Sustaining the Effectiveness of IoT-Driven Intrusion Detection over Time: Defeating Concept and Data Drifts

Omar Abdel Wahab

Abstract—Although the existing machine learning-based intrusion detection systems in the Internet of Things (IoT) usually perform well in static environments, they struggle to preserve their performance over time, in dynamic environments. Yet, the IoT is a highly dynamic and heterogeneous environment, leading to what is known as data drift and concept drift. Data drift is a phenomenon which embodies the change that happens in the relationships among the independent features, which is mainly due to changes in the data quality over time. Concept drift is a phenomenon which depicts the change in the relationships between input and output data in the machine learning model over time. To address data drifts, we first propose a series of data preparation steps that help improve the quality of the data and avoid inconsistencies. To counter concept drifts, we capitalize on an online deep neural network model that relies on an ensemble of varying depth neural networks that cooperate and compete together to enable the model to steadily learn and adapt as new data come, thus allowing for stable and long-lasting learning. Experiments conducted on a real-world IoT-based intrusion detection dataset, designed to address concept and data drifts, suggest that our solution stabilizes the performance of the intrusion detection on both the training and testing data compared to the static deep neural network model, which is widely used for intrusion detection.

Index Terms—Intrusion Detection; Internet of Things (IoT); Online Deep Learning; Concept Drift; Data Drift.

1 INTRODUCTION

The Internet of Things (IoT) is progressively becoming part of our modern life, owing to the wide array of benefits it brings to many critical sectors such as healthcare, industrial engineering and transportation systems. The main idea of the IoT is to equip devices (e.g., wearable devices, home gadgets, etc.) with sensing and computing capabilities, enabling them to monitor the environment, collect data and process these data to come up with effective decision-making frameworks. The IoT devices are overwhelmingly characterized by limited processing, memory, storage and networking capabilities [1]. To face this problem, these IoT devices often need to interact with the fog and cloud computing layers to accomplish their tasks efficiently. Cloud computing enables the IoT devices to accommodate their storage and analytical needs, taking advantage of the unlimited and high-performance computing and storage capabilities it provides. Nonetheless, given that the cloud servers are predominatingly located far from the IoT devices, communications with the cloud entail significant response times. Fog computing extends the benefits of cloud computing through bringing the data processing capabilities closer to the IoT devices [2]. The prevalent scenario involving the IoT, fog and cloud layers is fog nodes receiving data from the IoT devices, launching IoT-enabled data analytics applications to extract insights from the data, sending back the insights to the IoT devices in the range of milliseconds, and periodically sending data summaries to the cloud for long-term historical storage.

The IoT ecosystem is likely to be confronted with non-conventional security challenges. Besides the security vulnerabilities that the IoT faces due to the heterogeneity and resource limitations of the IoT devices, the interactions among the IoT, fog and cloud layers make room for additional vulnerabilities. Specifically, each of these layers has its own vulnerabilities that entice a large number of attackers. In addition, the IoT devices and fog nodes are heterogeneous by nature, giving attackers the freedom to alter their attacks strategies according to the special characteristics of every underlying IoT device and fog node type. Besides, the communications among the IoT, fog and cloud layers offer another possibility for attackers to launch a broad set of attacks. Also, the attacks that take place in any of these layers can easily propagate to the other layers through the communication channels unless effective security measures are taken. Therefore, there is a pressing need to come up with effective intrusion detection and prevention solutions that help protect the IoT ecosystem from these increasing attacks threats.

1.1 Problem Statement

Plenty of intrusion detection solutions have been lately proposed for protecting the IoT against advanced attacks [3], [4], [5], [6], [7], [8], [9], [10]. Many of these solutions rely on machine/deep learning to analyze the activities of the IoT devices and the communication traces of the IoT devices. The objective is to identify and discard the suspicious activities/patterns. Although these solutions usually
yield high accuracy and low false positive and negative levels in static scenarios, the performance stability of the current solutions becomes questionable in dynamic time-evolving settings. More specifically, the predictive power of the machine learning model that is used to recognize the intrusion starts to deteriorate over time, a phenomenon that is known as model drift. In case this drift is not detected and remedied on time, it can entail detrimental impacts on the performance of the machine learning-based intrusion detection solution. Technically speaking, model drift can be divided into two types, i.e., concept drift and data drift.

Concept drift is the result of statistical properties of the target class labels changing over time. In fact, a predictive machine learning model is concretely a problem of designing a mapping function $f$ that takes, as an input, data samples $X$ to predict a class label $y$, i.e., $y = f(X)$. This mapping is often designed in a static fashion, which implies that the mapping learned from historical data will remain valid on new future data and that the relationships among the input and output data will remain unchanged [11]. This might hold for some scenarios but is quite unrealistic in other scenarios wherein the relationships among the input and output data change as time evolves, leading to changes in the underlying mapping function. This implies that the predictions taken by a model trained on older data would no longer hold if the same machine learning model is trained on more recent data. Intrusion detection in the IoT is one of those scenarios wherein concept drift is likely to take place due to its heterogeneous nature, causing the performance of the underlying machine learning model to significantly drop.

To better illustrate the concept drift phenomenon in an IoT-based intrusion detection scenario, let’s take a real-world example. Suppose that a machine learning model was trained to identify DoS attackers in a healthcare-driven IoT ecosystem wherein wearable devices are deployed to monitor patients’ conditions to detect respiratory diseases. When the machine learning model was originally trained, there was a specific idea of what a DoS attacker is and what features were relevant to identify them. For example, an IoT device that sends twenty messages within one minute to a fog node or to another IoT device was deemed to be a DoS attacker. Then, with the outbreak of COVID-19, some wearable devices might need to send more than twenty messages per minute in the highly affected areas to report some abnormal respiration rates. With this new usage pattern, sending twenty messages becomes normal that not only DoS attackers would do it, leading the concept of DoS attackers to drift. Thus, unless the machine learning model has been updated to include these new changes, the model would predict benign machine as DoS attackers, resulting in high false positive rates.

On the other hand, data drift takes place when some change happens in the independent variables, rather than the target variables as is the case with concept drift. Mapping the data drift phenomenon onto our previous example, it would not be the definition of a DoS attacker that changes, but rather the values of the features that we rely on to define these attackers. Data drifts can occur either due to some intentional malicious behavior or due to natural unintentional circumstances. An example of intentional malicious behavior could be attackers deciding, after observing that the intrusion detection system is quite effective in identifying their attacks, to change their attack behavior to confuse the system (e.g., switching their attack from the network layer to the application layer). In this case, the detection system would result in high false negatives, since it is not aware of these new conditions. On the other hand, the natural unintentional circumstances that could lead to data drifts include:

- Intrusion detection sensors being replaced, which leads to modifying some units of measurement (e.g., from centimeters to inches). This phenomenon is called upstream data changes.
- Data quality problems, e.g., some broken or defective sensors that report illogical values (e.g., always reading 0).
- Covariate shift, i.e., the change in the relationships among the features or change in the distribution of the data (e.g., an intrusion detection system that is trained predominantly on DoS attacks, yet the real dataset has a large proportion of data probing attacks as well).

### 1.2 Contributions

To counter data and concept drifts in machine learning-based IoT-driven intrusion detection systems, we propose a two-phase solution. In the first phase, we propose a set of data preparation steps in order to mitigate the impacts of data drifts on the intrusion detection performance. Specifically, we employ a series of methods to detect and replace missing and inconsistent values in the data. We then use the Interquartile Rule method [12] to detect outliers, followed by a Principal Component Analysis (PCA) method [13] to reduce the curse of dimensionality. The objective of these data preparation steps is to mitigate the impacts of data drifts caused by upstream data changes and broken/defective IoT devices that report missing, illogical and inconsistent values. Thereafter, to mitigate the impacts of concept drifts, we employ an online Deep Neural Network (DNN) model [14] that automatically adapts the capacity of the neural network to improve the online predictive capability of the intrusion detection model. The online DNN begins with a shallow network that is characterized by a fast convergence and then progressively and automatically converts into a deeper model when new data are received to learn more complex relationships. The main idea is to modify the architecture of the traditional DNN through binding each hidden layer representation to an output classifier. Thereafter, the standard Backpropagation method is replaced by a Hedge Backpropagation [15] method that assesses the performance of each output classifier at every online round. The main idea of the Hedge algorithm is to keep the weights over different classifiers at each time period and then update these weights based on the observed loss. The Hedge backpropagation method capitalizes on the classifiers of varying depths with the Hedge algorithm, allowing us to train the DNN with an adaptive capacity as new data (with concept drift) come, while also fostering the share of knowledge among the shallow and deep networks. The main contributions of this paper are:
• Proposing a two-phase long-lasting IoT-driven intrusion detection solution that provides high and sustainable performance in the presence of data and concept drift scenarios. To the best of our knowledge, our solution is the first to address the problem of sustaining the detection performance in IoT intrusion detection systems in the presence of concept and data drifts.

• Employing an online deep neural network that relies on an collection of competing and cooperating deep neural network classifiers of varying lengths to foster continuous learning and adaptation of the model as new data (with concept drift) come.

• Adopting a series of data preparation steps (i.e., data cleaning, outlier detection and dimensionality reduction) to mitigate the impacts of data drifts on the intrusion detection performance.

We compare the performance of the proposed solution with a traditional deep neural network using a series of experiments on the DS2OS traffic traces dataset [16], which is designed to tackle model drifts in IoT-driven intrusion detection systems.

1.3 Organization

In Section 2, we discuss relevant related approaches that propose detection systems in the IoT and highlight the originality of our solution. In Section 3, we discuss the data preparation steps that are proposed to mitigate the impacts of data drifts. In Section 4, we present the details of the online deep neural network model that is proposed to address the concept drift problem. In Section 5, we explain the environment used to conduct our experiments, and present and discuss the experimental results. Finally, we summarize the main findings of the paper in Section 6.

2 Related Work

We discuss the following in the following the main machine learning-based intrusion detection systems in the IoT [17], [18].

In [3], Illy et al. propose an ensemble learning approach that is implemented in a fog-to-things environment, where the anomaly detection is done first at the level of fog nodes and then the attack classification is done at the level of the cloud. In [4], the authors propose an algorithmic hybridization of intrusion detection with a Deep Belief Network (DBN). The DBN examines and detects the active malicious behavior inside the IoT networks. In [5], the authors address the problem of accurately extracting feature information from big unlabeled data to identify intrusions. They propose a Softmax classification approach that is based on an improved deep belief network, with a pre-training technique that seeks to decrease the feature space of the original intrusion data.

In [6], the authors discuss the vulnerability of IoT networks to adversarial attacks. Adversarial attacks take place when an adversarial instance is fed as an input to a machine learning model. An adversarial instance represents a feature that has been intentionally perturbed with the intention of confusing the machine learning model to produce erroneous predictions. A variant of the Feedforward Neural Network (FNN) known as the Self-normalizing Neural Network (SNN) is proposed to classify the attacks. The main contribution of this work is exploring the process of normalizing the input features in a deep learning-based IDS adversely to improve the ability of the deep learning model to resist adversarial attacks.

In [7], the authors adopt deep learning to detect distributed attacks in Social Internet of Things. The performance of the deep model is compared with a traditional machine learning approach. Similarly, distributed attack detection is compared with centralized detection system. The authors of [9] put forward a hierarchical intrusion security detection model Stacked De-noising Auto-encoder Support Vector Machine (SDAE-SVM) based on a three-layer neural network. The objective is to investigate the applicability of deep learning de-noising auto-encoder for IoT security. A layer-by-layer pre-training and fine-tuning approach is implemented for dimensionality reduction. Similarly, an intelligent intrusion detection system for IoT environments is proposed in [19]. The system capitalizes on deep learning to identify the malicious traffic and is presented in the form of a Security-as-a-service solution that boosts the interoperability between several communication protocols that could be used.

The authors of [8] design a cloud-based distributed deep learning solution for detecting and mitigating Botnet and phishing attacks in the IoT. The solution includes two components that operate in a cooperative manner, i.e., (1) a distributed Convolutional Neural Network (CNN) model that is integrated as a micro-security add-on into the IoT devices to recognize phishing and application-layer Distributed Denial of Service attacks; and (2) a temporal Long-Short Term Memory (LSTM) model that recognizes Botnet attacks, and ingest embeddings from CNN to identify distributed phishing attacks across multiple IoT devices. In [20], the authors advance a deep learning-based solution for detecting Internet Of Battlefield Things (IoBT) malware through analyzing the device’s Operational Code sequence. The IoBT covers the complete realization of pervasive sensing, pervasive computing, and pervasive communication in military applications. The Operational Code sequence is converted into a vector space on which a deep Eigenspace learning approach is implemented to differentiate malicious applications from benign ones. The authors of [21] put forward a Particle Deep Framework for detecting attack behavior in IoT networks. The proposed framework included three tasks, i.e., (1) eliciting network data traces and validating their integrity; (2) capitalizing on the Particle Swarm Optimization (PSO) method to automatically tune the parameters of the deep learning network; and (3) Combining deep neural network with PSO to identify abnormal behavior in IoT-based smart homes.

Despite the importance of these approaches, they implicitly rely on the unrealistic assumption that the intrusion detection environment is static. Yet, in real-life, many factors such as the statistical properties of the target class labels, distribution of the data, relationships among features and data quality are subject to change over time. This makes these approaches quite vulnerable to concept and data drifts and thus prone to drastic performance deterioration. To address this problem, we first propose a series of data...
preparation steps to improve the quality of the data and to remove redundant and unnecessary dimensions, thus mitigating the impacts of data drifts. We then employ an online deep neural network that adaptively updates its capacity to improve the online predictive power of the intrusion detection model. We argue that online deep learning is essential in intrusion IoT-driven detection scenarios since the data comes in sequential order, making it unrealistic to train the whole dataset at once.

3 Data Preparation: Mitigating the Impacts of Data Drifts

In this section, we explain the data preparation steps that are proposed to mitigate the data drift in the proposed intrusion detection system. These steps are classified into two major categories, i.e., data cleaning and dimensionality reduction, both of which are discussed hereafter. Before we proceed we the data preparation steps, we present in Table 1 an explanation of the features that are contained in the used dataset to help the readers better understand the preparation steps. We also present in Table 2 a breakdown of the attacks represented in the dataset along with their explanation.

3.1 Data cleaning

Data cleaning is a process that aims to handle missing and noisy values in real-world datasets. We first start with handling missing values. Two types of missing values can be distinguished in the used dataset, i.e., standard missing values and inconsistent values. The reason of having missing/inconsistent values might be due to some broken or defective intrusion detection sensors that report null or illogical values. The standard missing values occur when no data value is registered for a certain feature in a certain observation or when a value of “null” or “NA” is found. This type of missing values is easily detected by the Pandas\textsuperscript{1} open-source data analysis and manipulation tool. In total, we detected 2,156 missing values in the feature “value” and 148 missing values in the feature “Accessed Node Type” over a total of 357,953 samples. On the other hand, inconsistent values are encountered when the value of a certain feature contradicts with the feature’s type (e.g., a value of “false” in the feature “age” that is supposed to only have numeric values). To detect inconsistent values, we iterate, through the values of the feature “value” which is of type numeric and try to convert these values to numeric. If the conversion cannot be done, this means that the value is not numeric and thus should be considered inconsistent. Moving to the “Accessed Node Type” feature which is categorical, we try to convert the values of this feature into the numeric type. If the conversion is possible, the value is identified as inconsistent. The replace the missing and inconsistent values, we employ the k nearest neighbors algorithm [22]. The algorithm first finds the k closest neighbors to the observation with missing/inconsistent data and then takes the weighted mean over these neighbors. The weight represents the distance to neighbor; meaning that the closer the neighbor is to the observation with missing/inconsistent values, the more weight it will have when computing the mean. The missing/inconsistent values are finally replaced with the computed weighted mean.

The next step is to detect outliers. An outlier is a data point that lies an abnormal distance from the other data points in the same feature space. The reason of having an outlier might be a variability in the measurement across sensing devices (e.g., a device being replaced) or it may due to some experimental error. Data outliers can mislead the training process, resulting in considerably less accurate models and longer training times. To detect the outliers in the used dataset, we opt for the Interquartile Rule [12]. The main advantage of this method is that it allows us to obtain, at a glance, several information about the dispersion of the dataset such as the minimum, maximum, median and quartiles, which makes it quite useful for assessing the symmetry (or skewness) problem of the data distribution. A quartile is a type of quantile that divides a series of data points into a set of equal parts, known as quarters. The first quartile (Q1) is the 25th percentile of the ordered data series, the second quartile (Q2) is the median, and the third quartile (Q3) is the 75th percentile. The Interquartile Rule method is composed of the six following steps [23]:

1. https://pandas.pydata.org/

- Sort the data series µ in ascending order
- Compute the median Q2 of µ
- Compute the 25th percentile Q1 as the median of µ’s lower half
- Compute the 75th percentile Q3 as the median of µ’s upper half
- Compute the interquartile range: IQR = Q3 − Q1
- Mark as outlier any data point that is > Q3 + IQR * 1.5 or < Q1 − IQR * 1.5

In the used dataset, the Interquartile Rule is only applicable on the value feature whose values are continuous. Based on our results, any value smaller than −27.23 or greater than 48.05 is an outlier. In our case, no outlier has been identified.

3.2 Dimensionality Reduction

Dimensionality reduction is the process of decreasing the number of dimensions (aka features) in the dataset. Having a big number of dimensions in the feature space can imply that the volume of that space is very large, and in turn, that the data rows that we have in that space often represent a small and non-representative sample. This can negatively influence the performance of machine learning, a problem that is often referred to as the curse of dimensionality. The curse of dimensionality happens when the density of the data samples exponentially dwindles with the increase of the dimensionality. Keeping on adding features without increasing the number of training samples in parallel leads the dimensionality of the feature space to grow and become sparser and sparser. This increased sparsity makes it easier for the machine learning model to discover a “perfect” solution, a problem that is known as overfitting. Therefore, it is often a good practice to decrease the number of input features, thus decreasing the number of dimensions of the feature space. The most common method for dimensionality reduction is that of Principal Component Analysis (PCA) [13], which we will be using in our solution. PCA enables
### Table 1: List of features in the dataset

| Dimension                  | Data Type    | Example                                             |
|----------------------------|--------------|-----------------------------------------------------|
| Source ID                  | GUID         | /agent3/lighthouse                                      |
| Source Address             | GUID         | /agent4/lightcontroller                                 |
| Source Type                | Categorical  | movement sensor                                       |
| Source Location            | Categorical  | Dining Room                                           |
| Destination Service Address| Categorical  | /agent1/lighthouse                                     |
| Destination Service Type   | Categorical  | Sensor service                                        |
| Destination Location       | Categorical  | Bathroom                                              |
| Accessed Node Address      | Categorical  | /agent2/movement/movement                              |
| Accessed Node Type         | Categorical  | /lightcontroller                                      |
| Operation                  | Categorical  | write                                                |
| Value                      | Continuous   | Amounts to Maximum                                    |
| Timestamp                  | Discrete     | 15200316000002                                        |
| Normality                  | Nominal      | Described in Table 2                                   |

### Table 2: Breakdown of the attacks in the dataset

| Attacks          | Frequency in the Dataset | Percentage of Overall Attack Types | Description                                           |
|------------------|--------------------------|-----------------------------------|-------------------------------------------------------|
| DDoS             | 5,780                    | 87.64%                            | A device performing access at a very high frequency   |
| Data Probing     | 442                      | 4.31%                             | A device attempting to take control over another one  |
| Malicious Control| 889                      | 8.87%                             | A device attempting to take control over another one  |
| Malicious Operation| 805                    | 8.03%                             | A device attempting to take control over another one  |
| Network Scan     | 1,559                    | 15.55%                            | A device accessing the network ranges of other devices|
| Spying           | 532                      | 5.30%                             | A device reading the values of other devices          |
| Wrong Setup      | 129                      | 1.2%                              | A device accessing its services in a wrong location   |
| Total            | 100,27                   | 100%                              | Amounts to 3% of the dataset                          |

3.2.2 Covariance Matrix

This step helps us understand how the variables vary from the mean, with respect to each other, i.e., if there is any relationship between these variables. In some cases, variables are highly correlated in the sense that they comprise redundant information. Thus, computing the covariance matrix enables us to characterize these correlations. A covariance matrix is an \( n \times n \) symmetric matrix (with \( n \) being the number of dimensions) whose entries are the covariances associated with all possible pairs of the variables. Having computed the values of the covariance matrix, we mainly need to look at the signs of the covariances. If the covariance is positive, this means that the two underlying variables are correlated, i.e., they increase/decrease together. If the covariance is negative, this means that the two underlying variables are inversely correlated, i.e., one variable increases when the other decreases.

3.2.3 Eigenvectors and Eigenvalues

The eigenvectors of the covariance matrix represent the directions of the axes where most of the variance (and thus most of the information) lies, i.e., the Principal Components. Eigenvalues simply are the coefficients associated with the eigenvectors, quantifying the amount of variance contained in each single Principal Component. Thus, eigenvectors and eigenvalues always come in pairs and have a number that is equal to the number of dimensions in the dataset. Having computed the eigenvectors and eigenvalues, the next step would be to sort the eigenvectors, largest to smallest, in order of their eigenvalues. The sorting step gives us the principal components based on their significance. To obtain the percentage of variance contained in each principal component, we simply divide the eigenvalue of each component by the sum of the eigenvalues.

3.2.4 Determining the Principal Components to Keep

Principal components are designed in such a way that the first principal component would include the largest possible
variance compared to the rest of the components. The second principal component would include the second largest variance and so on. The last step in the PCA process is to choose the number of principal components to keep for training the machine learning model. It’s worth mentioning that the choosing the number of principal components to keep is mainly subjective. For example, one might decide either to keep all the components or to dismiss the ones that have less significance. To help us make our decision, we plot in Figure 1 the cumulative explained variance ratio as a function of the number of principal components. This plot quantifies how much of the total 12-dimensional (the dataset’s dimensions except for the target variable) variance is contained in the first N components, with N being represented on the X-axis. By looking at the figure, we notice that the first two principal components account for approximately 60% of the variance and that need at least 9 principal components to obtain 100% of variance. Therefore, we choose to keep 9 principal components from the 12 components that we initially had.

4 ONLINE DEEP NEURAL NETWORK: MITIGATING THE IMPACTS OF CONCEPT DRIFT

In this section, we provide the details of the online DNN, which is mainly inspired from the work presented in [14]. The online DNN is proposed to counter the impacts of concept drift on the intrusion detection system. This is achieved through automatically adapting the capacity of the neural network to improve its performance in making online predictions in the presence of concept drifts.

Consider an online attack classification task whose aim is to learn a function $F$ on a collection of training samples $D = \{ (x_1, y_1), \ldots, (x_K, y_K) \}$ that come in a sequential manner, with $x_k$ being a $d$-dimensional sample, $y_k \in \{0, 1\}^C$ being the class label attributed to $x_k$ and $C$ being the number of class labels. Let the classification task be denoted by $y_k$ and the cumulative prediction error by $E_K = \frac{1}{K} \sum_{k=1}^{K} \tau(y_k \neq y_k)$. $E_K$ is a characteristic function whose value is 0 if the condition is false and 1 otherwise. In order to minimize the prediction error over the sequence of $K$ samples, we need to formulate a loss function. Let $\Gamma(F(x), y)$ be a cross-entropy loss function. At each online classification iteration, whenever a sample $x_k$ comes, the online DNN makes a prediction. Then, the environment uncovers the ground truth of the class label, allowing the online DNN to update the model using an online gradient descent methodology.

The prediction function $F$ is a series of stacked linear transformations, each of which is followed by a nonlinear activation. Given a data sample $x_k$ as an input, the (recursive) prediction function of a deep neural network with $H$ hidden layers ($l^{(1)}, \ldots, l^{(H)}$) using the Hedge Backpropagation [15] is defined by Equation (1):

$$F(x) = \sum_{h=0}^{H} \delta^{(h)} f^{(h)}$$

with

$$f^{(h)} = \text{softmax}(l^{(h)} \phi^{(h)}), \forall h = 1, \ldots, H$$

and

$$l^{(h)} = \alpha(W^{(h)} l^{(h-1)}), \forall h = 1, \ldots, H; l^{(0)} = x$$

where $\alpha$ is an activation function (e.g., sigmoid, ReLU, etc.). Equations (1) and (3) depict a feedforward process of the deep neural network. The hidden layers $l^{(h)}$ embody the feature representations that are learnt during the training process. $\phi^{(h)}$ and $\delta^{(h)}$ are newly introduced parameters and thus are not part of the traditional deep neural network. They hence need to be learnt (later in this section). The main idea of this prediction function is that, different from the original deep neural network wherein the final prediction is based on the feature representation $l^{(H)}$, the prediction is the result of a weighted combination of $H + 1$ classifiers learnt using the feature representations from $l^{(0)}$ to $l^{(H)}$.

The weight assigned to each classifier is represented by $\delta^{(h)}$ which is computed using the Hedge Algorithm [24]. At the first iteration, all the weights $\delta$ are uniformly distributed, where $\delta^{(h)} = \frac{1}{H+1}, h = 0, \ldots, H$. Then, at each iteration, the classifier $f^{(h)}$ carries out a prediction $\hat{y}_k^{(h)}$. Once the ground truth is uncovered, each classifier’s weight is updated according to the loss experienced by the classifier as per Equation (4):

$$\delta^{(h)}_{k+1} \leftarrow \delta^{(h)}_k \frac{\Gamma(F(x_k), y_k)}{\Gamma(F(x_k), y_k)}$$

where $\ell \in (0, 1)$ is a discount factor and $\Gamma(f^h(x), y) \in (0, 1)$ is an adaptive loss function. At the end of each iteration, the weights $\delta$ are normalized so that $\frac{1}{H} \sum_{h=0}^{H} \delta^{(h)} = 1$. The adaptive loss function is given in Equation (5):

$$\Gamma(F(x), y) = \sum_{h=0}^{H} \delta^{(h)} \Gamma(f^h(x), y)$$

In order to learn the parameters $\phi^{(h)}$ across the different classifiers, the online gradient descent methodology [25] is used as per Equation (6):

$$\phi^{(h)}_{k+1} \leftarrow \phi^{(h)}_k - \eta \nabla_{\phi^{(h)}} \Gamma(F(x_k), y_k)$$

$$= \phi^{(h)}_k - \eta \nabla_{\phi^{(h)}} \phi^{(h)}(f^{(h)}, y_k)$$

Where $\eta$ is the learning rate. The feature representation parameters $W^{(h)}$ can be updated through backpropagating the error derivatives from each single classifier $f^{(h)}$. 
Technically speaking, by applying the online gradient descent methodology and the adaptive loss function given in Equation (3), we can update the feature representation parameters $W^{(h)}_{k+1}$ as per Equation (7).

$$W^{(h)}_{k+1} = W^{(h)}_k - \eta \sum_{i=1}^{H} \delta^{(i)} \nabla W^{(h)}_i \Gamma(f^{(i)}, y_k)$$  (7)

where $\nabla W^{(h)}_i \Gamma(f^{(i)}, y_k)$ is calculated through backpropagating the classifier’s $f^{(i)}$ error derivatives.

The logic of the online DNN is further summarized in Algorithm 1.

Algorithm 1: Online Deep Neural Network

1. **Input**: Learning rate $\eta$
2. **Input**: Discount factor $\ell$
3. **Input**: Number of hidden layers $H$
4. **procedure** ONLINEDEEPNEURALNETWORK
5. **Initialize** $F(x)$ with $H$ hidden layers, and $H+1$ classifiers $f^{(h)}, \forall h = 1, \ldots, H$
6. **Initialize** $\delta^{(h)} = \frac{1}{H+1}, \forall h = 1, \ldots, H$
7. for sequence $k = 1, \ldots, T$ do
   8. Get $x_k$
   9. Use Equation (1) to predict $\hat{y}_k$
   10. Uncover the ground truth of the class label $y_k$
   11. Update the loss function, i.e.,
       $$\Gamma^{(h)}_k = \Gamma(f^{(h)}_k, y_k), \forall h = 1, \ldots, H$$
   12. Use Equation (6) to update $\phi^{(h)}_{k+1}, \forall h = 1, \ldots, H$
   13. Use Equation (7) to update $W^{(h)}_{k+1}, \forall h = 1, \ldots, H$
   14. Use Equation (4) to update $\delta^{(h)}_{k+1}, \forall h = 1, \ldots, H$
   15. Normalize $\delta^{(h)}_{k+1} = \frac{\delta^{(h)}_{k+1}}{\zeta_k}$ with $\zeta_k = \sum_{h=0}^{H} \phi^{(h)}_{k+1}$
8. **end procedure**

In all, the main advantages of the online DNN model are: (1) adjusting the depth of the neural network based on the performance of the classifier at that depth, achieving an online learning with multiple experts; (2) modeling the online learning process as an ensemble of multi-depth networks that cooperate (through the sharing of the feature representations) and compete (using the Hedge algorithm) to ameliorate the performance of final predictions, thus enabling long-lasting learning through allowing the model to constantly learn and adapt as more data come; (3) different from the traditional online machine learning models that undergo slow convergence in deeper network when the concepts drift occurs, the proposed online DNN quickly adapts to the concept drift thanks to the Hedge backpropagation method.

5 EXPERIMENTAL EVALUATION

In this section, we first explain the environment used to conduct our experiments and then provide experimental results and analysis.

5.1 Experimental Setup and Datasets

We implement a 16-layer online deep neural network using the online backpropagation algorithm. Each hidden layer of the network comprises 100 units. The rectifier (ReLU) is used as an activation function and the learning rate is set to 0.01. A classifier was assigned to each of the 15 hidden layers (except for the input layer) with depth from 2, ..., 16. The discount factor $\ell$ is set to 0.99. The implementation was carried out in Keras Python on Tensorflow.

The DS2OS traffic traces dataset is used. The main motivation for choosing this dataset stems from the fact is that it is designed to address model drifts in IoT-based intrusion detection systems. The dataset records communication traces between a multitude of IoT devices, all of which are part of a common middleware, i.e., the Distributed Smart Space Orchestration System (DS2OS). Eight types of IoT devices are considered, i.e., light controllers, thermometers, motion sensors, washing machines, batteries, thermostats, smart doors and smartphones. Each device is characterized by an address (e.g., agent2 / lightcontrol2), a type (e.g., light controller) and a location (e.g., kitchen). The dataset consists of 357,000 records spanning over 13 dimensions. In Table 1 we explain the meaning of each of these dimensions. Moreover, we give in Table 2 a breakdown of the attacks represented in the dataset along with their explanation. We compare the performance of our solution with the traditional deep neural network, which has been extensively used for intrusion detection in the IoT.

To split the dataset into training and test sets, the k-fold cross-validation approach is employed, while setting $k$ to 10. This approach divides the dataset into $k$ batches, each of which is selected every time to be the test set and the other $k-1$ batches are integrated together to form the training set. Then, the accuracy of the classifier is computed by averaging the error over all the $k$ iterations. The main advantage of cross-validation stems from its effectiveness in minimizing the bias of the classifications toward the dataset’s structure, given that every data sample needs to be part of the test set exactly once and of the training set $k-1$ times.

5.2 Results and Discussion

To start our comparisons, we first provide in Tables 3 and 4 the confusion matrices of our solution and the regular DNN respectively over the different attack types. The confusion matrix is a matrix that is used to quantify the performance of a classification model on a test data set whose true values are known. The columns of the confusion matrix represent the predicted classes and the rows represent the actual classes. The confusion matrix is used to compute the true positive (TP), false positive (FP), false negative (FN) and true negative (TN) of a machine learning model. True positive is a case wherein the actual class is positive and the model correctly classifies it as positive ($a_1 \neq a_i$). False positive is a case wherein the actual class is negative and the model incorrectly classifies it as positive ($a_1 = a_i$). False negative is a case wherein the actual class is positive and the model incorrectly classifies it as negative ($a_1 \neq a_i$). True negative is a case wherein the actual class is negative and the model correctly classifies it as negative ($a_1 = a_i$).

By looking at Table 3, we notice that our solution correctly classifies most of the attacks, with only 6 misclassifications in the case of DoS attacks (classified as normal), 2 misclassifications in the case of malicious control attack (one
Data Probing & Wrong Setup & Malicious Control & Malicious Operation & Scan & Spying & Normal & Normal

| DDoS | Data Probing | Malicious Control | Malicious Operation | Scan | Spying | Wrong Setup | Normal |
|------|--------------|-------------------|---------------------|------|--------|-------------|--------|
| DoS  | 320          | 0                 | 0                   | 0    | 0      | 0           | 26     |
| Data Probing | 0          | 64                | 0                   | 0    | 0      | 0           | 0      |
| Malicious Control | 0             | 0                 | 261                 | 0    | 0      | 0           | 0      |
| Malicious Operation | 0            | 0                 | 0                   | 0    | 0      | 0           | 0      |
| Scan | 0            | 0                 | 1                   | 0    | 0      | 0           | 0      |
| Spying | 0            | 0                 | 0                   | 0    | 0      | 0           | 0      |
| Wrong Setup | 0             | 0                 | 0                   | 0    | 0      | 0           | 28     |
| Normal | 6            | 0                 | 0                   | 0    | 0      | 0           | 0      |

We notice from Table 5 that our solution achieves comparable percentages of accuracy, precision, recall and F1 score to the regular DNN across all the attack classes with a slight improvement to our solution. Yet, by comparing Table 5 and Table 6 we notice that the performance of the regular DNN considerably drops for all the attack classes, especially the DoS and normal classes. For example, the accuracy of detecting DoS attacks drops from 96.1% on the training data to 86.2% on the testing data. Similarly, the accuracy of detecting normal cases drops from 94.5% on the training data to 77.1% on the testing data. The reasons for this significant drop are the concept and data drifts that took place in the testing data post training. On the other hand, our solution keeps a stable performance on the testing data, showing a large resilience to the concept and data drifts. For example, in the case of DoS attacks, the accuracy of our solution drops only from 98.5% to 98.4%. In the case of the normal class, the accuracy of our solution drops only from 97.7% to 96.8%. These results confirm the effectiveness of the data preparation steps as well as the online DNN in considerably reducing the impacts of data and concept drifts, respectively.

In Fig. 2, we study in more detail the accuracy of our solution and the regular DNN on both the training and testing data, while varying the number of data instances from 55,000 to 350,000. Several experiments are performed on different test sets and the average accuracy is taken. The first observation that can be drawn from the figure is that increasing the number of data instances has a positive impact on both approaches, on both the training and testing sets. The second observation is that accuracy gap between the training and testing data is almost negligible in our solution, compared to a large gap in the case of the regular DNN model. This again show the large resilience to the concept and data drifts that our solution enjoys compared to the regular deep neural networks.

In Figure 3, we study the false positive (Figure 3a) and false negative (3b) percentages entailed by our solution and the regular DNN, while varying the number of data instances from 55,000 to 350,000. Several experiments are performed on different test sets and the average false positive and false negative rates are taken. We notice from the figure that the false positive and false negative percent-
### Table 5: Training Data

| Attacks                | Our Solution |                       | Regular DNN |                       |
|------------------------|--------------|------------------------|-------------|------------------------|
|                        | Accuracy     | Precision              | Recall      | F1 Score               | Accuracy     | Precision              | Recall      | F1 Score               |
| DoS                    | 98.0%        | 97.9%                  | 97.8%       | 97.8%                  | 96.1%        | 96.0%                  | 96.0%       | 96.0%                  |
| Data Probing           | 100%         | 100%                   | 100%        | 100%                   | 98.5%        | 97.5%                  | 98.0%       | 98.0%                  |
| Malicious Control      | 98.9%        | 98.5%                  | 98.4%       | 98.4%                  | 95.7%        | 95.9%                  | 95.9%       | 95.9%                  |
| Malicious Operation    | 100%         | 100%                   | 100%        | 100%                   | 97.4%        | 97.9%                  | 97.9%       | 97.9%                  |
| Scan                   | 99.9%        | 99.7%                  | 99.5%       | 99.5%                  | 97.4%        | 97.9%                  | 97.9%       | 97.9%                  |
| Spying                 | 100%         | 100%                   | 100%        | 100%                   | 97.5%        | 97.4%                  | 97.1%       | 97.6%                  |
| Wrong Setup            | 100%         | 100%                   | 100%        | 100%                   | 98.1%        | 98.1%                  | 98.2%       | 98.2%                  |
| Normal                 | 97.7%        | 96.9%                  | 96.8%       | 96.8%                  | 94.5%        | 94.4%                  | 94.1%       | 94.1%                  |

### Table 6: Testing Data

| Attacks                | Online Deep Learning |                       | Regular DNN |                       |
|------------------------|----------------------|------------------------|-------------|------------------------|
|                        | Accuracy     | Precision              | Recall      | F1 Score               | Accuracy     | Precision              | Recall      | F1 Score               |
| DoS                    | 98.4%        | 97.4%                  | 97.3%       | 97.3%                  | 96.2%        | 96.4%                  | 96.4%       | 96.4%                  |
| Data Probing           | 100%         | 100%                   | 100%        | 100%                   | 90.5%        | 91%                    | 91%         | 91%                    |
| Malicious Control      | 98.9%        | 98.5%                  | 98.4%       | 98.4%                  | 89.4%        | 89.2%                  | 89.3%       | 89.3%                  |
| Malicious Operation    | 100%         | 100%                   | 100%        | 100%                   | 90.2%        | 90.5%                  | 90.5%       | 90.5%                  |
| Scan                   | 99.8%        | 99.7%                  | 99.5%       | 99.5%                  | 87.2%        | 88.1%                  | 87.2%       | 87.5%                  |
| Spying                 | 100%         | 100%                   | 100%        | 100%                   | 91.5%        | 91.7%                  | 91.4%       | 91.1%                  |
| Wrong Setup            | 100%         | 100%                   | 100%        | 100%                   | 91.4%        | 91.5%                  | 91.4%       | 91.8%                  |
| Normal                 | 96.8%        | 96.3%                  | 96%         | 96.6%                  | 77.1%        | 76.2%                  | 75.5%       | 76.8%                  |

Fig. 2: Accuracy: Our solution significantly reduces the accuracy gap between the training and test sets.

Fig. 3: Our solution significantly reduces the false positive and false negative percentages compared to the regular DNN.

Although the current intrusion detection systems in IoT environments achieve high accuracy and low false positive and negative rates, maintaining and stabilizing their performance over time remains questionable in the presence of concept and data drifts. In this work, we address this challenge and propose first a series of data preparation steps (i.e., data cleaning and dimensionality reduction) to mitigate the impacts of data drifts that affect the quality of the data and risk to change the relationships among the input features. Thereafter, we employ an online deep neural network model that represents the online learning process as an ensemble of multi-depth networks that cooperate (through...
sharing the feature representations) and compete (using the Hedge algorithm) to enable the model to constantly learn and adapt as new data come, thus achieving long-lasting learning. We conduct a series of experiments on a real-world dataset that is designed to represent model drifts in IoT-based intrusion detection systems. The results suggest that our solution significantly stabilizes the performance of the intrusion detection process not only on the training data but also on the testing data in the presence of concept and data drift scenarios, compared to the static deep neural network model which is widely used in IoT-based intrusion detection. Moreover, the results suggest that our solution reduces the false positive by approximately 6% and false negative by approximately 4.5%, compared to the static deep neural network model.

ACKNOWLEDGMENTS

This work has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] S. Arisdaakessian, O. A. Wahab, A. Mourad, H. Otrok, and N. Kara, “Fogmatch: an intelligent multi-criteria iot-fog scheduling approach using game theory,” IEEE/ACM Transactions on Networking, vol. 28, no. 4, pp. 1779–1789, 2020.

[2] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in Proceedings of the first edition of the MCC workshop on Mobile cloud computing, 2012, pp. 13–16.

[3] P. Ily, G. Kaddoum, C. M. Moreira, K. Kaur, and S. Garg, “Securing fog-to-things environment using intrusion detection system based on ensemble learning,” in 2019 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2019, pp. 1–7.

[4] N. Balakrishnan, A. Rajendran, D. Pelusi, and V. Ponnusamy, “Deep belief network enhanced intrusion detection system to prevent security breach in the internet of things,” Internet of Things, p. 100112, 2019.

[5] P. Wang, X. Song, Z. Deng, H. Xie, and C. Wang, “An improved deep learning based intrusion detection method,” in 2019 IEEE 5th International Conference on Computer and Communications (ICCC). IEEE, 2019, pp. 2092–2096.

[6] O. Ibitoye, O. Shafiq, and A. Matrawy, “Analyzing adversarial attacks against deep learning for intrusion detection in iot networks,” in 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019, pp. 1–6.

[7] 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, 2018.

[8] G. D. L. T. Parra, P. Rad, K.-K. R. Choo, and N. Beebe, “Detecting internet of things attacks using distributed deep learning,” Journal of Network and Computer Applications, p. 102662, 2020.

[9] Z. Lv, L. Qiao, J. Li, and H. Song, “Deep learning enabled security issues in the internet of things,” IEEE Internet of Things Journal, 2020.

[10] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, “Resource-aware detection and defense system against multi-type attacks in the cloud: Repeated bayesian stackelberg game,” IEEE Transactions on Dependable and Secure Computing, 2019.

[11] J. Gama, I. Zliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” ACM computing surveys (CSUR), vol. 46, no. 4, pp. 1–37, 2014.

[12] M. Hubert and S. Van der Venken, “Outlier detection for skewed data,” Journal of Chemometrics: A Journal of the Chemometrics Society, vol. 22, no. 3-4, pp. 235–246, 2008.

[13] S. Wold, K. Esbensen, and P. Geladi, “Principal component analysis,” Chemometrics and intelligent laboratory systems, vol. 2, no. 1-3, pp. 37–52, 1987.

[14] D. Sahoo, Q. Pham, J. Lu, and S. C. Hoi, “Online deep learning: Learning deep neural networks on the fly,” arXiv preprint arXiv:1711.03705, 2017.

[15] A. Gammerman and V. Vovk, “Hedging predictions in machine learning,” The Computer Journal, vol. 50, no. 2, pp. 151–163, 2007.

[16] “DS2OS traffic traces,” https://www.kaggle.com/francoisxa/ds2ostraffictraces?select=main&simulationAccessTraces.csv, accessed: 2021-01-27.

[17] M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, I. Ali, and M. Guizani, “A survey of machine and deep learning methods for internet of things (iot) security,” IEEE Communications Surveys & Tutorials, 2020.

[18] S. Hajheidari, K. Wakil, M. Badri, and N. J. Navimipour, “Intrusion detection systems in the internet of things: A comprehensive investigation,” Computer Networks, vol. 160, pp. 163–191, 2019.

[19] G. Thamilarasu and S. Chawla, “Towards deep-learning-driven intrusion detection for the internet of things,” Sensors, vol. 19, no. 9, p. 1977, 2019.

[20] A. Azmooddeh, A. Dehghantanha, and K.-K. R. Choo, “Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning,” IEEE Transactions on Sustainable Computing, vol. 4, no. 1, pp. 88–95, 2018.

[21] N. Koroniots, N. Moustafa, and E. Sitnikova, “A new network forensic framework based on deep learning for internet of things networks: A particle deep framework,” Future Generation Computer Systems, 2020.

[22] J. M. Keller, M. R. Gray, and J. A. Givens, “A fuzzy k-nearest neighbor algorithm,” IEEE transactions on systems, man, and cybernetics, no. 4, pp. 580–585, 1985.

[23] O. A. Wahab, J. Bentahar, H. Otrok, and A. Mourad, “Optimal load distribution for the detection of vm-based ddos attacks in the cloud,” IEEE transactions on services computing, vol. 13, no. 1, pp. 114–129, 2017.

[24] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” Journal of computer and system sciences, vol. 55, no. 1, pp. 119–139, 1997.

[25] M. Zinkevich, “Online convex programming and generalized inﬁnitesimal gradient ascent,” in Proceedings of the 20th international conference on machine learning (icml-03), 2003, pp. 928–936.

[26] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,” Nature, vol. 405, no. 6789, pp. 947–951, 2000.

[27] P. Refaeilzadeh, L. Tang, and H. Liu, “Cross-validation,” in Encyclopedia of database systems. Springer, 2009, pp. 532–538.