours worth of traffic flow and we use $T = 60$ to represent a 60 second window. This window size is chosen to reflect that the majority of attack types introduced to the system are rather long-term in nature (reaching up to 2 minute duration) as opposed to the overall short-term nature of flows (less than a second).

7. Experiments

The primary experimental question we would like to evaluate is whether the sequential nature of data plays an important role when choosing the best anomaly detection model and, if so, which deep autoencoder architecture for scoring anomalies according to reconstruction error performs best? Given that fully connected autoencoders are known to be a powerful tool for anomaly detection, we experimentally assess the utility of RNN-based autoencoder models and their variations in comparison to a fully connected baseline model, referred to simply as "Autoencoder" in the experiments.

A secondary question is whether the attentional encoder-decoder model is able to provide guidance towards the underlying causes of maliciousness. Because the attentional component is only tied to the temporal dimension, we are interested in seeing whether the malicious segments within the T-length window can be identified from the attention they receive during the decoding phase.

[7] Available by request

[8] https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon

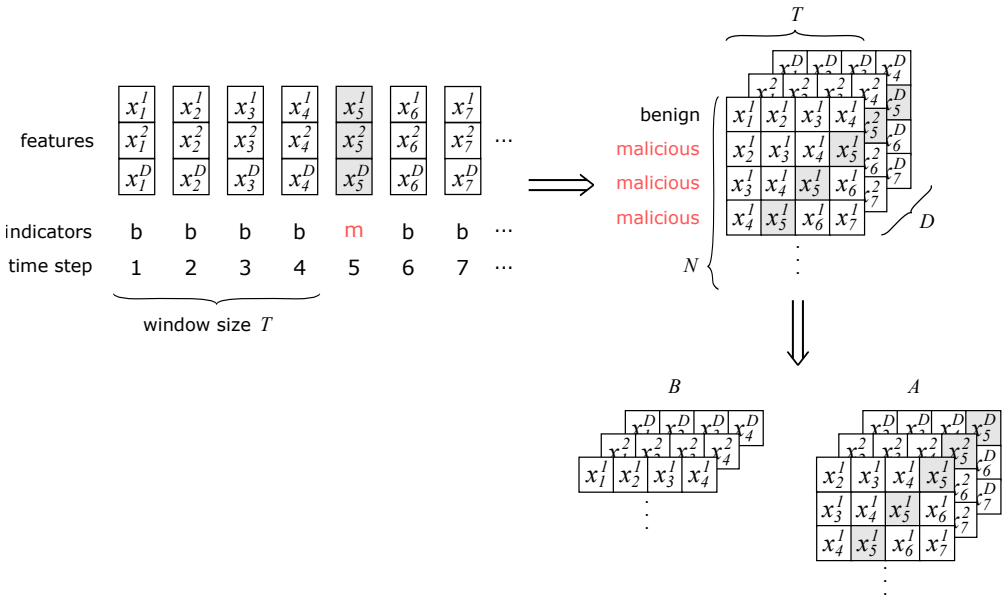


Figure 6. Data set construction process from a sequence of D-dimensional features. The example in the figure is for $D = 3$ and $T = 4$.

All the experiments are performed on a NVIDIA GTX1080Ti GPU-based machine using TensorFlow[9] and Keras[10] libraries.

7.1. Setup

Sequential data is transformed into 3D tensors, with size [samples, time steps, features] = $[N, T, D]$, which in the case of $D = 1$ simply yields a matrix (i.e., a 2D tensor). For fully connected autoencoders we do not consider the temporal dimension, and instead use a matrix of size $[N, T \cdot D]$. The specific values for N, T and D in each data set are given in Table 1.

Data splitting. Each data set is comprised of both benign B and malicious A samples. Set A is only used at test time, while benign set B is divided into a training part B_{train} and a test part B_{test}. A sample is malicious if any segment along temporal dimension t contains traces of maliciousness, that is, if any segment within a sample contains a time frame where an attack occurs, the complete sample is considered malicious. Otherwise, if none of the segments in the sample contain traces of attacks, it is considered benign. Fig. 6 depicts this data creation process alongside benign/malicious labeling of samples.

Parameter tuning. Hyper-parameters for the models (number of neurons n, learning rate α) are chosen via a 10-fold cross-validation procedure using only benign training data B_{train} and using (lower) reconstruction error as the validation criterion for selecting hyper-parameters. For most of the data sets, the number of samples is rather small compared to contemporary deep learning benchmark data sets, and we resort to architecturally rather shallow networks. All cross-validated hyper-parameters and the values tested are given in the Appendix D. The exception is CICIDS2017 data for which we chose a reasonable set of values that provided quick learning, but still resorted to having a stopping condition by monitoring the validation loss. The optimization is performed with Adam gradient method (Kingma & Ba, 2015).

[9] https://www.tensorflow.org/
[10] https://keras.io/

Model	Data set				
	Synthetic	A1-51	A1-56	Rank	CICIDS
Autoencoder	.890 ± .033	.642 ± .000	.535 ± .004	.774 ± .022	.525
LSTM	.898 ± .100	.644 ± .004	.556 ± .007	**.883** ± .032	.764
Bi-LSTM	.862 ± .131	.645 ± .004	**.561** ± .005	.861 ± .030	.785
S-LSTM	.820 ± .115	**.647** ± .006	.555 ± .009	.835 ± .041	**.807**
Bi-S-LSTM	.812 ± .107	.645 ± .003	.557 ± .013	.876 ± .026	.729
Encoder-Decoder	**.900** ± .116	.642 ± .000	.553 ± .008	.846 ± .031	.796
Attention ED	.464 ± .179	.643 ± .001	.538 ± .006	.802 ± .030	.499

Table 2. Precision at recall for all models and data sets. Bi stands for bidirectional, and S for stacked. Values represent average across 10-fold validation with standard deviation after the ± sign. For CICIDS data we only ran the models once.

Model	Data set				
	Synthetic	A1-51	A1-56	Rank	CICIDS2017
Autoencoder	.862 ± .050	.348 ± .001	.273 ± .003	.704 ± .024	.380
LSTM	.895 ± .105	.365 ± .007	.287 ± .004	**.864** ± .037	.674
Bi-LSTM	.843 ± .157	.364 ± .007	**.289** ± .004	.827 ± .045	.719
S-LSTM	.817 ± .117	**.368** ± .007	.286 ± .005	.753 ± .056	**.728**
Bi-S-LSTM	.798 ± .111	.356 ± .005	.287 ± .009	.827 ± .042	.642
Encoder-Decoder	**.895** ± .125	.354 ± .005	.284 ± .005	.789 ± .045	.715
Attention ED	.308 ± .209	.351 ± .002	.276 ± .005	.756 ± .041	.349

Table 3. Average precision at recall for all models and data sets. Bi stands for bidirectional, and S for stacked.

Model training. Once validation is done, the best performing combination of hyper-parameters is selected as the final set of parameters for a specific model. Training is done by using 80% of benign data, while the other 20% is used for monitoring the learning phase and early stopping. Both parts are based on the B_{train} set.

Evaluation. With the trained model, we compute the metrics on the held-out benign data (not used during previous two phases) and all malicious data, that is, $B_{\text{test}} \cup A$. We report $Prec(k)$, $Prec(R)$, $AP(k)$ and $AP(R)$ for all the models and data sets. R is the number of malicious samples given in Table 1.

7.2. Results

7.2.1. Sequential models and autoencoders

Tables 2 and 3 show $Prec(R)$ and $AR(R)$ for all data sets, respectively, while Figs. 7 and 8 show $AP(R)$ from different perspectives for easier comparison. If a ranking method is perfect (i.e., all R anomalies are ranked in the top-k) then both $Prec(R)$ and $AP(R)$ will be 1. An overall initial impression from the results is that sequence-based models are more suitable for the task at hand, with only marginal differences between them. The pure fully-connected Autoencoder still appears to be a reasonable approach in terms of computed metrics, but overall *lags behind most sequential methods* where it only outperforms the Attentional Encoder-Decoder. Somewhat surprisingly, the *Attentional Encoder-Decoder model underperforms* compared to its non-attentional variant, and completely fails in two cases. We conjecture that despite its explanatory promise, the attentional aspect adds complexity to these models and seems hinder the training process. The pure Encoder-Decoder models remain a reasonable choice as they offer comparable performance to standard LSTM-based models and perform best on one data set (Synthetic).

For both Yahoo! data sets, A1-51 and A1-56, all the models have difficulties distinguishing between benign and anomalous samples, which is indicated by rather low R-precision scores (around 0.64 and 0.55). This means that roughly 60% and 50% in the first R samples are normal/benign cases. Considering that other data sets have higher values for both $Prec(R)$ and $AP(R)$, it looks that normal and anomalous cases for Yahoo! data are very similar to each other. With a smaller number of samples for training and the low dimensionality $D = 1$ for both Yahoo! data

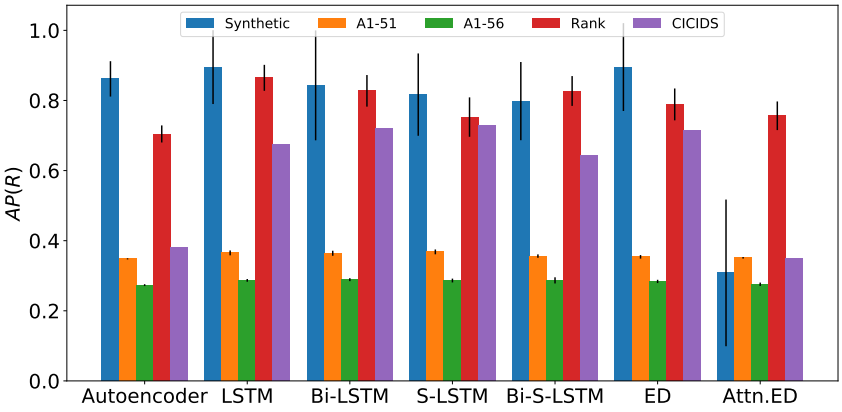


Figure 7. Average precision at recall for all models and data sets. Bi stands for bidirectional, S for stacked, ED for Encoder-Decoder and Attn.ED for Attentional Encoder-Decoder. Black vertical lines represent one standard deviation from the mean, and this information is absent for CICIDS data.

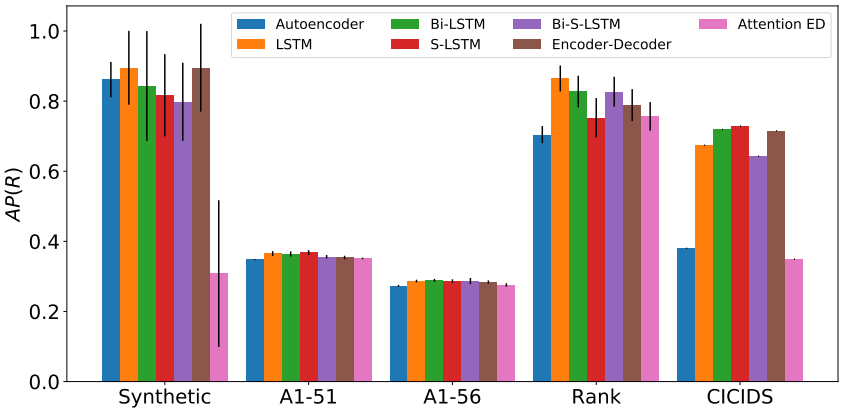


Figure 8. Average precision at recall for all models and data sets. Bi stands for bidirectional and S for stacked. Standard deviation is omitted for clarity. Black vertical lines represent one standard deviation from the mean which is absent for CICIDS data.

sets A1-51 and A1-56, all models are able to reliably capture a distribution that includes a significant portion of the malicious cases. Performance is stronger across all models for the Synthetic data, where the normal and anomalous dynamics are manually controlled. For Rank and CICIDS data, there is enough information in the $D > 30$ features to allow the models to distinguish between the two cases.

Overall, the two most promising models based on the results are Stacked and Bidirectional networks with the Stacked version being slightly better. The combined version (Bidirectional-Stacked-LSTM) does not further improve performance, suggesting that data complexity is fully captured with either an additional layer (stacked), or providing reversed inputs (bidirectional). On the other hand, if we exclude CICIDS data, the pure LSTM model offers the most stable results, i.e., it provides the best performance on Rank data while being very close to the top-performing models on the remaining data sets.

7.2.2. Quality of rankings

Fig. 9 offers more insight into the utility of the rankings produced by the models. The figures show the average precision while varying k, the number top-ranked elements considered. Here, we are focusing on a small subset of the anomalies compared to the overall number of anomalies (given in Table 1) with the rationale that network security analysts are rarely able to investigate more than 100 reports in a given day (or perhaps even week). The only exception is CICIDS data where we chose an optimistic large upper limit in order to observe the differences between the models.

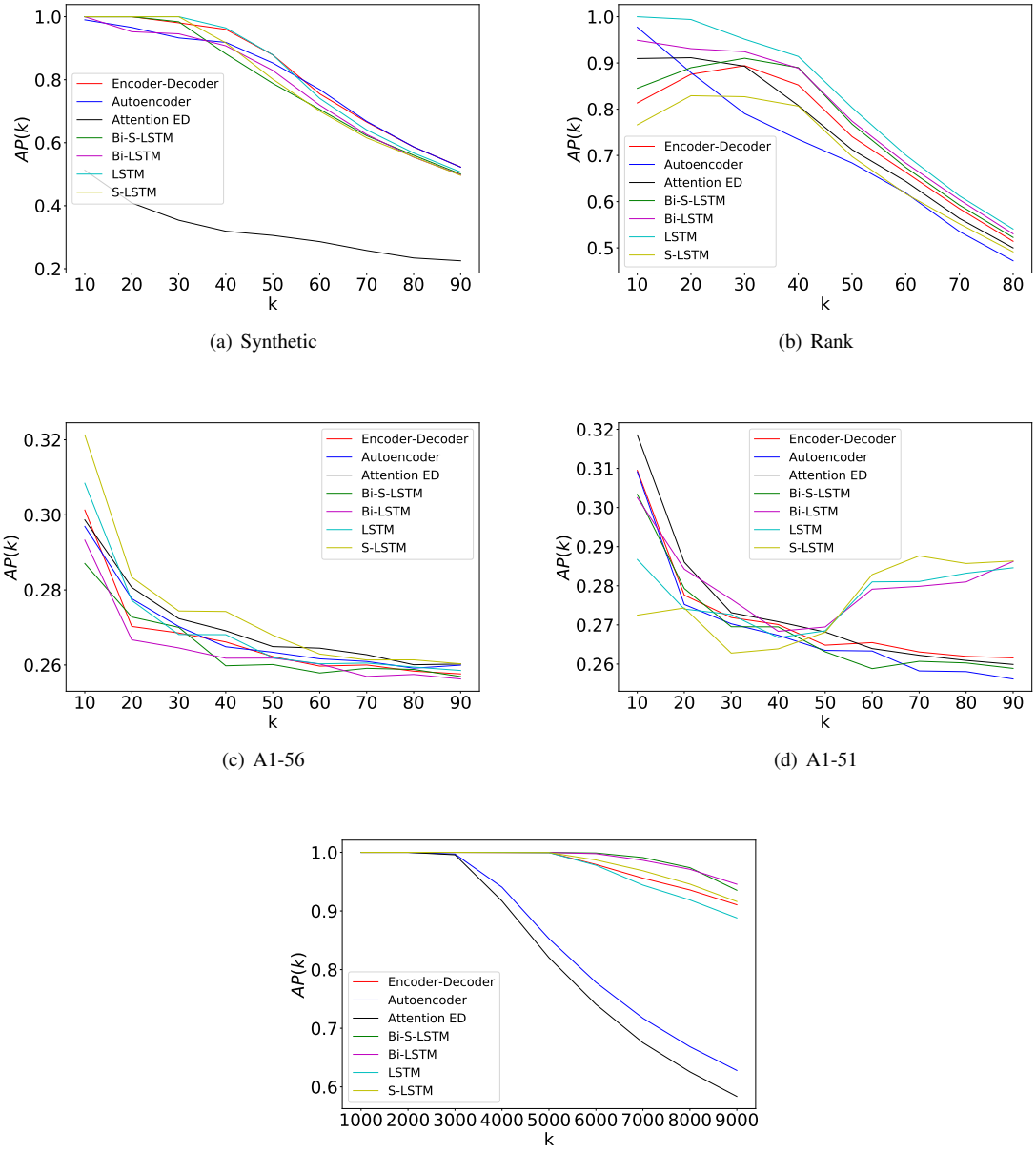
All sequential models (excluding the attentional model) seem to be close to each other in terms of performance; i.e, which model is best depends on the data set without a clear winner across data sets. For the Synthetic case, the Encoder-Decoder provides the most stable detection across this particular range for k, while for Rank the pure LSTM network is the most effective. For the Yahoo! data, we observe some abrupt changes over the interval of k shown, but the changes are in quite a narrow range ($[0.48, 0.53]$ for A1-56 data, and $[0.46, 0.54]$ for A1-51 data). These two data sets are more difficult to model as the number is samples is a limiting factor. The Stacked LSTM seems to be more appropriate for the first 100 samples, even though the stacked LSTM gives the lowest value for A1-51, while it alleviates this deficiency for the latter values. Finally, for the CICIDS data, we have the bidirectional models (both vanilla and stacked) outperforming other models. In this case, with a large number of samples, we notice that providing both forward and reverse context (i.e., bidirectionality) for the reconstruction improves the final outcomes.

Average Precision@k assesses the quality of rankings, which are highly correlated with the Precision@k plots, but there are noticeable subtleties. For A1-56, the Stacked LSTM clearly outperforms all other models, even though the overall results are not impressive with such low values compared to other data sets. For the Rank data set, the vanilla LSTM is again clearly the best model. For Synthetic data, the Encoder-Decoder and Autoencoder models outperform the rest. For A1-51, an Attentional Encoder-Decoder is rather surprisingly showing as the best for initial values of k, while the latter part of the curve is dominated by the Stacked LSTM, vanilla LSTM and Bidirectional LSTM. For CICIDS, the bidirectional models are outperforming all other approaches as observed previously.

If we consider only real-world data (the Rank and CICIDS data sets), we can see that the bidirectional models perform comparably to or better than the others. Although the vanilla LSTM is the best model for the Rank data set, both the bidirectional LSTM and bidirectional-stacked-LSTM are very close in comparison.

7.2.3. Attentional component and explainability

Based on the tables and figures, the attentional version of the Encoder-Decoder provides surprisingly poor results. In this particular scenario, we conjecture that the attentional component acts more as a constraint that is difficult to satisfy rather than augmenting the decoding phase with additional information. The difference between successful applications of attention in the literature and our use case here is the reconstruction problem we are adopting. This is exemplified by the trivial explanation shown in Fig. 10. The figure shows an attention map for one specific anomalous case in Rank data. Unsurprisingly, the best time step to use on the decoder's side is the same time step from the encoder's side. From a reconstruction perspective, this is a logical outcome, but from an explainability perspective, we do not gain any useful information that would benefit network security operators. In short, the attentional component focuses on information (timesteps) that is *beneficial for the current decoding step*, rather than pointing out which steps are potential outliers worthy of further inspection.



(a) Synthetic

(b) Rank

(c) A1-56

(d) A1-51

(e) CICIDS

Figure 9. Average Precision@k for different values of k.

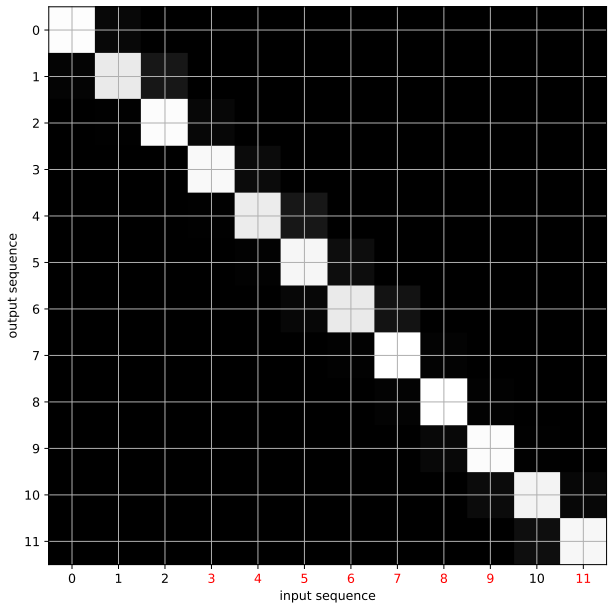


Figure 10. Attention map for an input sequence containing several anomalous time steps (red). White colors indicates higher values (maximum of 1). Numbers of both axes indicate timesteps starting at 0. The top-left square is input/encoder step at 0 and output/decoder step at 0. Attention component does not have any restrictions in terms of focus, that is, each decoding step can use *all* other timesteps from the encoder's end.

7.3. Discussion

7.3.1. Comparison to other methods

A direct comparison to other approaches in the literature is not possible since almost all published results focus on using labeled data and (supervised) classification-based metrics. In this paper, we argue that such an approach cannot be fully adopted in an operating environment and we suggest alternative metrics for evaluating performance of models. Operators and analysts have access to a substantial palette of different types of data, and most of the time, the data is unlabeled, noisy, and sequential but otherwise unstructured – characteristics that we tried to mimic in our evaluation.

7.3.2. Time series approach to modeling

One potential downside of our methodology is the break-up of data by time windows of specific (fixed) length. When a malicious event occurs in data, it is split across several, if not up to hundreds of samples. The models can in turn put all these samples at the top of the ranked list, thereby inflating the ranking scores. To avoid such pathological cases, we would instead need some way to split data based on events and aggregate data in a way that included all relevant information to predicting anomalousness of the event. However, how to do this is not always obvious. For example, the CICIDS data consists of network-related flows of packets, which would have to be accounted for on an individual basis. Similarly, the Rank data is a collection of heterogeneous observations originating from several different monitoring sensors (including network flows, file related events and process related events) where aggregation by event is not obvious.

While our autoencoding methodology can in principle incorporate any form of data, a common denominator in all network data is the time dimension and it seems natural to aggregate over this dimension. In an actual operating environment, one would choose the time-window pertinent to that particular environment, and based on the security analyst's needs – how often an IDS should return reports about the current state of the system? The one second window specified for CICIDS data in our experiment may prove to be inappropriate in practice, but given the volume of data in this particular case, this value was chosen to capture the overall short-term nature of network flows.

7.3.3. Operational deployment

Until now, our primary focus has been on investigating different sequential deep neural network architectures and providing a model enabling insight into its decisions, while actual operating implementation was of secondary concern. However, an important question for all of the models evaluated in this article is whether they can be deployed in a real operating environment and the resulting time and space complexity of doing so. While the volumes of modern network data available combined with the computational expense of deep learning may suggest these models could be impractical in deployed settings, the success of deep learning has led to new hardware solutions that have been designed to meet the computational demands of such models: new GPU models optimized for high parallel data processing, as well as cloud-based services offering resources on demand. A potential issue that still arises in an operating environment, however, is that data is not fixed-length, but comes in streams, requiring constant updates or re-training of models. This online scenario has already been touched upon in (Tuor et al., 2017) suggesting to expose each sample *only once* to non-sequential models, and augmenting sequential RNNs with auxiliary structures having their own update policies.

7.3.4. Further limitations

Data preprocessing and feature engineering are still used in our framework in order to provide meaningful representation to deep neural networks. As the complexity of monitoring systems increases, more data is ingested and needs to be checked whether it is beneficial for the model or not. It is difficult to envision an automatic mechanism for converting raw log files to numerical representations without the human intervention that would be applicable to all data sources. In the Appendix C, we have provided a list of constructed features used in the experiments, which are only applicable to our particular case and are not easily transferable to other use-cases. This is further complicated by the fact that log data comes from heterogeneous sources that might necessitate building completely disjoint models and requiring a unifying manager to aggregate over those models.

For security analysts, the most useful piece of information is whether a presence of a particular Advanced Persistent Threat (APT) is seen within the monitored system, for example, data exfiltration. These threats (or actual attacks) can also span a window of several days, and not just a single time step in the way we developed our framework. As such, threats usually contains multiple alerts that can be unrelated to each other. In this study, we are primarily focusing on raising alerts via anomaly based detection, while merging of alerts and threat recognition requires a higher level framework.

As mentioned in Section 4.6.1, the attentional encoder-decoder architecture is only able to highlight *time-steps* of significance. Although useful, each time step contains all the constructed features which may not contain enough information for the security analyst to start examining the anomaly. A more insightful clue would be to highlight features related to specific attributes, for example, file-related or network-related features, to give an incentive to the analyst where to start their investigation.

8. Conclusion

In this paper, we presented an experimental comparison of different deep learning architectures for the purpose of detecting anomalous behavior in a cyber-security setting. Our methodology is oriented around *autoencoder reconstruction modeling* where we aim to capture the underlying *normal* data distribution and then detect future deviations from this distribution. The driving assumption behind this approach is that a model trained on normal data (network traffic) should have difficulties reconstructing anomalous samples coming from a different distribution. Adopting this approach we are able to avoid the pitfalls arising from (supervised) classification-based strategies: 1) labeled data is difficult to acquire in many cases, 2) the classification approach can only distinguish between classes it was trained on, and 3) binary classification outcomes do not offer additional insight into the most dangerous threats in the system.

In order to assess reconstruction quality and detection rate for anomalous cases, we focused on ranking metrics that provide higher scores for systems that place true anomalies at higher ranks – an important metric for end-user network security analysts. Based on our analysis, sequential models prove to be more suitable for this ranking task than the arguably more popular fully-connected autoencoder approaches. Out of several tested variants, the most

promising deep architecture for anomaly detection is a stacked recurrent network, where additional recurrent layers offer improved distribution modeling in comparison to bidirectional connections and the encoder-decoder framework.

While sequential deep learning models prove effective at anomaly detection, an open question remains how to explain these anomalies to end users. The attentional model has an appealing property to provide explanations at each time step, however the particular anomaly detection model we used proved difficult to train for the reconstruction task and was not able to improve upon its non-attentional encoder-decoder variant. Future work should investigate improved training of such models as well as attentional components that explicitly focus on identifying anomalous segments of input data to pinpoint sources of anomalies for end users.

A major potential avenue for future research lies in exploring different forms of the attentional component for explanation. In this paper, we have utilized a version of attention that relies on a *forward-pass* through the network, that is, the attentional vector is computed at the same time as the reconstruction output. On the other hand, studies focusing on image processing (Selvaraju et al., 2017; Bau et al., 2017) primarily utilize a *backward-pass* through the network. In this second approach, a layer of interest is selected first, and then a modified backpropagation algorithm is applied in order to indicate which input features are responsible for each layers' activations. Since convolutional layers are dominant for image processing, identifying relationships between image sections and layer activations is more natural. In the context of streaming network data used in this article, it would be interesting to explore whether this backward-pass approaches carries more utility for establishing relationships between input features and final reconstruction values that would lead to improved attention-based explanations.

Another area for future investigation are model-agnostic explanation frameworks (Ribeiro et al., 2016, 2018; Lundberg & Lee, 2017) that leverage training data to establish a basis or neighbourhood around a test instance with the aim to (a) identify the most influential features for the classification of the test instance and (b) understand how the classification changes with respect to perturbations of the input data. The advantage of these frameworks is their flexibility to work with any black-box machine learning model. In this model-agnostic explanation scenario, deep learning architectures would not necessarily require any form of attentional component to support explanation.

As a final direction for future work, we could attempt to circumvent the need for manual feature engineering by training our anomaly detection algorithms to work directly with log data. That is, we could ingest log data in its textual form, and focus on deep architectures that are commonly used for learning from text sequences. Such an approach directly supports the use of heterogeneous sources of data since log files returned by all sources are primarily text-based. Even quantitative non-log based operational data, numerical values can be tokenized and log files can be treated as text documents (Ring et al., 2017). While such a text log-based approach would require custom architectures and vast volumes of data to train well, it also holds the promise of relaxing a number of limitations of existing work.

References

Abraham, A., Grosan, C., & Martin-Vide, C. (2007). Evolutionary design of intrusion detection programs. *IJ Network Security*, *4*, 328–339.

Aldweesh, A., Derhab, A., & Emam, A. Z. (2020). Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues. *Knowledge-Based Systems*, *189*, 105–124. doi:https://doi.org/10.1016/j.knosys.2019.105124.

Alom, M. Z., Bontupalli, V., & Taha, T. M. (2015). Intrusion detection using deep belief networks. In *2015 National Aerospace and Electronics Conference (NAECON)* (pp. 339–344). Dayton, OH, USA. doi:10.1109/NAECON.2015.7443094.

Atlantic Council (2017). http://www.publications.atlanticcouncil.org/cyberrisks/. Accessed 16 July 2018.

Aydın, M. A., Zaim, A. H., & Ceylan, K. G. (2009). A hybrid intrusion detection system design for computer network security. *Computers & Electrical Engineering*, *35*, 517 – 526. doi:https://doi.org/10.1016/j.compeleceng.2008.12.005.

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. *CoRR*, *abs/1409.0473*.

Bau, D., Zhou, B., Khosla, A., Oliva, A., & Torralba, A. (2017). Network dissection: Quantifying interpretability of deep visual representations. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 6541–6549).

Bilge, L., Kirda, E., Kruegel, C., & Balduzzi, M. (2011). EXPOSURE : Finding malicious domains using passive DNS analysis. In *NDSS 2011, 18th Annual Network and Distributed System Security Symposium*. San Diego, CA, USA.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Bivens, A., Palagiri, C., Smith, R., Szymanski, B., & Embrechts, M. (2002). Network-based intrusion detection using neural networks. *Intelligent Engineering Systems through Artificial Neural Networks*, *12*, 579–584.

Blowers, M., & Williams, J. (2014). Machine learning applied to cyber operations. In R. E. Pino (Ed.), *Network Science and Cybersecurity* (pp. 155–175). New York, NY: Springer. doi:10.1007/978-1-4614-7597-2_10.

Buczak, A. L., & Guven, E. (2016). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, *18*, 1153–1176.

Chandola, V., Banerjee, A., & Kumar, V. (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR)*, *41*, 1–58. doi:10.1145/1541880.1541882.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1724–1734). Association for Computational Linguistics. doi:10.3115/v1/D14-1179.

Creech, G., & Hu, J. (2013). Generation of a new ids test dataset: Time to retire the kdd collection. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)* (pp. 4487–4492). Shanghai, China. doi:10.1109/WCNC.2013.6555301.

Cybersecurity Ventures (2017). 2017 cybercrime report. https://cybersecurityventures.com/2015-wp/wp-content/uploads/2017/10/2017-Cybercrime-Report.pdf. Accessed 17 July 2018.

Diro, A. A., & Chilamkurti, N. (2018). Distributed attack detection scheme using deep learning approach for internet of things. *Future Generation Computer Systems*, *82*, 761 – 768. doi:https://doi.org/10.1016/j.future.2017.08.043.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*, 179–211.

Erfani, S. M., Rajasegarar, S., Karunasekera, S., & Leckie, C. (2016). High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, *58*, 121–134.

Gao, N., Gao, L., Gao, Q., & Wang, H. (2014). An intrusion detection model based on deep belief networks. In *2014 Second International Conference on Advanced Cloud and Big Data (CBD)* (pp. 247–252). Huangshan, China.

Gharib, A., Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2016). An evaluation framework for intrusion detection dataset. In *2016 International Conference on Information Science and Security (ICISS)* (pp. 1–6). Pattaya, Thailand. doi:10.1109/ICISSEC.2016.7885840.

Glasser, J., & Lindauer, B. (2013). Bridging the gap: A pragmatic approach to generating insider threat data. In *2013 IEEE Security and Privacy Workshops (SPW)* (pp. 98–104). San Francisco, CA, USA. doi:10.1109/SPW.2013.37.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, *18*, 602 – 610. doi:https://doi.org/10.1016/j.neunet.2005.06.042.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*, 1735–1780.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*, 107–116.

Hu, W., Liao, Y., & Vemuri, V. R. (2003). Robust support vector machines for anomaly detection in computer security. In *Proceedings of 2003 International Conference on Machine Learning and Applications* (pp. 168–174). Los Angeles, California, USA.

Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016). A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies* BICT'15 (pp. 21–26). New York City, United States. doi:10.4108/eai.3-12-2015.2262516.

Jordan, M. I. (1997). Chapter 25 - serial order: A parallel distributed processing approach. In J. W. Donahoe, & V. P. Dorsel (Eds.), *Neural-Network Models of Cognition* (pp. 471 – 495). North-Holland volume 121 of *Advances in Psychology*. doi:https://doi.org/10.1016/S0166-4115(97)80111-2.

Joshi, S. S., & Phoha, V. V. (2005). Investigating hidden markov models capabilities in anomaly detection. In *Proceedings of the 43rd Annual Southeast Regional Conference, Vol. 1* (pp. 98–103). Kennesaw, Georgia, USA.

Kim, G., Lee, S., & Kim, S. (2014). A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, *41*, 1690 – 1700. doi:https://doi.org/10.1016/j.eswa.2013.08.066.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of 3rd International Conference on Learning Representations (ICLR)*.

Krizhevsky, A., & Hinton, G. E. (2011). Using very deep autoencoders for content-based image retrieval. In *Proceedings of 19th European Symposium on Artificial Neural Networks (ESANN)*. Bruges, Belgium.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.

Li, Y., Ma, R., & Jiao, R. (2015). A hybrid malicious code detection method based on deep learning. *International Journal of Software Engineering and its Applications*, *9*, 205–216.

Lippmann, R. P., & Cunningham, R. K. (2000). Improving intrusion detection performance using keyword selection and neural networks. *Computer networks*, *34*, 597–603.

Ludwig, S. A. (2017). Intrusion detection of multiple attack classes using a deep neural net ensemble. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)* (pp. 1–7). Honolulu, HI, USA: IEEE.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 4765–4774). Curran Associates, Inc.

Luong, T., Pham, H., & Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* (pp. 1412–1421). Lisbon, Portugal: Association for Computational Linguistics. doi:10.18653/v1/D15-1166.

Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*, .

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*. Cambridge, UK: Cambridge University Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, *abs/1301.3781*.

Modi, C., Patel, D., Borisaniya, B., Patel, H., Patel, A., & Rajarajan, M. (2013). A survey of intrusion detection techniques in cloud. *Journal of Network and Computer Applications*, *36*, 42–57.

Moustafa, N., & Slay, J. (2015). Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 Military Communications and Information Systems Conference (MilCIS)* (pp. 1–6). Canberra, ACT, Australia.

Moustafa, N., Creech, G., & Slay, J. (2017). Big data analytics for intrusion detection system: Statistical decision-making using finite dirichlet mixture models. In I. Palomares Carrascosa, H. K. Kalutarage, & Y. Huang (Eds.), *Data Analytics and Decision Support for Cybersecurity: Trends, Methodologies and Applications* (pp. 127–156). Cham: Springer. doi:10.1007/978-3-319-59439-2_5.

Mukkamala, S., Janoski, G., & Sung, A. (2002). Intrusion detection using neural networks and support vector machines. In *Proceedings of the 2002 International Joint Conference on Neural Networks* (pp. 1702–1707). Honolulu, HI, USA volume 2. doi:10.1109/IJCNN.2002.1007774.

Muna, A.-H., Moustafa, N., & Sitnikova, E. (2018). Identification of malicious activities in industrial internet of things based on deep learning models. *Journal of Information Security and Applications*, *41*, 1–11.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1532–1543). Doha, Qatar.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* KDD '16 (pp. 1135–1144). doi:10.1145/2939672.2939778.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2018). Anchors: High-precision model-agnostic explanations. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence* (pp. 1527–1535).

Ring, M., Dallmann, A., Landes, D., & Hotho, A. (2017). Ip2vec: Learning similarities between ip addresses. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)* (pp. 657–666). doi:10.1109/ICDMW.2017.93.

Ring, M., Wunderlich, S., Scheuring, D., Landes, D., & Hotho, A. (2019). A survey of network-based intrusion detection data sets. *Computers & Security*, *86*, 147 – 167. doi:https://doi.org/10.1016/j.cose.2019.06.005.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1* chapter Learning Internal Representations by Error Propagation. (pp. 318–362). Cambridge, MA, USA: MIT Press.

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). Grad-cam: Visual explanations from deep networks via gradient-based localization. In *The IEEE International Conference on Computer Vision (ICCV)* (pp. 618–626).

Seok, S., & Kim, H. (2016). Visualized malware classification based-on convolutional neural network. *Journal of the Korea Institute of Information Security and Cryptology*, *26*, 197–208.

Sequeira, K., & Zaki, M. (2002). Admit: anomaly-based data mining for intrusions. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 386–395). Edmonton, Alberta, Canada.

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP,* (pp. 108–116). Funchal, Madeira, Portugal.

Shiravi, A., Shiravi, H., Tavallaee, M., & Ghorbani, A. A. (2012). Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security*, *31*, 357 – 374. doi:https://doi.org/10.1016/j.cose.2011.12.012.

Shone, N., Ngoc, T. N., Phai, V. D., & Shi, Q. (2018). A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *2*, 41–50. doi:10.1109/TETCI.2017.2772792.

Sommer, R., & Paxson, V. (2010). Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the IEEE Symposium on Security and Privacy 2010* (pp. 305–316). Oakland, California, USA.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. A. (2009). A detailed analysis of the kdd cup 99 data set. In *2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications* (pp. 1–6). Ottawa, ON, Canada. doi:10.1109/CISDA.2009.5356528.

Tsai, C.-F., & Lin, C.-Y. (2010). A triangle area based nearest neighbors approach to intrusion detection. *Pattern recognition*, *43*, 222–229.

Tuor, A., Kaplan, S., Hutchinson, B., Nichols, N., & Robinson, S. (2017). Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In *Workshop on Artificial Intelligence and Cyber Security* (pp. 224–231). AAAI.

Wang, Z. (2018). Deep learning-based intrusion detection with adversaries. *IEEE Access*, *6*, 38367–38384. doi:10.1109/ACCESS.2018.2854599.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*, 270–280. doi:10.1162/neco.1989.1.2.270.

Xiang, J., Westerlund, M., Sovilj, D., & Pulkkis, G. (2014). Using extreme learning machine for intrusion detection in a big data environment. In *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop* AISec '14 (pp. 73–82). doi:10.1145/2666652.2666664.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning* (pp. 2048–2057). Lille, France.

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, *5*, 21954–21961.

Zhang, J., Zulkernine, M., & Haque, A. (2008). Random-forests-based network intrusion detection systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *38*, 649–659.

Appendix A. Input based attention

We can introduce the inputs to the computation of context vector $\mathbf{r}_t$ Eq. (6) in the following manner for $\mathbf{x}_t = [x_t^1, \ldots, x_t^D]$:

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})}, \quad j \in \{1, \ldots, T\} \tag{A.1}$$

$$e_{tkj} = v_a^{\mathrm{T}} \tanh(\mathbf{W}_a \mathbf{h}_{t-1}^{\mathrm{d}} + \mathbf{U}_a \mathbf{h}_k^{\mathrm{e}} + \mathbf{Z}_a C(x_k^l)), l \in \{1, \ldots, D\} \tag{A.2}$$

$$e_{tk} = \sum_{j=1}^{D} e_{tkj} \tag{A.3}$$

where C is expansion operator that transforms a scalar value to a vector of those scalars. Then for every decoding step, we have a complete matrix measuring the influence of every step *and* every feature within those steps. Comparing to Eqs. (7), $\mathbf{Z}_a$ is a matrix representing new learnable weights connecting a single feature to the attention vector.

Appendix B. Information Retrieval metrics

In Information Retrieval, for a given query we are interested in returning all the relevant items/documents. Any kind of machine learning algorithm for that purpose returns a subset of all items in the domain. These retrieved items are compared against all relevant items to compute both *precision and recall*. Both of these metrics are computed for an *unordered set* of retrieved items and are defined as follows:

$$Prec = \frac{\#\text{retrieved relevant items}}{\#\text{retrieved items}}, \tag{B.1}$$

$$Recall = \frac{\#\text{retrieved relevant items}}{\#\text{relevant items}}. \tag{B.2}$$

That is, precision is the fraction of retrieved documents that are relevant, and recall is the fraction of relevant items that are retrieved.

For ranked retrieval, both precision and recall need to be modified to incorporate the order of retrieved items. This ordering is naturally given by the top-k retrieved items defined by the inherent scoring of the machine learning algorithm, for example, probabilities of belonging to relevant class. One method of measuring performance is *precision-recall graph*, that plots precision values at predefined recall levels. The whole graph can be further aggregated to provide a single number as a performance measure. Another measure (that we adopt for our experiments) is the Average Precision (AP) which has been shown to have good discrimination and stability. Average Precision is the average of the precision values obtained for the set of top-k items existing after each relevant items is retrieved and is defined as:

$$AP = \frac{1}{|R|} \sum_{i=1}^{N} Prec(i) \cdot Relevant(i), \tag{B.3}$$

with R being the total number of relevant samples, $Prec(i)$ is precision computed for the first i items in the set and $Relevant(i)$ is an binary function indicating whether item i is relevant or not. N includes all the items in domain, but once all R relevant items have been encountered, $Relevant(\cdot)$ is zero for the remaining items and we can stop the summation.

For our purposes in the intrusion detection context, *anomalousness indicates relevancy (what we want to be highly ranked), and retrieving a sample in the top-k would mean that the model labels it as anomalous*. With this interpretation, recall is not very informative, as it will always be constant no matter the ranking. Thus, we are omitting it from our evaluation and focus only on precision-related metrics.

Since network security analysts are inherently time-constrained in how many anomalies they can investigate in a fixed amount of time, our aim is to only examine the top-k highly ranked cases. This is analogous to the number of retrieved items a user is willing to investigate when performing web queries – they will be interested in only couple of cases among the top 10 or 20 retrieved items. Thus, we restrict our evaluations to the top-k variants of the metrics – precision@k and average precision@k which are defined as:

$$Prec(k) = \frac{\#\text{retrieved relevant items in the top-}k\text{ items}}{k} \tag{B.4}$$

$$AP(k) = \frac{1}{k}\sum_{i=1}^{k} Prec(i) \cdot Relevant(i). \tag{B.5}$$

For detecting malicious cases, we have to map notions of relevant and retrieved item to *malicious* and *labeled malicious* samples, respectively. Armed with this conversion to security context, we refine $Prec(k)$ and $AP(k)$ as our evaluation metrics in Section 5.

The downside of $Prec(k)$ is that it does not average well across different queries, since the size of the relevant set will influence the outcome (consider cases where relevant set size is less than k and larger than k). To overcome this issue, what is commonly used is $Prec(R)$ or R-precision, that is, precision computed on the first R items, where R is the number of relevant items. For R-precision, ideal score equals 1 no matter the size of the relevant item set.

Appendix C. Feature lists

Tables C.4 and C.5 show which features were extracted for Rank data and CICIDS2017 data respectively. Not all computed features carried useful information and some of them were either constant (at zero value) or heavily skewed towards 0. These where removed prior to further preprocessing steps.

Related event	feature calculation	included
process creation	number of created processes	true
	number of unique source ips	false
	total number of parent processes	true
	number of unique parents processes	true
	number of unique directories when creating processes	true
	number of unique commands when creating processes	true
	total number of users	true
	number of unique users	true
	total number of *parentImage_paths*	true
	number of unique *parentImage_paths*	true
	total number of images files	true
	number of unique images files	true
	total number of images directories	true
	number of unique images directories	true
process termination	number of processes	false
	number of unique processes	false
	number of parents processes	false
	number of unique parents processes	false

	number of unique directories when terminating processes	false
	number of unique commands when terminating processes	false
	total number of users	true
	number of unique users	true
file related	number of created files	true
	number of unique files	true
	number of unique source ips	false
	number of processes involved in this creation	true
	number of unique categories	true
	number of unique account types	false
	number of unique *hostnames*	true
	minimal difference between *creation_time* and previous_creation_time	true
	maximal difference between *creation_time* and *previous_creation_time*	true
	average difference between *creation_time* and *previous_creation_time*	false
sysmon related	total number of source ips	false
	number of unique source ips	false
	total number of source ports	false
	number of unique source ports	false
	total number of destination ips	false
	number of unique destination ips	true
	total number of destination ports	false
	total number of processes involved	false
	number of unique processes involved	false
	total number of account types	true
	hashtag of all source ports	true
	hashtag of all destination ports	true

Table C.4: Extracted features for Rank data. Related event indicates the underlying event for a subset of features, and column *included* indicates if the feature is included in the final construction of a data set. Certain features do not carry any information and are therefore removed.

Feature extracted	included	$f(x)$
number of unique source ips	true	x
number of unique source ports	true	x
number of unique destination ips	true	x
number of unique destination ports	true	x

number of started flows	true	$ln(x+1)$
number of stopped flows	true	$log_{10}(x+1)$
number of idle flows	true	$log_{10}(x+1)$
number of started tcp flows	true	
number of stopped tcp flows	true	
number of idle tcp flows	true	$log_{10}(x+1)$
number of started udp flows	true	$log_{10}(x+1)$
number of stopped udp flows	false	
number of idle udp flows	true	$log_{10}(x+1)$
number of started other flows	false	
number of stopped other flows	false	
number of idle other flows	false	
minimum payload size	false	
maximum payload size	true	x
mean payload size	true	x
payload size deviation	true	x
minimum header size	true	x
maximum header size	true	x
mean header size	true	x
header size deviation	true	x
number of fin flags found	true	$log_{10}(x+1)$
number of psh flags found	true	$log_{10}(x+1)$
number of urg flags found	false	
number of ece flags found	false	
number of syn flags found	true	$log_{10}(x+1)$
number of ack flags found	true	$log_{10}(x+1)$
number of cwr flags found	false	
number of rst flags found	false	
minimum flow idle-active time	true	$log_{10}(x+1)$
maximum flow idle-active time	true	$log_{10}(x+1)$
mean flow idle-active time	true	$log_{10}(x+1)$
flow idle-active time deviation	true	$log_{10}(x+1)$
minimum flow activity	false	
maximum flow activity	false	
mean flow activity	false	
flow activity deviation	false	
minimum flow idle time	false	
maximum flow idle time	false	
mean flow idle time	false	
flow idle time deviation	false	
forward flow payload	false	

forward flow header	false	
number of forward flow psh flags	false	
number of forward flow urg flags	false	
minimum forward flow idle-active time	false	
maximum forward flow idle-active time	false	
mean forward flow idle-active time	false	
forward flow idle-active time deviation	false	
backward flow payload	true	$log_{10}(x+1)$
backward flow header	true	$log_{10}(x+1)$
number of backward flow psh flags	true	$log_{10}(x+1)$
number of backward flow urg flags	true	$log_{10}(x+1)$
minimum backward flow idle-active time	false	
maximum backward flow idle-active time	true	$log_{10}(x+1)$
mean backward flow idle-active time	true	$log_{10}(x+1)$
backward flow idle-active time deviation	true	$log_{10}(x+1)$

Table C.5: Extracted features from CICIDS2017 PCAP files. f is the transformation function applied to shift the feature distribution into a more uniform range.

Appendix D. Hyper-parameters of models

Model	Parameter	Values
Autoencoder	learning rate α	0.01, 0.001
	number of layers L	0, 1, 2
	number of neurons n	100, 200
	bottle-neck layer neurons	10, 20
LSTM	learning rate α	0.01, 0.001
	number of neurons n	8, 16, 32
Bi/S - LSTM	learning rate α	0.01, 0.001
	number of neurons n	4, 8, 16
(Attention) Encoder-Decoder	learning rate α	0.01, 0.001
	number of neurons n	8, 16, 32

Table D.6. Parameters values tested in validation procedure for each model. Bi/S indicates all three combinations of bidirectional and stacked variants of LSTM. Learning rate α is the coefficient responsible for the magnitude of gradient updates during optimization. Number of layers L is how many layers are present in the network. For autoencoder this value is doubled, while LSTM variants in this study only have $L = 1$ or $L = 2$ (for stacked variant). Number of neurons n represents number of processing units within a single layer.

Data set	Model	Hyper-parameters			
		α	L	n	b.n. n
Synthetic	Autoencoder	0.001	1	200	20
	LSTM	0.01	1	8	
	Bi-LSTM	0.01	1	8	
	S-LSTM	0.01	2	16	
	Bi-S-LSTM	0.01	2	8	
	Encoder-Decoder	0.01	1	8	
	Attention ED	0.01	1	32	
Yahoo! A1-51	Autoencoder	0.001	1	200	20
	LSTM	0.01	1	32	
	Bi-LSTM	0.01	1	16	
	S-LSTM	0.01	2	16	
	Bi-S-LSTM	0.01	2	16	
	Encoder-Decoder	0.01	1	32	
	Attention ED	0.01	1	16	
Yahoo! A1-56	Autoencoder	0.001	1	200	20
	LSTM	0.01	1	32	
	Bi-LSTM	0.01	1	16	
	S-LSTM	0.01	2	8	
	Bi-S-LSTM	0.01	2	16	
	Encoder-Decoder	0.01	1	32	
	Attention ED	0.01	1	16	
Rank	Autoencoder	0.001	1	100	20
	LSTM	0.01	1	32	
	Bi-LSTM	0.001	1	16	
	S-LSTM	0.01	2	16	
	Bi-S-LSTM	0.001	2	16	
	Encoder-Decoder	0.001	1	32	
	Attention ED	0.001	1	32	
CICIDS	Autoencoder	0.001	2	400	50
	LSTM	0.01	1	20	
	Bi-LSTM	0.01	1	20	
	S-LSTM	0.01	2	20	
	Bi-S-LSTM	0.01	2	20	
	Encoder-Decoder	0.01	1	20	
	Attention ED	0.01	1	20	

Table D.7. Chosen values of hyperparameters for each model and data set. α is the learning rate, L number of layers, n the number of neurons inside each layer, and 'b.n. n' is number of neurons inside bottle-neck layer for the autoencoder. The standard LSTM model only has a single layer, while stacked variants of LSTM contain two layers.

Appendix E. Receiver-Operating Characteristic

As an alternative to ranking metrics, we can also evaluate our anomaly detectors according to Receiver-Operating Characteristic (ROC) curves and the summary statistic of Area under the Curve (AUC). While these evaluations do not provide the user-centric insights of ranking metrics that we focus on in this article, they do provide an alternative perspective on overall anomaly detector performance.

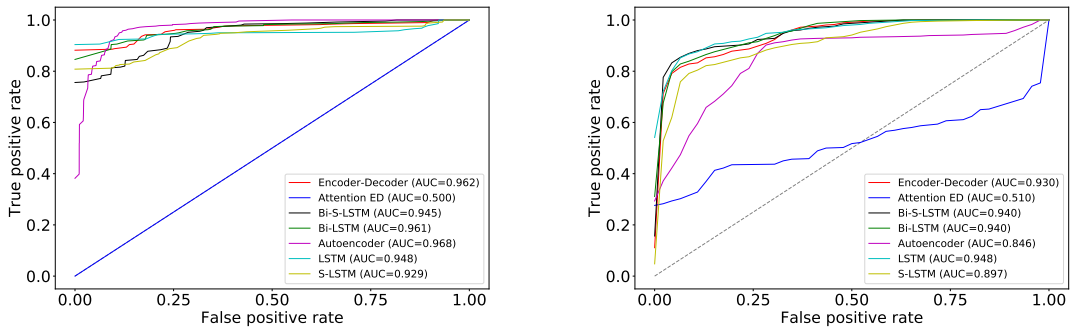
Figure E.11 shows ROC curves for all models across all data sets. The results are averaged over multiple runs (except for CICIDS). ROC curves present a plot of the false positive rate (FPR) vs. the true positive rate (TPR) for binary classifiers. The FPR corresponds to the proportion of negative instances that were mistakenly predicted as positive. The TPR corresponds to the proportion of positive instances that are correctly predicted as positive. Hence, the ROC curve shows a trade-off indicating the FPR that would result if we desire a certain TPR (and vice versa).

Also shown in the Figure E.11 legend for each model is its Area under the ROC Curve (AUC). Here, we can consider that the perfect tradeoff (a TPR of 1.0 at all FPR values) would have an AUC of 1.0 whereas the AUC of a random predictor should be lower bounded by 0.5 in general when the train and test data are from the same distribution. *However*, we remark that in our anomaly detection setting, the test data is by definition *very* different from the training data since the training data contains no anomalies, while the test data has anomalies. In this case, it is highly possible that an AUC is less than 0.5 as we observe in some of our experiments, which overall indicates the exceptional difficulty of these test cases.

We would like to stress that the curves are computed with access to test labels (similar to results in Section 7.2); however, lack of access to test labels in practice would prevent choosing an anomaly detection threshold in order to achieve a target TPR-FPR trade-off. Because we cannot properly tune this anomaly detection threshold in practice in the absence of labeled anomalies, we are unable to evaluate standard boolean metrics like precision, recall, accuracy and F1-score. Nonetheless, we can use ROC curves and AUC to obtain a sense of overall anomaly detector performance.

In all cases, one will note that the ranking of algorithms according to AUC strongly correlates with the ranking evaluations in Section 7.2. One minor exception is the non-sequential Autoencoder for the Synthetic dataset – while it does have a high AUC, we note that it achieves a very poor TPR at low FPRs, which is not true of the other models. This makes Autoencoder a poor choice for end users since they require high TPRs at low FPRs in order to reduce the amount of false positives in the subset of anomalies they can investigate. In this case, the second best AUC score of Encoder-Decoder provides both a high AUC and a high TPR and low FPR making it a better choice. It is interesting to note that the Average Precision metric of Section 7.2 directly identifies this end-user preference for Encoder-Decoder over Autoencoder supporting the strong correspondence between the ranking metrics evaluated in this article and end-user requirements for anomaly detection systems.

A final surprising result is that all models perform quite poorly on Yahoo! data (A1-51 and A1-56), even though the reconstruction errors are low (not shown). The periodic nature of the signal in this case makes it difficult for all models to correctly identify/rank malicious cases and reflects our above commentary regarding the mismatched train and test settings for anomaly detection and its potential negative impact on AUC. However, these low AUCs do correlate with the low ranking metrics reported for these same datasets in Section 7.2 indicating that ranking metrics in this regime may provide strong indication of very poor anomaly detection performance viewed from an ROC curve and AUC perspective.



(a) Synthetic

(b) Rank

(c) A1-51

(d) CICIDS

Figure E.11. ROC curves for the tested models. AUC scores are given next to model labels. Results for Yahoo! A1-56 data set is left out since the curves are quite similar to those of A1-51 – all models also underperform on A1-56.