Comparing Autoencoder and Isolation Forest in Network Anomaly Detection

Timotej Smolen, Lenka Benova
Slovak University of Technology in Bratislava
Bratislava, Slovakia
xsmolen, lenka.benova@stuba.sk

***Abstract*—Anomaly detection is essential to spot cyber-attacks within networks. Unsupervised anomaly detection methods are becoming more popular due to difficult and expensive process of labeling network data as well as their superior ability to detect unknown attacks when compared with supervised or signature-based solutions. In this paper, we use an LSTM-based Autoencoder (RAE) anomaly detection model trained in a fully unsupervised environment, with optimizations for minimal memory usage. Secondly, we compare RAE with an Isolation Forest model by analysing their results. RAE attempts to capture the profile of the data by dimensionality reduction and the use of LSTM layers enables it to leverage the data from previous requests. Reconstruction error is calculated to decide about the abnormality. We train models on a dataset of requests towards a webserver in an unsupervised fashion. Before training, significant feature engineering is done to process multiple categorical attributes. The training process of RAE is optimized for minimum memory usage. We evaluated the results based on our analysis of the data as well as their statistical features. A manual analysis revealed differing focuses between numerical and categorical attributes. Isolation Forest disregards most categorical attributes and emphasizes numerical values. RAE on the other hand detects missing features more effectively but largely disregards numerical attributes. As such, RAE might have a higher probability of detecting a zero-day attack when compared to Isolation Forest.**

I. Introduction

Network security had been a concern long before the Internet became as widespread as it is today. Attackers with malicious intent have always been here but their activity has risen together with the rise of Internet around the world [1]. Security measures had to be implemented to counter these activities. One of such measures is an Intrusion Detection System (IDS). Intrusion detection is a continuous action of observing traffic within a computer or a network and evaluating it in an attempt to detect intrusions, defined as ”attempts to compromise the confidentiality, integrity, availability, or to bypass the security mechanisms of a computer or network”. The system that automates this process is an IDS [2]. Signature-based IDSs use known attack patterns to detect intrusions. Their disadvantage however lies in their nature. As they use known patterns to detect attacks, they are usually unable to detect new attacks which intruders might have devised. This is one of the reasons that the use of anomaly-based network intrusion detection systems (ANIDS) has become more common [3].

To support the idea of detecting unknown attacks, unsupervised machine learning method are often preferred. These methods use unlabelled datasets and are therefore not biased towards only learning to detect the attacks they are trained on. Thus they are more suitable for detection of zero-day attacks. The use of unsupervised machine learning methods solves one more significant problem present in this domain. Although obtaining a substantial dataset of network traffic is not arduous nowadays, such data typically lacks labels. The annotation of such data can be a lengthy as well as expensive process that requires considerable expertise in the domain [4]. Experts who analyse or even label such data can also benefit from the use of an unsupervised ANIDS. It can be a tool at their disposal to highlight likely anomalous data, detect a previously unknown anomaly, get a second unbiased opinion or simply to double check their work.

The article aims at using unsupervised machine learning methods to primarily detect point anomalies, which is also the most researched type of anomaly. They are single pieces of data differing from others [5]. We opt for them as we probably do not have enough contextual attributes to detect contextual anomalies. Collective anomalies may also be partly detectable by RAE, however different groupings and division of our data (such as division per user) would be more suitable for detection of this type. We train the models using unlabelled dataset consisting of web server logs. By using unlabeled data, the model can be trained on a wide range of data types and formats and can adapt to new or rare kinds of anomalies. Unlabeled datasets offer a lot of information that can be used to train efficient anomaly detection models as they can come from real live traffic. The model may learn what normal behavior entails without being influenced by any prior beliefs or labels. It can also learn to recognize uncommon and previously undiscovered anomalies that might not have been present in the labeled dataset.

II. Related works

Most of the current work in anomaly detection in unsupervised environment uses algorithms such as clustering, OCSVM, Isolation Forest or various types of neural networks.

Aditya Vikram et al. [6] uses Isolation Forest with 100 estimators to detect anomalies in the form of cyber-attacks. It was trained and evaluated on public NSL-KDD dataset [7]. Even though the model was trained in an unsupervised way, the labels for the dataset are available. In the final results, AUC ROC score of 98.3% was achieved.

Xiao Chun-Hui et al. [8] use Isolation Forest to detect anomalies in network management system data in an unsupervised way. As no similar labelled are available, they simulated normal traffic data and added superficial anomalous data. They use different feature extractors for their time-series data and achieved AUC score of more than 90% regardless of the feature extractor.

Guo Pu et al. [9] use the same NSL-KDD dataset to detect cyber-attacks. First, they use sub-space clustering (SSC) to divide data into N subspaces. Afterwards they apply one-class support vector machine (OCSVM) on each subspace to evaluate. They achieve the best results in regards to ROC metric when compared with other clustering algorithms such as DBSCAN or K-means clustering. However, the computation time of the algorithm was significantly higher than others as well, with times of 238.88s, 8.15s, and 0.69s respectively for SSC-OCSVM, DBSCAN and K-means.

A. Tuor et al. [10] introduce a solution for insider threat detection using Recurrent Neural Network (RNN). They use a neural network consisting of LSTM layers on CERT Insider Threat Dataset v6.2 [11]. The main thing differing it from other solutions is the fact that instead of working with all data instances together, they make a separate model for each user. Their best LSTM model achieved result of 35.6% using Cumulative Recall at 1000 metric, compared to 34.8% with Isolation Forest.

Ming-Chang Lee et al. [12] introduce a lightweight RNN LSTM model called ReRe for anomaly detection in univariate time-series data. The datasets used were by Numenta anomaly benchmark [13]. This approach uses 2 separate LSTM networks that are trained on previous data points in time. Each network has only 1 hidden layer with 10 hidden units. Each network calculates an average error, the difference being one of the networks only uses normal data to calculate this error, the other one includes data marked as anomalous as well. Only when both of the networks mark the input data point as an anomaly is it marked as one. They achieve a result of 0.5263 precision at 7, which is better by a margin of 0.0263 when compared with a similar anomaly-based lightweight approach.

Wen Xu et al. [14] use an Autoencoder to detect anomalies in the NSL-KDD dataset. They use Autoencoder consisting of 5 layers with a set up of 122-32-5-32-122 units within the layers. They compare different loss functions and achieve the best result with accuracy of 90.61% using the mean absolute error.

Zhaomin Chen et al. [15] introduce a Convolutional Autoencoder (CAE) in the area of network anomaly detection. The difference between a conventional and convolutional Autoencoder is the use of convolution and deconvolution layers to act as encoder and decoder respectively. The reason for its use is, that a convolution layers have less parameters than dense ones. Similar to other solution, they use the NSL-KDD dataset. They compare the model with PCA and conventional Autoencoder and achieve the best results with AUC score of 98.84% using CAE. Conventional Autoencoder was a close second with AUC score of 98.57%.

Sepehr Maleki et al. [16] introduce an LSTM-based Autoencoder model for anomaly detection. It attempts to label the dataset and uses probability criterion based on the central limit theorem to mark the anomalous and normal data. It is evaluated on The Numenta Anomaly Benchmark as well as temperature measurements on an Industrial Gas Turbine (IGT) burner-tip thermocouple. It achieves F1 score of 0.95 and 0.97 in the datasets respectively.

III. System descriptions

We aim to detect various types of anomalies that are potentially present in our unlabelled dataset. We chose Isolation Forest as one of our models, as it is fast and easily trainable model with low memory requirements as opposed to some of the other algorithms such as OCSVM or various clustering methods. Isolation Forest shows great results in [6], [8]. We also apply an LSTM-based Autoencoder model on our problem. This model combines different approaches from network anomaly detection domain, which showed promising results. The structure of Autoencoder for dimensionality reduction as leveraged in [14], [15] is combined with a use of LSTM as proposed in [10], [12] to incorporate use of previous samples to the solution, which we believe can be beneficial. Similar structure is used in [16], although on univariate time-series data. Our whole system was developed using Python programming languages and its libraries. For preprocessing we used Numpy, Pandas and Scikit-learn libraries. We used Scikit-learn's implementation of Isolation Forest and Tensorflow library to create RAE.

A. Data

The web server logs used to develop our system were provided by an antivirus company and they represent one day worth of requests to download client modules towards their web servers. The total number of requests is 90910622. Data contains 11 attributes as seen in table I. 8 of these attributes are categorical, the other 3 are numerical (feature *status* is in numerical format but has categorical values so we consider it categorical). Fig. 1 shows the format of the data. We used a custom parser to convert the data from log format into a more suitable csv format. This parser divided each row into corresponding features with a use of a regular expression.

B. Feature selection and engineering

Attributes *user_hash* and *time_local* were not used in our system as we do not isolate each user and the order of the requests is preserved throughout the training. We have also decided not to use *uri* attribute, as interpretation of

```
$user_hash [$time_local] "$http_host"
"$request_method $uri $server_protocol"
$status_code $body_bytes_sent $request_length
$request_time $scheme;
```

Fig. 1. Format of data

TABLE I. ATTRIBUTES OF WEB SERVER LOGS

user_hash	identifiers of a user
time_local	local time in the Common Log Format
http_host	HTTP server host
request_method	HTTP request method
uri	path to the update being requested
server_protocol	server protocol version
status	response status
body_bytes_sent	request body length
request_length	request length including line, header, and body
request_time	request_processing time in seconds
scheme	HTTP scheme

this attribute requires expert knowledge. The structure of this attribute is also changing constantly, so our processing of the attribute could quickly become outdated. As attribute *http_host* contains 92 distinct values we have handpicked certain features of this attribute. We have created 7 boolean columns. 3 of them describe the format of the *http_host*, whether it is in IPv4, IPv6 format and the presence of port. 3 other created columns are filtered one-hot encodings, where only the columns encoding values with frequency more than 1% are preserved. The last created attribute is a one-hot encoding of a less populated but important '-' value, describing missing value in *http_host*. During Exploratory Data Analysis we have discovered that the dataset contains duplicates. Due to the fact, that the granularity of the *time_local* is in seconds, we assume this shows repetition of the same request within the second. Therefore we have created a *repeated* attribute, containing the number of such rows in the dataset and deleted the duplicated rows. As the number of distinct categories was significantly lower in other attributes, we used ordinal encoder for *scheme* (as it only contains 2 distinct values) and one-hot encoding for *status*, *server_protocol* and *request_method* resulting in a total number of 35 attributes in the dataset as represented in Fig. 2. Lastly, we have scaled our 4 (including *repeated*) numerical attributes onto a range from 0 to 1.

C. Isolation Forest

As our first tested model we chose Isolation Forest. Whereas most of the other machine learning algorithms aimed at anomaly detection design a characterization of normal data and identify the instances differing from this profile as anomalies, Isolation Forest isolates anomalies directly without such profiling. Isolation Forest method creates a forest made of Isolation Trees, where each tree corresponds to a test that separates instances recursively. As Isolation Forest makes an assumption about the data, that anomalies are far fewer than instances of normal data, anomalies should be isolated earlier and have shorter paths in the trees. Anomaly score is calculated based on the lengths of these paths [17].

```
body_bytes_sent, request_length, request_time,
scheme, repeated, is_ip, has_port, is_ipv6,
no_http_host, http_host__….eset.com,
http_host__….eset.com, http_host__X.X.X.X,
server_protocol__-, server_protocol__HTTP/1.0,
server_protocol__HTTP/1.1, request_method__-,
request_method__GET, …, request_method__PUT,
status__200, …, status__416
```

Fig. 2. Attributes of preprocessed data

After training Isolation Forest model with default parameters on our data and calculating anomaly scores for all of the data, we observed that more than 22% were marked as an anomaly. Therefore, we had to tune the contamination parameter of Isolation Forest, which decides, what percentage of data instances are to be considered anomalous. As we do not have the information about percentage of anomalous samples in our dataset, we were forced to tune the parameter manually. We arrived at the value of 0.05 (resulting in 5% of anomalies), which still seemed high but provided a bit more diverse results overall than lower values such as 0.01, in terms of which attributes were out of the ordinary in samples marked as anomalies. Furthermore, we tuned the number of estimators (Isolation Trees) and chose a value of 200, which seemed to show best overall results based numbers in tables II and III. Isolation Forest with 200 estimators showed some of the best results with lowest (most anomalous) average anomaly score for OPTIONS request methods and missing server protocol and second highest averages in numerical attributes.

D. Autoencoder

A specific type of a neural network which is trained to reconstruct its input is called an Autoencoder. The purpose of Autoencoder is to reconstruct the input data. In settings of an anomaly detection problem, it is assumed that an Autoencoder would learn a profile of the normal data. Therefore, when later comparing the reconstructed data (output), the loss between original and reconstructed data would be small in normal instances, but large in case of anomalies [18]. Autoencoder is a dimensionality reduction system in its nature, as it tries to learn the profile of the data in a reduced subspace [19]. This subspace is usually one of the middle layers within the

TABLE II. AVERAGE VALUES OF NUMERICAL ATTRIBUTES FOR ANOMALOUS INSTANCES

Estimators	Request length	Body bytes sent	Request time
50	618.9	98665.2	0.306
100	589.7	109376.3	0.340
200	594.4	108757.5	0.338
300	597.4	108526.9	0.336

TABLE III. AVERAGE ANOMALY SCORES FOR VALUES OF CATEGORICAL ATTRIBUTES

Estimators	Missing server protocol	Request method OPTIONS
50	-0.164	-0.090
100	-0.156	-0.098
200	-0.164	-0.102
300	-0.165	-0.100

Autoencoder, such as layer L_2 in Fig. 3. The data is first encoded into this subspace from the original format, and then decoded back into the original shape.

Recurrent neural network one (RNN) is type of Neural Network that contains loops within the network structure. This means that it can feed the output back into itself enabling it to consider context in the form of previous inputs [21]. This makes RNN a good choice when dealing with sequential data which can profit from being able to process neighboring data points [22]. When dealing with time-series data, as is our data, it means that the neural network model is able to process a certain amount of historical data points. Most often this means that the model either evaluates single input sequence as a whole, or uses the first $X - 1$ data points to help evaluate the Xth point.

We chose this Autoencoder structure with recurrent neural network layers, to create RAE, our second model. Specifically, we have used Long short-term memory layers as introduced in [23] to tackle the vanishing and exploding gradients problem. Our model consists of 4 LSTM layers using hyperbolic tangent activation function with 32, 16, 16 and 32 units in the respective order. The whole layout can be seen in Fig. 4. This model structure with firstly reducing and then increasing number of units within layers enables RAE to reduce number of dimensions and to learn representation of the data within this reduced space. The added value of the LSTM layers is that multiple samples can be processed at once, and so instead of a single instance we can give it a sequence with previous instances. This means that previous instances within the sequence can be used to help predicting the next one. To process the network traffic data, we therefore have to modify the generalized Autoencoder structure shown in Fig. 3. To

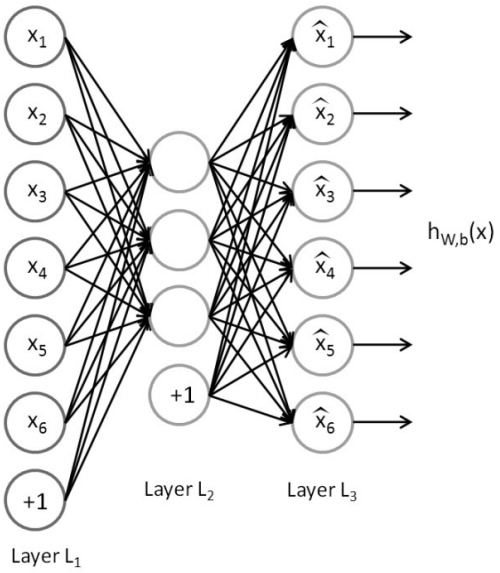

Fig. 3. Autoencoder structure [20]

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 10, 35)]	0
lstm_2 (LSTM)	(None, 10, 32)	8704
lstm_3 (LSTM)	(None, 16)	3136
repeat_vector (RepeatVector)	(None, 10, 16)	0
lstm_4 (LSTM)	(None, 10, 16)	2112
lstm_5 (LSTM)	(None, 10, 32)	6272
time_distributed (TimeDistri	(None, 10, 35)	1155

Fig. 4. Layers of our Autoencoder

fit our data into a model, we need to create sequences of $Timestep$ samples. This sequence is then fed to the model as a singular piece of data and processed accordingly, to leverage the previous inputs in the sequence to predict the next one. We compared optimizers such as Adam, Adagrad, Adamax and RMSProp, and while all of them produced very similar results in terms of loss, we chose to use Adam optimizer [24], as it is used in most of the state of the art solutions. We used a learning rate of $1e - 03$ and mean squared error, shown in equation 1, as our loss function, which is a common function used to calculate reconstruction error.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i}) \tag{1}$$

Early stopping halted training of our final model after 9 epochs as there was no improvement in value of loss. We use batch size of 256 because it strikes a good balance between the amount of data being processed at once and the computational efficiency of the training process. We have made a decision to use timestep of 10, thus resulting in every sequence acting as an input training consisting of 10 consecutive requests. This enabled the model to evaluate the whole sequence and take neighboring (time wise) requests into consideration. If the model were to be deployed into production, the sequence would be evaluated as a whole, or alternatively, a sequence would be flagged as an anomaly even if only a single request was flagged as one. However, for the ease of evaluation, and mainly for the ability to compare the results with Isolation Forest, we aggregated our losses for each data point and took only the maximum calculated loss per instance into account.

Due to the size of our data as well as hardware limitations, we made several optimizations to improve effectiveness of both training and evaluation of RAE. Our main limitation was the size of random access memory. We use a timestep of 10, which means that our data needs to be organized into sequences of 10 samples each. However, because our original dataset was already quite large, multiplying its size by 10 to fit our input layer resulted in a significant increase in memory usage. In fact, the resulting dataset was too big to fit into memory as a whole. To address this issue we have created a data generator. Instead of reading the data from RAM, this

generator reads a single batch directly from the file on the disk every time it feeds it to RAE. This resulted in much smaller memory usage, as the batches were only kept in the memory one at a time. Similar steps were used for prediction, as per the nature of Autoencoder, input and output are of the same size. We compared the training times of both approaches (reading from file or from memory) on a sample of 1000000 rows. Surprisingly, the memory optimization did not result in higher training times. The training time of RAE model when reading batches from file on a disk was 245.6 seconds. When reading data directly from memory the training time was 245.4 seconds. To store our data we used an hdf5 format, which enabled us to easily read only a specified part (batch) of a file from the disk.

IV. Results

Our test set consists of 1000000 samples from the dataset, not used during training of the models. As we are working in a fully unsupervised settings, meaning that the labels for our dataset are not at our disposal, we are unable to evaluate our models using metrics typically used such as accuracy, precision or AUC score. This makes the analysis of our results more difficult and the the angle at which we look at our results a bit uncommon, when compared with other works, where the labeled dataset is available for the evaluation part of the experiments. To evaluate our results we leverage statistical features of our data as well as our expertise. These results should be able to therefore be generalized for datasets from different domains, that exhibit similar distributions, features or characteristics. Besides predicted values, we have also looked at the speed at which models are able to evaluate data. RAE outperformed Isolation Forest in this aspect with time of 0.099s compared to 0.468s on predicting abnormality values on 1000 samples. Thus RAE seems to be quicker in evaluating, most likely due to the ability to leverage GPU.

In terms predicted values, we firstly look at the distributions of anomaly scores of the models. To make the comparison we scale the results of each model individually onto 0-1 scale. Fig. 5 shows, that the distributions differ significantly. Based on the plot we can observe that while RAE tends to reconstruct the data seemingly normal to him with minimal loss, Isolation Forest uses wider range of values for its anomaly score. Both of the plots converge with the anomaly score rising.

The mean absolute error between the Isolation Forest and RAE anomaly scores is 0.17, which only confirms what we observed in the plot. However, when looking at the maximum and minimum difference, we observe that 2725 instances have a difference higher than 0.8, meaning that an instance of data marked as anomalous by Isolation Forest was assumed normal by RAE. For threshold of 0.5, this would result in 53886 instances which is more than 5% of the test set. The minimum difference is -0.35, which shows that what is deemed as an anomaly by RAE, is usually not viewed as totally normal by Isolation Forest either. These numbers imply one of 2 things:

1) Some anomalies have been undetected by RAE.
2) Isolation Forest has high False Positive rate.

As the next step, we look at the average anomaly score of instances with positive values in some of the categorical attributes, which we identified as likely anomalous. These include certain uncommon HTTP status codes, presence of port in the *http_host* or the absence of a certain feature (displayed as '-' within the data). They can be seen in table IV, where we can observe that RAE seems to capture some of the anomalous attributes better than Isolation Forest. It has higher average anomaly score in missing request data as well as HTTP status 408. While it has a lower average anomaly score for *request_method_OPTIONS* or missing HTTP host, it is important to note that while there are 3104 instances with Isolation Forest score higher than 0.8 in our test set, there are only 14 with higher RAE score than 0.64. We can observe that Isolation Forest does better in capturing anomalous state of attributes connected to the HTTP host. However, it also seems to judge instances with specific HTTP host, whose frequency was more than 1% in the whole dataset as anomalous, even though it is unlikely that this attribute should contribute to higher anomaly score because of its frequency.

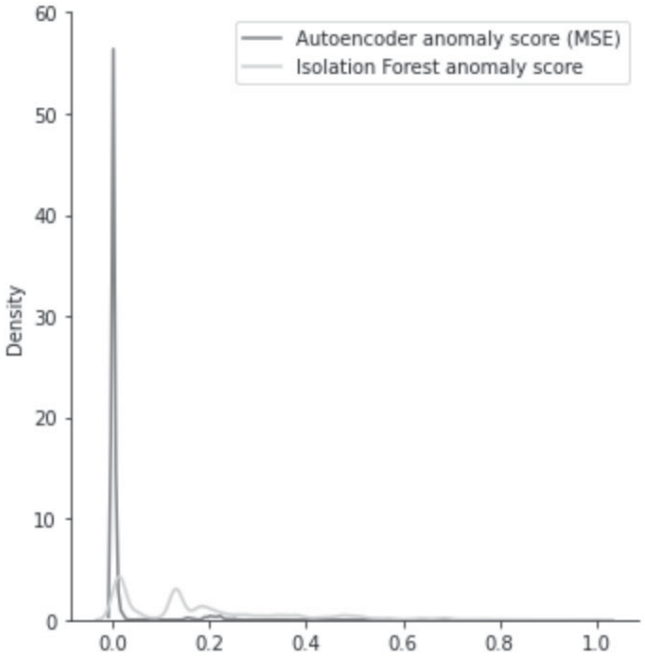

Fig. 5. Distributions of anomaly scores

TABLE IV
Average anomaly scores where value of attribute is 1

	iForest score	AE score
request_method__-	0.747378	0.990619
no_http_host	0.807269	0.648626
request_method__OPTIONS	0.536303	0.464310
status__408	0.815695	0.987441
is_ip	0.467981	0.024964
has_port	0.623714	0.289845
http_host__X.X.X.X	0.457052	0.006375

Our 4 numerical attributes have distribution similar to geometric, with high density in lower values of attributes and smaller in high values. Therefore we assumed correlation of these attributes with the scores calculated by our models. The correlations in table V, show that while Isolation Forest gives some weight to the *body_bytes_sent, request_length* and *repeated*, RAE does not consider any of these attributes except *request_length* significant and rather gives importance to some other categorical attributes. Interestingly, the one attribute both of the scores correlate with (*request_length*), has a negative Pearson correlation with them.

To make a final evaluation, we manually analysed 100 of rows of data with the highest anomaly score for each model. The cut off anomaly scores were 0.635761 for RAE and 0.947780 for Isolation Forest. Interestingly, there was no intersection between the 2 subsets.

Based on the top results Isolation Forest seems to disregard most of the categorical attributes. All of the samples used GET request method and received a HTTP status code 200. 46 of them were also addressed to one of the most frequent HTTP hosts. The focus is more on the attributes describing specific features of the host. Out of the top 100 samples, 59 had HTTP host in IPv4 format and 54 had port number appended at the end of it. Isolation Forest also gave significant importance to numerical values, such as *body_bytes_sent*, where the average value of the top 100 corresponded to approximately 99.98th percentile of the whole test set. Most of the samples are thus combinations of high numerical values and HTTP host in IPv4 format or with port present. Ergo it seems that lot of different types of anomalies slipped by undetected and that if a zero-day attack were to occur, our Isolation Forest might not be able to detect it. In fact, 35 of the 100 samples, such as sample 2 in table VI were only anomalous in *body_bytes_sent* attribute and 6, similar to samples 1 and 3 from table VI do not seem anomalous in any.

Whereas Isolation Forest puts the highest significance on the numerical attributes, RAE pretty much disregards them, with *body_bytes_sent* value of 0.0 in all of the 100 samples with RAE's highest anomaly score. RAE managed to detect all 7 samples with missing request method and gives high importance to missing HTTP host, with 98 our of 100 samples having 1 in this attribute. We can also see bigger diversity within some of other attributes, such as HEAD or OPTION request methods being included in the top 100 as well as some of the 4xx status codes. Some of different samples with high RAE anomaly score can be seen in table VII. As a lot of the samples seemed to be of similar kind with HTTP host missing, we briefly looked at the next 400 samples. The samples followed similar trend with more focus now on status codes or request methods (as it has already detected all of the samples with missing features). When doing this with Isolation Forest, we again received similar results, with more than 99% of the samples having status code 200 and GET request method, and detecting minimum of the samples with missing features as anomalous.

TABLE V. PEARSON CORRELATION BETWEEN SCORES AND NUMERICAL ATTRIBUTES

	iForest score	AE score
body_bytes_sent	0.136186	0.019042
request_length	-0.151371	-0.162446
request_time	0.028970	0.014049
repeated	0.123264	0.013464

TABLE VI. SAMPLES WITH HIGH ISOLATION FOREST ANOMALY SCORE -CATEGORICAL FEATURES WITH 0S IN ALL SAMPLES OMITTED

	Sample 1	Sample 2	Sample 3
body_bytes_sent	0.0596	0.6101	0.0596
request_length	0.1028	0.0889	0.0883
request_time	0.0003	0.0004	0.0395
http_host_...eset.com	1.0	0.0	0.0
http_host_...eset.com	0.0	1.0	1.0
server_protocol__HTTP/1.0	1.0	0.0	0.0
server_protocol__HTTP/1.1	0.0	1.0	1.0
request_method__GET	1.0	1.0	1.0
status__200	1.0	1.0	1.0
IF anomaly score	0.9596	0.9575	0.9662

TABLE VII. SAMPLES WITH HIGH RAE ANOMALY SCORE - CATEGORICAL FEATURES WITH 0S IN ALL SAMPLES OMITTED

	Sample 1	Sample 2	Sample 3
body_bytes_sent	0.0	0.0	0.0
request_length	0.0	0.0099	0.0064
request_time	0.0	0.0	0.0
no_http_host	1.0	0.0	1.0
server_protocol__-	1.0	0.0	0.0
server_protocol__HTTP/1.0	0.0	0.0	1.0
server_protocol__HTTP/1.1	0.0	1.0	0.0
request_method__-	1.0	0.0	0.0
request_method__GET	0.0	0.0	1.0
request_method__OPTIONS	0.0	1.0	0.0
status__400	1.0	0.0	0.0
status__403	0.0	1.0	1.0
AE anomaly score	1.0	0.8742	0.916

V. CONCLUSION

In this paper we apply LSTM-based Autoencoder (RAE) on a network anomaly detection problem. Its encoder reduces the dimensions of data and decoder reconstruct the same data into input format. Error is calculated between the original and reconstructed data, which acts as anomaly score. The use of LSTM layers enable use of sequential characteristics of the data. RAE is compared with Isolation Forest model. The analysis of one million samples from an unsupervised dataset revealed that RAE and Isolation Forest models differ significantly in their anomaly detection approaches. The mean absolute error between their anomaly scores is 0.17, with 2725 instances having a difference higher than 0.8. This discrepancy suggests that either the RAE has missed some anomalies or the Isolation Forest has a high false-positive rate. Further examination revealed that RAE tends to capture some anomalous attributes better than the Isolation Forest,

especially in cases of missing request or HTTP host data and HTTP status 408. However, Isolation Forest performs better in capturing anomalies related to HTTP host attributes. Both models demonstrate a negative Pearson correlation with the *request_length* attribute. A manual analysis of the top 100 anomalous rows for each model revealed no intersection between the two subsets and varying focuses on numerical and categorical attributes. Isolation Forest disregards most categorical attributes and emphasizes numerical values, while the RAE detects missing features more effectively but largely disregards numerical attributes. Therefore, RAE should be more suitable for zero-day attack detection.

ACKNOWLEDGMENT

This work was supported by the Slovak Research and Development Agency under the Contract no. SK-SRB-21-0059.

REFERENCES

[1] G. Fernandes, J. J. P. C. Rodrigues, L. F. Carvalho, J. F. Al-Muhtadi, and M. L. Proença, "A comprehensive survey on network anomaly detection," *Telecommunication Systems*, vol. 70, no. 3, pp. 447–489, Mar 2019. [Online]. Available: https://doi.org/10.1007/s11235-018-0475-8

[2] R. Bace and P. Mell, "Intrusion detection systems," Nov 2001.

[3] A. Khraisat, I. Gondal, P. Vamplew, and J. Kamruzzaman, "Survey of intrusion detection systems: techniques, datasets and challenges," *Cybersecurity*, vol. 2, no. 1, p. 20, Jul 2019. [Online]. Available: https://doi.org/10.1186/s42400-019-0038-7

[4] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, *Network traffic anomaly detection and prevention: Concepts, techniques, and Tools*, 1st ed. Springer Cham, 2017.

[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, Jul 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882

[6] A. Vikram and Mohana, "Anomaly detection in network traffic using unsupervised machine learning approach," in *2020 5th International Conference on Communication and Electronics Systems (ICCES)*, 2020, pp. 476–479.

[7] "Nsl-kdd dataset." [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html

[8] X. Chun-Hui, S. Chen, B. Cong-Xiao, and L. Xing, "Anomaly detection in network management system based on isolation forest," in *2018 4th Annual International Conference on Network and Information Systems for Computers (ICNISC)*, 2018, pp. 56–60.

[9] G. Pu, L. Wang, J. Shen, and F. Dong, "A hybrid unsupervised clustering-based anomaly detection method," *Tsinghua Science and Technology*, vol. 26, no. 2, pp. 146–153, 2021.

[10] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," *CoRR*, vol. abs/1710.00811, 2017. [Online]. Available: http://arxiv.org/abs/1710.00811

[11] "Insider threat test dataset," Nov 2016. [Online]. Available: https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=508099

[12] M.-C. Lee, J.-C. Lin, and E. G. Gan, "ReRe: A lightweight real-time ready-to-go anomaly detection approach for time series," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, Jul 2020.

[13] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, Nov. 2017.

[14] W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei, and F. Sabrina, "Improving performance of autoencoder-based network anomaly detection on nsl-kdd dataset," *IEEE Access*, vol. 9, pp. 140136–140146, 2021.

[15] Z. Chen, C. K. Yeo, B. S. Lee, and C. T. Lau, "Autoencoder-based network anomaly detection," in *2018 Wireless Telecommunications Symposium (WTS)*, 2018, pp. 1–5.

[16] S. Maleki, S. Maleki, and N. R. Jennings, "Unsupervised anomaly detection with lstm autoencoders using statistical data-filtering," *Applied Soft Computing*, vol. 108, p. 107443, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494621003665

[17] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[18] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," 2020. [Online]. Available: https://arxiv.org/abs/2003.05991

[19] U. Michelucci, "An introduction to autoencoders," *arXiv preprint arXiv:2201.03898*, 2022.

[20] A. Ng, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[21] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," *CoRR*, vol. abs/1912.05911, 2019. [Online]. Available: http://arxiv.org/abs/1912.05911

[22] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[23] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[24] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec 2014.