Gottfried Wilhelm Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Verteilte Systeme

Master thesis
Informatics (M.Sc.)

Anomaly detection in streaming data using autoencoders

Student:	B.Sc. Bin Li
First Supervisor:	Prof. Dr. Eirini Ntoutsi
Second Supervisor:	Prof. Dr. Wolfgang Nejdl
Date:	June 11, 2018

Declaration of Authorship

I hereby certify that this thesis has been composed by me and is based on my own work, unless stated otherwise. No other person's work has been used without due acknowledgement in this thesis. All references and verbatim extracts have been quoted, and all sources of information have been specifically acknowledged.

Hanover, June 11, 2018

B.Sc. Bin Li

Contents

List of Figures

List of Tables

Abstract

The amount and variety of data stream generated from applications of different domain grows steadily and are valuable for big data research. One of the most important core component is the anomaly detection for streaming data, which has attracted attention and investigation in plenty of application areas, for example, the sensor data anomaly detection, predictive maintenance, event detection. Those efforts could potentially avoid large amount of financial costs in the manufacture. However, different from traditional anomaly detection tasks, anomaly detection in streaming data is especially difficult due to that data arrives over time with latent distribution changes, so that a single static model doesn't fit streaming data all the time. An anomaly could become normal along with data changing in the stream, it is necessary to maintain a dynamic system that deals with this problem. In this paper, we propose a novel LSTMs-Autoencoder anomaly detection architecture specially designed for streaming data. This is a mini-batch based streaming processing approach. We experimented with streaming datasets containing different kinds of anomalies as well as distribution changes, and the results suggest that our model can sufficiently detect anomaly in data stream and update model timely to fit the latest data property.

Chapter 1

Introduction

Anomaly detection is a core component of data mining, and widely used in the manufacturing industry, e-commerce, internet applications etc. It could avoid inconvenient and reduce lose in many scenarios like machine health monitoring, credit card fraud detecting and spam email recognition, and could also be used as a preprocessing step to remove anomalies for datasets in many machine learning tasks. There are already plenty of anomaly detection techniques proposed in previous literatures, that solve this problem from variety perspectives, e.g. distance-based methods, clustering analysis, density-based methods etc.

There is no lack of anomaly detection approaches that perform good with respect to different kinds of data. Supervised approaches take anomaly detection as a binary classification problem of “normal” instances and “abnormal” instances, and all instance labels should be available in advance. The key difference to other classification problem is the amount of class label is extremely biased to the normal class. In order to avoid doing data augmentation or down sampling, unsupervised approaches are more direct solutions to this problem, which find out the instances that fit least to the majority as the anomalies. Furthermore, in most situations, partial labels are available, and semi-supervised and one-class models are more efficient. They learn the pattern from labeled normal data, test data that not fit the learned pattern perfectly is likely to be the anomalies. Different kinds of anomaly detection approaches fit different use cases and data character. However, majority of them are batch model, which means, all data should be available in advance. This becomes a shortcoming under today’s big data background. With the rapid development of hardware in the last decade, the situation of data acquisition and analysis has significantly been changed. Specifically, the IoT application. Assume that we collect data from sensors attached to IoT devices, the data comes continuously and everlasting. In the beginning, no static full set of data is available for model initialization in the traditional way. Besides, during data analysis, we should always consider the volume and velocity of data, which means, on one hand, with traditional batch classifiers, the infinity data stream will lead to out of memory, on the other hand, streaming data usually comes in a high speed that leaving the system few processing time, the model should work with only single look at each data point in the stream. In addition, the statistical property of data may also change over time, which is formally called ‘concept drift’. The model should always learn new knowledge from the stream and update its identification of anomaly automatically, while anomalies could be temporally. After a data distribution change, an anomaly possibly becomes normal in the new data environment. Data distribution changes should not be classified as anomaly, and anomaly show up rarely in over the stream, they should also not be oversighted. To this end, an anomaly detection system for streaming data should be able to 1) be initialized with only a small subset, 2) process streaming data and make prediction in real-time, 3) adapt data evolution over time. 4) model should be able to deal with the biased class problem.

LSTMs are a kind of recurrent neural network and proposed for temporal dependently data. As a neural network based model, LSTMs can deal with high-dimentional and non-linear data, with arbitary expansion of the model structure. In the last decade, LSTM are used widely in time series prediction, text prediction. And LSTMs-based autoencoder is a good choice for sequence to sequence problem, e.g. language translation, time series data embedding. Deep LSTMs have also shown good performance of capturing hierarchical information from time series like seperation of sentances. Recently, LSTMs-based autoencoders are also used for time series anomaly detection in order to capture the temporal information between data points. For example, Malhotra et al. introduced an autoencoder based anomaly detection approaches in [1],[2], and achieved good performance in multiple time series dataset. This kind of anomaly deteciton approaches overcomes reconstruction error based models using traditional vanille autoencoders (VAE). However, in those approaches, they still assume that the whole datasets are available beforehand and work on static data. Also, they didn’t consider the aforementioned online learning difficulties. Hence, we enhanced this kind of LSTMs-Autoencoder based static anomaly detection approaches with the online learning ability by implementing incremental model updating strategies with streaming data.

Neural networks, including autoencoders, are normally used in batch fashion, namely the whole training set is available, and trained by backpropagation. When come to online setting, only small subset accumulated data from stream are available for model initial training, which may be suboptimal. Assume that the initialization set is enough to train a convergent model, the further streaming data are used for further model updating to adjust latest streaming data changes and the patterns never seen ever. Unlike batch models, instead of aiming at best overall performance, online neural networks are learned to achieve best sequential performance for current streaming data. The difficulty is to detect when model should be updated according to latest data and updating with which part of data. The short-term changes of data distribution should not cause model variation, while permanent concept drifts should trigger model updating as soon as possible.

In this paper, we introduce a novel and robust incremental LSTMs-Autoencoder anomaly detection model, which designed specifically for time series data in a streaming fashion using Long Short-Term memory (LSTM) units, with also online learning ability for model updating. For each accumulated mini-batch of streaming data, the autoencoder reconstructs it with previous knowledge learned from normal data. Anomaly data (never used for training) is expected to cause significant larger reconstruction error than normal data. In addition, the model is able to update itself when detected data distribution changes.

Todo: Summary of experiment results...

The rest of this paper is organized as follow. In chapter 2, we collected previous works on anomaly detection and their shortcomings. We also refer some works on incremental neural network. In chapter 3, define the problem formally. In chapter 4, we propose our method for streaming data anomaly detection and discuss the advantage over previous works. In chapter 5, we describe the datasets used for experiments and the experimental set-up. In chapter 6, we present our experimental results. And in chapter 7, we summarize the work and discuss of success and deficiency.

Chapter 2

Related works

In this section, we present a survey on previous works in anomaly detection. Both batch models and online models are listed with respectively traditional machine learning approaches and neural network or autoencoder based approaches. And for the online case, we also survey works on neural network updating approaches.

2.1 Batch model

2.1.1 Classical machine learning based approaches

As an important component of data mining and machine learning, anomaly detection has been investigated using plenty efficient models. When talking about anomaly detection, the most intuitive solution may be detection of outliers from a dense cluster, or to find those data points that have obvious different property as their neighbors and so on. Considering the lack of label and extremely imbalanced dataset, unsupervised approaches are more widely used in practice, for example Local Outlier Factor (LOF).

In anomaly detection, the LOF is a common density-based approach. LOF shares some concepts with DBSCAN such as ‘core distance’ and ‘reachability distance’, in order to estimate local density. Here, points with substantially lower local density than their neighbors are considered as anomalies. LOF shows competitive performance in many anomaly detection tasks, especially when dealing with data with unevenly density distribution. However, when getting a numerical factor from LOF model, it is actually hard to define a threshold automatically for the judgement of anomaly.

Because normally the anomaly appears rarely in the dataset, and occurs usually in novel ways, it is expensive for labeling and hard to learn all kinds of anomalies in the training phase, so unsupervised models are commonly used. There are still a batch of semi-supervised or one-class anomaly detection models. The intuitive difference between anomaly detection and binary classification problem is the obvious few negative class data (anomalies). A typical one-class model is the One-class Support Vector Machine (OCSVM).

As an semi-supervised one-class classifier, OCSVM takes only normal data as input, and generates a decision surface to separate them from the anomaly states. By analyzing anomalies, the datasets are always bias to the normal part, and anomaly appear only rarely. So, this kind of one-class classifiers avoid making balance between the two classes. Besides, they also take advantage of classical support vector machine, with the help of kernel method, they can also deal with linearly not separable data. However, in the meantime, the choosing a proper kernel becomes a hard point of OCSVM. A suboptimal kernel function can seriously impact the performance.

Although classical machine learning approaches can handle most of the normal anomaly detection, there a still a lack of pervasive models that fit different kinds of data characters. Moreover, only few of those approaches could be directly or after some modification used for time series and streaming data, while they ignore the temporal dependency between samples.

2.1.2 Autoencoder-based batch approaches

LSTMs-Autoencoders are originally widely used for text generation because that the LSTMs are able to capture the context dependency between words and sentences. Text data are usually embedded into vector as input of autoencoder. And the tasks are either generate temporal relevant text on the decoder side or learn text representation in the hidden layer [1]. Similar problem is faced in the time series data mining due to the temporal dependency of values at each timestamp. So, in later works, LSTMs-Autoencoder are also used for time series data.

Sutskever et al. [13] use a deep LSTMs-based sequence to sequence model for language translation. In their work, the deep LSTMs encoder take single sentence as input, and learn a hidden vector of a fixed dimensionality, and then a different LSTMs decoder decodes it to the target sentence. As a translation task, they found that this encoder-decoder architecture can capture long sentences and sensible phrases, especially they achieved better performance with deep LSTMs in compare with shallow LSTMs. In addition, a valuable found is, reversing the order of words in the input sentence makes the optimization problem much easier and achieved better performance. The LSTMs based model outperforms non-LSTMs model on the long input sentence cases (more than 35 words) since its long-term memory ability.

Li et al. [7] did similar research on long paragraph text and even entire document generation using LSTMs-autoencoders. Their main contribute is the hierarchical sentences representation. The model learns words level, sentence level and paragraph level information with each respectively a LSTMs layer, so that the model captures very long-term temporal information. Moreover, they introduced an attention based hierarchical sequence to sequence model that connect the most relative part between encoder and decoder like the works around a final punctuation. They experiment with documents over 100 words, the results show that hierarchical and attention-based hierarchical LSTMs learns even better long-term temporal information than standard LSTMs-encoder-decoder models.

As autoencoders achieves great successes in text data and speech processing, they are also used on time series anomaly detection in terms of temporal dependently data. These models train autoencoders with only normal data, and anomaly data as unknown patterns. Then the autoencoder can only reconstruct normal patterns, large reconstruction error indicates anomaly. An early work [11] uses the vanilla autoencoder to detect abnormal status of the electric power system. In order to capture temporal information, they applied sliding window on the raw data as input. As anomaly scoring method, they evaluated each sliding window with respect to their reconstruction error. As some measures in the autoencoder output vectors are more sensible to anomalies than others, they use the average absolute deviation of reconstruction error as anomaly score. And the anomaly threshold is chosen by large amount of experiments over normal data.

An important reason of using autoencoder for anomaly detection is its ability of dealing with high-dimensional data. Sakurada et al. [12] experimented with time series data that consist of 10-100 variables with no linear correlation. Comparing with reconstruction using PCA or Kernel PCA techniques, using the autoencoder reconstruction error is more easily to recognize anomalies.

In further researches, Malhotra et al. [10][8] develop the application of LSTMs-autoencoder in sequence learning into anomaly detection problem. They proposed stacked LSTM networks model to learn high level temporal patterns. They show that LSTMs outperforms normal RNNs based anomaly detection model and avoid facing to the gradient vanishing problem. They also detect anomaly based on the reconstruction error. The scoring function is based on the parameters of a estimated normal distribution of a validation set. Their experiments show that the model performs good in variety kinds of datasets. A variation of this model [9] has been proved that achieves better performance in the anomaly detection tasks, while they tell that using a constant as input of decoder instead of read time series value improves the performance of model.

2.2 Online anomaly detection over data stream

2.2.1 classical online approaches

Recently, more and more attention is paid to streaming data mining, and many efficient classical models are modified to fit the online learning property. Cui et al. [2] proposed online anomaly detection approach with grid-based summarization and clustering algorithm, which can detect anomaly immediatly when data arrives. For the seen streaming data, they used summarization algorithm to reduce the run time and memory consumption over stream. And they give anomaly scores to instances to describe concrete anomaly degree. They show good performance in KDD dataset for network attack detection. However, they didn't take consider only isolated data point without any temporal information measuring. Another streaming data anomaly detection framework by Tang et al. [15] use sliding window to involve the contextual dependency and temporal changes in the stream. The anomaly detection algorithm is a modified version of LOF while LOF's high time complexity can not be directly employed to streaming data. They use comentropy to filter out most normal windows, and feed the rest to LOF.

2.2.2 Autoencoder-based online approaches

Zhou et al. [16] proposed an online incremental updating method for denoising autoencoders by modifying the hidden layer neurons in order to deal with the non-stationary streaming data properties. The kern ideal are two steps, adding hidden layer neurons to capture new knowledge, and merging hidden layer neurons if information is redundant. Their experimental result shows comparable or better reconstruction result than non-incremental approaches with only few data used during initialization. And they show that their incremental feature learning methods performs more adaptively and robustly to highly non-stationary input distribution.

Dong et al. [3] proposed a 2-step anomaly detection mechanism with incremental autoencoders. The implemented the system with ensembled autoencoders in multithreads to leverage parallel computing when large volumes of data arrive. Besides their 2-step mechanism check anomaly in the first step and verify anomaly data with previous and subsequent data (to differ between anomalous state and concept drift) to reduce false-positive rate in anomaly detection. In the experimental results, they show that their model outperforms commonly used tree-based anomaly detection model especially when concept drift presents and speed up the online processing speed with mini-batch learning and online learning in multithreads.

Ghazikhani et al. [4] introduced an online neural network model for streaming data towards to the two major problems of online learning, concept drift and imbalanced classes. In term of concept drift, they applied a forgetting function that weights recent instances to navigate the model to the drifted model, so that the model always learns pattern from latest data. Besides, for class imbalance, they proposed a error function for two-class imbalance problem with the basic idea that the error function generating higher error signals for instances in the minority class.

Kochurov el at. [6] designed incremental learning framework for deep neural networks based on Bayesian inference. They argued that, naïve deep learning approaches for incremental learning applies Stochastic Gradient Descent (SGD), which intent to keep previous learned model remembered, and enhanced with current batch of new data. However, by SGD, the neural network model is likely to converge to the local optimal of the latest batch of data with of preserve the previous knowledge. Their Bayesian framework estimate the posterior distribution over the weights of the model in the condition of previous knowledge and use the Bayesian rule to sequentially update the posterior distribution in the incremental learning.

In this work, we implement a LSTMs-autoencoder based incremental streaming data anomaly detection model. The LSTMs-autoencoder is close to the model by Malhotra et al. [8], and we design an online model updating strategy as well as the dataset used for model updating.

Chapter 3

Preliminaries

3.1 Definition of stream

Assume that a set of devices or data warehouses provide data continuously with a specific velocity V (here we only take about numerical data). A data stream DM is defined as

$$DS = \{(x_1, t_1), (x_2, t_2), ..., (x_T, t_T), ...\} \tag{3.1}$$

where x_T is the instance arrived at timestamp t_T represented by a multi-dimensional vector.

In order to feed the streaming data to LSTMs, the stream is accumulated to windows and batches as Figure 3.1. Every T instances are accumulated to a window as a single LSTM input. And MB is the predefine batch size, each batch contains MB windows.

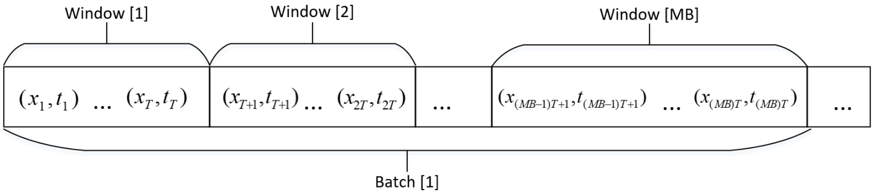
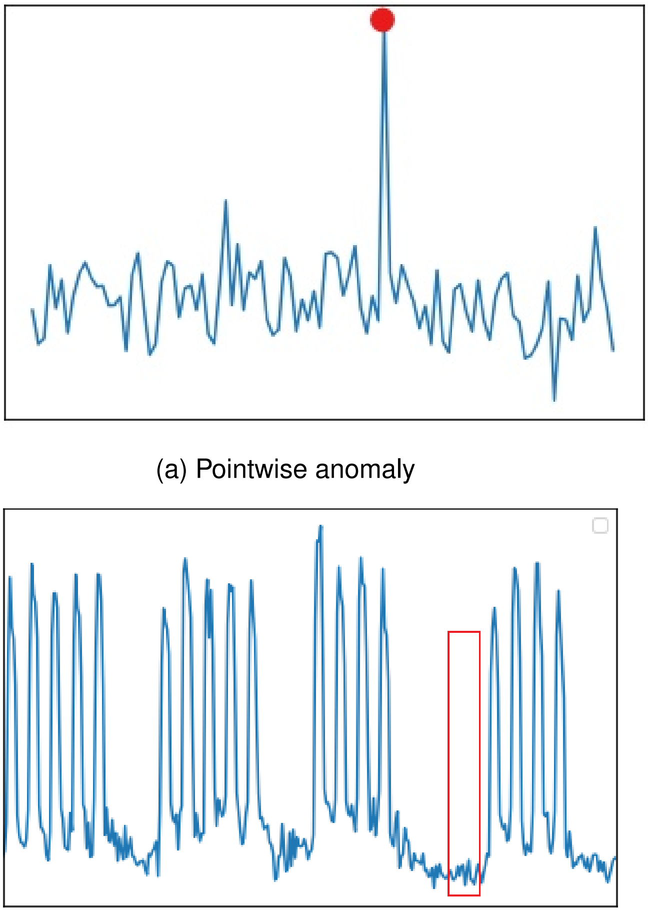

Window [1]
Window [2]
Window [MB]
(x_1, t_1) ... (x_T, t_T)
(x_{T+1}, t_{T+1}) ... (x_{2T}, t_{2T})
...
$(x_{(MB-1)T+1}, t_{(MB-1)T+1})$ ... $(x_{(MB)T}, t_{(MB)T})$
...
Batch [1]

Figure 3.1: Data stream

3.2 Definition of anomalies

Pointwise: A data point (instance) is anomalous(e.g. Figure 3.2a on the next page) if this point is distant from other observations according to some specific measurement metrics. This is used in fine-grained anomaly detection tasks, that need to find out every single anomalous instance, e.g. credit card fraud detection, spam email detection.

Window-based: Sometimes, a data point is apparently normal, but this point, or potentially together with its neighbors violates the overall periodicity or other character of the time series, it or they are also anomaly, which is called window-based anomaly, e.g. Figure 3.2b.



(a) Pointwise anomaly

(b) Window-based anomaly

Figure 3.2: Anomaly types

In the anomlay detection experiments, abnormal data is treated as positive class and normal as negative class.

		Actual value	
		Normal	Abnormal
Prediction	Normal	TN	FN
	Abnormal	FP	TP

Table 3.1: Confusion matrix

The target is to achieve higher true positive rate (equation 3.2, detect anomalies correctly) and while remain lower false positive rate (equation 3.3, wrong alarm). The evaluation metric is Area Under the Curve (AUC), where curve represents the receiver operating characteristic curve, and is created by plotting the true positive rate against the false positive rate at various threshold settings.

$$TPR = \frac{TP}{TP + FN} \tag{3.2}$$

$$FPR = \frac{FP}{FP + TN} \tag{3.3}$$

3.3 LSTMs

Recurrent neural networks(RNNs) are widely used for speech, video recognition and prediction due to its recurrent property that captures the temporal dependency between data in compare with other feed forward networks. However, the volume of RNN's memory is limited, and vanishing gradient is also a difficulty by training RNNs. Therefore, the long short-term memory networks (LSTMs) are a kind of reinforced RNN that is able to remember meaningful information in arbitrary time interval. A LSTM network is a recurrent neural network with neurons being LSTM units. Figure 3.3 shows a classical structure of a LSTM unit. LSTMs are able to capture long-term memory while there are a forget gate and a update gate in the LSTM unit, that select necessary previous information and new coming information according to the input data at each time step. The information is transferred to the next step within the cell state. Besides, each LSTM units also output its value by going through a softmax function.

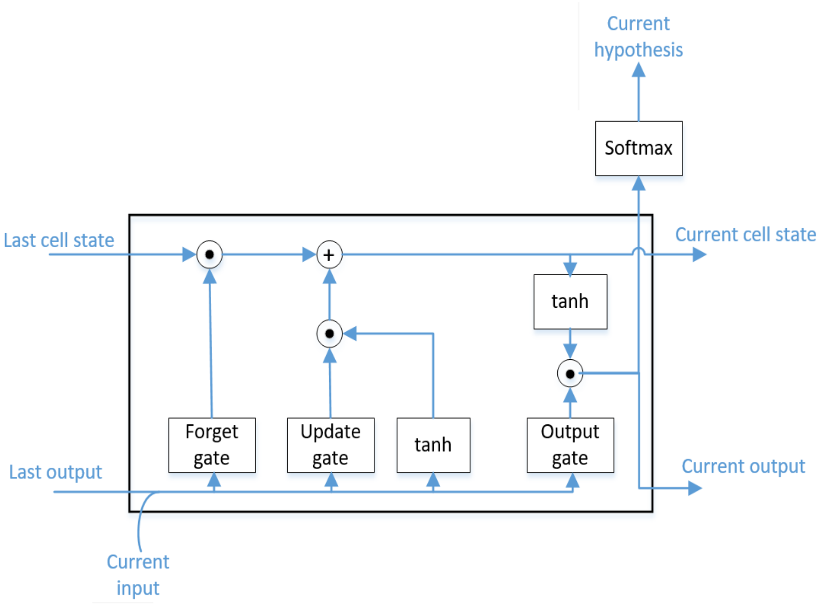


Figure 3.3: The LSTM unit

A LSTM unit can be unfolded over time, as shown in Figure 3.4 on the following page. The LSTM unit takes a data window as input, namyly takes one instance at a time. Therefore, the LSTM unit extracts useful and drop useless temporal information from the window of data.

Deep LSTM RNNs are built by stacking multiple LSTM layers. Note that LSTM RNNs are already deep architectures in the sense that they can be considered as a feed-forward neural network unrolled in time where each layer shares the same model parameters. It has been argued that deep layers in RNNs allow the network to learn at different time scales over the input[5]. Figure 3.5 is a example of stacked deep LSTM neural network, there are 3 LSTM layers, each can be unfolded into 5 time steps, so the LSTMs take a window in length 5 as input and the output is in same size.

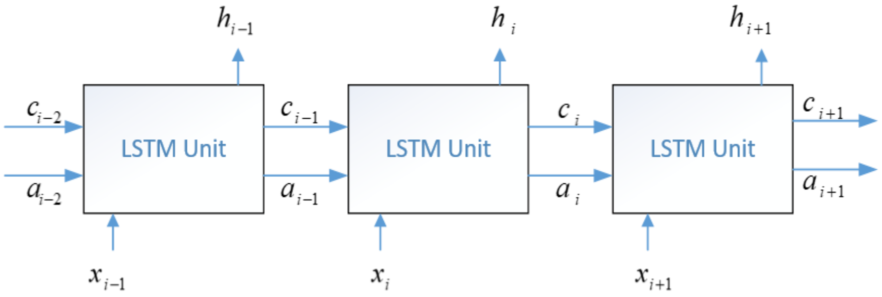
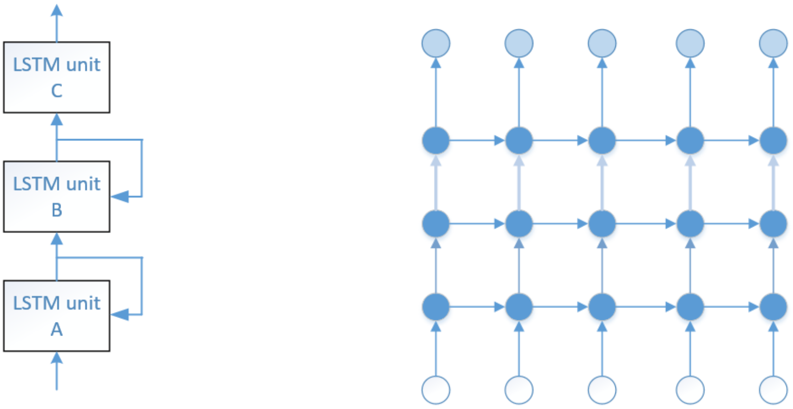


Figure 3.4: Unfolded LSTM unit



(a) Deep folded LSTMs

(b) Deep unfolded LSTMs. Each horizontal dark dot chain is an unfolded LSTM unit over time, hollow dots and grey dots are windows of inputs and outputs.

Figure 3.5: Deep LSTMs

3.4 Autoencoders

An autoencoder (Figure 3.6 on the facing page) is an artificial neural network with symmetrical structure. Normally an autoencoder has at least one hidden layer that consists of less neurons than input and output layers. And the basic aim of autoencoders is to reconstruct its own input and learn a lower dimensional representation (encoding) of input data in the hidden layer. Moreover, the autoencoders are also used for anomaly detection by measuring the reconstruction error between inputs and predictions. Normally the component between input layer and hidden layer is called encoder, and the symmetrical component between hidden layer and output layer is called decoder. For input χ, the objective function is to find weight vectors for encoder and decoder to minimize the reconstruction error (Equation 3.4).

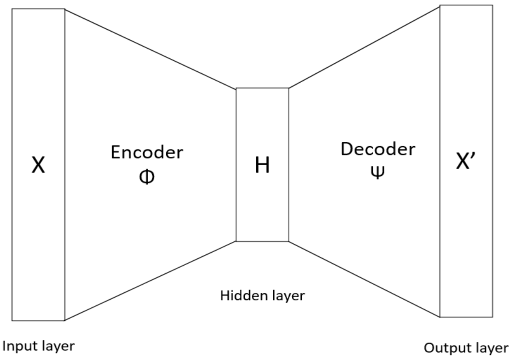
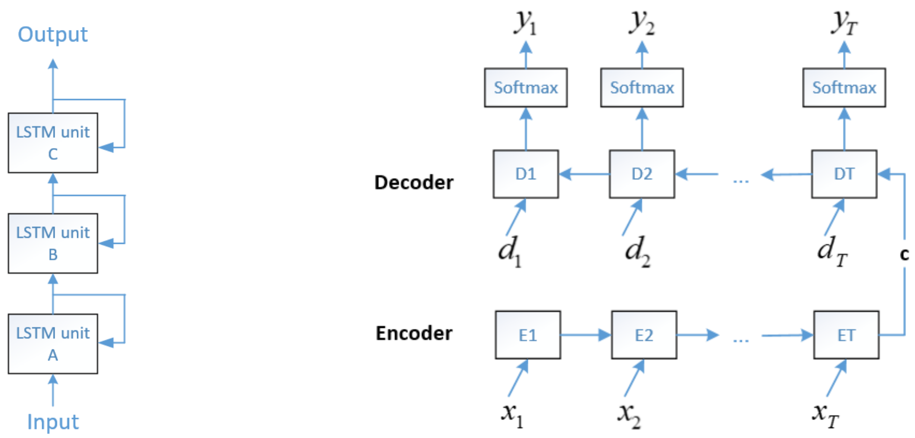


Figure 3.6: Autoencoder

$$\begin{aligned} \Phi :&\chi \rightarrow H \\ \Psi :&H \rightarrow \chi \\ \Phi, \Psi =&argmin \,\|\chi - (\Psi \circ \Phi)\chi\|^2 \end{aligned} \tag{3.4}$$

LSTMs-autoencoder has the same encoder-decoder architecture, while the neurons are LSTM units and connected in the way described in section 3.3. Figure 3.7 on the next page is a basic LSTMs-based autoencoder architecture with single LSTM layer on both encoder and decoder side. Our incremental LSTMs-autoencoder is based on this structure. The model takes window with length T as input (one instance in each step). The cell state carries sequence information and is passed through LSTM unit over time. When the encoder reaches the last encoder state, namely ET in Figure 3.7b on the following page, the cell state is actually the fix length embedding of the input window, and copied to the decoder as initial cell state of decoder, so that the input information is also transferred to the decoder. And the decoder predict the window in reversed order in order to make the optimization problem easier. To be notice is, different from aforementioned deep LSTMs in section 3.3, the encoder outputs at each time step are not directly used as inputs of decoder, while between the encoder and decoder is actually not the same logical connection as stacked LSTMs. Here, the outputs of encoder are ignored, and there are different works contributes to the research of decoder inputs. Cho et al. [1] feeds the input sequence to the decoder for a learning phrase representation task, Malhotra et al. [8] feed to decoder LSTM unit at each time step the prediction of last time step as input, and in a extended work [9] they feed the decoder always a constant vector for an anomaly detection task, because the finial cell state already carries all relevant information to represent the input window. In our model, we feed the decoder a constant vector.



(a) Folded LSTMs-Autoencoder

(b) Unfolded LSTMs-Autoencoder

Figure 3.7: LSTMs-Autoencoder

3.5 Streaming data generator: Apache Kafka

We utilize Apache Kafka as the streaming platform. Kafka is a widely used Publish/Subscribe architecture streaming system. It different from classical message queue technique with its fault tolerant, durable and large capacity properties. Different application or database can publich data to a specific topic of Kafka (topic is the data category mechanisms used in Kafka), and other processors can consume data from this topic (Figure 3.8). In our experimental setting, the data source is static databases, Kafka generate real-time data stream pipeline as data source publish records to the experiment topic, and furthermore the stream of records will be consumed by different consumers like our analysis model, visualization model etc. This configuration can be easily scaled up to more complicated and demanding real world use cases. Each record in the Kafka stream pipeline is in the form of [Key, Value, Timestamp], where keys are used for positioning and values carry the data record.

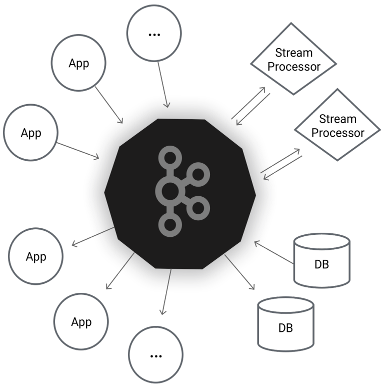


Figure 3.8: Kafka diagram[1]

[1] From: https://kafka.apache.org/

Chapter 4

Proposed model

4.1 Framework overview

The proposed model is a full flow from data stream generation, anomaly detection with autoencoder-based model and online model incremental updating. Figure 4.1 on the following page shows the general pipeline. The first received batches of streaming data are used for decision of model hyperparameters and the model initialization. Hyperparameters includes the hidden layer size, batch size, input window length as well as the number of epochs. Once the hyperparameters are learned, an autoencoder will be constructed and initialized with random weights. A subset of the streaming data is used for initial model training (only normal data used for training). Furthermore, the model is used for online anomaly detection, and evaluated based on the label provided by experts if available, or otherwise the model trust its prediction, and collect hard examples for retraining. Model will be retrained when the retraining condition is triggered. As shown in Algorithm 1, if a batch of streaming data is available, the model will start do prediction, evaluation, and check whether current window is useful to store for later retraining. If so, the window of data will be appended to the retrain buffer, with the instance order not been destroryed. As a consequence of anomalies' rare appearance, we keep all seen anomalous windows for determination of anomaly score threshold during retraining.

4.2 LSTMs-Autoencoder

4.2.1 Encoder-decoder architecture

The LSTMs-Autoencoder is consist of two LSTM units, one as encoder and the other one as decoder. The encoder inputs are fix length vectors with shape <MB, T, D>, where MB is the number of data windows contained in a mini-batch, T is the numbers of data points within each data window, and D represents the number of data dimensionality. Here, MB and T are learned as hyperparameter in the initialization phase. And on the decoder side, it will output exactly the same format data vector for each mini-batch. The LSTM unit copies its cell state for itself as one of the cell input at next timestamp. At the last timestamp of encoder, the cell state of LSTM unit is the hidden representation of the input

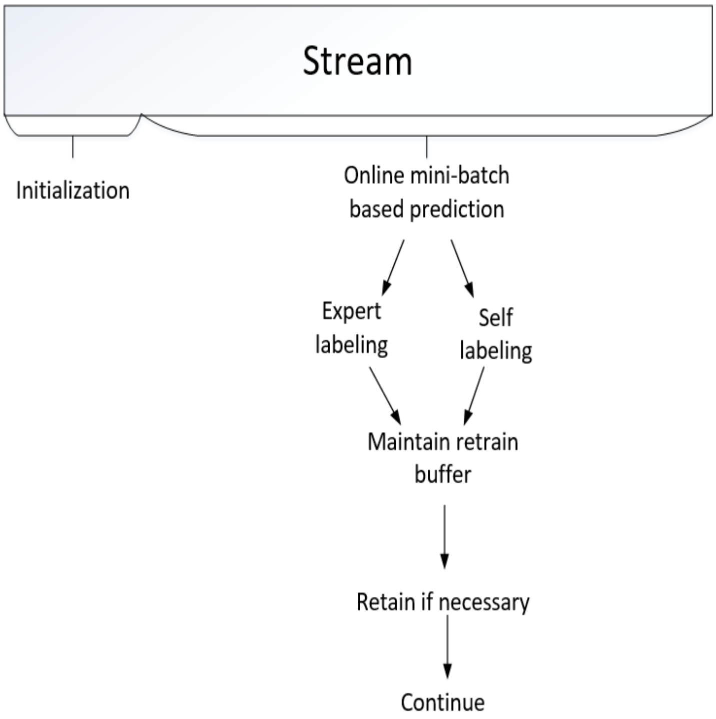


Figure 4.1: Online anomaly detection system flowchart

data vector and copied to the decoder unit as initial cell state, so the hidden information can be passed to the decoder. The size of hidden layer representation vector, namely the size of cell state is another hyperparameter need to be learn in the initialization phase. The larger the hidden vector, the more information can be captured during the process, so it is a feature highly depends on the data. Similar to previous study [13], we also train the encoder and decoder with time series in reverse order. For example, if the input data fragment are data points from timestamp t1 to t2, then the decoder will predict data point at t2 at first, and then back to t1 step by step, while this trick makes the gradient escarpment between last state of encoder and first state of decoder smaller and easier to learn.

In order to let the whole process happen online, the model initialization also utilizes streaming data. Once a small subset of streaming data is available, hyperparameters are learned, and then another dataset that consists only of normal data is collected from stream used for training. Assume that once an anomaly detection task is determined, the anomalous state is explicit defined and a subset of anomalous data is available for model initialization. We split the normal data into four subsets, N_1 for hyperparameters tuning, N_2 for model training, N_3 for early stopping, and scoring parameters learning, N_4 for testing. And abnormal data are split into two subsets, A_1 for decision of anomaly score threshold, A_2 for testing.

Algorithm 1: OnlineAnomalyDetection

```
input: normal buffer size: SN, abnormal buffer size: SA,performance threshold: P
needRetraining = False;
normalBuffer = [ ];
abnormalBuffer = [ ];
while True do
    if len(normalBuffer) >= SN and len(abnormalBuffer) >= SA then
        retrain(normalBuffer, abnormalBuffer);
    else
        data, labels = getBatchData();
        pred = predict(data);
        result = evaluation(pred,labels);
        foreach window in data do
            if 'anomaly' in label[window.index] then
                abnormalBuffer.append(window)
            end
            if result[window.index] >= P then
                continue;
            else
                if label == 'normal' then
                    normalBuffer.append(window);
                else
                    continue;
                end
            end
        end
    end
end
```

4.2.2 Online anomaly detection

The autoencoder reconstructs the input with its knowledge of normal data, so if the input data contains anomalies, the reconstruction error will be obviously large due to the lack of anomalous knowledge. For input $X^{(i)}$, the reconstruction error is

$$e^{(i)} = \left| X^{(i)} - X'^{(i)} \right| \tag{4.1}$$

similar to [8], the reconstruction error of data points N_3 is used to estimate the parameters μ and $\Sigma$of a normal distribution $\mathcal{N}(\mu, \Sigma)$ using maximum likelihood estimation. The anomaly score for a point $x_t^{(i)}$ is defined as

$$a^{(i)} = (e^{(i)} - \mu)^T \Sigma^{-1} (e^{(i)} - \mu) \tag{4.2}$$

During the initialization phase, a anomaly score threshold τ is also learned using N_3 and A_1 as

$$\tau = argmaxAUC(a(N_3), a(A_1)) \tag{4.3}$$

The anomaly score of every instance in a window is compared with the threshold, and values over the threshold are predicted as anomalies. If a window contains more than τ_N anomalous values, this window is predicted as anomaly.

4.3 Online learning

However, if we consider using the model for streaming data, the autoencoder might get outdated because of the relative small and simple initialization dataset and concept drift happed along with time. So the update of model is necessary. In this section, we introduce the incremental learning function of the LSTMs-Autoencoder.

4.3.1 Retraining dataset

Once the LSTMs-Autoencoder is initialized, it is ready for online prediction. There is a multi-thread setting in the online learning architecture. A sub thread collects data instances continuously from the stream, and in the meantime, the main thread is working on real-time anomaly detection as long as mini-batches of data is provided by the sub thread. For each single window in the mini-batch, every instance is reconstructed and calculated the anomaly score using Equation (4.2) on the previous page. The system maintains two data buffers for retraining (Figure 4.2 on the facing page), one for normal data, and the other one for anomalies. Considering the fact that a well mastered window leads to lower reconstruction error, and higher error indicates new features in the data, and we can measure this reconstruction error level by the predefined normal distribution on reconstruction error. After each batch, the label for each data window is determined by either expert or the model itself. We predefine a performance threshold for normal data. Normal data windows that containing more that performance threshold over-anomaly-score-threshold instances are regarded as not good mastered and will be appended into the normal buffer for retraining. As anomalies appear rarely in the stream, we collect all anomalous windows in the abnormal buffer for score threshold determination during retraining.

Because the out-of-date buffer not might be collect from previous concept drift time period, and not benefits to current retraining, we maintain the retain buffers with a queue structure, so that only a specific amount of most fresh data can stay in the buffer. To this end, when a retraining process is triggered, only not well mastered fresh normal data are used for retraining.

4.3.2 Retraining trigger

During the online processing, if the system detected that the model doesn't fit the current data any more, then the retraining is triggered and done with the latest collected data in the two buffers. During experiments we found that, anomalies only appears rarely in the stream, so it often happens that the model need retraining to fit the latest data, but still

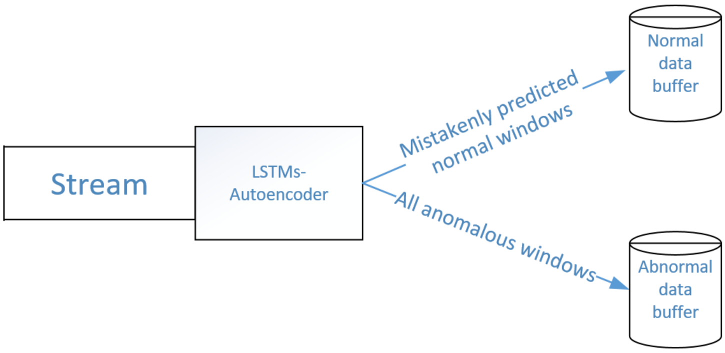


Figure 4.2: Retraining data buffer

lack of anomaly data in the buffer to update the threshold. To this end, we trigger the retraining so long as the anomaly buffer is not empty. If the anomaly data are not enough to make up a single batch, then we duplicate the anomaly buffer until the batch size. In case of the normal buffer reaches a predefined size and anomaly buffer is not empty, the model is retrained in a sub thread while the main thread keeps processing the stream.

The first retraining trigger strategy depends on the buffer size. In case of the normal buffer reaches a predefined size and anomaly buffer is not empty, the model is retrained in a sub thread while the main thread keeps processing the stream. The approach is suitable for larger, relative stationary data, while even concept drift happens, large amount of data arrives quickly to enrich the retrain buffers, and trigger retraining in time. And this approach highly depends on the performance threshold that decides when a data window from stream should be appended to the buffers, namely, retraining is not directly depends on the real-time prediction performance.

Another retraining trigger strategy is designed for smaller data set, where the waiting time of retrain buffer full might be long after concept drift happening. During the waiting time, there can be other concept drifts, and the prediction performance is suboptimal during this time period. So, for smaller data sets, the retraining trigger should directly relate to real-time performance. A simple way is, compare the batch performance with the first batch after last retraining or streaming beginning. The reason is, the model is only retrained with normal data, therefore every retraining brings new knowledge to the model, and improve the performance, so the batch performance should at least same as or better that the first batch performance, otherwise it indicates concept drift.

4.3.3 Model retraining

Once retraining process is triggered, the model will be retrained using data from retrain buffers. Windows of normal buffer are divided into retraining set and retraining validation set. Once the online phase starts, the LSTMs-Autoencoder is loaded into memory, and further model retraining are all done in memory. The retraining is a continuation of the initialization or previous retraining with identical data format. Parameters mu, sigma as well as threshold are learned from the retraining validation set and anomaly buffer data.

The parameters mu and sigma are the mean and variance (or covariance for multivariate data) of reconstruction error estimated by normal validation set during training. So we learn new parameters in the retraining using normal validation set as well.

Algorithm 2: retrain

```
input: normal buffer: nBuf, abnormal buffer: aBuf
retrainSet, valSetN = split(nBuf);
valSetA = aBuf;
Train(retrainSet);
mu, sigma, threshold = getParameters(valSetN, valSetA)
```

Chapter 5

Experimental setup

5.1 Datasets

We use 5 datasets in our experiments, PowerDemand, SMTP, HTTP, SMTP+HTTP and ForestCover, those are widely used streaming datasets in the streaming data mining area [8][3][14]. Statistical features are listed in Table 5.1. PowerDemand is a small univariate time series that records the power demand over a period of one year. Weekdays' demand is higher than weekends' and daytime is higher than nights, demand of special days (e.g. festivals) are abnormal. The experimental data is an subsampling of original set. We demonstrate a synthetic example with visualization using this dataset while the trends and anomalous states are relative obviously. SMTP, HTTP, SMTP+HTTP are streaming anomaly data extracted from KDD Cup 99 dataset. According to Tan et al. [14], HTTP contains sudden surges of anomalies and SMTP does not, but possibly exhibits some distribution changes within the stream. Because of the difficulty to point out where the distribution changes occur in the stream, the HTTP+SMPT dataset is derived by connecting SMTP and HTTP, so that a distribution change is occurred when the communication protocol is switched. The ForestCover dataset is from the UCI repository, which contains 6 kinds of forest cover types. Similar as Dong et al. [3], we defined the smallest class Cottonwood/Willow with 2747 instances as anomaly, and the rest 5 classes as normal class with distribution changes.

Table 5.1: Datasets information

Dataset	Size	Dimensionality	Anomaly proportion(‰)
PowerDemand	35 040	1	21.92
SMTP	96 554	34	12.25
HTTP	623 091	34	6.49
SMTP+HTTP	719 645	34	7.26
ForestCover	581 012	7	3.53

We separate each dataset into initialization set and streaming set, both contain normal and abnormal data. Further, the initialization set is divided into

- G(n&a): for grid search

- Tr(n): for model initial training
- P(n&a): for model parameter learning
- Te(n&a): for initial testing

where “n” represents normal data and “a” represents abnormal data. And the streaming set is published to Kafka to generate data stream (Figure 5.1).

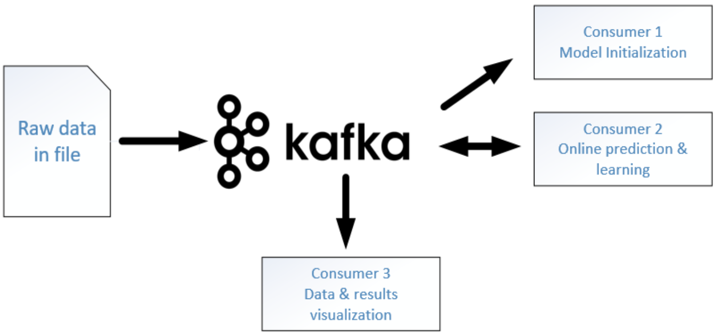

Raw data in file
kafka
Consumer 1 Model Initialization
Consumer 2 Online prediction & learning
Consumer 3 Data & results visualization

Figure 5.1: Data stream Publisher-Consumer architecture

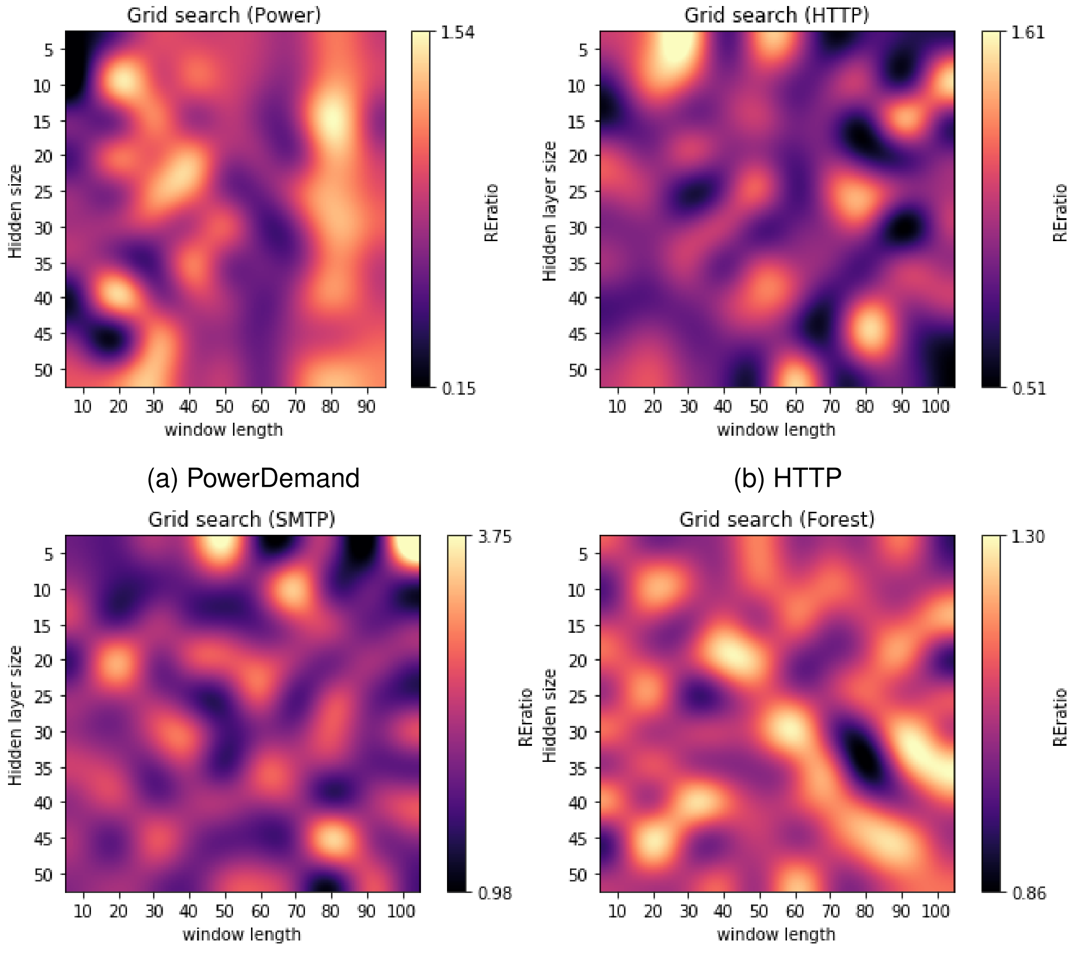
For each dataset, we use half data for initialization and the other half for online prediction. The experimental results reported are averaged over 10 runs. For each run, the model is given with random initial weights. Each subset used for training and prediction are preprocessed with locally, in order to scale them into $[0, 1]$ to fit the LSTM activation function.

5.2 Parameter tuning

For each dataset, we carry out a grid search step to tuning the model hyperparameters that fit the data best. Here we try multiple combinations of window length and hidden size for each data set. The grid search set G contains 5% -15% anomalies, and same amount of normal data together with the anomalies make up the testing set in grid search. The rest normal data is used for training. Because of the uncertainty of the random neural network weight initialization, we do each experiment 10 times and take the average result to reduce the impact. To be noted that during every divisions, the consistency of streaming data is persisted, or in other words, no random sampling took place. A good model should make the reconstruction error as large as possible in order to make the classification easier. Given dataset D, win is the input window, the average reconstruction error of D is given by Equation (5.1). The target function of grid search is given by the ration of average reconstruction error of abnormal and normal test grid search set G (Equation (5.1)).

$$ARE(D) = \frac{1}{T} \sum_{t=1, win \in D}^{T} (input_{t,win} - output_{t,win}) \tag{5.1}$$

$$REratio = \frac{ARE(G_a)}{ARE(G_n)} \tag{5.2}$$



(a) PowerDemand

(b) HTTP

(c) SMTP

(d) Forest

Figure 5.2: Grid search

As a result of grid search, the hyperparameter for different datasets is listed in Table 5.2

Table 5.2: Hyperparameters

Dataset	Window length	Hidden size	#Grid Search instance
PowerDemand	80	15	1 000
SMTP	100	5	5 000
SMTP+HTTP	100	5	5 000
HTTP	30	5	10 000
ForestCover	100	35	10 000

Chapter 6

Experimental results

6.1 Anomaly detection performance

With parameters learned from Section 5.2 on page 26, autoencoders are learned for each dataset with the beginning of streaming data. The anomaly detection performance is described by AUC. For each dataset, we compare the AUC of online phase that without and with continuously model and parameter retraining (Table 6.1). The Power demand set retrain trigger dependes on the batch performance, and for the rest datasets, retraining only triggered when retrain buffers are full.

Table 6.1: Performance

Dataset	AUC(without retraining)	AUC(with retraining)	#retrain
PowerDemand	0.91	0.97	1
HTTP	0.74	0.84	3
SMTP	0.76	0.82	2
ForestCover	0.69	0.77	3

6.2 Synthetic example

In order to show the benefit of model retraining along the stream, we demonstrate the online learning process of the small set Power demand in this section. The power demand dataset does not contain clear incremental or sudden concept drift, but the normal pattern still different slightly to each other. Lack of overall impression during the model initialization phase can lead to failures in the online phase.

6.2.1 Reaction of concept drift

Figure 6.1 on the following page shows 3 continual days power demand in normal state. Due to the lack knowledge of current pattern, the autoencoder reconstructs the input time series high than desired on day 1(left diagram). This could be caused by seasonal changes on the power demand, which is slightly, gradually, and not able to cause misclassify directly. However, the increase of normal data reconstruction error makes the margin between two classification classes smaller, and harder to make decision. As a consequence, the model retraining process is triggered after the second day with last seen data in the retrain buffer, and the model performs well again on the third day.

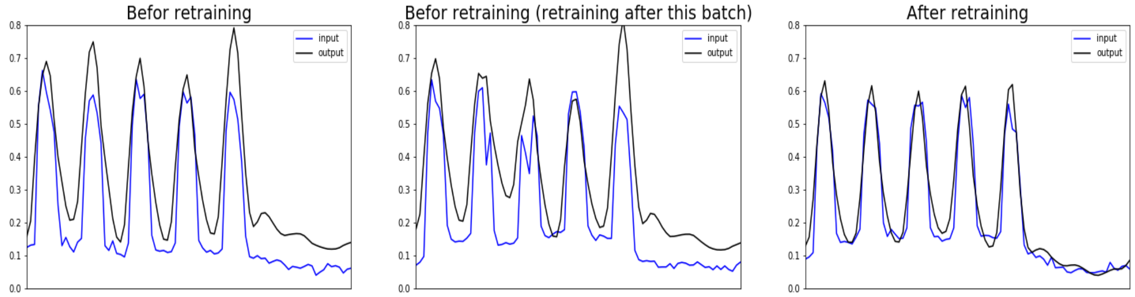


Figure 6.1: Retraining effect on Power Demand dataset

6.2.2 Retaining

During the online phase, the model is retrained two times, before batch No.10 and No. 27. After retraining, the normal data reconstruction error becomes lower while for abnormal data becomes higher, so that the classification becomes easier.

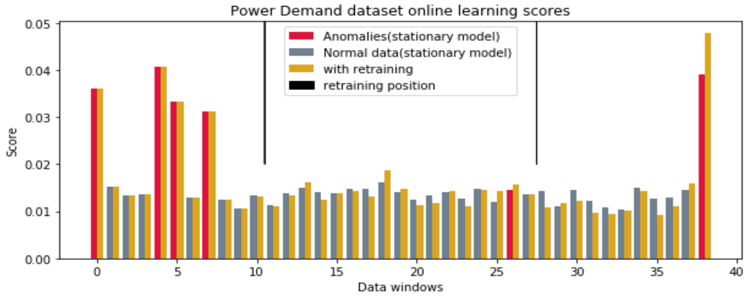


Figure 6.2: Power Demand dataset online learning scores: For each window, the anomaly score is the highest pointwise scores within the window

After each retraining process, the parameters mu, sigma and threshold of anomaly scores are also updated. Figure 6.3 shows the parameter changes over the stream. As there is no clear concept drift during the power demand stream, the parameters changes just slightly, and learn latest knowledge from the retrain buffer.

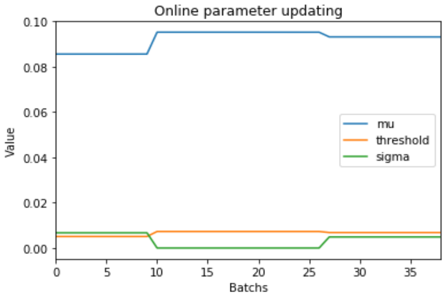


Figure 6.3: Online parameter updating

During the online phase, each normal window that is not given with a low enough anomaly score is appended to the retaining buffer to accumulate the retaining set. In order to find out what kind of data is used for retaining and how much retraining data is enough for model updating, we experiment with different retraining buffer size on the power demand stream.

6.3 Model retraining

6.3.1 Reaction of sudden and drastic concept drift

The main advantage of online model is its ability to take reaction against sudden data distributional changes in time. The SMTP+HTTP data set is composed by directly connect HTTP set after SMTP, so there is a sudden concept drift in between. The model is initialized with only SMTP data, so HTTP is completely unknown knowledge for the model. Figure 6.4 is a box plot of anomaly scores of normal instances from different part of the stream. The block B1 is statistic of normal instances' anomaly scores between the last model updating on the SMTP side and the concept drift happening, which is relative lower due to the good grasp of SMTP data. Once the concept drift takes place, namely, HTTP data arrives with the stream, more normal instances with higher anomaly score appears in B2. Although a retraining process is triggered soon after the concept drift, the normal instances' anomaly scores still increase due to lack of HTTP instance. Gradually, with the increasing amount seen HTTP data, the model gives normal data lower anomaly score again during B4 to B6. As a result, we can observe that, when a sudden concept drift happened in the stream, our model needs only 3 to 4 times retraining with totally 3500 instances for retraining to master the new data distribution again.

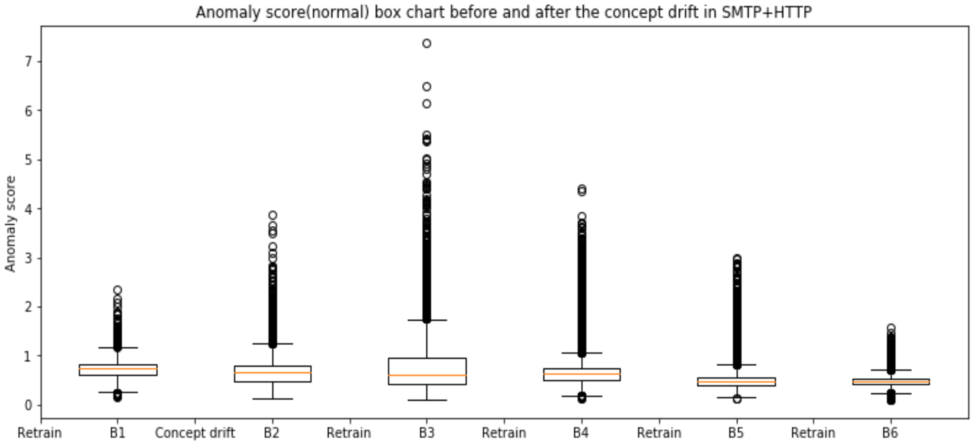
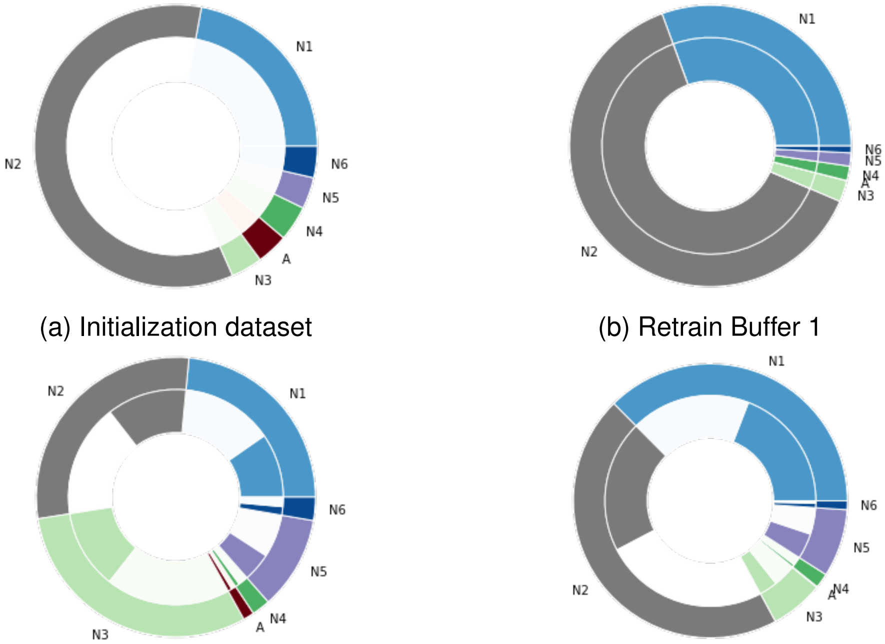


Figure 6.4: Anomaly score box chart of SMTP+HTTP

6.3.2 Reaction of serried and slight concept drift

Sometimes concept drift over the stream are slight, periodically, and potentially repeated. A single slight concept drift may not be able to trigger the retraining, but new knowledge should be saved into retraining buffer, so that once the model retrained with the fresh knowledge, the model should perform well when the same concept drift happens. We experiment with the FOREST dataset. There are 7 kinds of forest cover types as labels. We take the least type No.4 as anomaly while the rest 6 kinds as normal. Cover types appears alternately over the stream, so that it could be treated as slight concept drift.

The model is trained with hidden size 45 and window length 20. In the beginning, 3000 windows are used for initialization, and 26050 windows comes as stream. Every normal window contains more than 10 scores over threshold is treated as hard window and appended to retraining buffer. Also, every abnormal window is saved for threshold updating. When retrain buffer size reaches 750, a retraining process will be triggered. The model retraining is triggered 3 times over stream.



(a) Initialization dataset

(b) Retrain Buffer 1

(c) Retrain Buffer 2

(d) Retrain buffer 3

Figure 6.5: FOREST data initializaiton and retrain buffer data distribution: N1 to N6 represent normal states from the 6 different cover type classes, and A represents anomaly, namely type4. (a) is the initialization data distribution and (b), (c), (d) are the three retrain buffers' data distribution. Outer rings are all seen data since last retraining and inner rings are the portions added to retrain buffer.

Chapter 7

Conclusion

Bibliography

[1] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. 2014.

[2] Hongyin Cui. Online outlier detection over data streams. 2002.

[3] Yue Dong and Nathalie Japkowicz. Threaded ensembles of autoencoders for stream learning. 2017.

[4] Adel Ghazikhani, Reza Monsefi, and Hadi Sadoghi Yadi. Online neural network model for non-stationary and imbalanced data stream classication. 2013.

[5] Michiel Hermans and Benjamin Schrauwen. Training and analyzing deep recurrent neural networks. 2013.

[6] Max Kochurov, Timur Garipov, Dmitry Podoprikhin, Dmitry Molchanov, Arsenii Ashukha, and DmitryVetrov. Bayesian incremental learning for deep neural networks.

[7] Jiwei Li, Minh-Thang Luong, and Dan Jurafsky. A hierarchical neural autoencoder for paragraphs and documents. 2015.

[8] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. 2016.

[9] Pankaj Malhotra, Vishnu TV, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Timenet: Pre-trained deep recurrent neural network for time series classifi

cation. 2017.

[10] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. 2015.

[11] Marco Martinelli, Enrico Tronci, Giovanni Dipoppa, and Claudio Balducelli2. Electric power system anomaly detection using neural networks. 2004.

[12] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. 2014.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. 2014.

[14] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. Fast anomaly detection for streaming data. 2011.

[15] Xiaohong Tang and Chen Li. The stream detection based on local outlier factor. 2015.

[16] Guanyu Zhou, Kihyuk Sohn, and Honglak Lee. Online incremental feature learning with denoising autoencoder. 2012.