Intrusion Detection in the IoT under Data and Concept Drifts: Online Deep Learning Approach

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

21-02-2022 / 23-02-2022

CITATION

Abdul Wahab, Omar (2022): Intrusion Detection in the IoT under Data and Concept Drifts: Online Deep Learning Approach. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.19210197.v1>

DOI

[10.36227/techrxiv.19210197.v1](https://doi.org/10.36227/techrxiv.19210197.v1)

Intrusion Detection in the IoT under Data and Concept Drifts: Online Deep Learning Approach

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 detect data and concept drifts, we first propose a drift detection technique that capitalizes on the Principal Component Analysis (PCA) method to study the change in the variance of the features across the intrusion detection data streams. We also discuss an online outlier detection technique that identifies the outliers that diverge both from historical and temporally close data points. To counter these drifts, we discuss an online deep neural network that dynamically adjusts the sizes of the hidden layers based on the *Hedge* weighting mechanism, thus enabling the model to steadily learn and adapt as new intrusion data come. Experiments conducted on an IoT-based intrusion detection dataset 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; Online Deep Learning; Concept Drift; Cybersecurity.

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], [2]. 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 predominately 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 [3]. 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 [4], [5], [6], [7], [8], [9]. 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

• O. Abdel Wahab is with the Department of Computer Science and Engineering, Université du Québec en Outaouais, Gatineau, Canada.

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 [10]. 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 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 [11]. 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 discuss a (1) drift detection technique that capitalizes on the Principal Component Analysis (PCA) method [12] to study the major directions of feature variance across the intrusion detection data streams; and (2) online outlier detection technique that determines outliers not only based on historical data, but also based on temporally close data points. In the second phase, the aim is to mitigate the impacts of concept and data drifts. To do so, we employ an online Deep Neural Network (DNN) model [13] that automatically adapts the capacity of the neural network to improve the online predictive capability of the intrusion detection model. Online deep learning is effective in in scenarios where concept and data drifts occur as it proposes to dynamically tune the predictions of future as new data come. One major challenge in adopting deep learning in an online environment is deciding about the depth of the underlying neural network. A quite shallow network would make limited the ability of the deep learning model to detect complex attack patterns. Conversely, an excessively deep network would make the convergence of the model very slow. Usually, this problem is solved by having validation data on which the deep learning model is tuned to determine the appropriate network size that allows the model to do best on new data. Yet, this option is not realistic when it comes to online learning where the data come in a sequential manner. To overcome this problem, we adopt the *Hedge algorithm* [14], to assign a weight and an output classifier to each hidden layer in the deep neural network. At first, equal initial weights are assigned to each hidden layer. After the classifier has made its predictions at the current time moment and the ground truth is known, the weight of the classifier at the next time moment is tuned based on the observed loss. The final prediction is then carried out using a weighted aggregation mechanism over the different classifiers. The originality of our solution stems from the fact that it provides a comprehensive and generic mechanism for detecting and countering data and concept drift in intrusion detection scenarios. The solution is quite

generic in the sense that it is independent from any detection algorithm, data format or neural network size. In summary, 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.**
- **Employing an online deep neural network that dynamically adjusts the sizes of the neural networks based on the Hedge weighting mechanism. The objective is to foster continuous learning and adaptation of the model as new data (with concept drift) come.**
- **Discussing a drift detection technique that capitalizes on the PCA method to study the change in the variances of the features over time;**
- **Discussing an online outlier detection technique that considers both historical and temporally close data points to identify those observations that diverge from all available data points and those that diverge from the recent trends in the data.**

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 [15], 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 drift detection and online outlier detection techniques. In Section 4, we present the details of the online deep neural network model that is proposed to fight against drifts. 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 in the following the main machine learning-based intrusion detection systems in the IoT [16], [17].

2.1 Machine Learning-based Intrusion Detection in IoT

In [4], 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 [5], 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 [6], 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 [7], 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.

The authors of [8] 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 [18]. 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 [19] 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.

2.2 Online and Incremental Learning for Concept Drift

In [20], the authors first present a comprehensive survey on concept drift in machine learning and then discuss a learning framework that is resilient to this phenomenon, composed of three phases, i.e., concept drift detection, concept drift understanding, and concept drift adaptation. In the detection phase, data chunks are first collected from data streams and then abstracted to deduce the important features that have the highest impact on the system in the case of concept drift. Finally, the dissimilarity between the distribution of historical data and that of new data is studied and the statistical significance of the change is assessed. The understanding phase aims to provide contextual information about the drift, i.e., the time at which the drift starts as well as its duration (i.e., the *When*), impact (i.e., the *How*), and the regions that are affected by the drift (i.e., the *Where*). The adaptation

phase is concerned with designing mechanisms to tune the existing machine learning models based on the detected drift. In this work, we apply the drift detection and adaptation phases.

In [21], a comprehensive survey is conducted on IoT device identification and detection from passively collected traffic traces and wireless signals using machine learning. The literature is classified into four categories, i.e., device-specific pattern recognition, unsupervised device identification, deep learning-enabled device identification and abnormal device detection. The authors particularly stress the importance of *continual learning* (one type of incremental learning) in enabling the machine learning model to adapt to new devices' characteristics. That is, *continual learning* allows the neural networks to be incrementally and gradually trained with the arrival of new data, without letting the learning model to forget what has been learnt in the previous stages.

In [22], the authors propose to employ online machine learning to address the concept drift in IoT-based intrusion detection environments based on the interleaved-test-then-train strategy. That is, the machine learning model is tested and then trained once on each single new data chunk. In [23], the authors propose *ElStream*, an ensemble learning approach for concept drift detection in dynamic social Big Data streams. The main idea is to capitalize on the expertise of several classifiers to improve the predictive power of the machine learning model, making it more resilient to concept drifts. In [24], an online and incremental machine learning approach called Smart Traffic Management Platform (STMP) is discussed to deal with real-time concept drifts in unstructured big data streams with unlabeled target variables. The proposed approach combines heterogeneous data from many sources such as IoT devices, social media and smart sensors to improve the resilience to concept drifts. The online part of the approach is embodied by an online adaptive clustering algorithm that tunes the model as new data arrive. The incremental part of the approach is implemented using Growing Self Organizing Maps (GSOM) which generate a dynamic feature map according to the growing self-organizing process.

In all, two main machine learning approaches have been used in the literature to deal with concept drifts, i.e., online learning and incremental learning. Online learning proposes to tune the underlying machine learning model through analyzing few data samples at a time, on the fly. Incremental learning suggests to learn from batches of data at different time intervals with the aim of making the model more stable when given new learning tasks [24]. We argue that continual learning and incremental learning in general are not suitable for concept drift scenarios. This is because in continual learning once the model learns some knowledge, it has to remember and store the model, thus presuming that the learned knowledge would constantly remain valid. Yet, this assumption

Fig. 1: Overall Architecture of Our Solution

does not always hold in an intrusion detection scenario wherein the attack strategies are continuously changing, leading the properties of some classes to change. On the other hand, we believe that online machine learning is more effective in dealing with concept drifts as it imposes no limitations on the model in terms of not forgetting previous knowledge. Therefore, we propose in this work model drift detection and adaptation mechanisms based on online machine learning. The originality of our solution stems from the fact that it provides a comprehensive and generic mechanism for detecting and fighting against data and concept drifts in intrusion detection scenarios. The solution is quite generic in the sense that it is independent from any detection algorithm, data format or neural network size.

3 DRIFT DETECTION

In this section, we first discuss a PCA-based drift detection technique that allows us to recognize the drifts that occur in the data over time. Thereafter, we examine an online outlier detection mechanism that detects those data points that diverge from both the historical and recent trends in the data.

3.1 PCA-based Drift Detection

To detect the drifts in the data, we propose to employ the Principal Component Analysis (PCA) technique [12] as discussed in [25]. The main idea is to capitalize on PCA to measure and compare the variance of the features in the dataset at the current time moment t with that of the features in the dataset at the previous time moment $t - 1$. In case of the variance changes significantly between the two datasets, this means that some drift has happened in the distribution of the data. PCA is a dimensionality reduction technique which enables us to convert an array of n dimensions into a reduced array of $p < n$ dimensions. Dimensionality reduction is an essential step toward improving the quality of the dataset as, usually, some dimensions don't provide much information to the machine learning process. To reduce the dimensionality, the main idea of PCA is to derive the *principal components*, that tell us which combinations of dimensions have the

---

Algorithm 1: Drift Detection

```

1: Input:  $\beta_t$ : Sequential dataset at current time moment  $t$ 
2: Input:  $\beta_{t-1}$ : Sequential dataset at time moment  $t - 1$ 
3: Output: hasDrift: Boolean variable with a value of 1 if a drift is
   detected and 0 otherwise (initialized to 0)
4: procedure DRIFTDETECTION
5:   Standardize the feature values of  $\beta_t$  and  $\beta_{t-1}$ 
6:   Compute the covariance matrices  $\Sigma_{\beta_t}$  and  $\Sigma_{\beta_{t-1}}$  of the
   features
7:   Compute the eigenvectors  $E_{\beta_t}^{\Sigma_{\beta_t}}$  and  $E_{\beta_{t-1}}^{\Sigma_{\beta_{t-1}}}$  and eigenvalues
    $V_{\beta_t}^{\Sigma_{\beta_t}}$  and  $V_{\beta_{t-1}}^{\Sigma_{\beta_{t-1}}}$  of the covariance matrices of the datasets  $\beta_t$ 
   and  $\beta_{t-1}$ 
8:   Compute the degree of drift between current dataset and the
   dataset at the previous time moment as
    $Drift(\beta_t, \beta_{t-1}) = \max \arccos(V_{\beta_t}^{\Sigma_{\beta_t}} \cdot V_{\beta_{t-1}}^{\Sigma_{\beta_{t-1}}})$ 
9:   if  $Drift(\beta_t, \beta_{t-1}) \geq 60^\circ$  then
10:     hasDrift = 1
11:   end if
12: end procedure

```

---

biggest variance and therefore give most of the information. Besides using PCA for dimensionality reduction, we propose to use it for drift detection as well. We depict this process in Algorithm 1. The algorithm takes as an input a sequential dataset at current time moment t and a sequential dataset at previous time moment $t - 1$ and outputs a boolean variable indicating whether or not a drift has occurred between the two datasets. First, the values of the features of both datasets are standardized (line 5). The objective of this step is to standardize the range of the values of the initial dimensions in such a way to make them contribute equally to the analysis. Having standardized the values, the covariance matrices of the features of both datasets are computed (line 6). This step helps us understand how the features vary from the mean, with respect to each other, i.e., if there is any relationship between these features. Thereafter, the eigenvectors and eigenvalues of the covariance matrices are computed (line 7). 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 Components*. Thus, eigenvectors and eigenvalues always come in pairs and have a number that is equal to the number of dimensions in the dataset. Finally, the degree of drift between current dataset and the dataset at the previous time moment is computed by measuring the angle of intersection between the eigenvalues of both datasets (i.e., the variance of major directions of the datasets) (line 8) [25]. If this angle is greater or equal to 60° meaning that the directions of the datasets are significantly divergent, a drift is signaled; otherwise no drift is reported (lines 9-11).

3.2 Online Outlier Detection

Outlier detection is a primordial step in data preparation. An outlier is a data point that lies at 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. Traditional outlier detection approaches are often based on determining the standard data distribution and marking as outliers the data observations that deviate from this distribution. Yet, such an approach is not effective when it comes to intrusion detection in the IoT where no complete dataset is available for random access and hence the data distribution cannot be accurately determined. In addition, the distribution of the intrusion detection data is subject to change over time, thus disrupting the efficiency of traditional outlier detection approaches which assume a steady data distribution for the complete data. Therefore, we adopt in this work an online outlier detection technique that is based on the work discussed in [26]. This technique proposes to determine the outliers not only based on the entire dataset (available so far) but also based on the temporally close data points which are selected based on change in data distribution. To achieve this vision, deviations are determined in both global (all historical data points available so far) and local (temporally close data points which follow the same data distribution as the underlying data point) contexts. The global context is useful to identify those outliers that diverge from all available data points, while the local context help identify those outliers that diverge from the recent trends in the data.

Algorithm 2 depicts the online outlier detection process. The algorithm takes as inputs a time window, a data distribution function, the average neighbor density of all the history data points, and average neighbor density of the data points during the specified time window. For each data point, the algorithm first computes its neighbor density point (line 6) and then uses this density to compute the global and local deviation factors (lines 7 – 8). It then computes the standard deviations of these the global and local deviation factors (lines 9 – 10). The global and local deviation factors are then compared with their standard deviations. If either the local or global deviation factor is three times bigger than its standard deviation (i.e., significantly big) [27], the data point is considered to be an outlier (lines 11 – 14). In case a drift in the data is detected based on Algorithm 1 (line 15), the average local neighbor density during the specified time window is updated based on the new distribution of the data (lines 16 – 17).

4 DRIFT ADAPTATION: ONLINE DEEP NEURAL NETWORK

In this section, we provide the details of the online DNN, which is mainly inspired from the work presented in [13]. 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.

Algorithm 2: Online Outlier Detection

```

1: Input:  $f(x)$ : Data distribution function
2: Input:  $\omega$ : Time window
3: Input:  $\hat{D}_G$ : Average neighbor density of all the history data points
4: Input:  $\hat{D}_L$ : Average neighbor density of the data points in the recent time window  $\omega$ 
5: Output: Outliers: Array storing the data points that are identified as outliers
4: procedure ONLINEOUTLIERDETECTION
5:   for each new incoming data point  $x$  do
6:     Compute the neighbor density  $\theta_x$  of  $x$ 
7:     Compute the global deviation factor as  $\varrho_x = 1 - \frac{\theta_x}{\hat{D}_G}$ 
8:     Compute the local deviation factor as  $\rho_x = 1 - \frac{\theta_x}{\hat{D}_L}$ 
9:     Compute the standard deviation of  $\varrho_x$  as  $\sigma_{\varrho_x}$ 
10:    Compute the standard deviation of  $\rho_x$  as  $\sigma_{\rho_x}$ 
11:    if  $\varrho_x > 3 \times \sigma_{\varrho_x}$  or  $\rho_x > 3 \times \sigma_{\rho_x}$  then
12:      Mark  $x$  as outlier
13:      Add  $x$  to Outliers
14:    end if
15:    Run Algorithm 1 to detect drifts
16:    if a drift in the data occurs then
17:      Update  $\hat{D}_L$ 
18:  end procedure

```

In traditional machine/deep learning, the model is usually trained in a batch mode, meaning that a batch of large amount of data is used to train the machine/deep learning model at once. Thus, the obtained model would be static, as it is based on static relationships feeding the data features and target variable. Yet, in a very dynamic and evolving domain such as intrusion detection in the IoT where attackers are constantly adjusting their attack behavior, the attack patterns would undoubtedly change over time. Therefore, we adopt in this work an online deep learning strategy whereby the machine learning model handles a few data samples at a time and hence can be tuned on the fly. This helps bypass the concept drift problem as the model can continuously tune its hypothesis in the light of the newly received data¹.

Formally speaking, the online deep model observes, one at a time, a sequence $(I_1, \dots, I_t)$ of data instances, for which a model M_t is trained to predict a label. Thus, upon the receipt of a new data instance I_{t+1} , the label L_{t+1} associated with I_{t+1} is predicted using M_t . Similarly, upon the receipt of the next data instance I_{t+2} , the ground truth regarding L_{t+1} is available and hence the model is updated with historical data $(I_1, \dots, I_{t+1})$.

Adopting a deep learning approach in such an online setting entails several design choices. In fact, deciding upon the appropriate depth of the neural network is tricky. Having a overly simple network would make the power of learning complex patterns to be limited. On the other hand, having an overly deep network would result in a very slow model convergence, which contradicts with the main idea of online machine learning. In traditional batch learning models, this challenge is usually addressed using a validation phase, where the dataset is split into a training set and a validation set. After the training phase, the model is used on the validation set to tune the neural

network model and hence determine the structure of the neural network that best generalizes to new data. Yet, in practice, having validation data in an online learning setting is unrealistic as data come in a sequential manner. Therefore, fine-tuning the neural network model on validation data is not an option.

Consequently, an online fine-tuning method is needed. For this purpose, we propose to adopt the *Hedge Backpropagation* method discussed in [13]. In this method, each hidden layer is associated with an output classifier and a weight α . The final prediction is made using a weighted combination over the different output classifiers. To compute the weight α of each classifier, the *Hedge* algorithm proposed in [14] is employed. The algorithm starts by assigning equal initial weights to each hidden layer. Then, after the classifier has made a prediction and the ground truth is revealed, the weight of each classifier at the next time moment is updated by multiplying the weight at the current time moment by a discount factor parameterized by the loss undergone by the underlying classifier.

Consider an online attack classification task whose aim is to learn a function F on a collection of training samples $D = \{(x_1, y_1), \dots, (x_K, y_K)\}$ 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 $\hat{y}_k$ and the cumulative prediction error by $E_K = \frac{1}{K} \sum_{k=1}^K \tau_{(\hat{y}_k \neq y_k)}$. 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 at time moment t . Then at the next time moment $t+1$, 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 $(l^{(1)}, \dots, l^{(H)})$ using the *Hedge Backpropagation* [28] is defined by Equation (1):

$$F(x) = \sum_{h=0}^H \delta^{(h)} f^{(h)} \quad (1)$$

with

$$f^{(h)} = \text{softmax}(l^{(h)} \phi^{(h)}), \forall h = 1, \dots, H \quad (2)$$

and

$$l^{(h)} = \alpha(W^{(h)} l^{(h-1)}), \forall h = 1, \dots, H; l^{(0)} = x \quad (3)$$

where α 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 δ are newly introduced parameters and thus are not part of the traditional deep neural network.

1. <https://neptune.ai/blog/concept-drift-best-practices>

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* [14]. At the first iteration, all the weights δ are uniformly distributed, where $\delta^{(h)} = \frac{1}{H+1}, h = 0, \dots, H$. 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):

$$\delta_{k+1}^h \leftarrow \delta_k^h \ell^{\Gamma(f^h(x), y)} \quad (4)$$

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 δ are normalized so that $\sum_h \delta_k^h = 1$. The adaptive loss function is given in Equation (5).

$$\Gamma(F(x), y) = \sum_{h=0}^H \delta^{(h)} \Gamma(f^h(x), y) \quad (5)$$

In order to learn the parameters $\phi^{(h)}$ across the different classifiers, the online gradient descent methodology [29] is used as per Equation (6).

$$\begin{aligned} \phi_{k+1}^{(h)} &\leftarrow \phi_k^{(h)} - \eta \nabla_{\phi_k^{(h)}} \Gamma(F(x_k), y_k) \\ &= \phi_k^{(h)} - \eta \delta^{(h)} \nabla_{\phi_k^{(h)}} \Gamma(f^{(h)}, y_k) \end{aligned} \quad (6)$$

Where η 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 (5), we can update the feature representation parameters $W^{(h)}$ as per Equation (7).

$$W_{k+1}^{(h)} \leftarrow W_k^{(h)} - \eta \sum_{i=1}^H \delta^{(i)} \nabla_{W^{(h)}} \Gamma(f^{(i)}, y_k) \quad (7)$$

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

The logic of the online DNN is further summarized in Algorithm 3.

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

---

Algorithm 3: 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
6:    $f^{(h)}, \forall h = 1, \dots, H$ 
7:   Initialize  $\delta^{(h)} = \frac{1}{H+1}, \forall h = 1, \dots, H$ 
8:   for sequence  $k = 1, \dots, T$  do
9:     Get  $x_k$ 
10:    Use Equation (1) to predict  $\hat{y}_k$ 
11:    Uncover the ground truth of the class label  $y_k$ 
12:    Update the loss function, i.e.,
13:     $\Gamma_k^{(h)} = \Gamma(f_k^{(h)}, y_k), \forall h = 1, \dots, H$ 
14:    Use Equation (6) to update  $\phi_{k+1}^{(h)}, \forall h = 1, \dots, H$ 
15:    Use Equation (7) to update  $W_{k+1}^{(h)}, \forall h = 1, \dots, H$ 
16:    Use Equation (4) to update  $\delta_{k+1}^h, \forall h = 1, \dots, H$ 
17:    Normalize  $\delta_{k+1}^h = \frac{\delta_{k+1}^h}{\zeta_k}$  with  $\zeta_k = \sum_{h=0}^H \delta_{k+1}^h$ 
18:  end procedure

```

---

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) [30] 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, $\dots$, 16. The discount factor ℓ is set to 0.99. The implementation was carried out in Keras Python² on Tensorflow.

The *DS2OS traffic traces* dataset [15] 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 [31] is employed, while setting

2. <https://keras.io/>

TABLE 1: List of features in the dataset

Dimension	Data Type	Example
Source ID	GUID	/../light425
Source Address	GUID	/agent3/lightcontrol3
Source Type	Categorical	movement sensor
Source Location	Categorical	Dining Room
Destination Service Address	Categorical	/agent1/lightcontrol
Destination Service Type	Categorical	Sensor service
Destination Location	Categorical	Bathroom
Accessed Node Address	Categorical	/agent3/movement3/movement
Accessed Node Type	Categorical	/lightControler
Operation	Categorical	write
Value	Continuous	20.1757
Timestamp	Discrete	1520031696052
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
DoS	5,780	57.64%	A device performing access at a very high frequency
Data Probing	342	3.41%	A device writes anomalous data types
Malicious Control	889	8.87%	A device attempting to take control over another one
Malicious Operation	805	8.03%	A performing an operation it is isn't supposed to perform
Network Scan	1,559	15.55%	A device that accesses the network ranges of other devices
Spying	532	5.30%	A device reading the values of other devices
Wrong Setup	120	1.2%	A devices accessing its services in a wrong location
Total	100,27	100%	Amounts to 3% of the dataset

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 Table 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 machine learning model predicts a certain instance is an attack of type a_i (e.g., DoS), matching the actual ground truth. True negative occurs when an attack of type $a_j \neq a_i$ is not classified as being of type a_i . False negative is a case in which the machine learning model predicts an attack of type a_i as being another type $a_j \neq a_i$ of attack. False positive is a case in which the machine learning model predicts an attack of type that is not a_i as being of type 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 classified as scan attack and one classified as normal), and 26 misclassifications in the case of the normal class (classified as DoS attacks). On the other hand, by looking at Table

4, we observe that the static version of the DNN results in a significant number of misclassifications for all the attack classes. For example, in the case of DoS attacks, 437 misclassifications occurred, 58 as spying attacks and 379 as normal cases. In the case of scan attacks, 62 misclassification cases took place, 17 as spying and 36 as wrong setup attacks, and 9 as normal cases. In the case of the normal class, 1,477 misclassification cases took place (concept drift) across the different attack types. This shows that the static model is highly vulnerable to concept and data drifts and thus is not able to maintain its performance over time.

To better understand the impacts of the data and concept drifts on both compared solutions, we provide in Tables 5 and 6 several performance metrics (i.e., accuracy, precision, recall and F1 Score) on the training and the testing data respectively. Accuracy is the ratio of number of correct predictions to the total number of input data points. Precision is the number of correct positive results divided by the number of positive results predicted by the classifier. Recall (or sensitivity) is the proportion of actual positives that are identified as such. The recall tries to answer the following question: Of all the attack classes, how many did our model actually identify as such?. F1 score is the harmonic average of precision and recall.

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.

TABLE 3: Confusion matrix of our solution

	DoS	Data Probing	Malicious Control	Malicious Operation	Scan	Spying	Wrong Setup	Normal
DoS	1,261	0	0	0	0	0	0	26
Data Probing	0	64	0	0	0	0	0	0
Malicious Control	0	0	263	0	0	0	0	0
Malicious Operation	0	0	0	61	0	0	0	0
Scan	0	0	1	0	425	0	0	0
Spying	0	0	0	0	0	122	0	0
Wrong Setup	0	0	0	0	0	0	28	0
Normal	6	0	1	0	0	0	0	105,296

TABLE 4: Confusion matrix of the regular DNN model

	DoS	Data Probing	Malicious Control	Malicious Operation	Scan	Spying	Wrong Setup	Normal
DoS	830	0	0	0	0	0	0	551
Data Probing	0	50	0	0	9	0	0	320
Malicious Control	0	0	240	10	0	0	0	122
Malicious Operation	0	0	0	37	0	0	0	18
Scan	0	0	15	14	305	0	0	133
Spying	58	0	0	0	17	116	0	108
Wrong Setup	0	0	0	0	36	0	24	225
Normal	379	14	10	0	9	6	4	103,845

TABLE 5: Training Data

Attacks	Our Solution				Regular Deep Learning			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
DoS	98.5%	97.9%	97.8%	97.8%	96.1%	96%	96%	96%
Data Probing	100%	100%	100%	100%	98.5%	97.56%	98%	98%
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	99.8%	99.7%	99.5%	99.5%	97.4%	97.9%	97.9%	97.9%
Spying	100%	100%	100%	100%	97.5%	97.8%	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%	94.3%	94.1%	94.1%

TABLE 6: Testing Data

Attacks	Online Deep Learning				Regular Deep Learning			
	Accuracy	Precision	Recall	F1 Score	Accuracy	Precision	Recall	F1 Score
DoS	98.4%	97.4%	97.3%	97.5%	86.2%	86.4%	86.3%	86.3%
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%	90.9%
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%

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 shows 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 pos-

itive and false negative rates are taken. We notice from the figure that the false positive and false negative percentages entailed by our solution are quite smaller than those entailed by the regular DNN. For example, with 350,000 data instances, the false positive percentage entailed by our solution is 0.57% compared to 8.32% for the regular DNN. Similarly, with 350,000 data instances, the false negative percentage entailed by our solution is 0.58% compared to 5.98% for the regular DNN. Another conclusion that can be taken from this Figure is that the regular DNN model is more impacted by the concept drift than the data drift, as it entails higher percentages of false positive than false negative.

6 CONCLUSION

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 a comprehensive and generic solution for detecting and fighting against data and concept drift in IoT-based intrusion detection settings. We conduct a series of experiments on a real-world dataset that is designed to represent model drifts in IoT-based intrusion detection

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

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. Arisdakessian, O. A. Wahab, A. Mourad, H. Otrók, 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] G. Rjoub, J. Bentahar, O. Abdel Wahab, and A. Saleh Bataineh, "Deep and reinforcement learning for automated task scheduling in large-scale cloud computing systems," *Concurrency and Computation: Practice and Experience*, p. e5919, 2020.
- [3] 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.
- [4] P. Illy, 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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.
- [9] O. A. Wahab, J. Bentahar, H. Otrók, 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.
- [10] J. Gama, I. Žliobaitė, 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.
- [11] O. A. Wahab, A. Mourad, H. Otrók, and T. Taleb, "Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1342–1397, 2021.
- [12] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1–3, pp. 37–52, 1987.
- [13] 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.
- [14] 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.
- [15] "DS2OS traffic traces," <https://www.kaggle.com/francoisxa/ds2ostraffictraces?select=mainSimulationAccessTraces.csv>, accessed: 2021-01-27.
- [16] 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.
- [17] S. Hajjheidari, K. Wakil, M. Badri, and N. J. Navimipour, "Intrusion detection systems in the internet of things: A comprehensive investigation," *Computer Networks*, vol. 160, pp. 165–191, 2019.

- [18] G. Thamaras and S. Chawla, "Towards deep-learning-driven intrusion detection for the internet of things," *Sensors*, vol. 19, no. 9, p. 1977, 2019.
- [19] N. Koroniotis, 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.
- [20] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.
- [21] Y. Liu, J. Wang, J. Li, S. Niu, and H. Song, "Machine learning for the detection and identification of internet of things (iot) devices: A survey," *arXiv preprint arXiv:2101.10181*, 2021.
- [22] C. Nixon, A. Sedky, and M. Hassan, "Practical application of machine learning based online intrusion detection to internet of things networks," in *2019 IEEE Global Conference on Internet of Things (GCIoT)*. IEEE, 2019, pp. 1–5.
- [23] A. Abbasi, A. R. Javed, C. Chakraborty, J. Nebhen, W. Zehra, and Z. Jalil, "Elstream: An ensemble learning approach for concept drift detection in dynamic social big data stream learning," *IEEE Access*, vol. 9, pp. 66 408–66 419, 2021.
- [24] D. Nallaperuma, R. Nawaratne, T. Bandaragoda, A. Adikari, S. Nguyen, T. Kempitiya, D. De Silva, D. Alahakoon, and D. Pothuhera, "Online incremental machine learning platform for big data-driven smart traffic management," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 12, pp. 4679–4690, 2019.
- [25] J. Shao, Z. Ahmadi, and S. Kramer, "Prototype-based learning on concept-drifting data streams," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 412–421.
- [26] S. Sadik and L. Gruenwald, "Online outlier detection for data streams," in *Proceedings of the 15th Symposium on International Database Engineering & Applications*, 2011, pp. 88–96.
- [27] S. Papadimitriou, H. Kitagawa, P. B. Gibbons, and C. Faloutsos, "LocI: Fast outlier detection using the local correlation integral," in *Proceedings 19th international conference on data engineering (Cat. No. 03CH37405)*. IEEE, 2003, pp. 315–326.
- [28] A. Gammerman and V. Vovk, "Hedging predictions in machine learning," *The Computer Journal*, vol. 50, no. 2, pp. 151–163, 2007.
- [29] 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.
- [30] 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.
- [31] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-validation," in *Encyclopedia of database systems*. Springer, 2009, pp. 532–538.