Lifelong Anomaly Detection Through Unlearning

Min Du
University of California, Berkeley
min.du@berkeley.edu

Zhi Chen
University of California, Berkeley
zhichen98@berkeley.edu

Chang Liu
Citadel Securities
liuchang2005acm@gmail.com

Rajvardhan Oak
University of California, Berkeley
rvoak@berkeley.edu

Dawn Song
University of California, Berkeley
dawnsong@berkeley.edu

ABSTRACT

Anomaly detection is essential towards ensuring system security and reliability. Powered by constantly generated system data, deep learning has been found both effective and flexible to use, with its ability to extract patterns without much domain knowledge. Existing anomaly detection research focuses on a scenario referred to as *zero-positive*, which means that the detection model is only trained for normal (i.e., negative) data. In a real application scenario, there may be additional manually inspected positive data provided after the system is deployed. We refer to this scenario as *lifelong anomaly detection*. However, we find that existing approaches are not easy to adopt such new knowledge to improve system performance.

In this work, we are the first to explore the *lifelong anomaly detection* problem, and propose novel approaches to handle corresponding challenges. In particular, we propose a framework called *unlearning*, which can effectively correct the model when a false negative (or a false positive) is labeled. To this aim, we develop several novel techniques to tackle two challenges referred to as *exploding loss* and *catastrophic forgetting*. In addition, we abstract a theoretical framework based on generative models. Under this framework, our unlearning approach can be presented in a generic way to be applied to most zero-positive deep learning-based anomaly detection algorithms to turn them into corresponding lifelong anomaly detection solutions.

We evaluate our approach using two state-of-the-art zero-positive deep learning anomaly detection architectures and three real-world tasks. The results show that the proposed approach is able to significantly reduce the number of false positives and false negatives through unlearning.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; • **Information systems →** *Online analytical processing*; • **Computing methodologies →** Online learning settings.

KEYWORDS

anomaly detection; online learning; unlearning

ACM Reference Format:
Min Du, Zhi Chen, Chang Liu, Rajvardhan Oak, and Dawn Song. 2019. Lifelong Anomaly Detection Through Unlearning. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3319535.3363226

CCS '19, November 11–15, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6747-9/19/11…$15.00
https://doi.org/10.1145/3319535.3363226

1 INTRODUCTION

Anomaly detection is an indispensable step for security, owing to unavoidable vulnerabilities in complex computer systems and ceaselessly trying of sophisticated attacks [4]. Modern systems constantly produce system data that may reflect system status, which are valuable data source towards real-time anomaly detection.

Deep learning has been an effective approach with its ability to extract patterns from massive data [8, 23]. For example, a naive way is to train a supervised model with both normal and abnormal data, and use the trained model to assign labels in detection. A more desirable approach is to train a model that could detect *unforeseen anomalies* such as zero-day attacks [42]. That is, the deep learning model is capable of detecting anomaly types that are not known while it is being trained. To achieve this goal, anomaly detection methods that do not require any abnormal data for detection are preferred, which are referred to as "zero-positive" anomaly detection [23].

There have been several works proposed to handle this case. For example, LSTM-based anomaly detection models [8, 25] could be trained on normal time-series data to do forecasting, and detect a real data point as abnormal if it deviates from the predicted one. As another example, autoencoder-based anomaly detection models [15, 47] are proposed to detect anomaly over data that is time-insenstive, i.e., each data point is independent to each other.

A problem for zero-positive anomaly detection is that it may not always generalize. That is, an abnormal event observed during test time, which we refer to as *false negative*, can be classified as normal. In practice, administrator can manually inspect a small portion of the events and provide labels. However, it remains open for a deep learning-based approach how to effectively update the model with these new labeled data. For example, in a network traffic anomaly detection application, the emerging of new workloads may require the model to learn new patterns and to optionally forget old patterns *in a controlled manner*. Moreover, if system administrators provide feedback on false negatives (and false positives as well),

the model may also need to be updated in a cost-effective manner to better serve its goal.

In this paper, we focus on the *lifelong anomaly detection problem* to fill the gap. There have been several challenges to achieve the goal. First, so far, there have been no mechanism to make a deep learning-based anomaly detection model remember an *abnormal* instance. To make the model remember a normal instance x, most existing approaches learn a model to predict $\Pr(x)$, and make this probability high enough to be considered.

Therefore, to make the model remember x is abnormal, we need to decrease the predicted probability $\Pr(x)$. This is equivalent to make the model *forget* that x is a normal instance. Based on this idea, we develop an algorithm called *unlearn* to make a model unlearn that a false negative instance is normal. Note that the concept of *unlearning* has been proposed before by Cao et al. [3], which, unlike our work, focuses on unlearning samples that exist in the training dataset.

Second, naively decreasing the probability of $\Pr(x)$ will also likely make the model predict other normal events as abnormal. We refer to this issue as *exploding loss*, where the term *loss* is represented as $-\mathbf{log}\Pr(x)$. When $\Pr(x)$ is close to zero, the loss can be arbitrarily large. Maximizing it will cause deep learning models to become arbitrary, and not well-functioning to serve the anomaly detection task. Under the unlearning framework, we develop the *bounding loss* and *learning rate shrinking* techniques to mitigate this issue.

Third, since the lifelong anomaly detection problem will run in fashion that the model will keep being updated over time, it may forget previously observed examples. This issue is typically referred to as *catastrophic forgetting* in the deep learning literature [12]. A naive solution to this problem is to retrain the model with all previously observed examples. However, this naive approach is not practical, since the data set will be ever-growing over time, and retraining will soon become too costly. To tackle this issue, we develop an incremental learning approach to leverage a maintained important memory set to make the model not forgetting important past examples.

Fourth, we hope our approach can be generic to be applied to existing deep learning-based anomaly detection algorithms so that we can take the advantage of previous work. We observe that most existing approaches can be captured by a class of machine learning algorithms, called *generative models*. We thus abstract previous approaches in a theoretical framework. By taking advantage of this framework, we can easily present our unlearning algorithm in a generic way to make sure it can be applied to an arbitrary deep learning-based anomaly detection algorithm.

We summarize our main contributions as below.

(1) We are the first to examine the lifelong learning for deep learning based anomaly detection problems. To this aim, we propose the *unlearning* framework which can be applied to any deep learning-based zero-positive anomaly detection approach to turn it into a lifelong anomaly detection solution.
(2) We propose novel techniques to tackle the *exploding loss* and *catastrophic forgetting* challenges. While the former is unique to anomaly detection, the latter is a generic lifelong learning problem. We hope our solutions can inspire more studies of these issues.
(3) As a side product, we abstract a theoretical framework to apply a generative model for anomaly detection, so that our unlearning approach can be presented in a generic way. We hope this framework can shed new light to future deep learning-based anomaly detection research.
(4) We evaluate our approach using three real anomaly detection datasets, namely, HDFS log, Yahoo network traffic, and credit card transaction. We show that our proposed lifelong learning method could significantly reduce both the number of false positives and the number of false negatives. For example, for HDFS log, the experiment results show a reduction of up to 77.3% false positives and up to 76.6% false negatives, under different thresholds.

The rest of the paper is organized as follows. In Section 2, we formalize the anomaly detection problem and introduce the theoretical framework to apply generative models for anomaly detection. Then we present our unlearning framework and key technical novelties to handle the challenges in Section 3. In Section 4, we evaluate our proposed approach using various dataset and deep learning architectures. We discuss our observations in Section 5 and related work in Section 6. In the end, we conclude in Section 7.

2 LIFE-LONG ANOMALY DETECTION

In this section, we will first explain the zero-positive anomaly detection using a real-world example. Then we will formalize this problem, and introduce the state-of-the-art deep learning-based approach.

2.1 Motivation examples

System data collected in real-time could be continuous values such as CPU usage and temperature, or categorical values such as SYSCALL and function API calls. Each data point could either be a single value, or a vector. Moreover, system status and events evolve in time. The collected data points could either be analyzed as independent instances, or as time series events with time dimension involved. For example, an event *CPU-high* could itself indicate an anomaly, or only be an anomaly when it's following some event.

Previous work has extensively explored system data for anomaly detection, for example, using LSTM model on system logs [8] to detect execution anomalies, and autoencoder model on hardware performance counters to detect performance problems [15]. Because of the difficulty in obtaining abnormal labels, previous work typically circumvents this by training only on normal data, hoping to detect unforeseen anomalies that do not follow the learned normal pattern. However, it is possible that the training data is noisy, or new patterns emerge in detection, such that when the trained model is being used in detection, false positives or false negatives may appear.

Consider the following scenario. The training data may contain two instances: *ssh* → *program* → *exit* → *CPU-high* and *ssh* → *game* → *exit* → *cpu-high*. A zero-positive model trained on this is possible to learn *ssh* → ⋆ → *exit* → *CPU-high* as being a normal sequence, where ⋆ indicates an arbitrary event. However, while using this model for anomaly detection, it is possible to encounter *ssh* → *notepad* → *exit* → *CPU-high*, which is further detected as normal. Because of the incompleteness of the training data, the anomaly

detection model fails to recognize the suspicious *CPU-high* activity, which is possibly due to a non-logged attack. As another example, if the model is only trained on *ssh* → *program* → *exit* → *CPU-high*, it is possible to falsely detect *ssh* → *game* → *exit* → *CPU-high* as an anomaly.

In real-world scenarios while using the trained models for anomaly detection, there could be system admins reporting false negatives and false positives that the model fails to detect. In this paper, we focus on the life-long learning issue of such anomaly detection models, that is, how to incrementally update the model with the reported false negatives and false positives.

2.2 Problem definition

In this section, we first formalize the zero-positive anomaly detection problem, and briefly illustrate how generative machine learning models can be used to handle this problem.

Problem Definition 1 (Anomaly detection). *Consider an event sequence x_t, $(1 \le t \le T)$ such that each instance is sampled from a stationary distribution $\mathcal{D}$ with a probability of $1-\epsilon$, ($\epsilon \ge 0$ is a small constant). Detect for each followup instance x_t $(t > T)$ whether x_t is sampled from the same distribution.*

In this definition, the event sequence $x_1, ..., x_T$ is the training data. The distribution $\mathcal{D}$ defines whether an instance is normal (in distribution) or an anomaly. The constant ϵ controls the rate that anomaly can appear in the training data. When $\epsilon = 0$, it means that the training data does not include anomaly at all.

Previous work on zero-positive anomaly detection [8, 15, 26] assumes that the training dataset only contains normal data, i.e., $\epsilon = 0$. However, it is possible that human labeling may make occasional mistakes. In this work, we consider the anomaly detection problem with noisy data, which means that $\epsilon > 0$ though it is small, but we do not have the ground truth of which instances are abnormal or not. This definition is applicable to most real applications.

Distribution assumption. Different applications make different assumptions on the distribution $\mathcal{D}$ so that different machine learning models can be applied to learn this distribution. In general, we consider two types of distributions: *time-insensitive* and *time-sensitive*.

Definition 2.1 (Time-insensitive distribution). The value of a *time-sensitive distribution* $\mathcal{D}$ at any time t is independent to each other. Its probability mass function is defined by $\Pr(x_t)$.

Definition 2.2 (Time-sensitive distribution). The value of a *time-sensitive distribution* $\mathcal{D}$ at time t depends on all past history. The probability mass function is defined by $\Pr(x_t|x_{t-1}...x_1)$.

Life-long learning. In this work, we consider a special case of anomaly detection, called *life-long anomaly detection*. In the definition of Problem 1, we consider that no labels are provided on the training set. Learning from such a noisy data is inevitably inaccurate.

In real application scenarios, however, practitioners can manually examine a few suspicious examples to provide their labels. In this work, we are primarily interested in whether we can significantly improve the accuracy in such a case.

Problem Definition 2 (Life-long Anomaly detection). *Consider an event sequence x_t, $(1 \le t \le T)$ such that each instance is sampled from a stationary distribution $\mathcal{D}$ with a probability of $1-\epsilon$, ($\epsilon \ge 0$ is a small constant). In addition, we have a set of n pairs (t_i, l_i), such that $1 \le t_i \le T$ and $l_i \in \{-1, +1\}$. We know l_i is negative if x_{t_i} is sampled from $\mathcal{D}$, and l_i is positive if not. Detect for each followup instance x_t $(t > T)$ whether x_t is sampled from the same distribution.*

Note that manual inspection is a costly operation; thus, the total number of labeled pairs, i.e., n, must be small. Note that the additional data is most likely to be wrongly labeled by the deployed model. We refer to a pair (t_i, l_i) with $l_i = +1$ as a *false negatives*; and *false positive* for $l_i = -1$ to emphasize this fact. Although the focus of this paper is on false negatives, our framework is generic to handle both false negatives and false positives.

2.3 A theoretical framework for existing machine learning-based anomaly detection

In this section, we introduce a generic framework to describe machine learning-based anomaly detection approaches for Problem 1 using *generative models*. We will then explain recent works using deep learning models for anomaly detection to achieve the state-of-the-art results, and show that they can be modeled in the generic framework. In the end, we will explain how to train the models.

2.3.1 Anomaly detection using a generative model. It is easy to see that if we can learn the distribution (its probability mass function) under an event sequence, we can apply it to easily detect an anomaly. For example, given a new instance x_t for a time-sensitive distribution, we can simply check if $Pr(x_t|x_{t-1}...x_1) > \tau$ for a threshold τ to determine whether x_t is an anomaly.

Therefore, almost all anomaly detection algorithms rely on a class of machine learning models called *generative models* [27]. A generative model typically considers two classes of random variables, i.e., observable variables X (e.g., events) and hidden variables H. It assumes that the joint probability $\Pr(X, H)$ is modeled by a function $f_\theta(X, H)$, which is parameterized by θ. Thus, we can obtain the marginal probability of X as

$$\Pr(X) = \int_h f_\theta(X, H = h) \tag{1}$$

The learning algorithms search for a set of parameters θ to maximize the *likelihood* or optimize its equivalent transformation:

$$\begin{aligned} \theta^\star &= \operatorname*{argmax}_\theta \prod_{x_t} \Pr(X = x_t) \\ &= \operatorname*{argmax}_\theta \prod_{x_t} \int_h f_\theta(X = x_t, H = h) \end{aligned} \tag{2}$$

Using such a model, given a new instance x (i.e., $x_1...x_t$ in the time-sensitive case, or x_t in the time insensitive case), we can compute $\Pr(X = x)$ using (1) and check whether it is above a threshold τ. There have been various generative models used for anomaly detection, such as Gaussian mixture model [48], Principal Component Analysis [18], Long Short-Term Memory [8], and autoencoder [31].

2.3.2 Deep learning models. We now explain two deep learning models, namely long short-term memory and autoencoder, and how they are used for anomaly detection in recent literature.

Long short-term memory (LSTM). LSTM [17] is a type of *recurrent neural network* (RNN) to handle sequential data. LSTM has achieved the state-of-the-art on various sequence-based applications such as speech recognition [16].

In general, a RNN computes an *embedding* h_t (i.e., the hidden states) for a prefix of a sequence $x_1 \ldots x_t$. We use $h_t = \phi(x_1 \ldots x_t)$ to denote this embedding relationship. A RNN models the mapping ϕ as an incremental computation: $h_{t+1} = f_\theta(h_t, x_{t+1})$ parameterized by θ. In addition, a RNN models the conditional probability $\Pr(x_{t+1}|h_t)$ as a function $g_\rho(h_t, x_{t+1})$ parameterized by ρ. Different RNN models differ in their concrete choices of the functions f and g. We refer the readers to [17] for more details.

To apply LSTM for anomaly detection at time t, we can compute

$$h_{t-1} = f_\theta(h_{t-2}, x_{t-1}) \quad (3)$$

$$\Pr(x_t|x_1 \ldots x_{t-1}) = g_\rho(h_{t-1}, x_t) \quad (4)$$

where (3) can be recursively computed at each time point t. There are several ways to use $\Pr(x_t|x_1 \ldots x_{t-1})$ computed by (4) to detect if x_t is abnormal. One way (e.g., [8]) is to compare the probability against a pre-determined threshold τ as discussed above (see Figure 1 for example).

If the event space is continuous, an alternative way (e.g., [25]) is to use this probability density function to predict the next value $x_t^\star$ (i.e., as the one maximizing $\Pr(x_t^\star|x_1 \ldots x_{t-1})$); and compare whether x_t deviates from $x_t^\star$ too much (see Figure 2 for example).

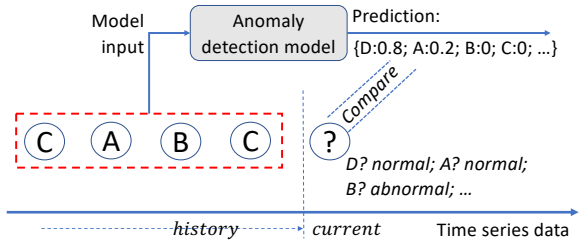


Figure 1: An example of LSTM-based anomaly detection on discrete event space. Given a history sequence of C, A, B, C, **LSTM computes the probability of the next events as** {D: 0.8, A: 0.2, B: 0, ...}, **and** $\tau = 0.1\%$. **If the next event (i.e., labeled with "?") is D or A, then its probability (**80% **or** 20% **is above the threshold, and thus it is normal. Otherwise, the probability is** 0%, **(since the summation of all probability is** 100%**) and the event is detected as abnormal.**

One benefit of LSTM is that it can automatically learn to forget non-essential events in a sequence. Therefore, it can effectively handle truncated data (i.e., the model runs from the middle of the event sequence) and noisy data (i.e., anomaly in the training data). In this work, we choose LSTM as the default model to time-sensitive distribution.

Autoencoder. When the events are independent in time, we do not need to consider the history $x_1 \ldots x_{t-1}$ when examining x_t. In this

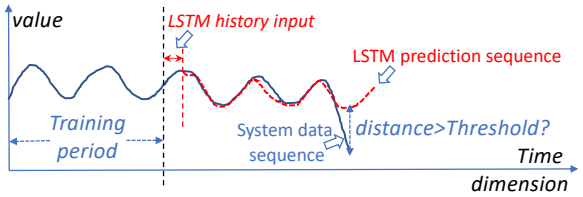


Figure 2: An example of LSTM-based anomaly detection on continuous event space. Suppose the network traffic of a website roughly follows the shape of a sine wave. LSTM model could learn this trend, and forecast the next value based on it. For example, given a history sequence of 1.2, 1.8, 2. 1.8**, LSTM model is possible to predict** 1.2 **as the next data point. To check if the real system generated next data point (e.g.,** 0.2**) is normal, we can compute the distance between this value and the predicted one. If the distance is larger than a threshold, we detect it as abnormal.**

case, an *Autoencoder* [14, 26] is an effective deep learning model for this task.

An autoencoder consists of an *encoder* ϕ and a *decoder* φ; both are parameterized functions. The encoder maps an input x into a hidden state $h = \phi(x)$, and the decoder maps h into another instance $x' = \varphi(x)$ in the input space. As a generative model, the joint probability $\Pr(x, h)$ can be modeled as

$$\Pr(X = x) \propto \exp(-||x - \varphi(\phi(x))||^2/2) \quad (5)$$

In (5), $\varphi(\phi(x))$ is the output of the autoencoder for input x. This is illustrated in Figure 3. It is easy to see that we can check whether the distance between the output and input, i.e., $||x - \varphi(\phi(x))||^2$, is smaller than a threshold, to detect if the event is normal (i.e., $\text{Prob}(X = x)$ is large enough).

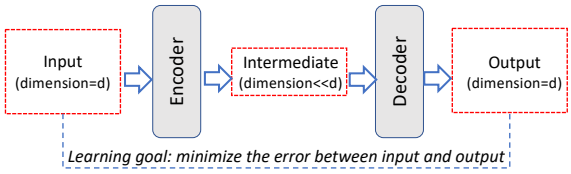


Figure 3: Autoencoder architecture.

2.3.3 Training models. So far, we discuss the models in an abstract way. Now, we provide more details for training a model so that the following discussions of our new approach can make sense.

To mathematically manipulate an event sequence, we encode each event into a numerical vector. For an event with continuous value, they can serve as a dimension of the vector directly. For an event with discrete value, we can encode them using so called *one-hot-encoding*. That is, given a value $x \in 1, \ldots, N$, we encode it as a N-dimension vector v, such that $v_i = 1$ if $x = i$, and $v_i = 0$ otherwise. For an event space with a mixture of multiple discrete and continuous values, we can convert them separately and concatenate the separate vectors into a bigger one.

To train a model, we need to solve the optimization problem defined by (1). It is equivalent to minimize $-\log\Pr(X)$. In deep

learning literature, we typically refer to an equivalent function of $-\mathbf{log}\Pr(X)$ as the *Loss function* $\mathcal{L}_\theta(X)$. For the time-sensitive case using an RNN model defined by (3-4), this can be defined as

$$\mathcal{L}_{\theta,\rho}(X) = \sum_{t=1}^{T} -\mathbf{log}\big(g_\rho(h_{t-1}, x_t)\big) \tag{6}$$

where h_t is defined recursively using (3) by setting $h_0 = 0$. For the autoencoder case, the loss function can be defined as

$$\mathcal{L}(x_t) = ||x_t - \varphi(\phi(x_t))||^2 \tag{7}$$

Deep learning relies on back-propagation [30] to compute the gradient of the loss function with respect to the parameters $\nabla_\theta \mathcal{L}_\theta(\S)$, and iteratively update parameters θ according to this gradient. Using the stochastic gradient decent (SGD)[14] as an example, θ is updated via

$$\theta_{\mathbf{new}} \leftarrow \theta_{\mathbf{old}} - \eta \cdot \nabla_\theta \mathcal{L}_\theta(\S) \tag{8}$$

where η is a small constant referred to as learning rate or step size. Other optimizers (such as Adam [20] and RMSProp [37]) rely on more complicated formula to update θ based on the gradient. In later discussions, when it is clear, we may omit the subscription of θ from the loss function $\mathcal{L}_\theta(x)$.

3 UNLEARNING FOR UPDATING ANOMALY DETECTION MODELS

When a machine learning model is deployed in applications, our goal is to update it with newly reported false positives or false negatives in a streaming fashion. Existing generative model-based learning approaches do not naturally support such an update. In this work, we propose an *unlearning* framework to handle such updates. The intuitive idea is that, once we know x_t is a false negatives, rather than maximize $Pr(x_t|x_1...x_{t-1})$, we want to minimize this probability. Directly doing it, however, will lead to several issues, such as exploding gradient and catastrophic forgetting. The unlearning approach implements this idea while not hurting the performance for other unlabeled data.

Note that false positives, which are normal events, can be handled in a regular way. In our presentation, we will explain the issues and our techniques with an emphasis of handling false negatives. At the same time, however, we will present the algorithm details in a generic way to handle both false negatives and false positives. In the following subsections, we will first provide an overview of our approach as well as the challenges, and then explain the two important techniques to tackle the challenges.

3.1 Overview of unlearning

We now consider the time-insensitive case. It is straightforward to extend the idea for the time-sensitive case. To illustrate the idea, we first consider the labeled set consists of one positive sample $(t, +1)$, such that x_t is falsely predicted as negative (i.e., normal example). In this case, we know that

$$\Pr(x_t) > \tau \tag{9}$$

for a threshold τ or, equivalently,

$$\mathcal{L}(x_t) < \tau' \tag{10}$$

for the corresponding threshold τ'. Our goal is thus to revise the model to decrease the probability of $\Pr(x_t)$, or equivalently to increase $\mathcal{L}(x_t)$. Notice that maximizing this is equivalent to minimize

$$\mathcal{L}_{\text{unlearn}}(x_t) = -\mathcal{L}(x_t) \tag{11}$$

Therefore, we can apply the same optimization algorithm for training to minimize $\mathcal{L}_{\text{unlearn}}(x_t)$ by computing its gradient

$$\nabla_\theta \mathcal{L}_{\text{unlearn}}(x_t) = \nabla_\theta\Big(-\mathcal{L}(x_t)\Big) = -\nabla_\theta \mathcal{L}(x_t)$$

Therefore, applying the update rule such as (8), we have

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} - \eta \cdot \nabla_\theta \mathcal{L}_{\text{unlearn}}(x_t) = \theta_{\text{old}} + \eta \cdot \nabla_\theta \mathcal{L}(x_t) \tag{12}$$

We can now generalize this idea to handle a labeled set $S = \{x_t, l_t\}$ by defining $\mathcal{L}_{\text{unlearn}}(S)$ as follows:

$$\mathcal{L}_{\text{unlearn}}(S) = -\sum_t l_t \mathcal{L}(x_t) \tag{13}$$

while the update rule corresponding to SGD (8) can be captured as follows:

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} + \eta \sum_t l_t \cdot \nabla_\theta \mathcal{L}(x_t) \tag{14}$$

The update rule for other gradient-based optimization algorithms can be adapted accordingly. Note that the learning process minimizes the loss. In contrast, our algorithm tries to maximize the loss for false negative examples. Therefore, we call our algorithm *unlearning*.

One great benefit of our algorithm is that it relies on the same component as training the model to optimize the unlearning loss function (13). Hence, our unlearning algorithm is generic and can be applied to arbitrary optimization-based deep learning models.

While the above idea is straightforward, a direct application of it suffers two major problems, namely *exploding loss* and *catastrophic forgetting*. We describe them and propose solutions below.

3.2 Handling exploding loss

Challenges. When handling false negatives, the unlearning algorithm essentially maximizes $\mathcal{L}(x_t)$, or equivalently minimizes $\Pr(x_t)$. However, $\Pr(x_t)$ can be arbitrarily close to 0. In this case, $\mathcal{L}(x_t)$ can be arbitrarily large. Therefore, the optimization algorithm may spend all the time to maximize this term even though it will increase the loss $\mathcal{L}(x_i)$ for those true negative events x_i. As a result, the learned model may not be effective at all.

Consider the following example. Assume we have a sequence of $x_1, ..., x_{t-1}$ that are labeled as negative, and x_t labeled as positive. The unlearning loss function is thus

$$\sum_{i=1}^{t-1} \mathcal{L}(x_i) - \mathcal{L}(x_t) \tag{15}$$

Without the last term of $-\mathcal{L}(x_t)$, minimizing the objective will reduce all individual terms $\mathcal{L}(x_i)$ for $i = 1, ..., t-1$. This is because every term has a lower bound (i.e., 0). Assume the minimal value of the entire summation is τ; then we know that each individual term is also upper bounded by τ as well. In this case, $\Pr(x_i)$ is lower bounded (e.g., by $\exp(-\tau)$), and thus we can rely on this lower bound to build a detection model.

When $-\mathcal{L}(x_t)$ is included, however, minimizing (15) may only maximize $\mathcal{L}(x_t)$, while making $\mathcal{L}(x_i)$ for any or all $i = 1, ..., t$ arbitrarily large. In this case, although we make sure the model predicts the probability $\Pr(x_t)$ to be arbitrarily close to 0%, the probability $\Pr(x_i)$ can also be close to 0% as well, for other $i \in \{1, ..., t-1\}$. As a result, the model may simply predict anything to be abnormal by minimizing the loss function (15).

To tackle the issue caused by exploding loss, we propose two techniques, namely *bounding loss* and *learning rate shrinking*. We explain them below.

3.2.1 Bounding loss. Clearly, the exploding loss issue is primarily introduced since the term $l_t \mathcal{L}(x_t)$ in (13) can be arbitrarily small when $l_t = -1$. Our idea is to provide a *lower bound* on this term. To this aim, we revise the term as

$$\text{ReLU}\big(BND - l_t \mathcal{L}(x_t)\big) \tag{16}$$

Here, ReLU is the rectifier linear unit [14], i.e., $\text{ReLU}(x) = \mathbf{max}(0, x)$. $BND > 0$ is a pre-determined constant, as a hyper-parameter to the algorithm.

Consider when $l_t = +1$. If $\mathcal{L}(x_t) \leq BND$, then

$$BND - l_t \mathcal{L}(x_t) \geq 0$$

and thus minimizing (16) is equivalent to minimize $-l_t \mathcal{L}(x_t)$ directly. On the other hand, if $\mathcal{L}(x_t) > BND$ already, (16) is always 0, i.e., the global minimum of (16).

When $l_t = -1$, on the other hand, we know that $\mathcal{L}(x_t)$ is lower bounded, and thus we can always choose BND so that $BND + \mathcal{L}(x_t) \geq 0$. In this case, minimizing (16) is equivalent to minimize $\mathcal{L}(x_t)$ as in (13) directly.

The final unlearning loss function is thus revised as

$$\mathcal{L}_{\text{unlearn}}(S) = \sum_t \text{ReLU}\Big(BND - l_t \mathcal{L}(x_t)\Big) \tag{17}$$

Another reason we choose to wrap a ReLU operation around is because it works compatibly with the back-propagation algorithm which computes the gradient, and its gradient computation is implemented in all major deep learning frameworks. Therefore, although we revise the loss function, we can still make sure that our approach is generic enough to be applied to all deep learning-based anomaly detection algorithms.

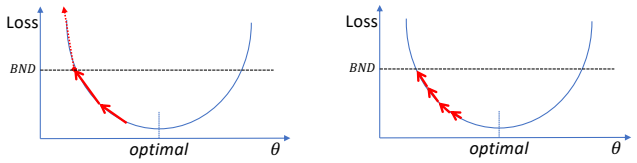
Choosing the bound *BND***.** One remaining issue is on how to choose BND. If it is chosen to be too large, (16) will not serve its design goal. If it is chosen to be too small, the corresponding loss function may not serve its goal to make the model unlearn the false negative.

One straightforward idea is to choose $BND = \tau'$, where τ' is the threshold defined in (10). In doing so, once $\mathcal{L}(x_t) > \tau$, the unlearning algorithm knows to stop. However, this simple idea has the problem that while the optimization algorithm minimizes the overall objective (17), it may stop at somewhere close to the local minimum of (16), while the term (16) is still positive. In this case, the model will still predict the false negative event as negative.

A practical approach is to set BND to be slightly higher than τ'. In fact, in our evaluation, we find that setting $BND = 2\tau'$ works well to solve this issue.

3.2.2 Learning rate shrinking. Choosing a proper learning rate is a practical question for all gradient-based optimization problems. In the standard minimization problem, the gradient size of the loss function $||\nabla_\theta \mathcal{L}||$ will get smaller as the algorithm approaches a local minimum.

In our problem, however, we want to maximize $\mathcal{L}(x_t)$; as a result, the gradient size may increase significantly after a few iterations, and using the same magnitude of the learning rate as the training algorithm will soon decrease the model's performance, even we bound the maximizing term with BND. This scenario is illustrated in Figure 4(a). There have been several algorithms such as RMSProp [37] and Adam [20] proposed to normalize the gradient size. However, our evaluation (see Table 7 in Section 4.3.2) shows that they cannot mitigate this issue.



(a) With a maximum bound (b) Controlled learning rate

Figure 4: Gradient ascent for unlearning.

To mitigate this issue, we need to significantly shrink the learning rate. In our evaluation, we find that choosing a learning rate at the scale of 1% of the initial learning rate used for training can effectively unlearn the given instance without causing drastic change (e.g., introducing significantly more false positives) for normal data.

Also, to make the learning procedure more effective, we evaluate $\mathcal{L}(x_t)$ after each iteration of gradient-based update, and stop as soon as it surpasses the preset bound BND. Figure 4 illustrates this process.

3.3 Preventing catastrophic forgetting

Challenges. In the lifelong learning scenario, when new labeled data is observed, the model is updated incrementally. Therefore, later update may overwrite the model to make it *forget* what has been learned in the past. In the deep learning literature, this is referred to as *catastrophic forgetting* [21].

One easy way to handle the catastrophic forgetting is that in each model update, all past data is put into the training set to compute for the update. Clearly, this approach is not practical, since the training set size is ever growing which makes model updating time inevitably long. In this work, we develop a technique so as to update the model with only a small set of examples. The basic idea is to keep the updated model "close" to the original model so that their performances on previously trained data are close as well.

Solution. Our idea is to update the parameters, from its old value $\theta^\star$ to its new value θ, so that their performance on old data does not change too much. A straightforward idea is to add a regularization term into the unlearning loss to penalize if the model differs too much:

$$\mathcal{L}(x_t) = \text{ReLU}\big(BND - l_t \mathcal{L}(x_t)\big) + \frac{\lambda}{2} \cdot ||\theta - \theta^\star||^2 \tag{18}$$

where λ is a hyper-parameter to control the regularization term. We call λ regularization weight. In our evaluation, however, we find this loss function will restrain the model from unlearning the new false negatives, no matter how we choose λ.

To mitigate this issue, we relax the regularization in the way so that unimportant parameters for previous examples have more freedom to move. To achieve this, we revise (18) as follows

$$\mathcal{L}(x_t) = \text{ReLU}\big(BND - l_t\mathcal{L}(x_t)\big) + \frac{\lambda}{2}\sum_i w_i(\theta_i - \theta_i^\star)^2 \qquad (19)$$

where w_i is a positive constant, θ_i indicates the i-th parameter, and i iterates over all parameters. This generalized form allows the regularization term to take weight in front of $(\theta_i - \theta_i^\star)^2$. Note that (18) is just a specialized form of (19) by taking each $w_i = 1$.

In general, we want to assign w_i to be large if changing $\theta_i^\star$ will cause more catastrophic forgetting, so that $(\theta_i - \theta_i^\star)^2$ can be small when optimizing $w_i(\theta_i - \theta_i^\star)^2$. Thus, we want to measure how likely changing $\theta_i^\star$ will cause more catastrophic forgetting.

To this aim, we maintain an *important memory set* M_t. This is a small set of labeled data that we want the model to remember at time t. Notice that this set is *disjoint* with the new labeled data that our unlearning algorithm wants to update at time t.

The weight w_i can be computed as

$$w_i = \frac{1}{|M_t|}\sum_{x\in M_t}\left(\frac{\partial\mathcal{L}(x)}{\partial\theta_i}\right)^2 \qquad (20)$$

Intuitively, this weight w_i is proportional to the square of the partial gradient of $\partial\mathcal{L}/\partial\theta_i$, which measures how much the loss changes if we move $\theta_i^\star$. Clearly, if the loss does not change much (i.e., close to 0), it is unlikely that changing $\theta_i^\star$ will cause catastrophic forgetting, and thus we can assign a smaller weight for w_i to allow $(\theta_i - \theta_i^\star)^2$ to be bigger. In our evaluation, we will demonstrate that such a rule can effectively mitigate the catastrophic forgetting issue while achieving the goal of unlearning (see Table 9 in Section 4.3.3).

Maintaining the important memory set. The remaining question is how to maintain the important memory set so that (1) it can be small enough to make the algorithm efficient; and (2) it contains enough important data.

To this end, we constitute the important memory set with two components. The first component is a random sub-sample of the validation set (i.e., a small set held out from the optimization algorithm during training to validate whether a model has achieved a good performance) used for training. We can re-sample this set at each iteration to achieve a good coverage of the entire set while keeping it small. The second component includes all manually labeled data in the past. During detection, once the model is updated with a new labeled data, the new data will be added to the important memory set. Note that we expect the second component to be small, so the whole important memory set can remain small over time.

4 EVALUATION

In this section, we evaluate the performance of the proposed lifelong learning mechanism, with two popular deep learning architectures - LSTM and autoencoder, and three real-world tasks: Hadoop system log anomaly detection, Yahoo network traffic anomaly detection, and credit card fraud detection.

We first perform an end-to-end evaluation on all tasks to show the effectiveness of the proposed technique and its wide applicability. In particular, we show that our approach is able to significantly reduce the number of false positives and false negatives, on different security data formats with different deep learning models. Further, we conduct experiments to systematically study the influence of different hyperparameters and the improvements brought by each sub-step in the incremental updating procedure.

4.1 Experiment setup

4.1.1 Datasets. In correspondence to the common security related data formats introduced in Section 2.1, the datasets we cover in evaluation include both discrete and continuous data, where the time dimension may or may not be useful. The dataset statistics are summarized in Table 1, and detailed as below.

Dataset type	Train	Test	
		normal	abnormal
HDFS log [40]	4,855	553,366	16,838
Yahoo network traffic [45]	168	1499	13
Credit card transactions [19]	56,856	227,846	387

Table 1: Dataset statistics.

Hadoop file system (HDFS) log dataset. Available in [40], the HDFS log dataset is generated through running Hadoop map-reduce jobs for 48 hours on 203 Amazon EC2 nodes. It contains various types of anomalies labeled by domain experts, such as "Write exception client give up" and "Receive block exception", details of which could be found in [44]. This dataset contains over 11 million log entries, where each log has a block identifier such as *blk_175691478784117391*. The log dataset could be further grouped into 575,059 block sessions by the block identifier each log has. Different blocks could be understood as concurrent threads, while the logs having the same block identifier are executed sequentially. Note that the labels are only provided at the block session level, i.e., each normal/abnormal label is associated with a block identifier, rather than a log entry. Over the past decade this log dataset has been extensively used for research in system log parsing and anomaly detection [7, 8, 24, 44]. The standard practice is to first map each log entry (e.g., *"Transaction A finished."*) into the corresponding log printing statement that prints out this log message (e.g., *LOG.info("Transaction %s finished.", ...)*. We could use a discrete key to represent each log printing statement, and the vocabulary set is simply the number of log printing statements in the source code. With that, the sequences of system log messages are parsed to sequences of discrete log keys, which further anomaly detection could be performed on. The state-of-the-art anomaly detection results on this dataset are achieved by DeepLog [8], which leverages a LSTM-based neural network for anomaly detection. As in DeepLog [8], our training dataset contains 4,855 normal block sessions, while the test dataset includes 553,366 normal sessions and 16,838 abnormal sessions. For detection, we say a log session is detected as normal if each log entry in this session is detected as normal; and abnormal if at least one log entry in this session is detected as abnormal.

Yahoo network traffic dataset. This dataset includes real-world

internet traffic data by Yahoo, which could be requested following the instructions in [45]. Each data point is a multi-dimensional measurement of traffic received by Yahoo services at a certain time point, which includes timestamp, value, seasonality, and etc.. The anomalies in the dataset have been manually labeled. We use this dataset as a continuous time series sequence data to validate our technique. Roughly the first 10% of the normal time series data are selected for training, and the rest are used as test data for anomaly detection, which contain 1, 499 normal data points and 13 abnormal data points, as shown in Table 1.

Credit card transaction dataset. This dataset is downloaded from Kaggle [19], which contains credit card transactions by European cardholders in September 2013. For purposes such as anonymity, this dataset only preserved the original "Time" and "Amount" features, while transformed all others using principal component analysis (PCA) in the original credit card transaction records. Although the dataset contains a feature "Time", different transaction records do not necessarily correlate in time dimension, since the transactions are made by different people. Although the purchasing history for a single person could be helpful to infer a person's social behavior patterns and further help to do fraud detection, the dataset prevents us from grouping transaction records by account holders because of anonymity etc.. As a result, we treat each transaction record as a standalone data point, and entirely depend on that for anomaly detection, without the data points before or after. The entire dataset has 492 frauds out of 284,807 transactions, with 30 features for each. Following [38], we randomly select 20% of the data, drop the abnormal records and use the rest 56, 856 for training. The rest 80% dataset are used for anomaly detection, which contains 227, 846 normal records and 387 abnormal ones.

4.1.2 Models and updating details. We choose LSTM to handle time-sensitive data, and autoencoder to handle time-insensitive data. To evaluate the effectiveness of the proposed approach, for each dataset and deep learning architecture being evaluated on, we first train an anomaly detection model on training data until it gets the finest anomaly detection results on test dataset. We refer to this initially trained model as BASELINE, and the model being incrementally updated as UNLEARN. Most previous works would simply use BASELINE for anomaly detection with real-time data points, and some are incorporated with naive continuous training using new *normal* data [8, 26]. However, none of the previous works are able to incrementally update the model with reported false negatives, or take actions to prevent forgetting of previously updated instances.

To show the improvement of the proposed incremental updating mechanism over initially trained model, we use BASELINE as a baseline, and a starting point for incremental updating in detection. We assume there are domain expert feedback on whether a data point is correctly detected, and use the feedback to continuously improve our model. A false negative feedback indicates that an anomalous data point is detected as normal, which we need to unlearn from the model; while a false positive indicates that a normal data point is not being properly learned by the model previously, and will be used for incremental relearning.

For all experiments, we assume a real-world data streaming scenario where the model makes a decision on each newly generated data point. We simulate the data stream using the test data. Each sample in the test data arrives at the server by their timestamp, and our server makes a decision on whether or not they are malicious. We also assume a domain expert will report if a decision is wrong, which is used to update the model immediately. The model is improved overtime and the wrong decisions are accumulated progressively as the number of false positives/negatives.

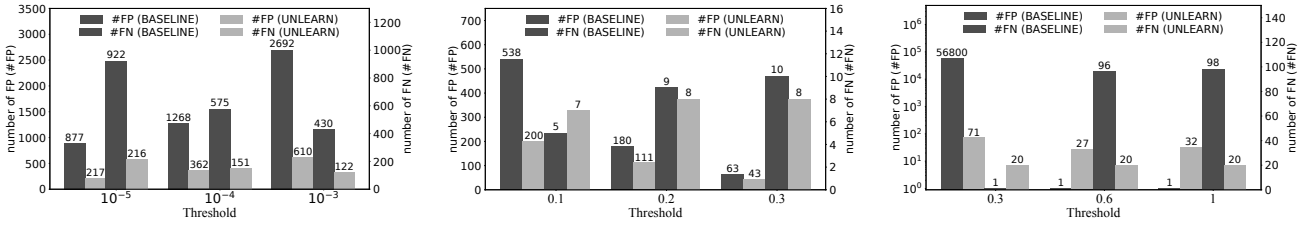
4.1.3 Evaluation criteria. Our goal of incremental updating is to reduce both false positives and false negatives. Therefore, we use the total number of false positives and false negatives as the evaluation metrics, which are denoted as #FP and #FN respectively. Unless otherwise mentioned, we count the number of #FP and #FN in a streaming fashion as well: when a new data point from the test data is seen in the stream, we increment the number of #FP or #FN if the domain expert reports our decision is wrong. Intuitively, if the proposed mechanism is able to effectively unlearn/relearn a data point that is reported to be falsely labeled, the model will not make the same mistake in future when encountering similar data points, so the total #FP and #FN will be reduced. Moreover, smaller #FP and #FN indicate more false positives and false negatives being reduced, and thus the more effective of the method. In some experiments F-1 score is also used as an overall measurement, the calculation of which could be found in [41].

4.2 End-to-end performance

In this section, we focus on presenting the overall effectiveness of the proposed method. For each dataset in Table 1, we first explain how anomaly detection is achieved utilizing existing deep neural networks, and then show the overall improvement with incremental updating by comparing #FP and #FN metrics.

4.2.1 LSTM anomaly detection on discrete sequence data. As explained in Section 4.1.1, HDFS log dataset is first parsed into a sequence of log keys, e.g., *C, A, B, C*, where *C* may represent a log entry printed by code *LOG.info("Transaction %s finished.", ...)*. Also, time dimension matters in the resulted discrete sequence data, since the ordering of system logs reveals the source code execution paths. Our baseline model is DeepLog, which is an anomaly detection model that achieves state-of-the-art anomaly detection results on HDFS dataset. The anomaly detection method on log key sequence is as described in Section 2.3, which utilizes a LSTM model, to take a fixed length of history key sequence as input, and output a probability distribution on the next key that may appear. A data point (i.e., a log key) is detected as abnormal if it has a predicted probability less than some threshold. Consequently, a false negative means that an abnormal log key has a predicted probability higher than the threshold. As a result, our goal of unlearning would be to reduce the predicted probability for the reported false negatives given its history input, and relearning is to increase the probability of the reported false positives.

Figure 5(a) shows the comparison of the initially trained model with and without incremental updating. Note that there are two y axes with different scales, where the left one indicates the number of false positives #FP, and the right axis shows the number of false negatives #FN. We select three different thresholds to showcase the effectiveness of our method. For BASELINE, #FP grows with



(a) HDFS log (b) Yahoo network traffic (c) Credit card transaction

Figure 5: Comparison of #FP and #FN between BASELINE and UNLEARN.

the increase of the thresholds, along with a decrease on #FN. In all cases, the UNLEARN with online incremental update is able to significantly reduce both #FP and #FN, compared with BASELINE.

4.2.2 LSTM anomaly detection on continuous sequence data. The Yahoo network traffic dataset introduced in Section 4.1.1 and Table 1 represents a continuous time series dataset. Each data point has 7 numerical features, and the history traffic trends could help to forecast the future ones. Note that the dataset needs to be scaled to prevent learning bias on value range, and we achieve it by simply dividing each value using the max value in the corresponding dimension. For this task, LSTM models could be leveraged for anomaly detection, by checking the error between the real data point and the forecasted one given its time sequence history [25], which we use as BASELINE. The loss function used for model training is mean squared error loss. Meanwhile, the error between the real data point and the model predicted one is also measured by mean squared error, and compared with a threshold to test if the real data point is an anomaly. Consequently, as illustrated in Section 3, the incremental unlearning procedure with given false negatives would be to increase the loss/error between the actual data point and the model prediction, while incremental relearning being the opposite.

The evaluation and comparison results could be found in Figure 5(b). For this dataset, the number of false negatives is extremely low, so false negatives are not unlearned as often. Also, the few false negatives have different types, for example, one in upward trend and another in downward trend in the time series data, so unlearning one does not affect much of another. As a result, the unlearning effect shown in Figure 5(b) is not as significant compared to previous HDFS dataset. Nevertheless, we show that, with three different thresholds, UNLEARN is able to evidently reduce the number of false positives.

4.2.3 Autoencoder anomaly detection on non-time series data. Unlike the datasets used in previous two experiments, the time dimension in the credit card transaction dataset presented in Section 4.1.1 is not necessarily useful for fraud detection, since each transaction record is made by a different person, and it's not possible to group transactions by card holders. Instead, we could simply use the 30 features of each transaction record for fraud detection of that record. More specifically, each transaction record is represented by a vector of 30 dimensions, and anomaly detection could be performed in the vector space to detect the anomalous vectors. Similar to the Yahoo network traffic dataset, each dimension is scaled through dividing each value by the max one in that dimension. Autoencoder as presented in Section 2.3 has been shown effective on this dataset [38], which utilizes normal transaction data to train a model, finds a desirable threshold, and detects anomalies by checking if the input-output error exceeds the threshold for each test data point. We use the implementation in [38] as the baseline model, and incrementally update it with new false positives and false negatives, to improve its performance.

The evaluation results are shown in Figure 3. Note that the left y axis indicating the number of false positives is in log scale. For this dataset, the error scores of normal data and abnormal data are pretty close. As a result, the numbers of false positives and false negatives differ significantly with the presence of different thresholds. In an extreme case where 1/4 of the test data are detected as anomalies (when threshold 0.3), UNLEARN is able to reduce almost all false positives (close to 99.9%), and introduce only 19 false negatives. For the other two thresholds, UNLEARN reduces most of the false negatives, with few false positives introduced.

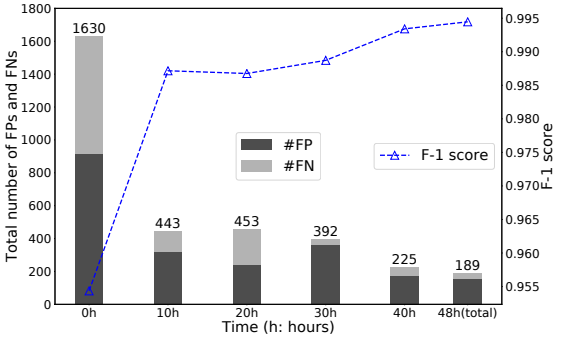


Figure 6: Improvement by UNLEARN overtime (HDFS log).

4.2.4 Improvements by UNLEARN overtime. In this section, we aim to explore the improvements brought by UNLEARN overtime, in terms of improving anomaly detection performance on test data. We use HDFS log dataset considering its large size. We construct the data stream in the same way as above, but the performance metric is calculated in a different way. We test each snapshot model at different time on the full test data to compute the #FP and #FN.

Here, a snapshot model at time H indicates that the model is updated in a streaming fashion, but only test samples with timestamp before H is used. Note that in previous experiments, each *snapshot model* at time H is only used to evaluate the test data right after H. By doing so, we can have a better understanding how snapshot models' performance change over time.

Figure 6 shows the results. The x-axis are the snapshot moment, and the y-axis shows the anomaly detection performance of the updated model. As can be observed, the total number of false positives and false negatives is much higher when there is no update at all (x=0). The model could be significantly improved with only a few updates ((x=10)), and slowly improved as time evolves. Also, the F-1 score improves from around 94.7% at the beginning, to 99.4% in the end.

4.2.5 Comparison of UNLEARN *with retraining.* As previously mentioned, to update the model with a newly labeled instance, a naive alternative solution is to retrain the model with both the new sample and all past training data. In this section, we compare the performance of UNLEARN with retraining, in terms of both efficiency and effectiveness.

However, for the anomaly detection models considered in this paper, only normal data samples could be used for training. As a result, our baseline retraining model is trained on a dataset that contains all past training data and all newly identified false positives in detection, while UNLEARN model is updated with newly labeled false positives and false negatives. Instead of comparing the results whenever a newly labeled sample appears and a model update is made, we only do a final comparison. That is, we assume the last false positive/negative is just reported, so now the retraining dataset contains all false positives and false negatives identified in detection as well as the original training dataset, and the UNLEARN model has finished updating with all newly labeled data. We obtain both models and apply each on entire test dataset to test the utility. The running time for retraining is simply the entire training time. For UNLEARN, we measure the time consumed by each unlearning and relearning step and show the average.

The results are shown in Table 2. The retraining process took 1736.42 seconds, which is not too long due to the relatively small training dataset, but is much longer than UNLEARN. Also, since retraining does not take false negatives into account, the final model produces more false negatives than BASELINE. Instead, UNLEARN which is updated with both false positives and false negatives improved BASELINE performance, and is better than retraining.

Threshold $T_p = 10^{-5}$

	time (seconds)	#FP	#FN	F-1 score
BASELINE	N/A	877	922	0.947
retraining	1736.42	21	2236	0.928
UNLEARN	1.12	157	32	0.994

Table 2: Comparison between UNLEARN **and retraining (HDFS log).**

4.2.6 Comparison of UNLEARN *with evolving clustering approach.* Before the popularity of deep learning, traditional methods such as clustering [43] have been extensively used for anomaly detection [4, 9]. In particular, DBSCAN [10] is a density-based clustering method which groups nearby points into the same cluster recursively, and identifies small clusters having points lower than a threshold as outliers.

To deal with evolving data streams where clusters need to be adapted to new data points in a streaming fashion, multiple improvements have been proposed such as CluStream [1] and DenStream [2], which are based on k-means and DBSCAN respectively [32]. Specifically, DenStream is an advancement over CluStream, and is able to discover new clusters with arbitrary shapes and identify outliers in an evolving data stream.

Although deep learning has outperformed traditional machine learning in many tasks, there haven't been many related works comparing the streaming versions of the two. In this section, we aim to compare the anomaly detection performance of DenStream and UNLEARN, as well as their base non-streaming versions, i.e., DBSCAN and autoencoder. We choose autoencoder to be the baseline for deep learning based anomaly detection instead of LSTM models, because the input data format it accepts is similar to clustering.

Dataset. For the datasets, we were not able to find publicly available datasets used in the streaming clustering papers [1, 2], and we found that the clustering performance on our credit card transaction dataset is extremely poor (F-1 score close to 0), which may not be suitable for clustering methods. To amend this, we use the banknote image dataset [13] which is claimed to be suitable for clustering based anomaly detection. The data are extracted from images taken from genuine and forged banknote-like specimens. The features for each image include variance, skewness and curtosis of the wavelet transforms of the images. There are totally 1372 images, of which 600 are fake. The train-test split statistics for each method is shown in Table 3. DenStream, autoencoder and UNLEARN all need initial normal data points to either form base clusters, or train an initial detection model. DBSCAN is an unsupervised approach which does not need training data, but we make sure that the test dataset for each method is the same for fair comparison.

The results of the four methods are shown in Table 4. For each method, we explore the hyper-parameters and report the best performed ones. As noted in the table, DenStream effectively improves the anomaly detection performance (F-1 score) over DBSCAN. The baseline deep learning alternative is comparable to DenStream, while its streaming version UNLEARN performs the best among all methods.

Algorithm	Train	Test	
		normal	abnormal
DBSCAN	N/A	586	600
DenStream/Autoencoder/UNLEARN	146	586	600

Table 3: Banknote dataset statistics.

4.3 Sub-components analysis

In this section, we divide the steps used in incremental updating and illustrate how each step helps to effectively unlearn/relearn the given instances, and more importantly, in a controlled manner.

	Clustering		Autoencoder	
	DBSCAN	DenStream	BASELINE	UNLEARN
#FP	20	610	43	122
#FN	535	300	383	0
F-1 score	0.402	0.611	0.624	0.882

Table 4: Comparison between UNLEARN and evolving clustering on Banknote dataset.

4.3.1 The effectiveness of loss bound BND. In this section, we show the necessity of applying a maximum loss bound *BND* for unlearning, and how it could potentially mitigate the explosive increase of false positives.

We first conduct a set of experiments to compare the system performance with different *BND* values. In these experiments, we only perform unlearning with given false negatives, but no action is taken for false positives. For each *BND* value, we analyze the number of false negatives being reduced and the number of false positives being incurred. An overly low *BND* value would restrain the unlearning of false negatives, while a *BND* value that is too high may explosively increase false positives. An appropriate *BND* value should be able to significantly reduce false negatives without incurring too many new false positives. The results are shown in Table 5. As in the table, the larger *BND* is, the more effective of the false negative unlearning, but the more false positives may occur.

Threshold $T_p = 10^{-5}$

	BASELINE	UNLEARN with only unlearning (*BND*=)				
		5	8	10	12	15
#FP	877	855	1257	2593	2616	39828
#FN	922	1049	114	8	0	0

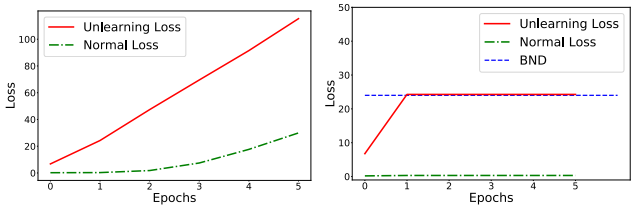
Table 5: #FP and #FN change with different loss bounds *BND* for unlearning (HDFS log). Relearning is disabled.

To understand why it happens, we plot the loss change of a particular instance being unlearned, as well as the average loss of normal examples in Figure 7. The two experiments plotted in the two sub-figures have the same learning rate.

The left figure shows the loss change without any bound, while the right figure indicates the loss change with maximum loss bound *BND* applied. Without any bound, the unlearning loss keeps increasing, resulting in a potentially exponential change on average normal loss. The resulted model could incur significantly more false positives, because of the loss increase in normal examples.

In the right figure, when maximum loss bound *BND* is applied, the loss of the unlearning instance simply increases until *BND*, before causing visible effect on average normal loss. Note that a low *BND* may terminate the unlearning process of an instance before it's fully unlearned (i.e., detected as anomaly), which is acceptable since it's better to cause many more false positives with the unlearning of a single instance.

4.3.2 The influence of learning rate η. As described in Section 3.2.2, the learning rate η has a significant effect on unlearning progress, especially for cross-entropy loss, where the absolute value



(a) without *BND* (b) with *BND*

Figure 7: Comparison of loss change with/out *BND* applied (HDFS log).

of gradient could be infinitely large. This set of experiments focus on evaluating the influence of different η, hoping to suggest the best practice for finding desirable learning rates.

As in Section 4.3.1, we first conduct experiments to measure the influence of different learning rates with only incremental unlearning enabled. Both the number of false negatives and the number of false positives are being analyzed for each learning rate. Ideally, the unlearning procedure should reduce as many false negatives as possible, and incur as few new false positives as possible.

Table 6 shows the number of FPs and FNs for both BASELINE, and UNLEARN. This is to show the increase of #FP without any relearning step. The learning rate used for training is $\eta = 0.1$. As in Table 6, in all cases, the majority of false negatives are reduced. #FP has modest increase when learning rates are appropriate. However, when the learning rate is big (comparable to what's used in training), the increase of #FP could be disastrous, as shown by the statistics of $\eta = 10^{-2}$ and $\eta = 10^{-1}$. Although low learning rate would generally be safe, it may cause the unlearning progress to be slow, as indicated in the "Average updates" statistics, which is the average number of updates for each instance to be successfully unlearned.

To better understand how the learning rate difference would affect the loss of normal examples while increasing the loss for abnormal examples, we further plot the loss change of both the example to unlearn, and the average loss of normal examples. Figure 8 shows the result. In the left figure when learning rate is big, the unlearning loss increases rapidly, along with a sudden change in average normal loss, causing a big increase in #FP. In comparison, the right figure shows the result of loss change with a low learning rate. The loss of the instance to be unlearned increases slowly until being unlearned, while the average normal loss stays low, without visible change.

In Table 6, the optimizer being used is gradient descent optimizer, where the learning rate is consistent in the whole training process. We also conduct experiments on other optimizers where the learning rate would automatically decay with the training progress. Specifically, we choose Adam and RMSProp optimizers in model training, with an initial learning rate of 10^{-3}. As shown in Table 7, the observations are similar - choosing a learning rate that is 1% of the training one is safe and moderate for unlearning.

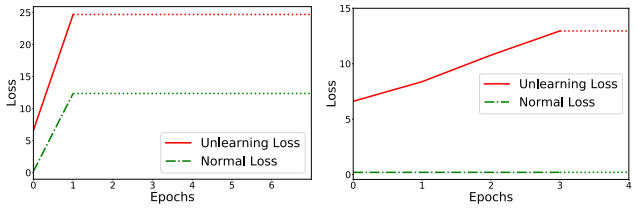
For relearning, the influence of learning rate is smaller, since the learning goal is to minimize the loss, and the gradient becomes smaller itself when close to converge. Table 8 shows the increase of

#FN while reducing #FP with only incremental relearning enabled. Although relearning is more tolerant to learning rate difference, small learning rate still provides better performance in terms of reducing #FP and in the meantime controlling the increase of #FN. Also, note that the "Average updates" statistics increase when learning rate becomes larger, which could be due to that the optimal minimal loss is hard to approach with a big step size.

Threshold $T_p = 10^{-4}$

	BASELINE	UNLEARN with only unlearning (η=)			
		2×10^{-4}	10^{-3}	0.01	0.1
#FP	1268	3061	3427	449525	552654
#FN	575	12	5	2	1
Average updates		1.75	1.2	1	1

Table 6: #FP and #FN change with different learning rates for unlearning (HDFS log). Relearning is disabled.



(a) $\eta = 10^{-1}$ (b) $\eta = 5 \times 10^{-4}$

Figure 8: Loss change of the example being unlearned, and the average loss change of normal examples (HDFS log).

Threshold $T_p = 10^{-4}$

	BASELINE	Adam (η=)			RMSProp (η=)		
		10^{-5}	10^{-4}	10^{-3}	10^{-5}	10^{-4}	10^{-3}
#FP	1268	2841	8859	552654	5605	181347	552654
#FN	575	25	4	1	9	2	1
Average updates		16.08	5.5	1	8	3.5	1

Table 7: #FP and #FN with different optimizers and learning rates for unlearning (HDFS log). Relearning is disabled.

Threshold $T_p = 10^{-3}$

	BASELINE	UNLEARN with only relearning (η=)			
		10^{-4}	10^{-3}	10^{-2}	10^{-1}
#FP	2692	249	140	86	7
#FN	430	1951	2015	2675	2752
Average updates		5.1	1.6	1.6	2.7

Table 8: Number of FPs and FNs with the change of learning rate for relearning (HDFS log). Unlearning is disabled.

4.3.3 The effectiveness of regularization weight λ. Finally, we evaluate the effect of regularization weight λ, which reflects the amount of previous information to remember. The larger λ is, the more previous information to remember, while $\lambda = 0$ indicates basic incremental updating without prevention from forgetting of previous learned/unlearned examples. However, an overly big λ could also restrain the updating of current examples. The results are shown in Table 9. Compared with the baseline case when $\lambda = 0$, #FP drops with increasing λ, showing that fewer previously learned examples are forgotten. Also, when λ is too big (e.g., 5×10^5), the increase of #FN and decrease of #FP shows that the unlearning of current examples is restrained, while more weight is being put on not forgetting previously learned examples. Nevertheless, Table 9 shows that a wide range of λ suffices to improve the results.

	BASELINE	λ for unlearning				
		0	5×10^2	5×10^3	5×10^4	5×10^5
#FP	22	162	165	143	95	32
#FN	90	44	43	40	43	79

Table 9: Number of FPs and FNs with the change of regularization weight λ (credit card transaction data).

The importance of applying regularization. To show the importance of applying λ to prevent catastrophic forgetting, we apply the following three models on *training dataset* to compare the performance: 1) BASELINE model which is not updated in testing; 2) UNLEARN model with λ regularization applied; and 3) UNLEARN model without regularization ($\lambda = 0$). Table 10 shows the results. Note that the training dataset only contains normal samples, so only false positives are incurred in the process. The BASELINE model without any update is exactly the same model trained on the dataset, and thus performs the best. UNLEARN with λ regularization slightly increased #FP on training dataset. However, UNLEARN without regularization ($\lambda = 0$) may explosively increase #FP. The comparison results show that applying regularization, i.e., the second term in Equation 19 effectively prevents catastrophic forgetting.

Threshold	BASELINE	UNLEARN	
		$\lambda = 5 \times 10^3$	$\lambda = 0$ (no regularization)
10^{-5}	0	4	16
10^{-4}	4	37	3026

Table 10: #FP on HDFS training dataset.

4.3.4 The influence of important memory set M_t. As introduced in Section 3.3, UNLEARN prevents catastrophic forgetting by limiting the update on model weights that are important to an *important memory set* M_t. We vary the portions of the two components in M_t, namely, a random sub-sample of the validation set used for training, and the set of all newly labeled samples in testing, to evaluate the influence of the two components. For this experiment, we use HDFS dataset. The first component is a 5% hold-out validation dataset in training which we call as validData, and the second

component is referred as newData. M_t is composed with α portion of $validData$ and β portion of $newData$. Table 11 shows the results with varying α and β. As can be observed, newly labeled data in test is more effective in improving the performance on test data, while combining with valid dataset gives the best performance.

Threshold $T_p = 10^{-5}$

BASELINE	#FP	#FN	α	β	#FP	#FN	α	β	#FP	#FN	α	β	#FP	#FN
	877	922	0	1	331	149	1	0	378	732	1	1	157	32

Table 11: #FP and #FN with $M_t = (\alpha \cdot validData) \bigcup (\beta \cdot newData)$ (HDFS log).

The importance of using M_t. As in Section 3.3, if not having M_t, an alternative is to use Equation 18 which simply restrains the update on all existing model parameters. We compare the updated model performance between using Equation 18 and using 19, on HDFS test dataset. With an anomaly detection threshold of $T_p = 10^{-5}$, the #FP and #FN for Equation 18 is 162 and 782, but for Equation 19 is 157 and 32 respectively. Applying memory set and calculating the weights important to it effectively improves the unlearning performance.

5 DISCUSSION

A major contribution of our work is a novel approach for unlearning. Although it is often accompanied with more false positives, we note that in system anomaly detection, the merit of fewer false negatives in fact worth the cost of more false positives. That is because reported false positives could be further checked by system admin, and then fed into the model for incremental learning. However, a false negative may never be found out, until a more disastrous event occurs due to the un-discovery of it.

There are previous works that explore specific models like Bayesian ones to adapt with new data [2, 11], which have also shown effective in anomaly detection [22, 32]. In contrast, the proposed UNLEARN scheme is achieved by manipulating the loss functions, which is a general approach for deep learning based methods that involve gradient descent steps.

Since the model is being incrementally updated in anomaly detection process, it is natural to think whether an attacker would try to inject adversarial examples to pollute the model. However, this is not quite easy for the usage scenario of UNLEARN, where the newly labeled data for update are proposed by the same party who uses the model for detection, and thus have no incentive in breaking the model.

In this paper, we only suggest to use a lower learning rate compared with the one used for training. However, studying optimizers that could adaptively change the learning rate for incremental updating where only micro adjustments are needed would be an interesting research direction to explore.

6 RELATED WORK

Deep learning, and recurrent neural networks (RNNs) in particular, have been found to be extremely effective in detecting and predicting events related to security, for example, to identify functions in code binaries [35]. RNNs and Echo State Networks have also been used for unsupervised feature extraction, which expresses great effectiveness for malware detection [29]. Similar to DeepLog [8], Tiresias [34] uses stacked RNNs to predict the next security event to happen based on a history of events.

Anomaly detection is an essential component for security. Compared with supervised anomaly detection which requires labeled anomalies [46], and unsupervised anomaly detection which assumes outliers to be rare [31], zero-positive anomaly detection has recently gained much interest in that it can fully enjoy the benefits brought by deep neural networks, while having no assumption on the anomalies to detect. AutoPerf [15] trains an autoencoder network utilizing hardware performance counters collected from normal system execution, to detect hardware performance problems. Kitsune [26] leverages an ensemble of autoencoders to detect network anomalies, which also requires training data to be clean (normal). Previous works have also explored time series dimension for system anomaly detection, which is important since real-time system status and actions are inevitably affected by previous system events. For this series of work, RNNs such as LSTM are extensively used, because of the ability to model sequence information. Anomaly detection on continuous time series data is mostly done by learning a LSTM model for forecasting, and then compute the error score by calculating the difference between LSTM-generated data point and the real system data point [6, 23, 25, 36]. For discrete time series data [5], DeepLog [8] uses LSTM to train a supervised classifier utilizing normal system log data, to predict the next system log and compare with the actual system generated one. Similar ideas could also be applied to system command sequences and function API call sequences [5]. In summary, recent works on zero-positive anomaly detection mostly used autoencoder for non-time series data analysis, and LSTM for time series data analysis, or different variations of them such as ensembles.

Although many previous works have validated the flexibility and effectiveness of zero-positive anomaly detection through deep learning, an essential part that's not fully studied is the incremental update of the initially trained models. Since only normal data are used for training, previous works mentioning incremental update either use all encountered normal data to update the model [15, 26], or only utilize false positives [8]. None of the previous works have mentioned how to update the zero-positive model to deal with reported false negatives. Moreover, as suggested in Section 4.3.2, naive incremental learning introduces many false negatives, while the restraint of which is not explored.

The concept of machine unlearning has been proposed and studied before. However, we note that the scenarios presented in previous work significantly differ with the goal of updating zero-positive anomaly detection models, and thus is hardly possible to be directly applied. Cao and Yang systematically studied machine unlearning in [3]. They transfer machine learning from raw data to summation forms. In terms of unlearning, only the summations involving the data to forget is recalculated, and the machine learning model is updated on the new summation forms, which is asymptotically faster than retraining from raw data. Although this work has its potential to be extended to deep learning, the method is proposed for unlearning of the data previously used for training, instead of unlearning a testing sample. Neural Cleanse [39] also mentioned the concept of unlearning, to fix a model that is previously trained

by trojan attack data samples. The proposed method is to retrain the model with the trojan attack data samples and corresponding correct labels, hoping to overwrite the attack labels previously used for training. However, in zero-positive anomaly detection, the alternative correct label for a sample to unlearn may not be known. Moreover, only replacing the label having maximum predicted probability does not guarantee to reduce the predicted probability of a false negative, which is required for it to be detected as abnormal.

Finally, lifelong learning has been previously proposed and studied for deep learning in general [21, 28, 33]. The problem to solve is to train the same machine learning model for multiple tasks, which may not even have the same set of labels, such as digits classification and face recognition. For each training task, the corresponding dataset may be trained over for multiple iterations, unlike our setting which is entirely online incremental updating. Inspired by the usage of fisher information matrix in [21], we leverage a similar idea to restrain the parameter change to a small memory set which contains previously updated examples.

7 CONCLUSION

In this paper, we study the problem of lifelong learning for deep learning based anomaly detection. We first summarize the general problem of zero-positive anomaly detection, and how previous work address it by utilizing various generative models, for example, autoencoder models for non-time series data and LSTM models for time series data. Leveraging the fact that all such models aim to minimize the loss for normal data in training process, we propose a new objective function that aims to maximize the loss to unlearn reported abnormal samples. Moreover, we propose to add a maximum bound to control the loss increase, and applying a regularization term to prevent catastrophic forgetting while updating the model with new test samples. The newly proposed loss objective could also be generalized to the incremental learning of new negative examples, and similar observations (e.g., shrinked learning rates, regularization term) are also applicable to incur fewer false negatives. Finally, we use three real-world datasets utilizing different deep neural architectures for anomaly detection, to show the wide applicability of our proposed incremental updating technique.

ACKNOWLEDGMENTS

The authors appreciate the valuable comments provided by the anonymous reviewers. We thank Ruoxi Jia from UC Berkeley and Idan Amit from Palo Alto Networks for the insightful discussions. This work was supported by the CLTC (Center for Long-Term Cybersecurity) at UC Berkeley.

REFERENCES

[1] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. 2003. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*. VLDB Endowment, 81–92.
[2] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. 2006. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*. SIAM, 328–339.
[3] Yinzhi Cao and Junfeng Yang. 2015. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 463–480.
[4] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41, 3 (2009), 15.
[5] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2012. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering* 24, 5 (2012), 823–839.
[6] Sucheta Chauhan and Lovekesh Vig. 2015. Anomaly detection in ECG time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 1–7.
[7] Min Du and Feifei Li. 2016. Spell: Streaming parsing of system event logs. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 859–864.
[8] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1285–1298.
[9] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. 2002. A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*. Springer, 77–101.
[10] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise.. In *Kdd*, Vol. 96. 226–231.
[11] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2004. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *2004 Conference on Computer Vision and Pattern Recognition Workshop*. IEEE, 178–178.
[12] Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences* 3, 4 (1999), 128–135.
[13] Stefan Glock, Eugen Gillich, Johannes Schaede, and Volker Lohweg. 2009. Feature extraction algorithm for banknote textures based on incomplete shift invariant wavelet packet transform. In *Joint Pattern Recognition Symposium*. Springer, 422–431.
[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.
[15] Justin Gottschlich, Abdullah Muzahid, et al. 2017. AutoPerf: A Generalized Zero-Positive Learning System to Detect Software Performance Anomalies. *arXiv preprint arXiv:1709.07536* (2017).
[16] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6645–6649.
[17] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[18] Ling Huang, XuanLong Nguyen, Minos Garofalakis, Michael I Jordan, Anthony Joseph, and Nina Taft. 2007. In-network PCA and anomaly detection. In *Advances in Neural Information Processing Systems*. 617–624.
[19] Kaggle. 2013. Credit Card Fraud Detection. https://www.kaggle.com/mlg-ulb/creditcardfraud [Online; accessed 19-April-2019].
[20] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[21] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.
[22] Christopher Kruegel, Darren Mutz, William Robertson, and Fredrik Valeur. 2003. Bayesian event classification for intrusion detection. In *19th Annual Computer Security Applications Conference, 2003. Proceedings*. IEEE, 14–23.
[23] Tae Jun Lee, Justin Gottschlich, Nesime Tatbul, Eric Metcalf, and Stan Zdonik. 2018. Greenhouse: A Zero-Positive Machine Learning System for Time-Series Anomaly Detection. *arXiv preprint arXiv:1801.03168* (2018).
[24] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Ye Xu, and Jiang Li. 2010. Mining Invariants from Console Logs for System Problem Detection.. In *USENIX Annual Technical Conference*. 1–14.
[25] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long short term memory networks for anomaly detection in time series. In *Proceedings*. Presses universitaires de Louvain, 89.
[26] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).
[27] Andrew Y Ng and Michael I Jordan. 2002. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in neural information processing systems*. 841–848.
[28] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural Networks* (2019).
[29] Razvan Pascanu, Jack W Stokes, Hermineh Sanossian, Mady Marinescu, and Anil Thomas. 2015. Malware classification with recurrent networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 1916–1920.
[30] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. 1988. Learning representations by back-propagating errors. *Cognitive modeling* 5, 3 (1988), 1.
[31] Mayu Sakurada and Takehisa Yairi. 2014. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. ACM, 4.

[32] Mahsa Salehi and Lida Rashidi. 2018. A survey on anomaly detection in evolving data:[with application to forest fire risk prediction. *ACM SIGKDD Explorations Newsletter* 20, 1 (2018), 13–23.
[33] Joan Serrà, Dídac Surís, Marius Miron, and Alexandros Karatzoglou. 2018. Overcoming catastrophic forgetting with hard attention to the task. *arXiv preprint arXiv:1801.01423* (2018).
[34] Yun Shen, Enrico Mariconti, Pierre Antoine Vervier, and Gianluca Stringhini. 2018. Tiresias: Predicting security events through deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 592–605.
[35] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi. 2015. Recognizing functions in binaries with neural networks. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 611–626.
[36] Adrian Taylor, Sylvain Leblanc, and Nathalie Japkowicz. 2016. Anomaly detection in automobile control network data with long short-term memory networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 130–139.
[37] T. Tieleman and G. Hinton. 2012. Lecture 6.5 - RMSProp, COURSERA: Neural Networks for Machine Learning. *Technical report* (2012).
[38] Venelin Valkov. 2017. Credit Card Fraud Detection using Autoencoders in Keras. https://github.com/curiousily/Credit-Card-Fraud-Detection-using-Autoencoders-in-Keras/blob/master/fraud_detection.ipynb [Online; accessed 19-April-2019].
[39] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao. [n.d.]. Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks. In *Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks*. IEEE, 0.
[40] Wei Xu. 2009. HDFS Log Dataset. http://iiis.tsinghua.edu.cn/~weixu/sospdata.html [Online; accessed 19-April-2019].
[41] Wikipedia contributors. 2019. F1 score — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=F1_score&oldid=911716685. [Online; accessed 31-August-2019].
[42] Wikipedia contributors. 2019. Zero-day (computing) — Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Zero-day_(computing)&oldid=895202836. [Online; accessed 16-May-2019].
[43] Rui Xu and Donald C Wunsch. 2005. Survey of clustering algorithms. (2005).
[44] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. 2009. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 117–132.
[45] Yahoo Research. 2015. A Benchmark Dataset for Time Series Anomaly Detection. https://yahooresearch.tumblr.com/post/114590420346/a-benchmark-dataset-for-time-series-anomaly [Online; accessed 19-April-2019].
[46] Ke Zhang, Jianwu Xu, Martin Renqiang Min, Guofei Jiang, Konstantinos Pelechrinis, and Hui Zhang. 2016. Automated IT system failure prediction: A deep learning approach. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 1291–1300.
[47] Chong Zhou and Randy C Paffenroth. 2017. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 665–674.
[48] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Daeki Cho, and Haifeng Chen. 2018. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. (2018).