Using deep learning to detecting abnormal behavior in internet of things

Mohammed Al-Shabi, Anmar Abuhamdah
Department of Management Information System, College of Business, Taibah University, Tayba, Medina, Saudi Arabia

Article Info

Article history:

Received Jun 11, 2021
Revised Jul 19, 2021
Accepted Aug 14, 2021

Keywords:

Deep learning
Detection of abnormal behavior
Internet of things
Long short-term memory neural networks
Machine learning

ABSTRACT

The development of the internet of things (IoT) has increased exponentially, creating a rapid pace of changes and enabling it to become more and more embedded in daily life. This is often achieved through integration: IoT is being integrated into billions of intelligent objects, commonly labeled "things," from which the service collects various forms of data regarding both these "things" themselves as well as their environment. While IoT and IoT-powered devices can provide invaluable services in various fields, unauthorized access and inadvertent modification are potential issues of tremendous concern. In this paper, we present a process for resolving such IoT issues using adapted long short-term memory (LSTM) recurrent neural networks (RNN). With this method, we utilize specialized deep learning (DL) methods to detect abnormal and/or suspect behavior in IoT systems. LSTM RNNs are adopted in order to construct a high-accuracy model capable of detecting suspicious behavior based on a dataset of IoT sensors readings. The model is evaluated using the Intel Labs dataset as a test domain, performing four different tests, and using three criteria: F1, Accuracy, and time. The results obtained here demonstrate that the LSTM RNN model we create is capable of detecting abnormal behavior in IoT systems with high accuracy.

Corresponding Author:

Anmar Abuhamdah
Department of Management Information System, Faculty of Business Administration, Taibah University
Janadah Bin Umayyah Road، Tayba, Medina 42353, Saudi Arabia
Email: aabuhamdah@taibahu.edu.sa; anmar_81@hotmail.com

1. INTRODUCTION

Internet of Things (IoT) is a technical term that has spread to both the digital and physical worlds following its introduction by British scientist Kevin Ashton, a technology pioneer and the founder of the first AutoID lab at Massachusetts Institute of Technology [1]. The driving concept of IoT was to connect common digital devices (often everyday or household electrical appliances) with one another in ways that enable users to monitor their status and take in accurate information from the network of devices without needing to be in close proximity to any of these devices. Put differently, IoT is a system comprised of various components (that may include machines, computing devices, people, or animals) that each possess unique identifiers (UID) and a network connection that enables them to transfer data without human interaction. In other words, IoT is describing the network's physical objects as the "things" of its title, which have been enhanced using technologies that enable them to link to and exchange data with other systems or devices using the internet [1]–[3].

Moreover, the IoT extends the internet of behavior (IoB), which is the interconnection of devices that results in a massive diversity of new data sources. While the behavior of an IoT network and its devices may sometimes be abnormal and could indicate a security threat, identifying this kind of behavior and distinguishing valid concerns from actual security threats is considered a major challenge in IoT [4]–[6]. There are several methods that can be applied here, but machine learning is considered one of the most promising areas capable of solving various different problems. Researchers have applied various techniques to improve the accuracy of machine learning (ML) algorithms, which have done much work to extend and improve the capacities of machines. ML processes work from the fact that learning, a natural part of human behavior, can be made into a vital part of machines as well. Deep learning (or deep neural network [DNN]) is a subcategory of ML that has been used in numerous applications to notable result, demonstrating its applicability for many real-world applications [5], [7]. Also called hierarchical learning, DL has been growing in popularity since 2006 as a new field in ML research, and deep learning (DL) techniques have evolved over the past years and affected many areas. DL can be defined as a subfield within ML that depends on the representations of several learning levels that correspond to a hierarchical structure of features, factors, or concepts, and high-level features, where concepts are defined by relying on lower-level concepts, and concepts at various levels may help the lower level itself define many top-level concepts [8].

The goal of the present study is to investigate and evaluate a DL model capable of detecting abnormal behavior in IoT networks. The model we propose here depends on DL, taking advantage of structures such as deep neural networks (DNN), deep certainty networks (DCN), and recurrent neural networks (RNN). In doing so, it seeks results that may realize the vision behind the current trend of researchers using DL in several fields of application, such as healthcare, banking services, industry, and many more.

In this work, we adopt DL techniques in order to propose a model for detecting abnormal suspicious behavior in IoT systems. Our model utilizes long short-term memory (LSTM) recurring neural networks (RNNs), a subset of DNNs. In order to assess the model's performance, we perform four different tests on Intel Labs dataset and compare the test results based on two criteria, F1 and accuracy. This remainder of this work is organized as follows. Section 2 presents a literature review, while Section 3 describes the proposed methodology for achieving our own model or algorithm. From there, Section 4 presents the results of these experiments and a discussion of our findings, and finally Section 5 serves as a conclusion, summarizing our research findings and providing recommendations for future work in this field.

2. LITERATURE REVIEW

There are several methods applied in different fields. However, ML is considered one of the current areas that may solve many different problems. Researchers have applied various techniques to enhance the performance and precision of ML algorithms in order to improve their results. Learning is a natural part of human behavior that can be turned into a dynamic part of machines as well. deep learning (DL) and deep neural networks (DNNs) constitute a subcategory of ML that has been analyzed and applied in numerous applications, where it achieves notable results that hold promise for various real-time applications [5], [7]. DL also sometimes called hierarchical learning, has been in use since 2006 as a new subfield of ML research, though over the past years it has evolved further. DL depends on learning several levels of representations (which correspond to a certain hierarchical structure of features, factors, or concepts) and high-level features. Here, important concepts are often defined by relying on lower-level concepts, while elsewhere this organization may help the lower level itself define many top-level concepts [8]. The following sub-sections describes various techniques of both ML and DL algorithms and then providers an overview of several previous studies.

2.1. Machine learning

Machine learning (ML) describes a field of computer science intended to enable computers to learn without programming directly from humans [5], [9]. ML was initially defined in 1959 by IBM engineer Arthur Samuel, who described the nascent area as follows: “A field of study that gives computers the ability to learn without being explicitly concluded” [10]. ML algorithms can be divided into supervised learning algorithms (including classification, regression, standardization, support vector machines (SVM), decision trees, Bayesian, and others) and unsupervised learning algorithms (such as clustering, neural networks, and others) [11]–[13]. Beyond these two types, several ML algorithms perform multiple functions [5], [7].

2.2. Deep learning techniques

Deep learning (DL) is one facet of a larger family of representational ML methods. DL builds from the fact that concepts and images can be represented in several ways, but at the same time there are certain representations of these images that make learning tasks faster and more efficient [8]. DL is the field of research that is interested in finding this definition of representations and how to teach machines using it. Table 1 summarizes the most important differences between ML and DL approaches and methods [14].

DL also includes its own methods, such as neural networks (NN) and recurrent neural networks (RNN) [15]. Different criteria are typically utilized in order to evaluate the performances of these methods such as the accuracy and F1-score (F1). Accuracy, which is constituted of the ratio of correctly projected views to total views, constitutes the most widely-used and accepted measure of performance. Meanwhile, F1 results take both positive and false positives into account, providing results that are only more useful than accuracy if researchers have a non-distribution equal for class.

Table 1. Comparison between machine learning and deep learning

	Machine Learning	Deep Learning
Data dependencies	Excellent tool for small/medium data set	An excellent tool for a Big data set
Devices dependencies	It works on standard devices (low-end)	Requires powerful devices
Engineering features	Request to understand the features that represent data	Not requested to better understand the features that represent data
Execution time	A few minutes to hours	Up to weeks, because of the calculation of weigh
Expandability	Some algorithms are easy to interpret, others are nearly impossible	Difficult to understand or impossible

Neural networks (NN) are a set of algorithms inspired by the performance of the human brain, which takes a large set of data and processes it to extract patterns from data. NN are classified according to the model of the connection between neurons to single-layer networks and multi-layer networks and this is called a structure. According to the method of assigning the weights accompanying the links to learning (which may be done in the presence of a supervisor or learning unsupervised), this is called "training" and is the process of the machine actually learning [15].

The main activity of NN is the classification and coding process. NN are approaches as general functions, which means that neural networks can learn to approximate any function (i.e. fx()) of the type of general approximation functions [16]. Recurrent neural networks (RNN) describe a NN model first proposed during the 1,980 by researchers attempting to model time series [17]. RNN is considered to be the first algorithm that uses a memory to save or remember the previous entries from a large set of serial data. The structure of this network is like a standard multi-layer receiver but differs in allowing the connections between hidden layers that associated with a specific time delay. The model will preserve knowledge about the past through these connections, enabling it to discover the temporal relations in the data between events far apart from one another. Traditional NNs assume that every input and every output is independent of every other input and output, which in many cases does not provide optimal efficiency. For example, when attempting to predict the next word or phrase in a full sentence, better and faster results will be possible if the previous word(s) are known. RNNs rely on a similarly iterative chain of information, and then perform an identical task at each segment of the chain, wherein the output relies on prior calculations while also preserving memories that keep the information calculation). Figure 1 illustrate a pattern of RNNs [18], [19].

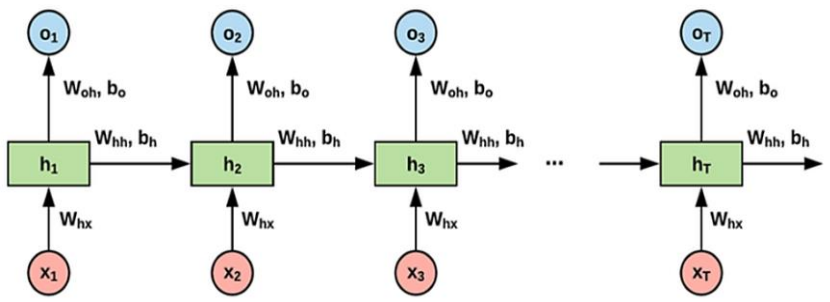


Figure 1. Pattern of recurrent neural networks (RNN)

In Figure 1, x_T represent the input data (i.e. x_1 , x_2 , x_3, …, x_T), while, o_r represents the next expected (predicate) data (i.e. o_1 , o_2 , o_3, …, o_r), and h_r represents the hidden layers that contains information about the previous income data. The following equation is used in the network training process:

$$h_r = f\left(WW^{(hh)}\, h_r + W^{(hx)}\right) \tag{1}$$

where h_T holds the information about the previous data in the chain and calculated by using the previous h_{r-1} beam and the x_r input current beam. A non-linear activation function f (usually *tanh*, *sigmoid*, or *ReLU*) is also applied upon the final assembly. Here, it is reasonable to take it that h_0 is a zero beam: $y_r = softmax(W^{(s)} h_r)$ that calculates the expected beam of data at a given time step, or t. In this instance, we use the *softmax* function to get a beam (1, V) whose sum of all elements is equal to one. This probability distribution provides us with the following (and most probable) income indicator from that income data:

$$J^{(r)}(\theta) = \sum_{i=1}^{|V|} (yy_{ri} \log y_{ri}) \tag{2}$$

Here we use what is known as loss function to account for the continued loss of cross entropy at each t in order to calculate the level of error between the expected statement and the actual result, which is usually referred to as the average squared error or mean squared error (MSE). Regrettably, if we implement the steps above, then the results will not be satisfactory. This is because the simple model of RNNs comes with a major weakness, often termed the vanishing hierarchy problem that can prevent these networks from achieving accuracy. In short, this issue stems from the way in which each step during the RNNs training will utilize the exact same weights as calculation y_r [20]. This result is also calculated during the back propagation; here, whenever we move backwards, an error signal develops, which may be greater or smaller depending on the particular circumstances. This error's presence, though, means that the RNN itself is having difficulty storing remote data in the chain and because of this, is making predictions based on simply its most recent inputs. As a result, many recent models have had extreme difficulty in either (or both) training RNNs on the one hand and the inability to handle long strings using tanh or ReLU as activation functions on the other.

These limitations of RNNs can be handled with more powerful models, including LSTM and gated recurrent unit (GRU) that can resolve the recurring issues of fading gradients in ways that enable researchers to still implement RNNs usefully [20]. LSTMs in particular are able to learn short-term dependencies for a longer period, as introduced by Olah [19] and later revised and circulated by many more researchers [19], [21]. LSTM RNNs are designed specifically to try and avoid these long-term dependency problems that cause the fading gradient issue in other RNNs. Its advantage in this area is that its default performance method is memorizing information inputs for longer periods of time than many other RNNs do [21]. Because of these advantages, our proposed model will adopt this type of RNN. All RNNs utilize a series of recurring units, and in more standard types of RNN, this recurrent unit usually a very simple structure, for example one tanh layer. With LSTM RNNs, though, the recurrent unit has a dissimilar structure, utilizing four neural network layers that interact in particular ways, rather than a single layer that follows a simpler pattern. Figure 2 depicts the LSTM RNN symbols, while Figure 3 shows the recurrence units in an LSTM RNN with its gates and Figure 4 demonstrates the LSTM RNN gates [14], [21], [22].

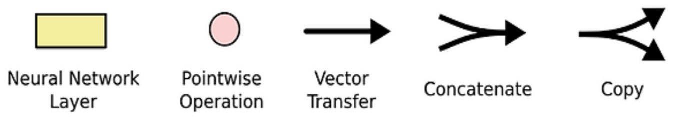


Figure 2. LSTM RNN symbols [13]

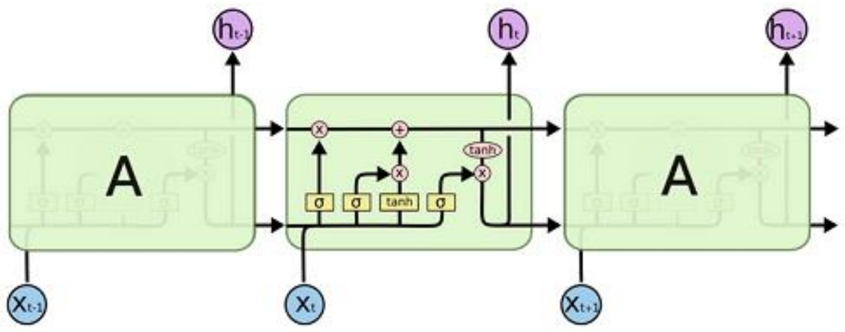


Figure 3. Recurrence units in LTSM

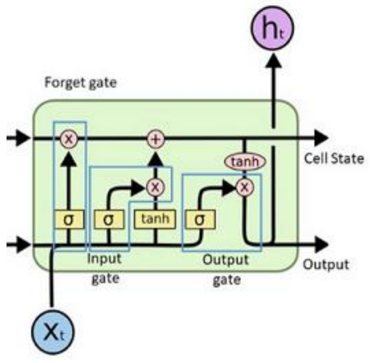


Figure 4. LTSM RNN gates RNN with its gates

As shown in Figure 4, the recurring units in the LSTM have three gates (forget gate, input gate, output gate):

1) Forgetting gate: decides how long you should remember and forget what is left as in the following equation. This portal defines information that must be removed from the cell in a specific time step (t) as decided using the $sigmoid$ (σ) function. The function considers the previous state (h_{r-1}) and the current content of both the current input (x_r) and outputs, which will be a number whose value is either *0* (to ignore) or *1* (to keep) for each number in the case of cell (c_{r-1}).

$$f_r = \sigma\left(W_f \,.\, [h_{r-1}, x_r] + \ b_f\right] \tag{3}$$

2) Input gateway update: decides the amount added from this unit to the current state as in the following equations and then defines any new information that will be stored in that cell case. There are two parts of this particular step. First, the $sigmoid(\sigma)$ layer determines which values require updating, and next, the $tanh$ layer creates a new candidate filter beam to be added on here.

$$i_r = \ \sigma(W_i \,.\, [h_{r-1}, x_r] + b_i) \tag{4}$$

$$\tilde{C} = tanh(W_c \,.\, [h_{r-1}, x_r] + b_c) \tag{5}$$

The second step at this gate combines those two actions in order to create and then update the status. Thus, we will update the state of the old cell $C_{r-1}$to the status of the new cell C_r as demonstrated in the following (6). The previous gate and the steps it has set in motion have already determined the actions to be precipitated here, so all that is required now is there implementation. Here, we can multiply the old case by f_r in order to forget any things that we had earlier determined to let go of at this stage. From here we then add $i_r \ * \ \tilde{C}$. These are the new candidate values, which were measured by how much we decided to update in each individual case.

$$C_r = \ f_r * \ C_{r-1} + \ i_r * \ \tilde{C}r \tag{6}$$

3) Output gate: Decides which segment of the current cell will be made into the output as in the following equations. The $Sigmoid$ function determines the values that should be allowed by *0* and *1*. The *tanh* function gives weight to those values passed and determines their level of significance as calculated from *-1* to *1* and then multiplies it by the $sigmoid(\sigma)$ output, thereby leaving only the parts that we have already decided to output

$$O_r = \ \sigma(W_o \ .\, [h_{r-1}, x_r \,] + b_a) \tag{7}$$

$$h_r = O_r * tan\, h(C_r) \tag{8}$$

2.3. Previous studies

The literature includes several approaches that have been applied to solve different problems, A selection of these includes the following. Argyros *et al.* [22] present the deep packet analysis mechanism used the REST call of websites. They attempt to learn correct communication patterns using the black box analysis methodology. Their approach is based on symbolic automation and periodic exploration in order to identify patterns.

DeMarinis and Fonseca [23] compare IoT data traffic with Internet data passage. They concluded that IoT data traffic has more regular patterns than others, which shows the benefit of using periodic exploration. Javaid *et al.* [24] proposed the use of DL applications to detect suspicious behavior in networks, which they did using the NSL-KDD dataset. In their work, they followed a self-learning DL scheme in which the advantage of obtaining unsupervised advantages from training data was employed using a sparse autoencoder. The two attributes of "attack" or "natural" were learned and then the DL was applied to a new test dataset to evaluate the speed and accuracy of its classification. They used the K-fold cross-validation technique to evaluate the model's performance, obtaining reasonable but not outstanding results. Another work related to the present study regards the application of automatic coding to detect abnormalities, which was introduced by Sakurada and Yairi [25]. Here the natural state of the network was learned by means of automatic coding through the mechanism of reducing non-linear features. Sakurada and Yairi were able to demonstrate that normal records from their test dataset had a small reconfiguration plan but this error was significant for revealing abnormal records in the dataset itself.

DL methods were also applied by Li *et al.* [26] to detect malicious code by using automatic code extraction and deep belief networks (DBN) to constitute a detection class. Their article showed that a hybrid method, such as the extraction technique they utilized, could be more efficient and yield more accurate results than could be achieved by utilizing DBN alone. This work also verified that "deep" networks outperform their shallow counterparts in detecting cyber attacks. Wang *et al.* [27] also investigated the possibilities of DL systems for the detection of malicious code. In this work, de-noising autoencoders were implemented to achieve deeper feature learning and thus distinguish between malicious and natural code, particularly JavaScript. Results demonstrated a promising degree of accuracy and seemed effective when applied to web applications, but it would be difficult to apply these findings to distributed loT/Fog systems.

Hasan *et al.* [28] applied several traditional machine learning techniques to predict attacks and detect abnormalities within IoT systems and eventually reached the following values of accuracy: 98.3% achieved when using the "logistic regression" mechanism, 98.2% achieved from the automated support beam mechanism, and 99.4% resulting from use of the decision tree mechanism, the "random purpose" mechanism, and the industrial neural networks mechanism all at once. Where Pahl and Aubet [29] used k-means algorithm and the BIRCH algorithm for classification and obtained 96.3% accuracy, Diro and Chilamkurti [30] developed an anomaly detector based on machine learning methodology using neural network mechanisms and an open source data set NSL-KDD with an accuracy of 98.27%. Alghuried [31] also used the same dataset on a model for detecting anomalies regarding IoT systems using traditional ML techniques and a combination of two particular ML algorithms: inverse weight cluster (IWC) and decision tree algorithm (C4.5). From this, their model achieved 97.0% accuracy. Meanwhile, D'angelo *et al.* [32] proposed an ML model for classification doubling that achieved 94.1% accuracy. Finally, Bru *et al.* [33] used DNNs to detect attacks within the home environments connected to the internet of things similar to [34], [35]. Table 2 summarizes the relevant research that has been conducted on the subject of detecting suspicious behavior in IoT through the use of ML techniques.

Table 2. Several methods to detecting suspicious behavior in IOT using machine learning techniques

Publication date	Publishing group	Research methods	Dataset Used	Accuracy ratio	Error ratio MSE
2021	Agarwal *et al.* [35]	- Support vector machines (SVM)	UNSW-NB15	- 97.7777	-
		- K-nearest neighbor (kNN)		- 93.3333	
		- Naïve Bayes (NB)		- 95.5555	
2020	Zerkouk and Chikhaoui [34]	- Long short term memory (LSTM)	SIMADL	- 94%	-
		- Convolutional neural network (CNN		- 93%	
		- CNN-LSTM		- 98%	
		- Autoencoder-CNN-LSTM		- 84%	
2019	Hasan *et al.* [28]	- Logistic regression.	Especially [36]	- 98.3%	- 1.7%
		- support vector machines		- 98.2%	- 1.8%
		- Decision trees		- 99.4%	- 0.6%
		- Random forest		- 99.4%	- 0.6%
		- Neural networks		- 99.4%	- 0.6%
2018	Pahl and Aubet [29]	K-means and BIRCH	Especially [36]	96.3%	3.7%
2018	Diro and Chilamkurti [30]	Deep neuron networks	NSL-KDD [37]	98.27%	1.73%
2017	Alghuried [31]	IWC and C4.5	Intel Lab Data [38]	97.0%	3.0%
2015	D'Angelo *et al.* [32]	U-BRAIN	NSL-KDD [37]	94.1%	5.9%
2018	Brun *et al.* [33]	Deep neuron networks	Especially [36]	unavailable	unavailable

3. METHOD

This work is intended to develop an algorithm or model with its process to detect abnormal suspicious behavior in IoT systems. As discussed earlier, powerful DL techniques such as RNNs with LSTM as introduced by Olah [19], have been used to great effect in many fields. LSTM in particular has proven able to learn short-term dependencies for a longer period, and LSTM RNNs are designed specifically to try and avoid the common long-term dependency problem that causes the fading gradient of a neuron network. An LSTM RNN can memorize the needed information for longer times, which enables it to avoid this problem more easily by default [21]. LSTM RNNs also employ four layers that interact in particular ways as described above, adding a further degree of protection. Because of these capacities, we adopted LSTM RNNs to fulfill our proposed model for finding a solution that can identify malicious and unauthorized uses of IoT. The followed subsection details both the datasets used to train our model and the algorithm's own design.

3.1. Dataset

Input data is a data set that we must process before beginning to utilize it in developing our proposed model. This is necessary because some datasets may not include an attribute label, since they were not initially intended or prepared for research purposes on the detection of suspicious behavior. Also, such datasets may include a large number of features that such research will not require, so we should ignore these and keep only the features that we need. Finally, we will generate anomalies (or “abnormal” results) and add them to the input data manually.

The benefits of adopting such an approach to a training dataset [39] are i) the input data has been addressed and defined in some way to the ML model; ii) the input data set has been configured into an eligible data set that will help teach the ML model to detect anomalies, and thus is suitable for research purposes; iii) the input data has been modified to meet research needs in ways that overcome the usual difficulties of obtaining an IoT income dataset at prohibitively high cost.

The dataset used in this work is from Intel Labs and it includes over 2.3 million reads from 54 sensors distributed across the Intel Berkeley research laboratory [38]. As noted by Intel Labs, each of these sensors here can collect particular types of data (light, voltage, temperature, and humidity) and will transfer all of this data back to the gateway on a schedule of once every 31 seconds. In addition, each reading in the dataset is marked by the date, the time, and the ID of the sending sensor. This Intel Labs dataset is freely and readily available, which means that it has been used as a dataset for many different problems and it was not gathered or presented specifically for research into uncovering data on suspicious behavior in IoT devices. To overcome this potential shortcoming, we processed the dataset as discussed next.

Processing over 2.3 million pieces of data can be computationally expensive. To resolve this, we took a random sample from the total number of readings and saved it in a separate file. We also omitted unnecessary features (columns) such as the identifying information (time, date, and sensor ID) as well as both light, and voltage: these were omitted due to our research focus on a proposed anomaly detection. Finally, anomalies were generated and added to that processed file of input data manually [39].

3.2. The algorithm

As mentioned earlier, our proposed model or algorithm is an adapted LSTM RNNs [19] intended to detect certain issues in IoT. The model consists of several steps as in Figure 5. As shown in Figure 5, there are several steps represents the mechanism or process that we follow the build the model. These steps (i.e., read dataset, feature definition, encoding of variables, divide the dataset [leaning and training], reducing MSE error [actual output], obtained model trained, realization of proposed model, convert to TensorFlow data, and predict the test data) are used to build the model in five stages (i.e., initialization, pre-processing, realization, training, and results stages).

In the initialization stage, all the test intermediaries are first defined and then configured with their fixed values. Also, both the training and the testing datasets are examined and stored in the specified variables. Record defaults are also configured and variables that must be reconfigured are made in order to prevent math errors. In the stage of data pre-processing, the feature and records names for each dataset are examined and extracted from the variables specified in the initialization stage and then placed in their respective elements. During the realization stage, a five-layer LSTM RNN model is utilized. At this point, the random weights, biases, and nodes that are randomly generated for the models are defined as well as the computing function (ReLU). The training stage of the algorithm architecture is part of the code that carries out the model test run. To accomplish this, we utilized three datasets (one apiece for training, testing, and frequency or epochs data) and recorded our model's results on each one. These results were then used in order to calculate metrics for success such as the cost and accuracy function. The results of these success metrics are used in order to improve the computed learning rate and various training steps.

The TensorFlow platform is a second-generation ML system that was created to succeed the earlier system DistBelief. TensorFlow is an open-source library collecting various models and algorithms for numerical computing, ML, and DL. Our adopted model was programmed by using Python to provide a suitable API for the front end of building applications which were then saved in high-performance C++ language. Figure 6 provides an overview of sample TensorFlow code for a basic LSTM cell.

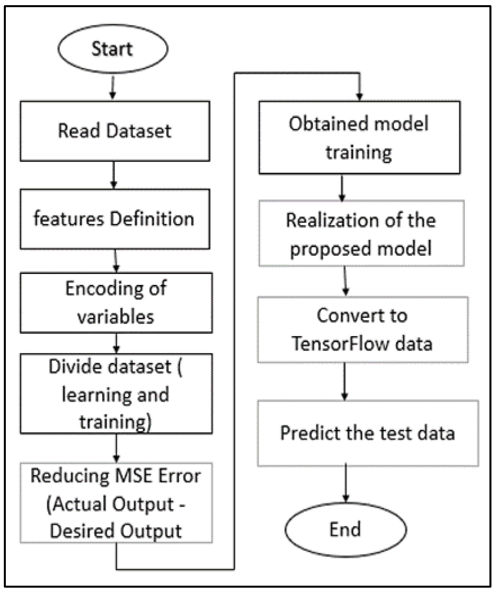
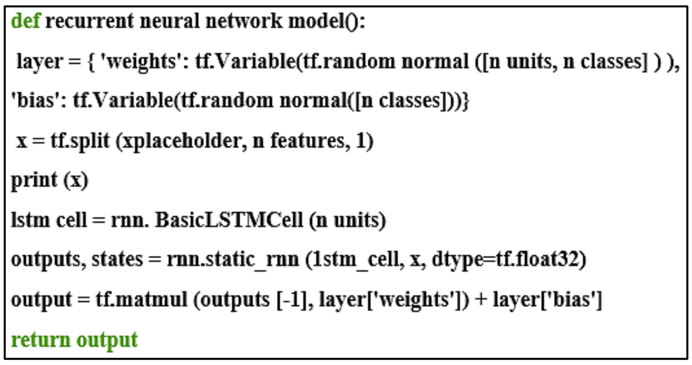

Start
Read Dataset
features Definition
Encoding of variables
Divide dataset (learning and training)
Reducing MSE Error (Actual Output - Desired Output
Obtained model training
Realization of the proposed model
Convert to TensorFlow data
Predict the test data
End

Figure 5. the process of the proposed model or algorithm



```
def recurrent neural network model():
 layer = { 'weights': tf.Variable(tf.random normal ([n units, n classes] ) ),
'bias': tf.Variable(tf.random normal([n classes]))}
 x = tf.split (xplaceholder, n features, 1)
print (x)
lstm cell = rnn. BasicLSTMCell (n units)
outputs, states = rnn.static_rnn (1stm_cell, x, dtype=tf.float32)
output = tf.matmul (outputs [-1], layer['weights']) + layer['bias']
return output
```

Figure 6. Sample TensorFlow code for a basic LSTM cell

4. EXPERIMENTAL RESULTS

4.1. Experimental settings

This research conducted preliminary experiments to determine the suitable configuration of the model for a proper data analysis. At first, it is implemented with its default parameter settings. Then, its parameter values are carefully tuned to obtain the desired results. Using TensorFlow (Python), experiments are conducted on a Windows 10 machine with CoreTM i7-7700HQ processor, 3.80 GHz CPU and 8 of RAM.

4.2. Experimental results

This work conducted over four tests to evaluate the performance of the model. These tests were performed on the basis of seven mediators (i.e., weights, biases, number of layers, learning rate, number of nodes, size of the dataset, and iterative periods). In the seven mediators, the weights and biases were randomly generated, while the values of layers, nodes, and learning rate remained constant and the set of repetitive data and periods were variable in all tests.

To evaluate our model’s performance, we utilized two criteria: time spent (i.e., MSE) and accuracy. These two metrics enabled us to evaluate the results that our model provided for the training data. Table 3 shows the mediators used in the first, second, third and fourth tests. Where in the first test our model performed upon small datasets (e.g., 5,000 data points apiece) in 100 periods. In the second test our goal was to determine whether our model could maintain a certain level of accuracy in detection: we were seeking a high rate with reasonable time similar to the one achieved in the first test while also increasing the data sets' volume and periods by 100%. Our third test and fourth test proceeded in the same way: for each one, we increased the dataset volume and dataset period by 100% beyond the previous test and observed whether our model could maintain a high degree of accuracy and a reasonable timeframe.

Following Table 3, Table 4 shows the first, second, third, and fourth test’s training results in terms of scores for accuracy, time, and F1. Likewise, Figure 7 illustrates the graph of loss obtained by our model from TensorFlow for the first test, Figure 8 for the second test, Figure 9 for the third test, and Figure 10 for the fourth test.

Table 3. Tests mediators

Test mediators	data sets volume	Periods	Learning rate	nodes	Layers	Weights	Biases
First test	5000	100	0.1	40	5	Random	Random
Second test	10000	200	0.1	40	5	Random	Random
Third test	20000	400	0.1	40	5	Random	Random
Fourth test	40000	800	0.1	40	5	Random	Random

Table 4. Tests training results

First test		Second test		Third test		Fourth test	
Epoch	Loss	Epoch	Loss	Epoch	Loss	Epoch	Loss
0	0.65207316	100	3.89E-15	-	-	-	-
10	1.35772E-13	110	3.89E-15	-	-	-	-
20	1.35771E-13	120	3.89E-15	-	-	-	-
30	1.35771E-13	130	3.89E-15	-	-	-	-
40	1.35771E-13	140	3.89E-15	340	2.96E-08	740	6.20644E-08
50	1.35766E-13	150	3.88485E-15	350	3.01E-08	750	5.29224E-08
60	1.35755E-13	160	3.88253E-15	360	3.3E-08	760	6.13865E-08
70	1.35727E-13	170	3.88021E-15	370	3.25E-08	770	5.21E-08
80	1.35672E-13	180	3.87789E-15	380	3.19E-08	780	6.29E-08
90	1.35572E-13	190	3.87558E-15	390	3.08E-08	790	5.13865E-08
F1 Score:	0.499670616		0.499670616		0.519473631		0.494739875
Accuracy Score:	0.998683332		0.998683332		0.996681998		0.998568974
Time spent	507		2436		8931		23459

The results in Table 4 for the first test and Figure 7 demonstrate that the model we have developed offers a remarkably high rate of detection and a correspondingly low rate of false alarms. Moreover, the value of the loss continues to decrease with the continuation of the model’s training, but even the first test already achieves a 99.87% accuracy rate in detection. These results demonstrate that LSTM RNN technology is a highly valuable means of detecting suspicious behavior in IoT accurately and efficiently.

Results in Figure 8 shows that the loss decreased when the training model data was increased. Meanwhile, the results in Table 4 for the second test show that the accuracy is 99.87%, which is comparable with the first test despite the increased. Results also show that the training time of the second test is increase by 4 times, or 480.473%. This increased time is due to increasing both the frequency and the dataset size. These results also support our assertion that LSTM RNN technology can detect suspicious behavior in IoT data efficiently and accurately.

Results in Table 4 for the third test and Figure 9 shows that the model has a very high detection rate and a low false alarm rate, with 400 iterations the model completed the detection with accuracy of 99%.67. Results also shows that the training time is decreased to 366.63 % (which is decreased 3.6 times) while comparing with the second test. Meanwhile, the accuracy of the final detection decreased to 99.67%, but this is still considered a very high accuracy, which supports the fact that LSTM RNN is able to detect suspicious behavior in IoT accurately and efficiently.

Results in Figure 10 show the model’s different training stages, demonstrating how the entire process took 6 hours, 30 minutes and 59 seconds to complete from beginning to end. Results in both Figure 10 and Table 4 for the fourth test demonstrate how the model begins to decrease its loss rates as it continues with training until it reaches a stage where the loss values become very close to each other. Results also reveal that the developed model has a remarkably high rate of detection and a very low rate of false alarms. Over 800 iterations were conducted by the algorithm, and the model achieved a detection accuracy of 99.86%. While comparing the results between the third and fourth tests, we notice that the accuracy is increased by 0.19% in the model final detection, which indicates that the accuracy of detection in the fourth test is higher than the third test. Furthermore, the training time is decreased to 262.67%, where in the third test was 366.63%. This supports the fact that LSTM RNN technology is able to detect suspicious behavior in IoT accurately and efficiently.

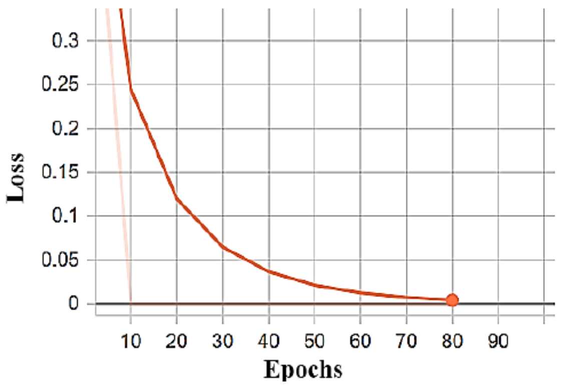


Figure 7. The graph of a loss in the first test

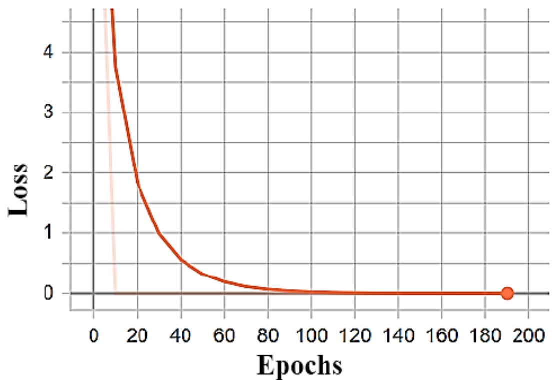


Figure 8. The graph of a loss in the second test

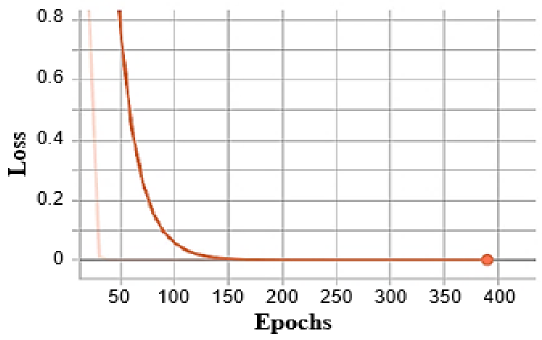


Figure 9. The graph of a loss in the third test

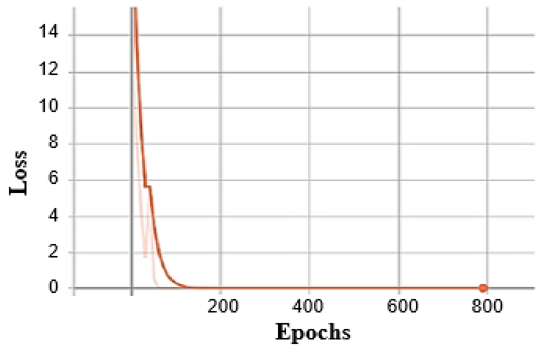


Figure 10. The graph of a loss in the fourth test

4.3. Results discussion

Figure 11 shows the relationship between 4 tests based on datasets size, final accuracy and frequency, while Figure 12 shows the data set size and accuracy analysis (where the horizontal axis represent the four tests results and the vertical axis represent the accuracy), Figure 13 illustrates the relationships between frequency and accuracy of model detection (where the horizontal axis represent the four tests results, the blue line respresent the accuracy results achieved and the dark red dash line represent the frequency results achieved) and Table 5 summarizes the four tests' results based on the following mediator:

a) The final accuracy of the fully-trained LSTM RNN model: represents ultimate results achieved by the model.
b) Average mean square error (MSE): represents the MSE values achieved with training outputs.
c) Dataset: represents the volume of data points and records used during the model's training process.
d) Frequency: represents the overall number of times that the model was trained in behavior features.
e) Time: represents the time needed for the model to complete each test.
f) Percentage increase in time: represents time as a percentage increase across all four tests.

Figure 11 shows a comparison between the four tests based on the datasets used and the results achieved for time, accuracy, and frequency. As this figure demonstrates, the first and second tests each achieved 99.87% accuracy, where the model shows the data with distinctive features. However, as the size of the dataset is increased by 100% in the third test, there was also a 0.2% decrease in accuracy, resulting in a final accuracy of 99.67%, which may have resulted from the increased dataset size or the model's exposure to data with different characteristics. However, when we increased the size of the dataset and maintained similar characteristics, the rate of accuracy detection increased again, rising back up to 99.86%. This result supports our hypothesis that the model's accuracy depends at least in part on the characteristics of the data that it is trained on.

Frequency describes the ultimate number of times that the algorithm has been exposed to different data attributes over the course of its training. From the various forms of testing done on our proposed model, it was determined that use of a larger dataset would mean that the model would require additional iterations in order to learn differences between different data attributes and increase its own ability to distinguish among them. Therefore, we did not use a uniform iteration of times across all stages of training, but instead increased times as the datasets' own sizes increased. Figure 12 shows a comparison between the four tests based on the model accuracy and Figure 13 shows the relationships between frequency and accuracy of model detection.

Table 5. The results of the model for the four tests

Standard	First test	Second test	Third Test	Fourth Test
Final accuracy	0.9987	0.9987	0.9967	0.9986
Average (MSE)	0.0013	0.0013	0.0033	0.0014
Date set	5000	10000	20000	40000
Frequency	100	200	400	800
Time (Second)	507	2436	8931	23459
Increase % in time	0	380.473	266.63	162.67

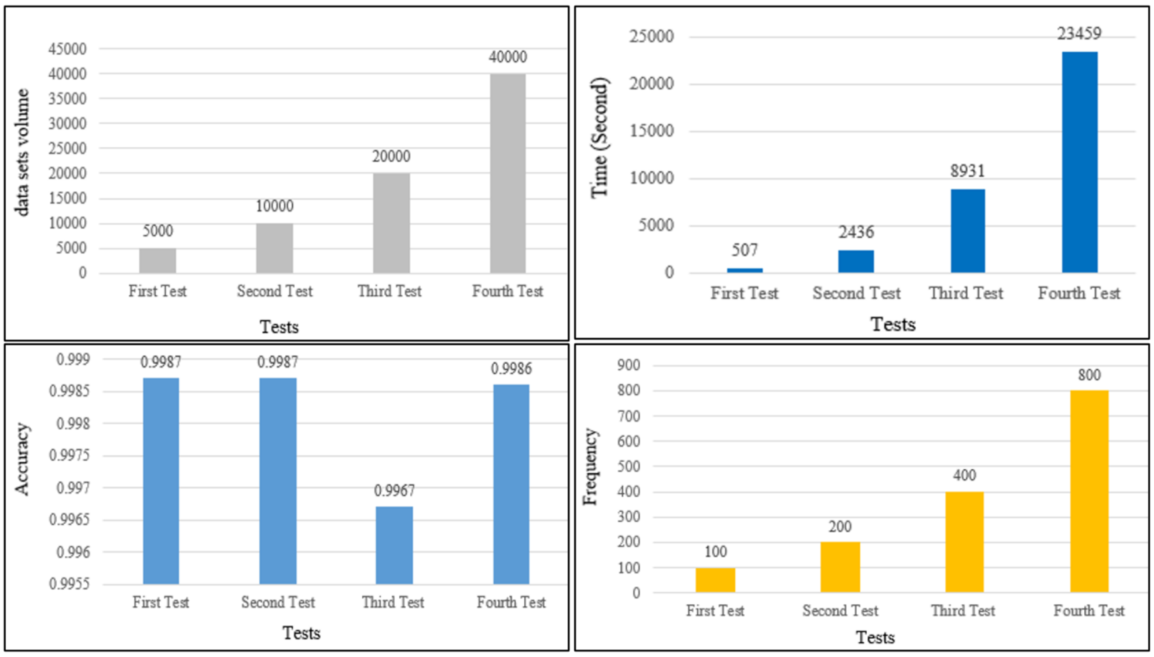


Figure 11. Relationship between 4 tests

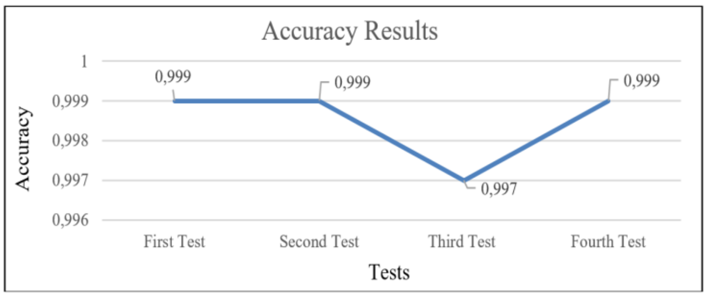


Figure 12. Data set size and accuracy analysis

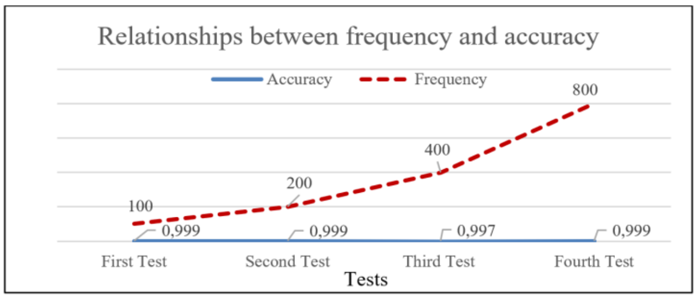


Figure 13. Accuracy and frequency analysis

Figure 13 shows a comparison among the four tests (as in horizontal axis) based on the accuracy and frequency the model achieved in each one, while also comparing the results of its training with the results achieved by models trained using traditional ML algorithms, which the researchers suggested in [30]. Here we noticed significant differences between the final detection of accuracy rates achieved by each model and the weighting of the cuff of the model we developed with an average detection accuracy of 99.8% (across the four tests) versus 97% for the traditional model. This difference indicates a profound learning proficiency in detecting suspicious behavior within IoT networks.

5. CONCLUSION

This work has investigated the detection of abnormal behaviors in IoT networks, which constitute one of the most important and widespread networks in use today. An adapted LSTM RNN was programmed using Python and used to produce our model. The second-generation Google ML framework called TensorFlow was used to determine the efficacy and efficiency of our proposed model, along with the LSTM's own features and the following mathematical formulas. To best evaluate our model's performance, four tests were performed based on seven mediators and two criteria, wherein the mediators of data size and period were changed with each test in order to evaluate the effectiveness of the model. Where all of the tests performed were assigned different values of volume and iterative periods, randomly-generated values are used for weights and biases as well as the various layers, nodes, and learning rate until the model was able to achieve the best rate of accuracy detection. The results obtained demonstrate that our model performs better when the dataset increases, and its accuracy does not depend on the platform used to train it. The results also shows that the model developed has an average detection accuracy of 99.8%, meaning that LSTM RNN is able to detect suspicious behavior for different types of IoT sensor data. However, the model may produce even better results in terms of accuracy when using a higher-quality processor. Thus, we recommend that future work trains its model on computing platforms other than the CPU, perhaps such as graphics processors (GPU) for example, and then compare the results achieved with each of these approaches.

REFERNCES

[1] K. Ashton, "Kevin Ashton May Change the World," *RFID Journal*. pp. 25–36, 2002.
[2] D. R. Kiran, "Internet of Things," *Production Planning and Control*, pp. 495–513, Jan. 2019, doi: 10.1016/B978-0-12-818364-9.00035-4.
[3] A. Khanna and S. Kaur, "Internet of things (IoT), applications and challenges: a comprehensive review," *Wireless Personal Communications*, vol. 114, no. 2, pp. 1687–1762, May 2020, doi: 10.1007/s11277-020-07446-4.
[4] S. Y. Lee, S. R. Wi, E. Seo, J. K. Jung, and T. M. Chung, "ProFiOt: abnormal behavior profiling (ABP) of IoT devices based on a machine learning approach," in *2017 27th International Telecommunication Networks and Applications Conference, ITNAC 2017*, Nov. 2017, vol. 2017-Janua, pp. 1–6, doi: 10.1109/ATNAC.2017.8215434.
[5] P. A. Savenkov and A. N. Ivutin, "Methods of machine learning in system abnormal behavior detection," in *In: Tan Y., Shi Y., Tuba M. (eds) Advances in Swarm Intelligence, ICSI 2020*, Springer International Publishing, 2020, pp. 495–505.
[6] M. Koutli, N. Theologou, A. Tryferidis, and D. Tzovaras, "Abnormal behavior detection for elderly people living alone leveraging IoT sensors," in *Proceedings - 2019 IEEE 19th International Conference on Bioinformatics and Bioengineering, BIBE 2019*, Oct. 2019, pp. 922–926, doi: 10.1109/BIBE.2019.00173.
[7] M. Bertolini, D. Mezzogori, M. Neroni, and F. Zammori, "Machine learning for industrial applications: A comprehensive literature review," *Expert Systems with Applications*, vol. 175, p. 114820, Aug. 2021, doi: 10.1016/j.eswa.2021.114820.
[8] L. Deng and D. Yu, "Deep learning: Methods and applications," *Foundations and Trends in Signal Processing*, vol. 7, no. 3–4. Now Publishers, pp. 197–387, 2013, doi: 10.1561/2000000039.
[9] J. Bell, *Machine learning: hands-on for developers and technical professionals*. Wiley, 2020.
[10] A. L. Samuel, "Some studies in machine learning using the game of checkers," *in IBM Journal of Research and Development*, vol. 3, no. 3, pp. 210–229, Jul. 1959, doi: 10.1147/rd.33.0210.
[11] M. Al-kmali, H. Mugahed, W. Boulila, M. Al-Sarem, and A. Abuhamdah, "A machine-learning based approach to support academic decision-making at higher educational institutions," Oct. 2020, doi: 10.1109/isncc49221.2020.9297177.

[12] J. H.Alkhateeb, A. A.Turani, A. Abuhamdah, M. J. A. Sara, and M. F. J. Klaib, “An effective deep learning approach for improving off-line Arabic handwritten character recognition,” *International Journal of Software Engineering and Computer Systems*, vol. 6, no. 2, pp. 104–112, 2020, doi: 10.15282/ijsecs.6.2.2020.7.0076.

[13] J. Brownlee, “A tour of machine learning algorithms,” *Machine Learning Algorithms*, 2019.

[14] S. A. Bini, “Artificial intelligence, machine learning, deep learning, and cognitive computing: what do these terms mean and how will they impact health care?,” *International Journal of Software Engineering and Computer Systems*, vol. 33, no. 8, pp. 2358–2361, Aug. 2018, doi: 10.1016/j.arth.2018.02.067.

[15] L. K. Hansen and P. Salamon, “Neural network ensembles,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990, doi: 10.1109/34.58871.

[16] J. Zurada, “Introduction to artificial neural systems,” *West Publishing Co.*, 1992.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, doi: 10.1038/323533a0.

[18] B. C. Csaji, “Approximation with artificial neural networks,” Etvs Lornd University, Hungary, 2001.

[19] C. Olah, “Understanding LSTM networks,” *github*. 2015, Accessed: Mar. 02, 2021. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs.

[20] S. Hochreiter, “The vanishing gradient problem during learning recurrent neural nets and problem solutions,” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, Apr. 1998, doi: 10.1142/s0218488598000094.

[21] R. C. Staudemeyer and E. R. Morris, “Understanding LSTM-a tutorial into long short-term memory recurrent neural networks,” *arXiv preprint arXiv:1909.0958*, Sep. 2019.

[22] G. Argyros, I. Stais, A. Kiayias, and A. D. Keromytis, “Back in black: towards formal, black box analysis of sanitizers and filters,” May 2016, doi: 10.1109/sp.2016.14.

[23] N. DeMarinis and R. Fonseca, “Toward usable network traffic policies for IoT devices in consumer networks,” in *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, Nov. 2017, pp. 43–48, doi: 10.1145/3139937.3139949.

[24] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, “A deep learning approach for network intrusion detection system,” *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ACM, 2016, doi: 10.4108/eai.3-12-2015.2262516.

[25] M. Sakurada and T. Yairi, “Anomaly detection using autoencoders with nonlinear dimensionality reduction,” in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis-MLSDA’14*, 2014, pp. 4–11, doi: 10.1145/2689746.2689747.

[26] Y. Li, R. Ma, and R. Jiao, “A hybrid malicious code detection method based on deep learning,” *International Journal of Software Engineering and its Applications*, vol. 9, no. 5, pp. 205–216, May 2015, doi: 10.14257/ijsia.2015.9.5.21.

[27] Y. Wang, W. Cai, and P. Wei, “A deep learning approach for detecting malicious JavaScript code,” *Security and Communication Networks*, vol. 9, no. 11, pp. 1520–1534, Jul. 2016, doi: 10.1002/sec.1441.

[28] M. Hasan, M. M. Islam, M. I. I. Zarif, and M. M. A. Hashem, “Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches,” *Internet of Things*, vol. 7, p. 100059, Sep. 2019, doi: 10.1016/j.iot.2019.100059.

[29] M. O. Pahl and F. X. Aubet, “All eyes on you: distributed multi-dimensional IoT microservice anomaly detection,” in *14th International Conference on Network and Service Management, CNSM 2018 and Workshops, 1st International Workshop on High-Precision Networks Operations and Control, HiPNet 2018 and 1st Workshop on Segment Routing and Service Function Chaining, SR+SFC 2*, 2018, pp. 72–80.

[30] A. A. Diro and N. Chilamkurti, “Distributed attack detection scheme using deep learning approach for internet of things,” *Future Generation Computer Systems*, vol. 82, pp. 761–768, May 2018, doi: 10.1016/j.future.2017.08.043.

[31] A. Alghuried, “A model for anomalies detection in internet of things (IoT) using inverse weight clustering and decision tree,” 2017.

[32] G. D’angelo, F. Palmieri, M. Ficco, and S. Rampone, “An uncertainty-managing batch relevance-based approach to network anomaly detection,” *Applied Soft Computing*, vol. 36, pp. 408–418, Nov. 2015, doi: 10.1016/j.asoc.2015.07.029.

[33] O. Brun, Y. Yin, and E. Gelenbe, “Deep learning with dense random neural network for detecting attacks against IoT-connected home environments,” *Procedia Computer Science*, vol. 134, pp. 458–463, 2018, doi: 10.1016/j.procs.2018.07.183.

[34] M. Zerkouk and B. Chikhaoui, “Spatio-temporal abnormal behavior prediction in elderly persons using deep learning models,” *Sensors*, vol. 20, no. 8, p. 2359, Apr. 2020, doi: 10.3390/s20082359.

[35] A. Agarwal, P. Sharma, M. Alshehri, A. A. Mohamed, and O. Alfarraj, “Classification model for accuracy and intrusion detection using machine learning approach,” *PeerJ Computer Science*, vol. 7, p. e437, Apr. 2021, doi: 10.7717/peerj-cs.437.

[36] F. X. Aubet, “DS2OS traffic traces,” *Kaggle*. 2018, Accessed: Mar. 06, 2021. [Online]. Available: https://www.kaggle.com/francoisxa/ds2ostraffictraces.

[37] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the KDD CUP 99 data set,” Jul. 2009, doi: 10.1109/CISDA.2009.5356528.

[38] P. Bodik, W. Hong, C. Guestrin, S. Madden, M. Paskin, and R. Thibaux, “Intel lab data,” *Intel Berkeley Research lab*. [Online]. Available: http://db.csail.mit.edu/labdata/labdata.html.

[39] R. Fu, K. Zheng, D. Zhang, and Y. Yang, “An intrusion detection scheme based on anomaly mining in Internet of Things,” in *4th IET International Conference on Wireless, Mobile & Multimedia Networks (ICWMMN 2011)*, 2011, pp. 315–320, doi: 10.1049/cp.2011.1014.