KDD99Cup dataset summery

|  |  |  |  |
| --- | --- | --- | --- |
|  | 10% KDD | KDD | Stmp+http |
| Size | 73M | 725M | 101M |
| Training (normal) | 6.8M |  |  |
| Validation (normal) 1 & 2 | 2M & 2M |  |  |
| Validation (anomaly) | 14M |  |  |
| Test (normal) | 2M |  |  |
| Test (anomaly) | 42M |  |  |

Related works

* **LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection (Autoencoder, Anomaly detection,batch)[1]**

The EncDecAD (short for Encoder-Decoder scheme for Anomaly Detection) is Long Short Term Memory Networks(LSTMs) based anomaly detection architecture for time series. The utilization of LSTMs enabled the model to avoid missing the temporal information in the time series. The EncDecAD model consists of two parts, namely an encoder and a decoder, which have identical LSTMs based structure. The encoder takes a mini-batch of time series as input and outputs a hidden vector, and then follows the decoder, taking the hidden vector as input and trying to reconstruct the original input mini-batch as output. The reconstruction error is the main criterion of anomaly likelihood, while the model is only trained with normal instances, so for unacquainted anomalous data, the reconstruction error is considered to be relative higher.

According to their experimental results, the EncDecAD model could separate normal and anomalous points with a large margin. It works on different kinds of datasets, even non-periodic unpredictable data. However, The model is designed only for batch data, and need to train on the whole dataset. Time series data always comes continuously along with time, and sometime also with concept drift. If we consider the velocity and volume feature of time series data, an online incremental model is then necessary. Also, in the online fashion, it is challenging to learn from a massive stream of data the optimal number of features.

* **Online Incremental Feature Learning with Denoising Autoencoders**

**(Autoencoders over streams)**

The authors introduced an incremental feature learning algorithm to determine the optimal model complexity for streaming data based on the denoising autoencoder. The main idea is feature adding and merging. Specifically, it adds new features to minimize the objective function’s residual, and if features are redundant, then merge them to prevent overfitting and obtain a more compact feature representation. The result shows that it’s a good way to learn features from a large dataset by starting with a small set of initial features and automatically adjusting the number of features. And this method leads comparable or lower reconstruction and classification error than the stationary fashion.

* **Threaded ensembles of autoencoders for stream learning**

**(outlier detection over streams, streaming autoencoders, concept drift)**

This paper proposes a multi-threaded neural network model to deal with streaming data. On each thread running a autoencoder model and two buffers connecting data stream and model, in order to avoid model being idle or data stream being delayed. The thread ensemble enables the model with continuous learning capacity. They use a reconstruction error based criterion for anomaly detection, and maintain a buffer containing only normal data for decision of anomaly threshold. In order to distinguish between anomaly and concept drift, they check the points before and after a specific data point, while continuous founded anomalies indicate the possibility to be concept drift.

However, they didn’t talk about how the arrival rate of data stream and the fluctuations in the volume of data would impact the detection of anomalies. In addition, how emergent of the concept drift happens could also be a influence factor.

**(new/emergent class detection over streams)**

Structure

Abstract

1. Introduction
   1. Background, use case examples, importants
   2. Existing anomaly detection methods
   3. Challenges of current data\Time dependency
   4. Lack of model for time series and streaming data anomaly detection
   5. Autoencoder based data streaming AD model
2. Related works
3. Outlier detection on static data
   1. Distance based
   2. Density based
   3. Neighborhood based
   4. Waveform based
   5. One-class SVM
   6. HMM based

But those models do not consider the temporal dependency

1. Outlier detection for streams
2. Deep Learning approaches: unsupervised Autoencoder based models

(solved the multi-dimensional time series anomaly detection problem)

1. EncDecAD
2. TimeNet

But those model always needs have all data in advance, could not deal with data changes (they work with stationary data).

1. Neural network based online learning architecture
   1. Add & merge
   2. Threaded ensembles of autoencoders for streaming learning

But not exactly take time series as input (didn't apply any window, namely temporal combination)

1. From the super-/unsuper-/semisupervised learning perspective
2. Basic concepts
   1. Definition of a stream
   2. Method of processing the stream
   3. Definition of an outlier (point & window)
   4. Autoencoders | LSTMs
3. An autoencoder-based framework for unsupervised anomaly detection
   1. Overview / architecture of the framework
   2. Autoencoder component initialization
      1. EncdecAD based architecture (input, output, hidden layer)
      2. LSTM Input format
      3. Reconstruction error, anomaly score, parameters

Autoencoder might get outdated--> how to update the autoencoder

* 1. Online learning for batch-based outliers
     1. Updating stratergy

* + - Model
      1. Start from scratch -- if reconstruction error continuously being high
      2. Continue training with last-seen data -- if reconstruction error shortly high
    - Parameters(mu, sigma, threshold)
      1. Update, if prediction performance bad (e.g. F-beta low, miss alarm etc.)
      2. To avoid overfitting, previous parameter still as a part of the new parameter
    1. Maintenance of dataset for retraining
* Keep storing N batches streaming data in the buffer
* Store summarization of all seen batches
* Label data that the model mistakenly predicted, to pay more attention on them by retraining

1. Experimental setup
2. Datasets description
   1. Amount of anomaly
   2. Normal, anomaly proportion
   3. Periodicity
   4. Dimensionality
3. Initialization with first n batches streaming data
   1. Wait until accumulated enough data for initializing the model
   2. Split into normal sets
4. Sn: Training normal set
5. Vn1: Validation normal set1 for early stopping
6. Vn2: Validation normal set2 for parameter learning
7. Tn: for testing of training

And anomaly sets

1. Va: Validation anomaly set for parameter learning
2. Ta: for testing of training
   1. Dropout rate of autoencoder
   2. Save to disk
3. Streaming data generation
   1. Apache Kafka
   2. Kafka setup and configuration
   3. Deal with latency problem
4. Evaluation metric
   1. #False alarm, # miss alarm
   2. F-beta for performance
   3. Reconstruction error of normal data for model fitness of data
   4. \*Area under the curve based on anomaly score

1. Experiment results
2. Hyperparameter grid search
3. Generally performance
4. When updating is triggered
5. Reaction of concept drift, non-significant anomalies
6. Comparasion of performance before/after updatng, with/without updating
7. Runtime comparasion
8. Conclusion
   1. How online learning helps the model to adjust the stream trend
   2. Is the general performance comparable to troditional batch approaches
   3. Possiable reasons of suboptimal performance during experiment
   4. Future works

Abstract

Data stream is a data format appears in plenty of big data research scenarios, for example, manufactural sensors, production line data etc. Here anomaly detection plays an important role for use cases like predictive maintenance, event detection, and could potentially avoid large amount of financial costs. However, different from traditional anomaly detection tasks, anomaly detection in streaming data is especially difficult while data comes along the time with latent changes, so the model doesn’t fit the data all the time.

In this paper, we introduce a novel autoencoder based anomaly detection methods specially designed for streaming data. The model takes mini-batches of data from the stream as input, and try to reconstruct it using autoencoder, and the anomaly likelihood is informed from the reconstruction error. Experimental results shows that our model can sufficiently detect anomaly from data stream and update model online to fit the latest data.

**Key words**

LSTMs, autoencoders, anomaly detection, online learning

Introduction­

(Problem introduction, problem importance)

Anomaly detection is an important problem in data mining, and widely used in the manufacturing industry, commercial world, internet company etc. It could avoid or reduce lose in many scenarios like machine health monitoring, credit card fraud detecting and spam email classification, and could also be used as a preprocessing step to remove anomalies for datasets. There are already plenty of anomaly detection and outlier detection techniques proposed in literature, that solve this problem from variety perspectives, e.g. distance-based methods, clustering analysis, density-based methods etc.

(why problem hard, short coming of previous works)

There is no lack of approaches that perform really good for anomaly detection, however, most of them are focusing on batch data, 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 also been changed. Specifically, the IoT application. Assume that we collect data from sensors attached to IoT devices, the data comes continuously and everlasting. During data analysis, we should always consider the volume and velocity of data, which means, on one hand, with traditional batch-based classifier, the infinity data stream will lead to out of memory, on the other hand, streaming data usually comes with a high speed that leaving the system few processing time. 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 definition of normal and anomalous automatically. 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.

(short coming of approaches similar to this paper, problems need to be solved, lead to inspiration & good ideas)

Malhotra et al. introduced similar autoencoder based anomaly detection approaches in [1],[2], and achieved good performance in multiple different time series dataset. However, in this approach, they assume that the whole datasets are available beforehand, and didn’t considered the aforementioned online learning difficulties. Hence, we enhanced this kind of autoencoder based anomaly detection approaches with the online learning ability by using incremental knowledge learning and model updating strategies based on the streaming data.

(high level ideas and concepts of approach in this paper)

In this paper, we introduce a novel and robust incremental autoencoder-based anomaly detection model, which designed specifically for time series data in a streaming fashion using Long Short-Term memory (LSTM) units as neurons, with also online learning ability for model updating. The model need only one pass of the streaming data. For each accumulated mini-batch of streaming data, the autoencoder try to reconstruct 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 update itself online according to criterions based on performance.

(summary of tested examples, summary of results)

(Todo: about label: semi-supervised, label for scoring, one-class for training)(Todo: we experimented with datasets: \_, \_ , \_. The model shows robustness )

Related works

There is already pretty much research based on anomaly detection, some of them referred to deal with streaming data.

**Classical machine learning based approaches**

As an important component of data mining and machine learning, anomaly detection has been investigated using plenty efficient models.

**LOF**

In anomaly detection, the Local Outlier Factor(LOF) is a common distance-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 use get a numerical factor from LOF model, it is actually hard to define a threshold automatically for the judgement of anomaly.

**OCSVM**

Another widely used model is the domain based One-class Support Vector Machine. As an unsupervised 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.

Although classical machine learning approaches can handle most of the normal anomaly detection, only few of those approaches could be directly or after some modification used for time series or streaming data, while they ignore the temporal dependency between samples.

**Autoencoder-based anomaly detection approaches**

LSTMs-Autoencoders are originally widely used for text generation. 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. As text data are relevant between sentences or in the sense of words within a sentence, it is similar to the streaming data temporal dependency problem.

Sutskever et al. \cite{seq2seq} 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. did similar research on long paragraph text or even entire document generation using LSTMs-autoencoders. Their main contribute is the hierarchical sentences representation. They learn words level, sentence level, paragraph level and document level each with respectively a LSTMs layer, so that the model captures very long-term temporal information. Moreover, they introduced a 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 shows 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 \cite{eps} 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. Sakurada et al. \cite{ dimensionalityreduction} 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. \cite{lstmad}\cite{encdecad} 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. The 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 \cite{timenet} has been shown that achieves better performance in the anomaly detection tasks. The author tells that, using a constant as input of decoder instead of read time series value improves the performance of model.

**Online incremental learning with autoencoders**

Zhou et al. 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, merging hidden layer neurons if there are information redundancy, and adding hidden layer neurons to capture new knowledge. 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 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.

Preliminaries

1. **Definition of a stream(time series, dimensionality, volume, velocity, label)**

Assuming that there are some devices or data warehouse that generate data continuously with a velocity **V** (here we only taking about numerical data). The data stream from 1st timestamp until ith timestamp is descripted as:

i=1,2,3….

Where represent the instance at timestamp t in the data stream. And we assume the volume of data stream is infinity, which means, there are always available data instances generated by the data source.

To be more generally, we consider as either univariate or multivariate, is defined as

N = 1,2….

Where is the feature space of the data stream with size N. For each instance , the label = {0,1} tells either the instance is normal or abnormal.

1. **Method of processing the stream(sliding window)**

For further online processing and detection, we generate mini-batches upon the data stream. The streaming data is accumulated as window **W,** and a mini-batch consist of one or more windows.

t=1,2,3…. , **WN**= 1,2,...

t=1,2,3…. , **WN**= 1,2,... **BN**=1,2,…

Where is a window with length **WN** start from instance at timestamp **t**, is a mini-batch consists of **BN** windows starting from window .

1. **Definition of an outlier (point & window)**

**Pointwise**

A data point (instance) is anomalous 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**

A window is anomalous if the window contains one or more anomalous data points. For most of the window-based anomaly detection algorithm, they only calculate the anomaly score of a given window, it’s hard and sometimes not necessary to find out which data points of this window are anomalous.

The target is to achieve higher true positive rate (predict normal data correctly) and while remain lower false positive rate (miss classify anomalies as normal).

1. **LSTMs and autoencoder**

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 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.

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.

A single LSTM unit can be unfolded over time. The LSTM unit take a data window as input, one data point at a specific time point for each step. Therefore, the LSTM unit extracts useful and drop useless temporal information for 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. ( TrainingandAnalyzingDeepRecurrentNeural Networks)

An autoencoder 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 () of the autoencoder, and the symmetrical component between hidden layer and output layer is called decoder (). For input X, the objective function is to find weight vectors for encoder and decoder to minimize the reconstruction error.



LSTMs-autoencoder has the same encoder-decoder architecture, while the neurons are LSTM units and connected in the way described in section \ref{LSTMs}. \Fref{fig:encdecad} 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 \Fref{fig:encdecad2}, 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 \ref{LSTMs}, 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. \cite{phraserepresentation} feeds the input sequence to the decoder for a learning phrase representation task, Malhotra et al. \cite{encdecad} feed to decoder LSTM unit at each time step the prediction of last time step as input, and in a extended work \cite{timenet} 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.

**Proposed model**

**Overview / architecture of the framework**

The proposed model is a full flow from data stream generation, anomaly detection with autoencoder-based model and online model incremental updating. Apache Kafka is used as the stream generator as shown in \Fref{fig:kafka}. 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 will be retrained when the retraining condition is triggered. As aforementioned in section \ref{sec:apachekafka}, topic is the data category mechanisms in Kafka. The streaming data are published to a topic, and the prediction results are send back to another Kafka topic for visualization.

**The Consumer2 in** \Fref{fig:kafka} is actually the core component of the LSTMs-autoencoder model. Once the initialized model is available, the online phase is then start. As shown in Algorithm \ref{alg:pipeline}, if a batch of streaming data is available, the model will start do prediction, evaluation, and check whether current batch is useful to store for later retraining.

**Kafka structured data stream generation**

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. In the experimental setting, our data source is static databases, Kafka generate real-time data stream pipeline as data source publish records to the specific topic (the data category mechanisms used in Kafka), 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.

**Autoencoder component initialization**

* + 1. EncdecAD based architecture (input, output, hidden layer)

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 <Batch\_num, Step\_num, Elem\_num>, where Batch\_num is the number of data windows contained in a mini-batch, Step\_num is the numbers of data points within each data window, and Elem\_num represents the number of data dimensionality. Here, Batch\_num and Step\_num are learned as hyperparameter in the process beginning. And on the decoder side, it will output exactly the same format data vector for each mini-batch. As introduced in last section, 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 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[sutskever et al 2014], 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, N1 for hyperparameters tuning, N2 for model training, N3 for early stopping, and scoring parameters learning, N4 for testing. And abnormal data are split into two subsets, A1 for decision of anomaly score threshold, A2 for testing.

* + 1. Reconstruction error, anomaly score, parameters

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.

**Online learning for batch-based outliers**

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. The main contribution of this paper is the incremental learning setting of the autoencoder model.

**Retraining dataset**

Normally when 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 Kafka publisher, 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 \ref{eq:score}. The system maintains two data buffers for retraining, 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. Normal data windows with average mean error over $\mu$ 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 threshold determination during retraining. To this end, when a retraining process is triggered, only wrong predicted normal data, those not well mastered, are used for retraining.

**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 lack of anomaly data in the buffer to update the threshold. To this end, we separate the updating of model and threshold, namely, when the retraining is triggered, update threshold only if there is enough abnormal data, otherwise only retrain model with the normal buffer. In case of the normal buffer reaches a predefined size, the model is retrained in a sub thread while the main thread keeps processing the stream.

**Retraining**

There are two retraining strategies, continue training and start from scratch. Once the retraining is triggered, the system examines the normal buffer. The normal buffer is divided into two parts, hard examples and extreme hard examples with the boundary being $\mu+2\sigma$. When the number of extreme hard examples in the buffer exceeds a specific proportion, it means that a great change happened in the stream, and the model is retrained from scratch. Otherwise the model still contains valuable information, so it is continue trained with the buffer data.

Alternative: We use the hidden vector as the low-dimensional representation of input data. Hidden vector of all normal data is used to check, whether a new coming data is similar to the previous normal data.

Similar to the model initial training, parameters $\mu$ and $\sigma$ are learned from a sub retraining set. They are combined with the previous parameters to generate the new one. If the anomaly buffer is large enough, a new threshold will also be learned, and combined with the previous value.

* + 1. Updating strategy

* + - Model
      1. Start from scratch -- if reconstruction error continuously being high
      2. Continue training with last-seen data -- if reconstruction error shortly high
    - Parameters(mu, sigma, threshold)
      1. Update, if prediction performance bad (e.g. F-beta low, miss alarm etc.)
      2. To avoid overfitting, previous parameter still as a part of the new parameter
    1. Maintenance of dataset for retraining
* Keep storing N batches streaming data in the buffer
* Store summarization of all seen batches
* Label data that the model mistakenly predicted, to pay more attention on them by retraining

1. Experimental setup

**Datasets description**

We use 5 datasets in our experiments, Power demand, SMTP, HTTP, SMTP+HTTP and ForestCover, those are widely used streaming datasets in the streaming data mining area \cite{encdecad}\cite{threaded}\cite{tan}. Statistical features are listed in Table. Power demand 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. 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. \cite{tan}, 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. \cite{threaded}, we defined the smallest class Cottonwood/Willow with 2747 instances as anomaly, and the rest 5 classes as normal class with distribution changes.

**Datasets separation**

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.

**Parameter tuning for each dataset** (window length, number of hidden neurons, epochs, batch size)

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.

For each dataset, 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. The aim function is \Fref{eq:target}. A good model should make the reconstruction error as large as possible in order to make the classification easier.

Where N and A are the number of normal and anomalous examples in the testing set.

Transform the observations to have a specific scale. Specifically, to rescale the data to values between -1 and 1 to meet the default hyperbolic tangent activation function of the LSTM model. (from internet)

Each experiment 10 times, in order to reduce the impact of the random initial weights of LSTMs

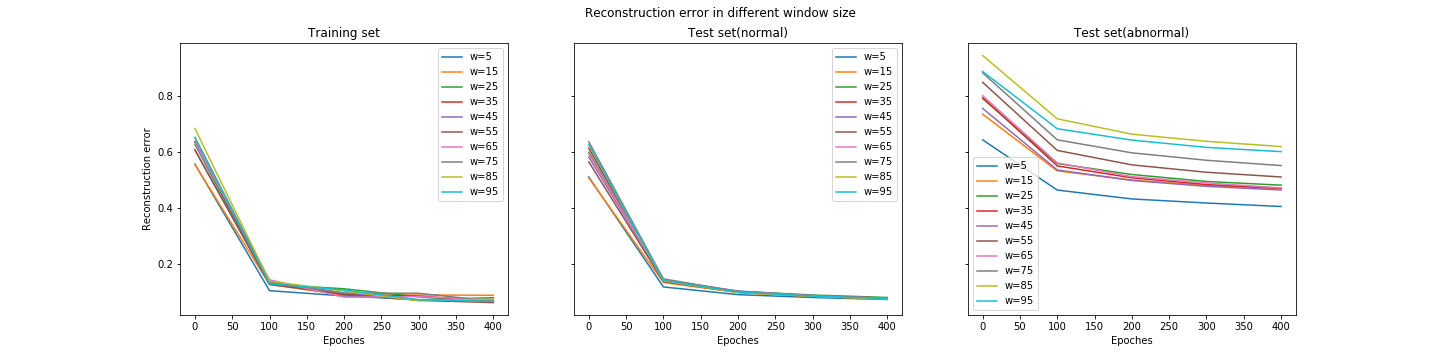
Epochs: 100 -1000, 100

Hidden neurons: 1 – 100, 5

Window length: 5 – 100, 10

Batch size: 1 – 10, 1

**Reconstruction error & window size**



Firstly, we experiment with different window size with 30 hidden units. As shown in FIGURE, with training epochs raising, the reconstruction error of all data deceases. The reconstruction error of normal testing data doesn’t change a lot with different window size, however, larger window size generally makes the anomalous data reconstruction error larger, so that could be easier to be separated from normal data. With experiment on other hidden layer size setting, the results show similar character.

In order to figure out how many hidden neurons are necessary to capture the information of input data, we experiment with different hidden layer size and window length. The target is maximum difference between normal and anomalous reconstruction error. If we fix epochs to 400, FIGURE shows that more hidden layer will make the difference larger. For the smtp dataset, there is a knee point around 15-20 neurons, the increase over 20 neurons show no more remarkable performance improvement. And in this experiment, larger window size also goes towards out target.

1. Initialization with first n batches streaming data
   1. Wait until accumulated enough data for initializing the model
   2. Split into normal sets
2. Sn: Training normal set
3. Vn1: Validation normal set1 for early stopping
4. Vn2: Validation normal set2 for parameter learning
5. Tn: for testing of training

And anomaly sets

1. Va: Validation anomaly set for parameter learning
2. Ta: for testing of training
   1. Dropout rate of autoencoder
   2. Save to disk
3. Streaming data generation
   1. Apache Kafka
   2. Kafka setup and configuration
   3. Deal with latency problem
4. Evaluation metric
   1. #False alarm, # miss alarm
   2. F-beta for performance
   3. Reconstruction error of normal data for model fitness of data
   4. \*Area under the curve based on anomaly score
   5. Retrain efficiency (time & effect)

To test the performance of the online anomaly detection system, less false alarm and more correct alarm are two basic criteria.

1. Experiment results

With parameters learned from \Fref{sec:parametertuning}, 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.

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. \Fref{fig:power\_retraining} 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. A model retraining process is triggered after the second day with last seen data, and the model performs well again on the third day.

After each retraining process, the parameters mu, sigma and threshold of anomaly scores are also updated. \Fref{fig:parachanges} shows the parameter changes over the stream.

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.

1. Reaction of concept drift, non-significant anomalies

A 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 new knowledge.

Another experiment on concept drift is to detect, how the different kinds of forest cover types in the FOREST dataset, which are all treated as normal state except type 4, relative to the model updating.

1. Comparison of performance before/after updatng, with/without updating
2. Runtime comparasion
3. Conclusion
   1. How online learning helps the model to adjust the stream trend
   2. Is the general performance comparable to troditional batch approaches
   3. Possiable reasons of suboptimal performance during experiment
   4. Future works