Received: 31 October 2016 | Revised: 20 April 2017 | Accepted: 13 August 2017

DOI: 10.1111/coin.12146

ORIGINAL ARTICLE

WILEY Computational Intelligence

Threaded ensembles of autoencoders for stream learning

Yue Dong[1] | Nathalie Japkowicz[2]

[1]Department of Computer Science, McGill University, Montreal, QC, Canada

[2]Department of Computer Science, American University, Washington, DC, USA

Correspondence
Yue Dong, Department of Computer Science, McGill University, Montreal, QC, Canada.
Email: yue.dong2@mail.mcgill.ca

Abstract

Anomaly detection in streaming data is an important problem in numerous application domains. Most existing model-based approaches to stream learning are based on decision trees due to their fast construction speed. This paper introduces *streaming autoencoder* (SA), a fast and novel anomaly detection algorithm based on ensembles of neural networks for evolving data streams. It is a one-class learner, which only requires data from the positive class for training and is accurate even when anomalous training data are rare. It features an ensemble of threaded autoencoders with continuous learning capacity. Furthermore, the SA uses a 2-step detection mechanism to ensure that real anomalies are detected with low false-positive rates. The method is highly efficient because it processes data streams in parallel with multithreads and alternating buffers. Our analysis shows that SA has a linear runtime and requires constant memory space. Empirical comparisons to the state-of-the-art methods on multiple benchmark data sets demonstrate that the proposed method detects anomalies efficiently with fewer false alarms.

KEYWORDS

anomaly detection, autoencoders, ensemble learning, multilayer perceptrons, one-class learning, stream mining

1 | INTRODUCTION

Anomaly detection on streaming data is a research area of increasing importance.[1] Anomalies or outliers are rare events that are statistically different from normal instances. Anomaly

detection on streaming data is the problem of identifying outliers from normal data in nonstatic data streams.

According to Tan et al[1], there are many special characteristics in detecting anomalies from data streams. First, data streams arrive continuously without stopping; thus, any offline batch learning algorithm that requires storing the entire stream for analysis is not suitable for stream learning. Second, data streams for anomaly detection contain mostly normal data instances (positive instances) as the size of the anomaly class might be small compared with the normal class. In this case, one-class classifiers are needed when the anomalies are not available for training. Third, streaming data usually come with concept drift where the distribution of data changes. Therefore, the model must learn adaptively and continuously.

When anomaly instances exist in the training data, supervised learning algorithms such as very fast decision trees (VFDTs) and its variations[2-5] can be used for streaming anomaly detection. These algorithms are based on the dynamic decision trees, which are very inefficient and slow at dealing with concept drift. Some other authors[1,6] proposed fast one-class learners which randomly built decision trees in advance without data. These algorithms' performance highly depend on the random trees in the ensemble and the last subsample in data streams. Moreover, these algorithms have high false-positive rates in anomaly detection because they make judgments based on a small subsample from the last window in the data stream. There have been other streaming anomaly detectors based on incremental neural networks (NNs). These algorithms suffer from the convergence problems and do not predict well before substantial amounts of data are passed.

In response to the aforementioned problems, this paper proposes a novel, fast, and scalable one-class algorithm, ie, streaming autoencoders (SAs), for detecting anomalies in data streams. The proposed method, which deals with concept drift through adaptive learning, has low false-positive detection rates with a 2-step checking scheme and performs well early on with little data required.

Streaming autoencoder is based on ensembles of autoencoders with multithreads and mini-batches. Streaming autoencoder learns incrementally with mini-batches from the streaming data. Afterward, the trained model is used to classify outliers with continuous learning; thus the positive (normal) testing data predicted by the model can be used to update the model. This provides an anytime learner with the ability of adapting to the evolving data. In particular, it can still perform well early on when there is not much data. It is also important to note that when large volumes of data arrive, the technique will leverage a parallel computing model with multithreads to distribute the computational load. By doing so, we can have a scalable model for detecting anomalies in real time.

In addition, SA uses a 2-step detection mechanism to check that points detected in the first phase of the system are real anomalies. This is achieved by an additional step using adjacent window comparisons to eliminate the possibility of concept drift. When an anomaly is detected, the system automatically compares the data points from 2 different time windows before and after the anomaly occurs. As shown later, using the profiles between 2 windows to distinguish anomalies from concept drift in data streams helps reduce the false-positive rate in anomaly detection.

Streaming autoencoders have several advantages over other popular models. First, only one pass of the data is required for the algorithm to learn. Second, it uses fixed amounts of memory no matter how large the data sets are. Third, SA only requires data from the normal class, which could work when a data stream contains very few, if any, anomalies. Fourth, SA has the ability to deal with concept drift as it learns the data continuously with incremental updates. Finally, the model separates streaming data into multiple threads to speed up the training process. An ensemble of autoencoders from those threads also improves the detection accuracy. Empirical comparisons

to the state-of-the-art anomaly detectors in data streams demonstrate that SA detects anomalies efficiently with low false alarm rate.

The following are the main contributions of this paper:

- We introduce a fast and accurate NN-based anomaly detector, ie, SAs, for data stream mining.
- We use an additional checking step on the proposed algorithm to reduce the false-positive rate in anomaly detection.
- Empirical experiments on multiple benchmark data sets demonstrate our method performs favorably in terms of area under curve (AUC) and runtime when compared with other state-of-the-art algorithms.

The rest of this paper is organized as follows. In Section 2, we discuss the related works of stream learning algorithms and their shortcomings. In Section 3, we propose our method for mining streaming data and discuss the advantages of our model over existing models. In Section 4, we describe the data sets and the experimental design. In Section 5, we present the results and compare them with the state-of-the-art stream mining algorithms. In Section 6, we conclude with remarks and discussions.

2 | RELATED WORKS

In this section, we survey previous works in anomaly detection and data stream mining. Since Domingos and Hulten[2] proposed VFDT in 2000, decision tree–based algorithms have been mainstream in data stream mining. We therefore mainly focus on works based on decision trees and point out their deficiencies. In view of these deficiencies, we then look at works that use NNs. Finally, we turn to various methods that have been proposed to improve NN-based algorithms in data stream mining, including the use of mini-batches and 2-step verifications.

2.1 | Anomaly detection

There have been many works proposed in the past for anomaly detection, but most of them do not work on streaming data. The most popular batch anomaly detectors include the algorithms based on classification,[7] clustering,[8] and distance[9] as the traditional machine learning algorithms. A few one-class learners were also proposed to solve class imbalance problems in anomaly detection. Examples are one-class support vector machine[10] and isolation forest.[11] These batch learning methods require loading the entire data set for training.[1] Therefore, they are not suitable for processing streaming data.

Recent works on anomaly detection are more focused on stream learning. Such examples include half-space trees (HSTa) algorithm,[1] randomized space Forest (RS-Forest),[6] ensemble of random cut trees algorithm,[12] and subspace embedding–based methods.[13,14] Most of these algorithms are inspired by HSTa algorithm, where random decision trees were built in advance without data. Once an ensemble of trees with the same lengths are built, the data points from the last window in the data stream are passed to the leaves of each tree, then anomalies are those instances, which lie in the leaves without other data instances.* These algorithms' performances highly depend on the random trees in the ensemble and the last subsample in data streams. More-

*Anomalies can be also identified if the number of data instances in these leaves fall below a threshold.

over, these algorithms have high false-positive rates in anomaly detection because only small subsamples of training data are used. Using a small sample for monitoring will cause more leaves to be empty or near empty, and the data points that fall into these leaves will be identified as anomalies.

2.2 | Dynamic tree-based data stream mining algorithms

Supervised learning algorithms are also used for streaming anomaly detection if the labels from both classes are available during training. For example, VFDTs[2] based on Hoeffding trees or its variations[3-5] are still widely used these days.

These algorithms are based on dynamic trees, whose leaves are built from different chunks of stream data. However, it is not natural to use dynamic decision trees when learning has to be continuous, which it frequently does due to concept drift. When concept drift is present, the decision tree structure is unstable in the setting of stream mining, and it is costly to adjust the trees' shape with new data.

We study VFDT in this paper as an illustration of the disadvantages of dynamic tree-based models because it is representative for dynamic tree-based models. The model is a supervised learning model, which requires the labels from both normal and abnormal classes. This is sometimes infeasible when abnormal instances are rare. However, the biggest problems for VFDT are its slow training time and inability to adjust to concept drift.

Very fast decision tree builds each node of the decision tree based on a small amount of data instances from a data stream. According to Domingos and Hulten,[2] VFDT works as follows: "given a stream of examples, the first ones will be used to choose the root test; once the root attribute is chosen, the succeeding examples will be passed down to the corresponding leaves and used to choose the appropriate attributes there, and so on recursively."

To decide how many data instances are necessary for splitting a node, the Hoeffding bound is used. It works as follows: consider r as a real valued random variable, n as the number of independent observations of r, and R as the range of r, then the mean of r is at least $r_{\text{avg}} - \epsilon$, with probability $1 - \delta$

$$P\left(\bar{r} \geqslant r_{\text{avg}} - \epsilon\right) = 1 - \delta,$$

where

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}.$$

With the Hoeffding bound, VFDT can guarantee that the performance of Hoeffding trees converges to that of a batch learning decision tree.[2] However, this data structure is not "stable" because this convergence only works when there is no concept drift. Subtle changes of the data distribution can cause Hoeffding trees to fail as demonstrated in the HTTP+SMTP data set from our experiment.

To mitigate the issue of concept drift, the same authors introduced concept-adapting VFDTs. Concept-adapting VFDT adapts better to concept drift by monitoring changes of information gain for attributes and generating alternate subtrees when needed. However, it still cannot solve the issue completely because the algorithm only monitors the leaves. Due to this disadvantage of decision tree–based models, other researchers tried to stay away from having to regrow decision trees altogether.

2.3 | NN-based data stream mining algorithms

Unlike algorithms that alter decision trees' structure dynamically as streaming data arrives such as VFDT, NNs naturally handle well the task of incremental learning with the ability of dealing with concept drift. When continuous, high-volume, open-ended data streams come, NNs learn by passing the data in smaller groups (mini-batch learning) or one at a time (online learning) before updating the weights. In this manner, they can learn by seeing each example only once and therefore do not require examples to be stored.

Many researchers have proposed algorithms based on incremental NNs.[15-18] These algorithms pass the data points one by one and update the weights every time after scanning a data point. The shortcoming of these algorithms is that the results are highly dependent on the random initial weights. Moreover, these incremental learning algorithms with stochastic gradient descent (SGD) might never converge if inappropriate learning rates are used.[19]

To avoid the convergence issue of SGD and to have a smaller empirical risk E_n, the mini-batch gradient descent (MB-GD) algorithm with random sampling[20,21] became a popular practice for updating the NNs. However, due to the random sampling, this approach is only used for static data sets, where all the data are accessible. In addition, although MB-GD is faster than gradient descent (GD), it is still significantly slower than SGD.

Streaming autoencoders use both SGD and MB-GD for updating the weights. Therefore, it combines the advantages of both GD algorithms. Each thread of SA is equipped with 2 buffers. The first batch of data in a thread are trained with mini-batch learning in multiple epochs. By doing so, the model will give a reliable prediction at the beginning of the streams since its initial weights will not be as random. The autoencoders are then trained with SGD for faster processing of the streaming data. By dividing the data into multiple threads for parallel processing, implementing buffers for receiving and sending data, and learning with MB-GD followed by SGD, the speed of SA is comparable to tree-based algorithms.

Similar to random tree-based algorithms in 2.1, an issue with SAs is the high false-positive rate in detection.[22] A few papers tackled this problem for NN-based methods by using a 2-step verification scheme.[23,24] These methods maintain some profiles across multiple sensors to distinguish point anomalies from contextual anomalies. These techniques are quite different from the ideas presented in this paper, where the second step verification is added to distinguish point anomalies from the concept drift.

In summary, the model proposed in this paper overcomes the disadvantages of existing NN-based models by the following means: (i) using mini-batch learning on chunks of stream data collected in the buffers to minimize the effect of random initial weights at the beginning of stream learning; (ii) dividing the data into multiple threads and processing them in parallel to speed up the training; (iii) updating weights with SGD for fast processing when more data arrive in the buffers; (iv) forming ensembles of NNs (autoencoders) to improve the prediction accuracy; and (v) adopting a 2-step verification scheme to reduce the false-positive rate in anomaly detection.

3 | THE PROPOSED METHODS

This section provides detailed introduction of the proposed algorithm, ie, SAs. We first introduce the basics of autoencoders. Then, we present the implementation details of SA. Next, we discuss the techniques SA used to deal with concept drift during model training and testing. The major notations used in this section are summarized in Table 1.

TABLE 1 Table of major notations

Notation	Description
$\mathcal{X} = R^d$	Feature space with d dimensions
X	A random variable take values from the feature space $\mathcal{X}$
$\mathbf{x} \in \mathcal{X}$	An instance in feature space $\mathcal{X}$
$\mathbf{x}'$	A reconstruction instance of $\mathbf{x}$
ϕ	A shaping function for autoencoders
ψ	An activation function for autoencoders
$\mathbf{h}$	The latent representation in autoencoders of an instance
$\mathbf{w}$	Weight matrix in an autoencoder
$\mathbf{b}$	A vector of bias in autoencoders
$L(\mathbf{x}, \mathbf{w})$	Loss function of an instance $\mathbf{x}$
$E_n(\mathbf{w})$	The empirical risk for n samples
η	Learning rate for updating autoencoders
$\mathbf{z} = (\mathbf{x}, y)$	A pair of data instance and its corresponding label

3.1 | Autoencoders

An autoencoder is a feedforward NN with an input layer, an output layer, and one or more hidden layers. What distinguishes autoencoders from other types of NNs is that the input layer and the output layer of an autoencoder have the same size. Autoencoders have the goal of reconstructing their own inputs X.

We usually call the connections between the input layer to hidden layers of autoencoders as "encoders." Similarly, we call the connections between hidden layers to the output layer as "decoders." Then, an autoencoder can be defined as a composition of 2 maps ϕ and ψ such that the goal is to minimize the reconstruction errors of inputs X:

$$\phi : \mathcal{X} \to \mathcal{F}$$

$$\psi : \mathcal{F} \to \mathcal{X}$$

$$\arg\min_{\phi,\psi} \|X - (\psi \circ \phi)X\|^2$$

In our study, we consider autoencoders with only one hidden layer. An autoencoder with one hidden layer takes the input $\mathbf{x} \in \mathbb{R}^d$ and maps it to $\mathbf{h} \in \mathbb{R}^p$ (p is the number of hidden units in the hidden layer) with a nonlinearity activation function ψ_1:

$$\mathbf{h} = \psi_1(\mathbf{w}\mathbf{x} + \mathbf{b}).$$

Then, $\mathbf{h}$ is mapped back to $\mathbf{x}'$ through the decoding process, where $\mathbf{x}'$ is called a reconstruction of $\mathbf{x}$:

$$\mathbf{x}' = \psi_2(\mathbf{w}'\mathbf{h} + \mathbf{b}').$$

The reconstruction error can be defined in terms of many ways. The most common ones are the squared error and the cross entropy. Autoencoders are trained to minimize reconstruction errors usually defined as the follows:

$$L(\mathbf{x}, \mathbf{w}) = \mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \left\|\mathbf{x} - \sigma_2\left(\mathbf{w}'\left(\sigma_1(\mathbf{w}\mathbf{x} + \mathbf{b})\right) + \mathbf{b}'\right)\right\|^2.$$

The empirical risk for n samples is the loss $L(\mathbf{x}, \mathbf{w})$ averaged over n samples

$$E_n(\mathbf{w}) = \frac{1}{n}\sum_{i-1}^{n} L(\mathbf{x_i}, \mathbf{w}).$$

To minimize the empirical risk, autoencoders need to be trained through weights adjusting. There are different error propagation mechanisms and GD optimizers for this task. As mentioned earlier in Section 2.3, 2 kinds of GD optimization procedures are used with backpropagation in this paper:

1. MB-GD: parameter updates are carried out after accumulating a fixed size n of training data. The gradient is then computed as the average of losses over this n data points:

$$\begin{aligned}\mathbf{w}(k) &= \mathbf{w}(k-1) - \eta\frac{\partial E_n}{\partial \mathbf{w}}\left(\mathbf{w}(k-1)\right)\\ &= \mathbf{w}(k-1) - \eta\frac{1}{n}\sum_{i=1}^{n}\frac{\partial L}{\partial \mathbf{w}}\left(\mathbf{x_i}, \mathbf{w}(k-1)\right).\end{aligned}$$

2. SGD (online gradient): parameter updates are performed every time a single example is passed through the network $\mathbf{x_t}$:

$$\mathbf{w}(t) = \mathbf{w}(t-1) - \eta\frac{\partial L}{\partial \mathbf{w}}\left(\mathbf{x_t}, \mathbf{w}(t-1)\right),$$

where $\mathbf{w}$ is the weights (including the bias), η is the learning rate, L is the loss function of a particular data point, and E_n is the empirical risk of n data examples.

While constructing an autoencoder, we usually want to compress the data to find a good representation. This is because the autoencoder can become useless by learning an identity function. This only happens when the hidden layers are larger than the input layer. In this case, we usually use denoising autoencoders or autoencoders with dropout to avoid learning the identity function.[25] In our model, we use autoencoders with dropout since it gives the best results throughout our experiments.

3.2 | Streaming autoencoders

The proposed method, ie, streaming autoencoder, is based on ensembles of incremental multi-threaded NNs with buffers. Each NN is built from a particular subspace (one thread) of the data stream. In each thread, we installed 2 buffers for collecting and processing data. A master thread is used to distribute the streaming data. The 2 buffers in each thread are used alternatively for receiving data from the stream and sending the data to NNs for training. This guarantees that each autoencoder is continuously learning without being idle and the data, which have not been processed in time, are stored.

This model is specifically designed for anomaly detection where the percentage of negative instances is very small. Since the autoencoders are only trained on normal instances in the data set, this is an unsupervised learning model. This model has the advantage of dealing with the class imbalance problem, namely, when the abnormal instances are rare or not available for training.[26] Moreover, this model has the ability to reduce the false-positive rate in anomaly detection by using an extra anomaly checker, as explained in Section 3.4.

In this section, we present the implementation details and algorithmic flows of SAs. We first talk about the training process of SA, then turn into the testing process with the 2-step verification scheme. Next, we discuss our algorithm in terms of time and space complexity.

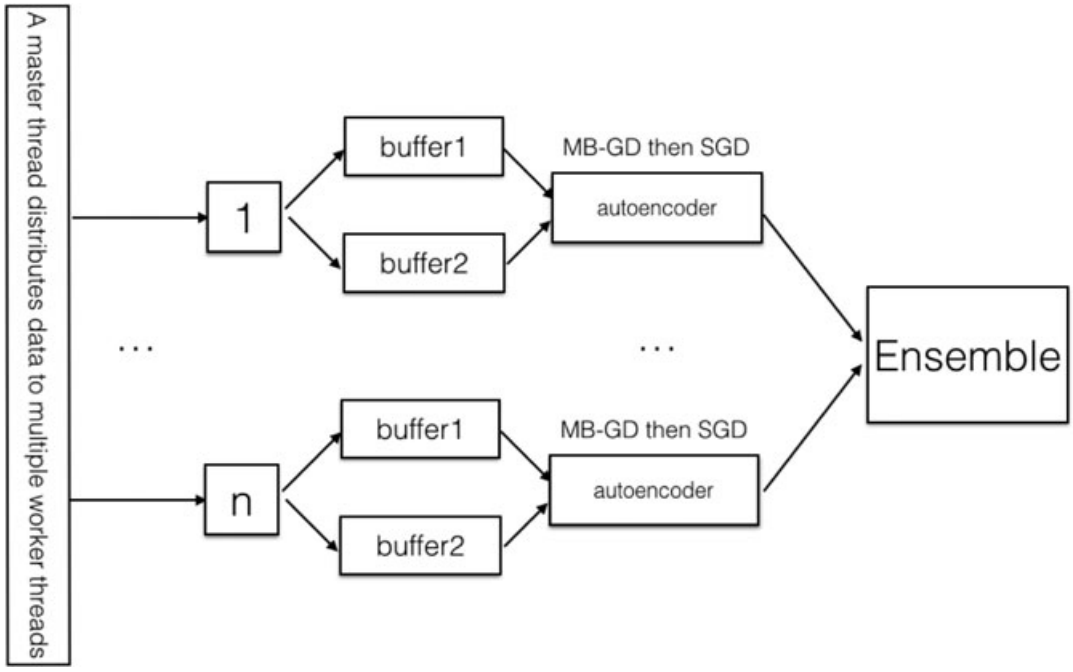

FIGURE 1 An illustration of the training pipeline for the proposed model, ie, streaming autoencoders (SAs). MB-GD, mini-batch gradient descent; SGD, stochastic gradient descent

3.3 | Training SAs

Streaming autoencoder is trained with a combination of MB-GD algorithm and SGD algorithm. To facilitate a better set of initial weights, the first set of data points collected in each buffer are trained in multiple epochs with MB-GD. After that, the models are trained incrementally with SGD on each evolving data thread. The system then operates with multiple threads of incremental autoencoders for anomaly detection.

The general pipeline for training the SA is shown in Figure 1, and the pseudocode is shown in Algorithm 1. SA is roughly trained with the following steps:

1. Initialize M threads and $2M$ buffers for parallel processing of data streams.
2. For each data point x_i in the stream:
   - x_i goes to the first empty buffer it encounters;
   - when the buffer is full, start training the corresponding autoencoder with the data from the buffer.

During training, using MB-GD followed by SGD overcomes the disadvantages of only using SGD. As mentioned in the work of Cun and Yann,[27] "simple batch algorithms converge linearly to the optimum $\mathbf{w}_\mathbf{n}^*$ of the empirical cost E_n. Whereas online algorithms may converge to the general area of the optimum, the optimization proceeds rather slowly during the final convergence phase." In some cases, it might not converge to the optimum weights at all if the learning rate is not chosen properly.

However, according to Cun and Yann,[27] "new examples in the data stream will follow the underlying asymptotic distribution $p(\mathbf{z})$ if the amount of such data points are nearly unlimited."

Algorithm 1 Streaming Autoencoders

Input: S is a sequence of positive (normal) data stream examples,
t is the number of epochs trained at the beginning of each thread,
B is the buffer size,
M is the number of threads
Output: SA is an ensemble of streaming autoencoders
Buffers is a set of buffers with latest data points in the stream stored in the buffers

procedure StreamingAutoencoders(S, B, M)
  Initialize M threads
  In each thread i, initialize 2 buffers B_{2i+1} *and* B_{2i+2} *of size B*
  In each thread i, initialize an autoencoder i with firstBatch[i]=False
  $k = 0$
  for each data sample $x_j \in S$ **do** InsertAndTrainBuffer(x_j, k, t, firstBatch)
  return SA= $(A_1, \ldots, A_M)$; Buffers= $(B_1, \ldots, B_{2M})$

procedure InsertAndTrainBuffer($x_j, k, t,$ firstBatch)
  while BufferIsFull(k) == False **do**
   Insert x_j into buffer k
  if autoencoder $floor((k-1)/2)$ is idle **then**
   if firstBatch=False **then**
  ▷ *mini-bach train autoencoder*$_{floor((k-1)/2)}$ *with data from buffer k* in *t epochs*
    miniBatchTrain($floor((k-1)/2)$, buffer k, t)
    firstBatch=True
    Clear buffer k
   else
  ▷ *online train autoencoder*$_{floor((k-1)/2)}$ *with data from buffer k one by one*
    onlineTrain($floor((k-1)/2)$, buffer k)
    Clear buffer k
  k=(k+1)% n
  return SA= $(A_1, \ldots, A_M)$; Buffers= $(B_1, \ldots, B_{2M})$

In this case, the optimal weights $\mathbf{w}(t)$ are approximated directly and therefore the empirical risk converges to the expected risk E_∞ over the underlying data distribution. Thus, using mini-batch training followed by online learning (as proposed in SA) has the advantage of finding the area of local optimal for the first k samples, then using possibly unlimited data in the stream for $\mathbf{w}$ converge to the optimal $\mathbf{w} *$ where the E_∞ is minimized.

After passing all training data, SA is ready for classification. When a testing instance arrives, the M NNs trained from M threads will vote for the output. If the labels of testing data arrive later, we can use them to update the model.

3.4 | Thresholds for predictions and the 2-step verifications

In order to be used for anomaly detection, the autoencoders are trained on normal (positive) instances, as shown in Algorithm 1. Once trained, they have the ability to recognize abnormal instances in new data (those instances with a large reconstruction error). Therefore, the reconstruction error is used as the measurement to profile the degree of anomaly for autoencoders.

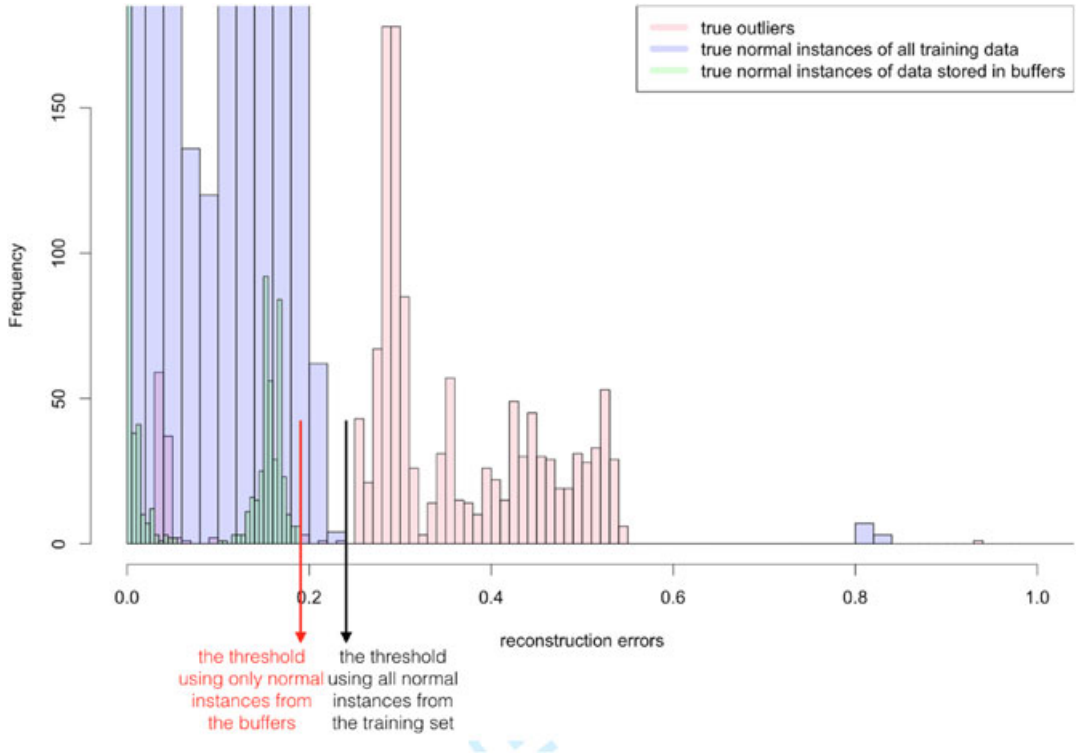

FIGURE 2 An illustration of how to empirically decide the threshold of the reconstruction error for streaming autoencoders [Color figure can be viewed at wileyonlinelibrary.com]

When the labels of training data are available, we could pass them to the model for deciding a better threshold that discriminates normal and abnormal instances. We installed a window of flexible size h to collect all anomalies appearing in the training data streams. These anomalies are then passed to each autoencoder for deciding the threshold of reconstruction error. The threshold of reconstruction error is then set empirically, as discussed in the paper.[26] The basic idea is to find the threshold which gives the best separation of the 2 classes in training data as Figure 2 shows.

Note that using Algorithm 2 with the empirically decided threshold for testing will predict more anomalous instances than a batch learner. This is because only the positive instances stored in the buffer are used to decide the threshold in SA. However, we implemented a checking step (Algorithm 3) where we distinguish anomalies from concept drift and therefore reduce the false-positive rate in detection. Normal test instances (with or without labels) are then queued into the buffers for continuously learning. The pipeline of SA for testing is shown in Figure 3.

3.5 | Time and space complexities

The key operation in SA is updating the weights each time from a subset of data collected in a buffer. The first subset in the data stream are trained with MB-GD and the rest are trained with SGD. Training k data samples from a buffer with MB-GD or SGD uses constant time.[†] Since only

[†]Although the constant time of training k examples with MB-GD and SGD differs.

Algorithm 2 AnomalyDetect

Input: T is a sequence of testing examples,
t is the number of epochs trained at the beginning of each thread,
SA is a set of autoencoders,
w is the window size,
Buffers is a set of buffers with latest data points in the stream stored in the buffers
Thresholds is a vector of thresholds for autoencoders in the ensemble
Output: Results is a list of predicted labels,
SA; Buffers are sets of autoencoders and buffers updated by predicted normal testing data

```
1:  procedure ANOMALYDETECT(T, SA, w, t, Buffers)
2:      for each testing data sample x_i ∈ T do
3:          if pred(x_i) == False then
4:              Results[i]="normal"
5:              InsertAndTrainBuffer(x_i, k, t, firstBatch=False)
6:          else if INDEEDANOMALOUS(x_i, w, Buffers) == 'normal' then
7:              Results[i]="normal"
8:              InsertAndTrainBuffer(x_i, k, t, firstBatch=False)
9:          else
10:             Results[i]="abnormal"
        return SA=(A_1, ..., A_M); Buffers; Results
11:
12:
13: procedure PRED(x,Thresholds)
14:     pass x through each trained autoencoder in the ensemble
15:     for each autoencoder A_i do
16:         if L_i(x, w) > Thresholds[i] then
17:             pred[i] == "abnormal"
18:         else
19:             pred[i]== "normal"
        return mode(pred)
```

the first subset of data in the stream are trained with MB-GD in multiple epochs, SA can be trained very fast.

The updating that uses backpropagation with Stochastic gradient descent is very fast compared with MB-GD. Training an NN with backpropagation can be slow. However, with multi-threaded processing and online learning, the speed improves dramatically.

An ensemble of autoencoders has training time complexity $O(t + \frac{n-BM}{BM})$, which is linear for an ensemble with fixed buffers size **B**, t epochs for the first training window, and ensemble size M (same as thread size). Adding a checking step in SA makes the testing time longer. However, for each anomaly detected, the checking time is constant. Therefore, the testing time for SA is also linear with respect to the number of testing instances.

With this fast processing, we can safely discard the normal data, which have been processed. In order to choose the thresholds for the reconstruction errors, we store all the negative instances (size h) in SA. Other than that, only the data points in the buffers and the weights of the M NNs are being stored in memory. The space needed to store $2M$ data points is constant and the space needed to store M NNs is also constant. The space complexity for SA is $O(1 + h)$, which is almost a constant since h is usually small.

Algorithm 3 AnomalyCheck

Input: x_i is a testing instance which is detected as anomaly in pred(x_i)
Output: Anomaly = True or False

procedure INDEEDANOMALOUS(x_i, w, Buffers)
  extract w data points ($w_1 = i - w, \ldots, i - 1$) from buffers before i
  train an autoencoder A_{before} on these w normal instances
  accumulate w instance after data i
  train an autoencoder A_{after} on these w unknown label instances
  if pred(x_i, A_{before}) == "abnormal" and pred(i, A_{after}) =="abnormal" **then**
    return "abnormal"
  else
    return "normal"

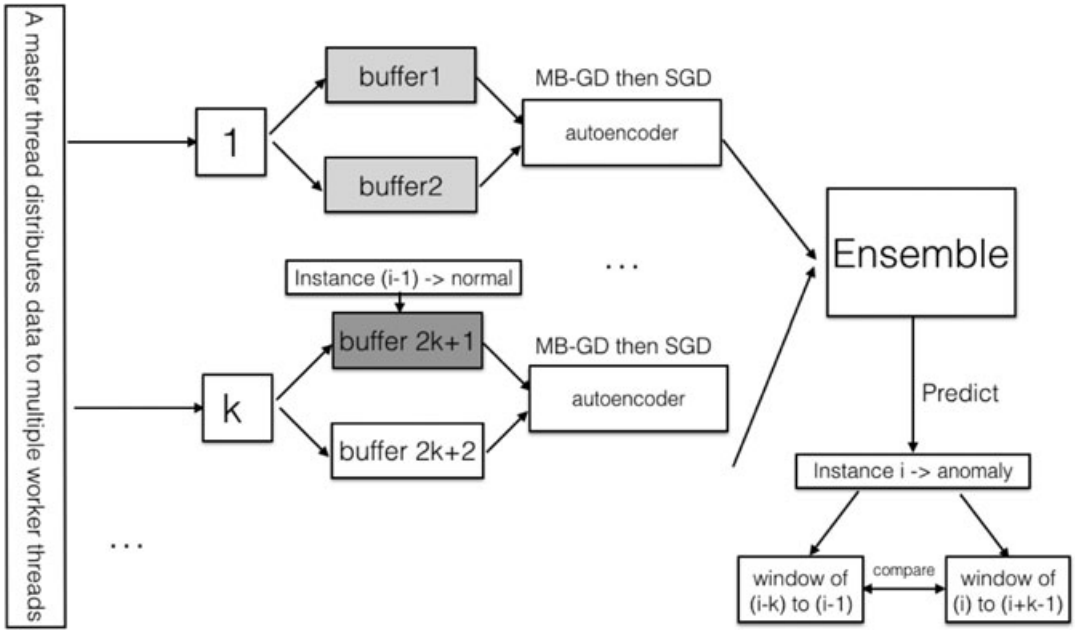

FIGURE 3 An illustration of the testing and continuously learning process for streaming autoencoders (SAs). MB-GD, mini-batch gradient descent; SGD, stochastic gradient descent

In terms of accuracy, SA optimizes incremental NNs since the initial weights are adjusted by mini-batch training from the first B data instances. This minimizes the problem brought by the random initial weights. Therefore, the performance of SA is less random than that of incremental NNs. Moreover, having ensembles of autoencoders from different segments of the data stream improves the performance.[28] Ensembles of classifiers usually have better performances than a single classifier as long as individual classifiers are diverse enough. In SA, each autoencoder in the ensemble is trained on a unique subset of the data stream. Therefore, the individual autoencoders in the ensemble are diverse enough to give a better performance than a single classifier.

3.6 | Dealing with concept drift

One requirement for stream learning algorithms is the ability to deal with concept drift. In data streams, the underlying data distribution can change over time. This phenomenon is called *concept drift*. Concept drift can cause predictions to become less accurate over time.[29]

Kelly et al[30] presented 3 ways in which concept drift may occur: the chance of prior probabilities of classes, the change of class-conditional probability distributions, and the change of posterior probabilities. Our study is restricted to the second type of concept drift, where the conditional distribution of the output changes.

Dealing with concept drift in training phase: when the conditional distribution changes in the output, there are 2 solutions to prevent the deterioration of prediction accuracy.[30] Active solutions (called concept drift detectors) explicitly detect concept drift in the data generating process. Usually, such models have no ability to learn the stream other than detecting distribution changes. On the contrary, passive solutions build models for learning data streams, which are continuously updated. This can be achieved by keeping a sliding window that stores the most recent data points in the stream[29] or using an ensemble of classifiers.[31]

The model proposed in this paper is a passive predictive model with adaptive learning. It uses blind adaptation strategies[31] without any explicit detection of concept drift. As Elwell and Polikar[31] explained, "the blind approaches forget old concepts at a constant rate whether changes are happening or not. As time goes on, the newly arrived data tend to erase away the prior patterns."

In NNs, learning is always associated with forgetting. The newly arrived data update the weights of NNs, which makes the effects of old data in the stream become less and less important. Since streaming data arrive at a high speed, the weights in the models get updated rapidly, and the model adjusts to the concept drift quickly.

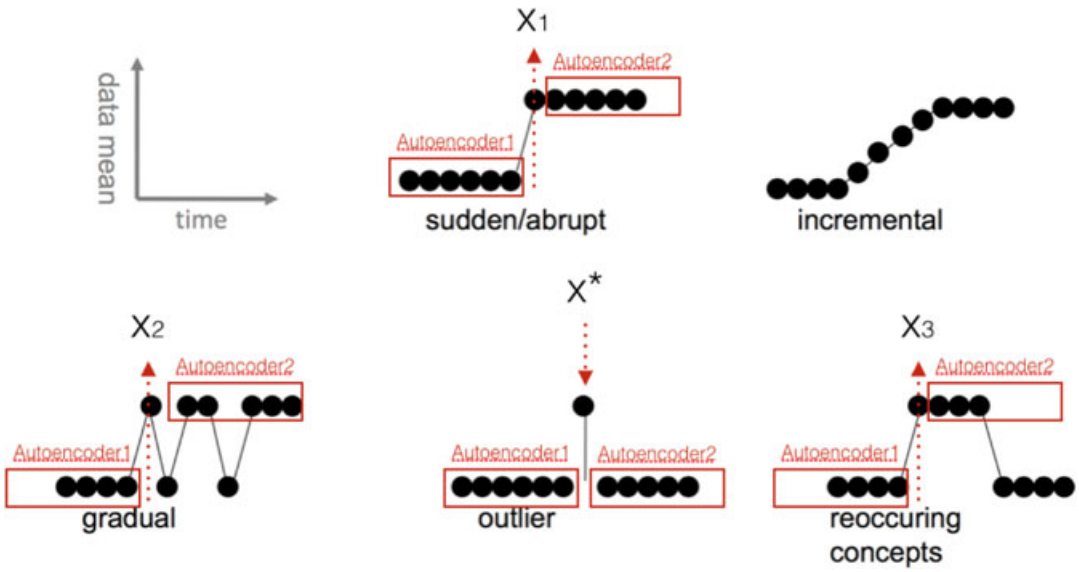

FIGURE 4 Configuration of changes over time. Only x^* is the outlier we want to detect, and the rest x_1, x_2, and x_3 will usually be detected in the first phase of streaming autoencoder (SA) but eliminated in the second phase [Color figure can be viewed at wileyonlinelibrary.com]

Moreover, the use of threaded ensemble learning makes our model less sensitive to noises and more stable in adapting to concept drift. The ability of continuously learning from data streams while keeping previously learned knowledge is known as the stability-plasticity dilemma.[32] This dilemma happens because "there is a trade-off between handling noise in a stable way and learning new patterns."[32] Since we split the data into multiple threads, the chance of noises affecting all threads is small compared with a single NN. If a concept drift is truly happening, it will show up in multiple threads and be detected and adapted by the majority vote of the NNs in the ensemble.

Dealing with concept drift in testing phase: with the additional checking step, SA has the ability to distinguish real outliers from concept drift during the testing phase. Figure 4 shows 5 different types of changes over time that may occur in a single variable data stream, the detailed description of these changes can be found in the works of Gama et al[33] and Kuncheva.[34]

As shown in Figure 4, only x^* is the real anomaly we want to detect. The rest of x_1, x_2, and x_3 will be detected by most of the outliers detectors and by the first detection phase of SA. However, x_1, x_2, and x_3 are points that really represent concept drift rather than real outliers. In the checking phase of SA, these points will likely be labeled as normal because the autoencoders built from adjacent windows before and after $x_i, i = 1, 2, 3$ will disagree which indicate concept drift are detected instead of outliers.

4 | EXPERIMENTAL SETUP

In this section, we first describe the data sets that are used in the experiments. We then discuss the experimental settings for our algorithm and the baseline algorithms.

4.1 | Data sets

The data sets used in our experiments are summarized in columns 2 to 4 of Table 3. We used typical benchmark data sets for comparisons. SMTP, HTTP, and SMTP+HTTP are streaming anomaly data published from the KDD Cup. Note that SMTP+HTTP is constructed by using data from SMTP followed by data from HTTP. This data set involves concept drift as more details can be found in the work of Tan et al.[1]

The fourth data set COVERTYPE is from the UCI repository. It has some distribution changes since there are multiple different cover-types in the normal class.[1] We used the smallest class Cottonwood/Willow with 2747 instances as the anomaly class. The next data set we used is the SHUTTLE data set from the UCI repository. This data set has no distribution change.[1]

Weather data set is a sensor data set we collected from the Meteorological Assimilation Data Ingest System.[‡] It consists of meteorological data collected from the Road Weather Information System. We use sensor readings such as temperature, humidity, dew points, etc, to predict the road surface conditions. We extracted the Road Weather Information System data from December 1, 2014 to December 1, 2015 in the area of Iowa restricted in the square defined from (43.492237, −96.573944) to (40.415721, −91.404758).[§] We use classes 1, 2, 4, and 9 as normal instances, and all the other classes (Snow/Ice Warning, Black Ice Warning, etc) as anomalies.

Table 2 summarizes the data sets used in the experiments.

[‡]https://madis-data.noaa.gov/madisPublic1/

[§](43.492237, −96.573944) is the upper left corner of the square and (40.415721, −91.404758) is the lower right corner.

TABLE 2 Benchmark data sets

Data Set	Data Size	Dimensionality	Anomaly, %
HTTP	623 091	41	6.49
SMTP	96 554	41	12.25
SMTP+HTTP	719 645	41	7.26
COVERTYPE	581 012	54	0.47
SHUTTLE	14 500	10	5.97
Weather	1 719 781	32	3.53

4.2 | Experimental settings

We compared the performance of SA with 2 popular tree-based anomaly detectors for stream learning: Streaming HSTa[1] and VFDT.[2] We also compared our model with streaming multilayer perceptrons (SMLP),[35] a supervised model developed in our previous paper. SMLP is a model with ensembles of multilayer perceptrons, which only works when anomaly instances are available for training.

The HSTa was proposed recently as a one-class anomaly detector for streaming data.[¶] It is an algorithm based on random trees. We used the default parameters settings as described in the original paper (ie, window size = 512, ensemble size = 30, and tree depth = 15). Very fast decision tree is a state-of-the-art classification algorithm for data streams. RMOA package in R is used for the comparison of VFDT (the underlying program is written in Java). The default parameters are used with a chunk size of 512.

We implemented SMLP, SA (without the buffers and the checking step), and SA2 in Python with h2o deep learning model. The parameter settings for SMLP, SA, and SA2 (SA with an additional checking step) are 8 threads with chunk sizes of 512. For SAs, we use input dropout of 0.1 and tanh functions with 50% dropout as the activation functions.

The area under the curve (AUC) based on anomaly scores is used as the evaluation metric. In HSTa, SMLP, and VFDT, the anomaly scores are calculated as the probability that a point is predicted as normal. Thus, a true anomaly instance generally has a low anomaly score, while a normal point has a high probability and therefore has high anomaly scores. For SA and SA2, we use the inverse of the reconstruction error as the anomaly score, because a testing data point with a smaller reconstruction error is more likely to be a normal instance.

Based on the anomaly scores and the true labels, we then computed the AUC scores for all anomaly detectors. The AUC is the main indicator reported in this paper as in Table 3, which ranges from 0 (the worse) to 1 (the best).

All experiments were conducted on a 2.7-GHz Intel Core i5 CPU with 8-GB RAM. The order of the data instances is preserved in each data stream. We split the data stream into 80% for training (from the beginning of the stream) and 20% for testing. During the testing stage, all algorithms perform test-and-train process each time a new segment (chunk size 512) of data arrives. The results reported are averaged over 10 runs; each run is obtained using a different random seed for all nondeterministic algorithms. Pairwise *t* tests with 5% significance level were used for comparisons.

[¶]https://sites.google.com/site/analyticsofthings/recent-work-fast-anomalydetection-for-streaming-data

TABLE 3 This table presents the AUC scores of the algorithms considered in this experiments. The last row calculates the average AUC over all data sets. $v/^*$indicates that SA2 (SA with an additional checker) performs significantly better/worse than the corresponding method (SMLP, VFDT, or HSTa) based on the paired t tests with a significant level of 5%

	AUC				
Data Set	**SA**	**SA2**	**SMLP**	**VFDT**	**HSTa**
HTTP	0.977	0.990	0.990	0.996*	0.995*
SMTP	0.997	0.999	0.998v	0.987v	0.880v
SMTP+HTTP	0.989	0.987	0.899v	0.867v	0.946v
COVERTYPE	0.973	0.982	0.994*	0.991*	0.980v
SHUTTLE	0.992	0.994	0.988v	0.990v	0.999*
Weather	0.923	0.904	0.909	0.877v	0.886v
Average	0.975	0.976	0.963	0.951	0.948

Abbreviations: AUC, area under the curve; HSTa, half-space tree; SA, streaming autoencoder; SMLP, streaming multilayer perceptrons; VFDT, very fast decision tree.

5 | EXPERIMENTAL RESULTS

Comparative results. Table 3 presents the AUC scores of the algorithms considered in this experiments. First, VFDT and SMLP are both multiclass classifiers. They are expected to perform better than other one-class learners because VFDT and SMLP use data from both classes. In contrast, HSTa and SAs use only "normal" data for training, by which we mean data from the positive class.

Based on Table 3, we can conclude SA2 outperforms other algorithms on the majority benchmark data sets. We could calculate the pairwise win-loss-tie statistics after running the t tests. The results between SA2 and SMLP, VFDT, HSTa, are 3-1-2, 4-2-0, 4-2-0, respectively. SA2 is indeed a competitive algorithm based on the win-loss-tie scores and the averaged AUC over all data sets. Moreover, when there is a strong concept shift as in SMTP+HTTP, SA, and SA2 perform significantly better than the other approaches. SA and SA2 also adapt better in Weather data set, where concept drift happens due to the seasonal changes. Therefore, we conclude from our observations that SA2's performance is better than that of the other systems.

Runtime comparison. Table 4 summarizes the average training and testing time over 10 independent runs for each method. Compared with the other one-class learner, SA and SA2 perform faster than HSTa on all data sets except COVERTYPE. We believe the reason why SA and SA2 are slower in learning COVERTYPE is the high variance of the normal data in COVERTYPE. This is due to that all normal instances are from difference types of forest cover-types, and they are not similar to each other in general. These one-class learners (SA,SA2, and HSTa) perform significantly faster than the other 2 multiclass learners, namely, VFDT and SMLP.

Very fast decision tree is the slowest method among all because it needs to modify the tree structure constantly. Very fast decision tree's runtime is on average about 10 times slower than the other algorithms considered, as shown in Table 4. More interestingly, we found that SAs (SA and SA2) are always much faster than SMLP in the training stage. This is because it is faster to update the autoencoders when the normal data instances are very similar to each other. A more detailed discussion on this issue can be found in the work of Japkowicz and Hanson.[36]

TABLE 4 Training time and testing time comparisons. All algorithms perform a test-and-train process continuously while the testing data arrives. Note that HSTa only use the data in the last window from training data; thus, the training time is almost zero. Thus, only the testing time (runtime) is recorded

	Training Time, s				Testing Time, s			
Data Set	**SA and SA2**	**SMLP**	**VFDT**	**SA**	**SA2**	**SMLP**	**VFDT**	**HSTa**
HTTP	20	41	349	12	22	13	399	49
SMTP	6	27	53	1	2	2	53	10
SMTP+HTTP	25	35	363	12	23	13	494	57
COVERTYPE	33	102	358	10	22	15	460	30
SHUTTLE	2	7	10	10	20	13	16	6
Weather	73	112	1260	21	31	27	951	203

Abbreviations: HSTa, half-space tree; SA, streaming autoencoder; SMLP, streaming multilayer perceptrons; VFDT, very fast decision tree.

Unlike VFDT, our NN-based models use incremental learning by adaptively changing the weights. The backpropagation in multiple threads is very efficient. During the testing stage, all algorithms perform a test-and-train process each time a new segment (chunk size 512) of data arrives. The testing time of SA2 is longer than SA because of the additional verification step. However, the test-and-train process of SA2 is still significantly faster than that of VFDT, and the overall runtime of SA2 is still less than that of HSTa.

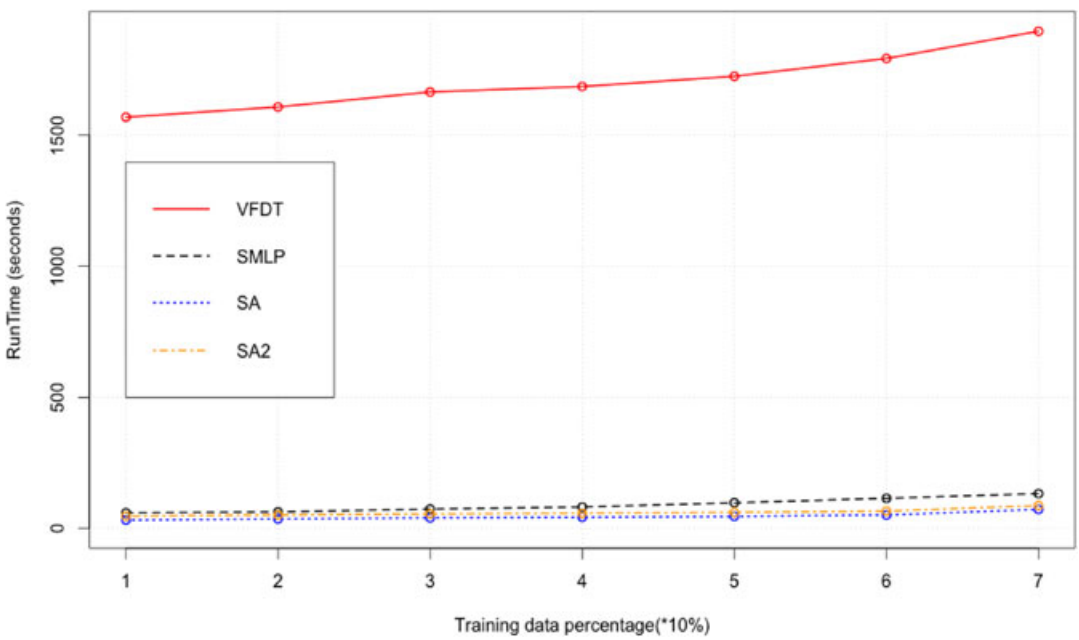

FIGURE 5 Runtime (training + testing) comparison between SMLP, streaming autoencoder (SA), and very fast decision tree (VFDT) in Weather data set with increasing training data sizes. Note that half-space tree (HSTa) has a constant runtime (203 seconds) when the size of testing data is fixed. This is because HSTa only uses the last window of training data for learning. Therefore, the total runtime is constant no matter how large the training data set is. SMLP, streaming multilayer perceptrons [Color figure can be viewed at wileyonlinelibrary.com]

FIGURE 6 The green bars represent the false-positive rate (fpr) for streaming autoencoder (SA), and the gray bars represent the fpr for SA2. We compare the fprs (false alarms) for SA and SA2 as follows. The x-axis represents our 6 data sets, and the y-axis indicates the fpr in anomaly detections. This process is repeated 10 times for each training set, and the averaged fpr is reported [Color figure can be viewed at wileyonlinelibrary.com]

We also recorded the runtime on different sizes of training data. Figure 5 shows the runtime comparison between SMLP, SA, SA2, and VFDT using the Weather data set.

In this experiment, the running time of SMLP, SA, and SA2 are linear of the data size n because each instance only requires a constant process time. We can see from Figure 5 that when the data size increases from about 200k to 1.2 million, the running time increases from 59 to 132 seconds for SMLP, 31 to 71 seconds for SA, and 46 to 86 seconds for SA2.

False-positive rate study. As we explained earlier, SA2 used a 2-step verification process to check if an instance is indeed an anomaly. This reduces the false-positive rate, as shown in Figure 6. Comparing the AUC of SA and SA2 from Table 3, we can see that SA2 is not always performing better than SA in terms of AUC. However, SA2 always outperforms SA in terms of low false alarm rate. This is due o the fact that some anomalous instances detected in SA are eliminated in the verification stage of SA2.

In our experiments, we found that the performance of our model is not too sensitive to the number of hidden units we choose in the hidden layers. Suppose the dimensionality (the number of attributes) of the data sets is n. Varying the number of hidden units from $2/3 * n$ to $2 * n$ gives similar results. The effect of randomly picking parameter values reduces as the ensemble size goes up. We can conclude that the insensitivity of our model to the parameters is due to ensemble learning. Random parameter settings in each learner might result in a weak learner, but an ensemble of several weak learners can lead to a strong learning algorithm.

6 | CONCLUSIONS

In this paper, we proposed a fast and scalable anomaly detector, ie, SAs, for data stream mining. The proposed algorithm has the following features for stream learning: (i) it processes endless

data streams in one pass without the need to store the entire data set from streams and (ii) it has the ability to deal with concept drift by learning continuously and adaptively. Moreover, it has the ability to learn different anomalies even if they have not been observed in the training data.

We have identified the key weaknesses for the commonly used tree-based algorithms in this paper. The empirical studies show that our NN-based model performs more favorably than tree-based algorithms in evolving data streams, especially when concept drift is present. Compared with other tree-based algorithms, SA2 performs favorably in terms of the win-tie-loss scores and the averaged AUC scores. Our algorithm is comparable to VFDT in terms of AUC scores and outperforms VFDT significantly in terms of runtime. Compared with HSTa, a state-of-the-art algorithm for one-class learning, our model outperforms it in terms of AUC scores and has a speed comparable to that of HSTa. Compared with our previous multiclass model SMLP, SA2 performs better in terms of win-tie-loss on the 6 data sets tested. Moreover, we are able to improve the speed of our model by 20% to 40% over the previous models (SMLP and SA). The time and space complexities of our new model are also significantly better with linear training time and constant memory space requirement. Moreover, SAs can be trained using only data instances from the positive class, which is an advantage when the abnormal instances are rare or not even available for training.

In summary, the main contributions of this paper are achieved by the following 3 means:

1. We used autoencoders with mini-batch learning and online learning in multithreads for fast processing speed.
2. We implemented an additional checking step in the testing phase to reduce the false-positive rate for detections.
3. We demonstrated empirically that our method performs favorably in terms of AUC and runtime when compared with other state-of-the-art algorithms on multiple benchmark data sets.

Note that our model will work well when the data arrive fast and in large quantities. This, however, is not always the case in data stream mining. One avenue for future work research is to consider what will happen with different rates of arrival and fluctuations in the volume of data. For example, the system could be made more adaptive so that when data arrive slowly, some of the threads in our model could be dropped and the number of learners in the ensemble reduced. On the other hand, when our model gets overwhelmed by a large volume of data, we could increase the number of buffers and threads.

ORCID


Yue Dong http://orcid.org/0000-0003-2161-8566

REFERENCES

1. Tan SC, Ting KM, Liu TF. Fast anomaly detection for streaming data. In: Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI'11, vol. 2; July 16-22, 2011; Barcelona, Spain.
2. Domingos P, Hulten G. Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2000; Boston, MA.
3. Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2001; San Francisco, CA.
4. Rutkowski L, Pietruczuk L, Duda P, Jaworski M. Decision trees for mining data streams based on the McDiarmid's bound. *IEEE Trans Knowl Data Eng*. 2013;25(6):1272-1279.

5. Rutkowski L, Jaworski M, Pietruczuk L, Duda P. Decision trees for mining data streams based on the Gaussian approximation. *IEEE Trans Knowl Data Eng*. 2014;26(1):108-119.
6. Wu K, Zhang K, Fan W, Edwards A, Philip SY. Rs-forest: a rapid density estimator for streaming anomaly detection. Paper presented at: 2014 IEEE International Conference on Data Mining; 2014; Shenzhen, China.
7. Abe N, Zadrozny B, Langford J. Outlier detection by active learning. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2006; Philadelphia, PA.
8. Heller KA, Svore KM, Keromytis AD, Stolfo SJ. One class support vector machines for detecting anomalous windows registry accesses. In: Proceedings of the Workshop on Data Mining for Computer Security; November 2003; Melbourne, FL.
9. Bay SD, Schwabacher M. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2003; Washington, DC.
10. Schölkopf B, Williamson RC, Smola AJ, Shawe-Taylor J, Platt JC. Support vector method for novelty detection. Paper presented at: NIPS; 1999; Denver, CO.
11. Liu FT, Ting KM, Zhou ZH. Isolation forest. Paper presented at: 2008 Eighth IEEE International Conference on Data Mining; 2008; Pisa, Italy.
12. Guha S, Schrijvers O. Robust random cut forest based anomaly detection on streams. In: Proceedings of the 33rd International Conference on Machine Learning; 2016; New York.
13. Huang H, Kasiviswanathan SP. Streaming anomaly detection using randomized matrix sketching. In: Proceedings of the VLDB Endowment, vol. 9; 2015; Kohala Coast, HI.
14. Pham D-S, Venkatesh S, Lazarescu M, Budhaditya S. Anomaly detection in large-scale data stream networks. *Data Min Knowl Disc*. 2014;28(1):145-189.
15. Polikar R, Upda L, Upda SS, Honavar V. Learn++: an incremental learning algorithm for supervised neural networks. *IEEE Trans Syst Man Cybern Part C (Appl Rev)*. 2001;31(4):497-508.
16. Carpenter GA, Grossberg S, Markuzon N, Reynolds JH, Rosen DB. Fuzzy artmap: a neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Trans Neural Netw*. 1992;3(5):698-713.
17. Furao S, Ogura T, Hasegawa O. An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Netw*. 2007;20(8):893-903.
18. Shen F, Hasegawa O. Self-organizing incremental neural network and its application. In: International Conference on Artificial Neural Networks; 2010; Thessaloniki, Greece.
19. Bottou L. *Stochastic Gradient Descent Tricks*. Neural Networks: Tricks of the Trade. Berlin Heidelberg: Springer; 2012:421-436.
20. Wilson DR, Martinez TR. The general inefficiency of batch training for gradient descent learning. *Neural Netw*. 2003;16(10):1429-1451.
21. Ruder S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747; 2016.
22. Maselli G, Deri L, Suin S. Design and implementation of an anomaly detection system: An empirical approach. In: Proceedings of Terena Networking Conference (TNC 03); 2003; Zagreb, Croatia.
23. Hayes MA, Capretz MAM. Contextual anomaly detection framework for big sensor data. *J Big Data*. 2015;2:2. https://journalofbigdata.springeropen.com/articles/10.1186/s40537-014-0011-y
24. Brzeziński D. Mining data streams with concept drift [Ph.D. thesis]. Poznan University of Technology; 2010.
25. Bengio Y. Learning deep architectures for AI. *Foundations Trends® Mach Learn*. 2009;2(1):1-127.
26. Japkowicz N, Myers C, Gluck M. A novelty detection approach to classification. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence, IJCAI'95, vol. 1. San Francisco, CA: Morgan Kaufmann; 1995.
27. Bottou L, Le Cun Y. Large scale online learning. In: Proceedings of the 16th International Conference on Neural Information Processing Systems; 2003; Whistler, BC.
28. Wang H, Fan W, Yu PS, Han J. Mining concept-drifting data streams using ensemble classifiers. In: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2003; Washington, DC.
29. Widmer G, Kubat M. Learning in the presence of concept drift and hidden contexts. *Mach Learn*. 1996;23(1):69-101.

30. Kelly MG, Hand DJ, Adams NM. The impact of changing populations on classifier performance. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 1999; San Diego, CA.
31. Elwell R, Polikar R. Incremental learning of concept drift in nonstationary environments. *IEEE Trans Neural Netw*. 2011;22(10):1517-1531.
32. Carpenter GA, Grossberg S, Reynolds JH. Artmap: supervised real-time learning and classification of nonstationary data by a self-organizing neural network. *Neural Netw*. 1991;4(5):565-588.
33. Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A. A survey on concept drift adaptation. *ACM Comput Surv (CSUR)*. 2014;46(4):44. http://dl.acm.org/citation.cfm?id=3084225
34. Kuncheva LI. *Combining Pattern Classifiers: Methods and Algorithms*. Hoboken, NJ: Wiley; 2004.
35. Dong Y, Japkowicz N. Threaded ensembles of supervised and unsupervised neural networks for stream learning. Paper presented at: Canadian Conference on Artificial Intelligence; 2016; Victoria, BC.
36. Japkowicz N, Hanson SJ. Adaptability of the backpropagation procedure. Paper presented at: International Joint Conference on Neural Networks IJCNN'99; 1999; Washington, DC.

How to cite this article: Dong Y, Japkowicz N. Threaded ensembles of autoencoders for stream learning. *Computational Intelligence*. 2018;34:261–281. https://doi.org/10.1111/coin.12146