Unsupervised Real Time Anomaly Detection for Streaming Data

CÉSAR AUGUSTO VELÁSQUEZ PINEDA, Technische Universität Dortmund, Germany

Abstract: Streaming data is becoming increasingly important, particularly in industrial environments or IoT applications, where sensors are constantly measuring the state of variables. Detecting anomalies for such data is not paramount in order to execute control or feedback actions whenever needed, however, this comes with its own challenges in terms or the data processing involved. This paper focuses on one paradigm for anomaly detection in streaming data: the HTM or Hierarchical Temporal Memory, explains and reports on the results acquired by it in streaming applications. Furthermore, the suitability and performance of the model is compared with that of two state of the art paradigms for anomal detection in time series: LSTMs and Autoencoders. The advantages and shortcomings of ech paradigm are in the end compared and discussed, in order to provide insight on the more plausible applications for each, in the context of time series and streaming data analysis.

ACM Reference Format:
César Augusto Velásquez Pineda. 2022. Unsupervised Real Time Anomaly Detection for Streaming Data. 1, 1 (February 2022), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Streaming data from sensors prevails in most industries, and it has a growing trend with IoT devices that are key to the rise of Industry 4.0. It is therefore essential to monitor the data inflow coming from these devices, since this data allows for a good functioning of a productive process, machine, or many other applications. Detecting when the behaviour of such data is anomalous has in consequence a deciding role in deciding when and how action should be taken upon a system in order to keep it performing as desired. In this work, streaming data of time series nature is explored in this context.

When speaking about anomaly detection in time series data, a distinction needs to be done between steady and streaming data. In the former the data is analyze as a batch, i.e. a model can be learned for the whole time span comprised within it in order to look for anomalous behavior. Streaming data is, on the other hand, updated constantly, and it is typical that an anomaly detection paradigm for such data is expected to detect anomalous behavior as the data arrives, i.e. in real time, instead of batches. This is a simple but deciding aspect, since the challenges to overcome in anomaly detection of streaming data with respect to bath processing are several and very distinct, as stated by [2] and they can be summarized in the following aspects:

(1) Online processing (every time step should be identified as anomalous or not before the next one arrives).
(2) The model should be fast and adaptive to fulfill the condition 1. For this, it is important that the learning of the algorithm is continuous and not dependent of the entire dataset.
(3) Due to the, in general, high velocity of data income from sensors, human interaction, supervision or labeling of data is implausible, from which learning should be unsupervised in nature.
(4) Adaptability of models to concept drift, i.e. the possibility that the system statistics may deviate due to a change in the environment from which the data comes from (like updates on the sensors for instance).
(5) Minimization of false positives and false negatives, since otherwise the outputs of the model might end up being ignored or considered largely as not relevant.

Author's address: César Augusto Velásquez Pineda, cesar.velasquez@tu-dortmund.de, Technische Universität Dortmund, Dortmund, Germany.

In this paper, one plausible model paradigm to tackle the above issue is presented and described in detail: the HTM, whcih stands for Hierarchichal Temporal Memory. In Section 2 the algorithm is presented, both its mechanism of action, logic behind its construction and how it is used for streaming data. Then, Section 3 introduces briefly two other paradigms for the same problem at hand, the LSTMs and Autoencoders, both popular approaches for time series data in the context of anomaly detection. Upon doing that, Section 4 compares and discusses the three paradigms, again in the context of their applicability to anomaly detection in streaming data. Finally, Section 5 Summarizes the main ideas discussed throughout this text.

2 ANOMALY DETECTION AND THE HTM PARADIGM

2.1 Time series and Anomalies

A time series can be defined as a collection of observations of a single random variable at different time points, i.e. $X = \{x^{(1)}, x^{(2)}, ..., x^{(n)}\}$ where $x^{(t)} \in \mathbb{R}^m$ where m is the dimension and t is the timestamp (time interval where the variable was measured) [4]. Typically, m is equal to 1 and the time intervals are equally spaced. A time series can be steady, i.e. the dataset to be analysed is the set of observations ranging from $x^{(1)}$ to $x^{(n)}$, where no further data is added during inference and forecasts are made only based on this data observations. On the other side, the time series can consist of *streaming data*, which means that the data is continuously being gathered and updated, i.e. at the present time period the time series data has the shape $X_t = ..., x^{(t-3)}, x^{(t-2)}, x^{(t-1)}, x^{(t)}$, and at the next one it will be $X_{t+1} = ..., x^{(t-3)}, x^{(t-2)}, x^{(t-1)}, x^{(t)}, x^{(t+1)}$ [2]. As a consequence, while producing forecasts on the data, a model fitted to streaming data will always need to update itself based on the last data collected in order to produce accurate forecasts based on the latest gathered data.

Anomalies in the context of time series analysis, anomalies correspond to instances (single observations) or collective instances (multiple observations) whose values deviate from the expected [7]. In this context, *unexpected* is also to be understood as *statistically unlikely*. How the deviation is measured depends on the specific anomaly detection paradigm implemented. Figure 1 shows the main types of anomalies in time series data.

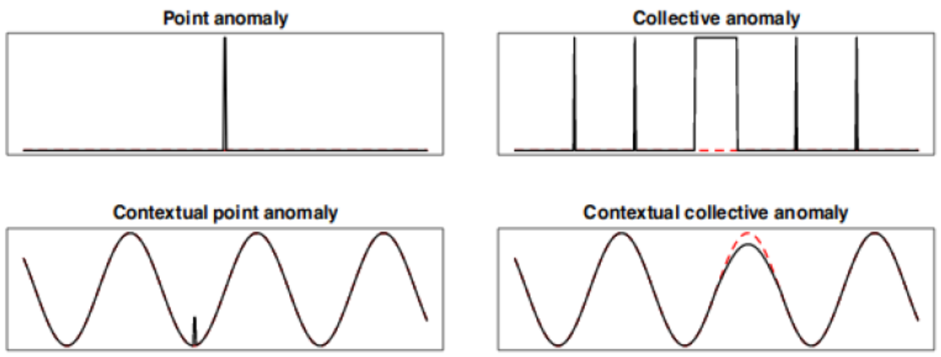

Fig. 1. Different types of anomaly in univariate time series data. Taken from: [4]

2.2 HTM algorithm overview

HTM, or Hierarchical Temporal Memory is an algorithm developed by Numenta [2], which aims to reproduce in a more Biologically similar way, the human processing logic as compared to neural networks.

In the context of time series data, the HTM model is used in order to make a prediction of the system state in the next time step x_{t+1}. Using this prediction, once the actual value is known, a prediction error can be calculated, which is in turn used to calculate an anomaly likelihood, which is the one that will in the end conclude whether this observation corresponds to an anomaly or not. Overall, the anomaly detection block diagram using HTM is showed in Figure 2. The HTM block is a whole process on its own, whose block diagram is further explained in Figure 3.

As can be seen in Figure 3, the HTM algorithm takes the current time state of the system (a sensor measurement at the current time for example) and feeds it to an *encoder*. The encoder creates an embedded representation of x_t in the form of a binary sparse array. This array is passed to a *spatial pooler* which has the task of guaranteeing that the level of sparsity in any embedded signal x_t is the same, i.e. the ratio of the number of ones with respect to the number of zeroes is constant, all this while maintaining the semantic meaning of the embedding. This representation is then fed to the most important building block of the HTM, which is the *sequence memory*. The memory is represented as an array with several mini columns (as shown in Figure 4). According to both the present state x_t and the past observations $(x_{t-1}, x_{t-2}, ...)$ the cells (circles inside the grid) of the memory are activated. This structure helps the algorithm learn patterns and make predictions based on both the present state but also the context it comes from. With this mechanism, the model makes a prediction $\pi(x_t)$ for the next state, i.e. for x_{t+1}. In Figure 4, it can be seen for instance, that even though B and C are present in the sequence in the same relative positions, the active cells differ (although the activated columns are the same), because they appear in a different context (with a sequence starting in A with respect to a sequence starting with X), which in turn leads to a different prediction based on this past context and different cell activations.

2.3 Prediction Error

As mentioned before, x_t is encoded via the encoder and the spatial pooler into a sparse binary representation, namely $a(x_t)$. Now, according to Section 2.2, the prediction of the HTM algorithm for the current time step was done while passing in the observation $x_t - 1$, yielding the prediction $\pi(x_t - 1)$ (which is also represented in the form of a sparse binary vector of the same size as a_t). The prediction error is calculated using both of this measures as follows:

$$S_t = 1 - \frac{\pi(x_{t-1}).a(x_t)}{|a(x_t)|}$$

The prediction error, is 0 if $\pi(x_{t-1})$ is exactly the same as $a(x_t)$ i.e. if the predicted representation turns out to be the materialized real observation. It is 1 on the other side if $\pi(x_{t-1})$ have no bits in common with $a(x_t)$ i.e. if they are orthogonal. It is noteworthy that the prediction error can adjust to concept drifts, since at shift points it will peak, but then subside as the system converges to the new normal metrics, as the concept drift settles.

2.4 Anomaly likelihood

The anomaly likelihood is the final output of the anomaly detection model with HTM. The prediction error is not used directly as a measure to conclude on whether an observation is anomalous or not because of the inherent noise that may exist within the data. Sometimes peaks may occur, but they may not represent themselves an anomaly, from which just thresholding the precision error would lead to many false positives (many events tagged as anomalous when they are actually not). Because of this possibility, the anomaly likelihood is computed from the prediction error. This measure is based on a rolling normal distribution over the last predicted error values within a sliding window with size, i.e. number of observations W. This rolling distribution has the sample mean (μ_t) and the sample variance (σ_t^2) as parameters, which are calculated as follows:

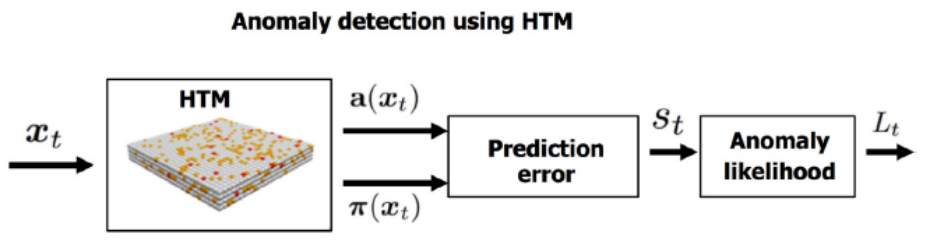

Fig. 2. Block diagram of data flow through the HTM Anomaly Detection algorithm. Taken from: [2]

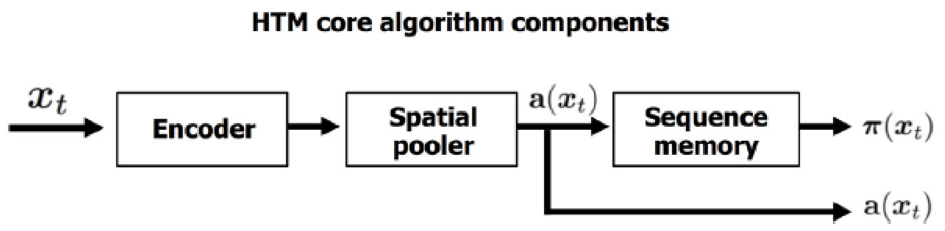

Fig. 3. Block diagram of the core HTM algorithm (HTM block in Figure 2). Taken from: [2]

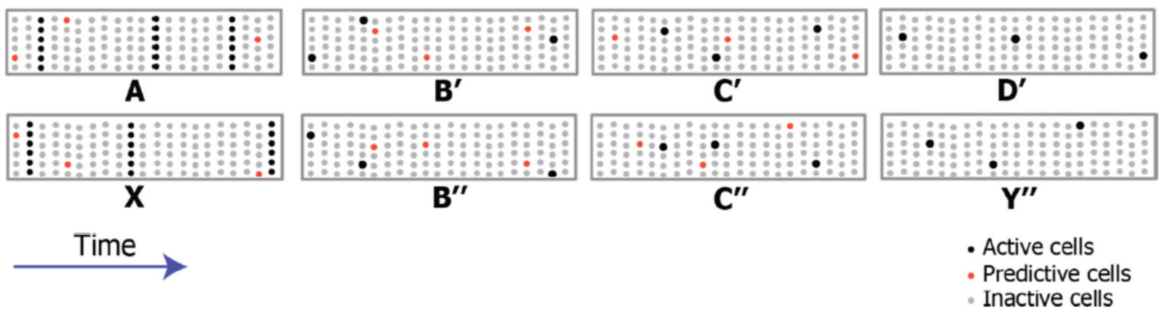

Fig. 4. HTM sequence memory illustration. Taken from: [2]

$$\mu_t = \frac{\sum_{i=0}^{i=W-1} s_{t-i}}{W}$$

$$\sigma_t^2 = \frac{\sum_{i=0}^{i=W-1} (s_{t-i} - \mu_t)^2}{W-1}$$

The short term average of the prediction errors is calculated as:

$$\tilde{\mu_t} = \frac{\sum_{i=0}^{i=W'-1} s_{t-i}}{W'}$$

Where W' is a window for a short term moving average, and $W' << W$. From these measures, the anomaly likelihood is calculated as the complement of the tail probability:

$$L_t = 1 - Q(\frac{\tilde{\mu_t} - \mu_t}{\sigma_t})$$

L_t is therefore the value that is thresholded via a user defined value ϵ, as:

If $L_t >= 1 - \epsilon$, then an anomaly is detected. According to [2] for values very close to zero, it would be very unlikely to get alerts with a likelihood higher than the threshold value ϵ.

2.5 Multiple data sources simultaneously

Until now, everything that has been described involves one data source, i.e. one time series from which the anomalies are detected. However, anomalies in many industrial environments may depend on the measurements of multiple sensors simultaneously, which brings the challenge of the multidimensionality of the input data. [3] states that the framework described in Section 2.2 can be extended to multiple models. In this approach, all data sources are combined into a single anomaly likelihood before executing the thresholding, because this way interactions between them can be accounted for while detecting the anomalies (otherwise each model would just have an anomaly score, and decisions would still be made out of individual anomaly scores per sensor instead of from the combined effect of the data, as Figure 5 shows.

For each data source ($M+1$ in this example), the HTM architecture is implemented, in order to output the prediction errors S_t^0 to S_t^M. Then, all of these predictions errors are combined into a function that computes the overall anomaly likelihood L. In [3] it is assumed, for the sake of simplicity, that the underlying distributions of each of the $M+1$ prediction errors are independent (although in reality this not always holds, for instance, when there are multiple sensor measuring variables of the same process, in which case their values may be correlated to some extent) from where the joint anomaly likelihood is calculated as:

$$L_t = 1 - \prod_{i=0}^{M-1} Q(\frac{\tilde{\mu_{t_i}} - \mu_{t_i}}{\sigma_{t_i}})$$

3 RELATED WORK

HTMs are naturally of interest for implementation in applications with streaming data. However, there are other paradigms that are currently subject of research for the similar applications. In the following setions, two of the most relevant ones, namely LSTMs and Autoencoders will be briefly explained and discussed, in order to discuss and compare the different approaches.

Anomaly detection paradigms for time series can be grouped in two main categories, namely forecasting-based and reconstruction-based. The former are those that intend to, at each time step, forecast the value of the response variable, after which an error measure is calculated once the actual observation is measured. From this forecast error, anomalies are determined and flagged as such. The HTM model, as well as LSTM Recurrent Neural Networks (which will be hereafter discussed) work typically under this paradigm. On the other side, the reconstruction-based models intend to, given some data, reconstruct it through learning its patterns. The reconstructed data is compared to the actual data, and the anomalies are detected according to which parts of the data present a reconstruction error higher than some accepted threshold. Autoencoder Neural Networks are a good example of this kind of anomaly detection paradigm.

3.1 LSTM Neural Networks

RNNs, or Recurrent Neural Networks, are a type of neural network that can model sequential data, and they do so by passing information from on time step to the next one, being able to capture temporal dependencies and learn from them. Examples of data that are modeled through RNNs are time series, speech or written language, music, and any other kind of data that has a sequential temporal dependency. The LSTM, or Long Short Term Memory model is a kind of RNN that was developed in order to allow the traditional RNNs to capture longer contextual dependencies within a sequence. This is naturally advantageous, since it can learn dependencies further away in time than a simple RNN. The structure of an LSTM is shown in Figure 6.

Within a single cell of an LSTM, the cell state h_t is the main variable implicated. It has the information of the sequence, both the present state and the context that the network decides to keep through learning (and it does so through keeping, updating and dropping information using multiple gates appearing as σ in the figure and transformations (tanh in a typical LSTM cell as the one showed). Because of the gating structure, if some data from several time steps in the past is considered relevant to predict the present state during learning, then the gates will allow to keep this information flowing from one time step to the next one to preserve its semantic meaning in the encoded data.

When applied in the context of anomaly detection, LSTMs work similarly to the HTM algorithm, i.e. the network makes predictions on the next time step(s) and the result is compared to the actual observed value. An anomaly score is calculated as a function of the discrepancy between the predicted and observed (the higher the discrepancy, the higher the anomaly score). The final classification "anomalus/not anomalous" is finally made through a threshold parameter ϵ, analogous to what was shown in Section 2.4. When the anomaly likelihood (probability) is higher than this threshold, then an anomaly is detected. Some of the features of LSTMs relevant to the anomaly detection problem in time series are:

Specifically, in the context of anomaly detection in time series data, time series fall under the forecasting paradigm, as well as the aforementioned HTM algorithm. Unlike HTM, LSTM learn parameters in a supervised manner, which might be an impeachment for working with high inflow rate data. Because of the typically big network architectures an enormous number of parameters, it is also typical for LSTMs to be retrained by batches, i.e. the model parameters are not updated upon income of new data points automatically, as was the case for the HTM model. LSTMs do not need to assume independence of measurements to handle multidimensional data, since they can take such data as input directly by construction, facilitating and providing flexibility to the model in terms of the possible relation within variables therein. Since LSTMs learn by batches instead of online, they are naturally more succeptible to not adapting as fast as HTMs to concept drifts in the data.

3.2 Autoenconders

Just as LSTMs are a popular method for time series data analysis because of its power to learn temporal dependencies in order to forecast, Autoencoder Neural Networks are also worth mentioning when discussing anomaly detection, both for time series as well as for other types of data (such as images). Unlike HTM of LSTMs, autoencoders do not forecast on the data but instead recreate it based on a low dimensional representation, and the compare the recreation with the original to look for relevant discrepancies. The main mechanism of an autoencoder is depicted in Figure 7.

As can be seen in the figure, the original data (blue column) is first fed to the encoder (typically an LSTM-like neural network) which has as output a lower dimensional representation of the data (red column). Then, the original data is recreated from the lower dimensional representation via the decoder, which is another LSTM-like neural network whose output has the same dimension as the originally fed data. For many applications, autoencoders

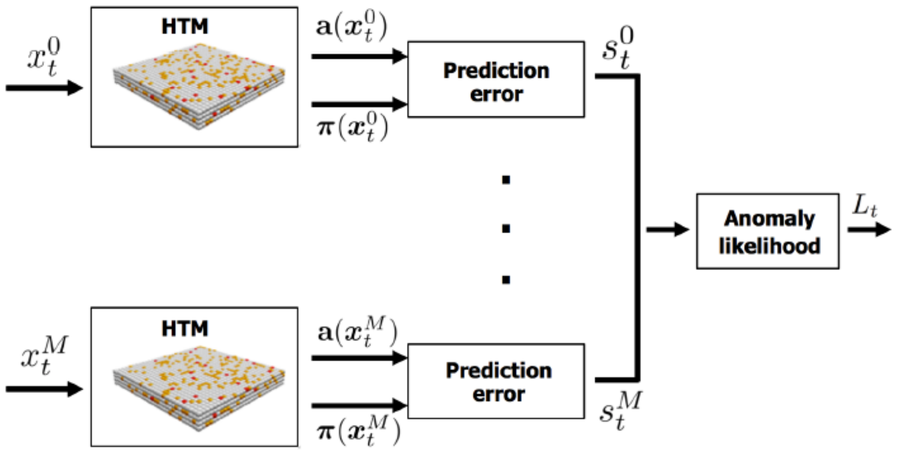

Fig. 5. System with anomaly likelihood calculation over multiple independent HTM models. Taken from: [3]

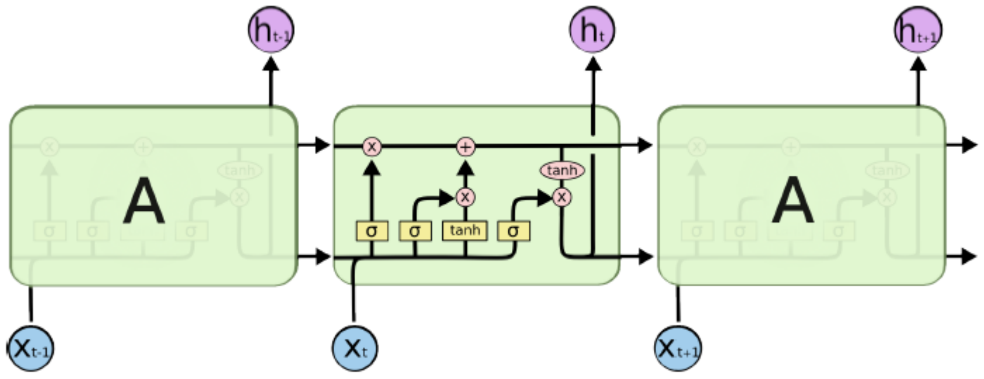

Fig. 6. Configuration of an LSTM cell. Taken from: [4]

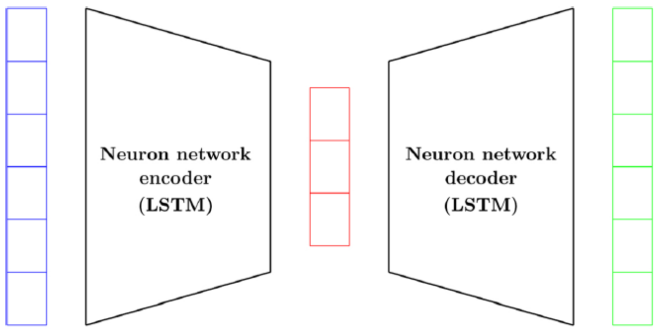

Fig. 7. LSTM Autoencoder diagram. Taken from: [6]

have the advantage of the dimensionality reduction, i.e. the data is represented in lower dimensions than it originally has. In anomaly detection approaches however, the finally recreated representation is the subject of interest. Since the whole structure learns the main patterns of the data, the decoded representation is typically denoised, as compared to the original data. Then, the reconstructed data is compared against the original and if their difference go beyond some threshold, then they are classified as anomalies. Similiarly to the HTM approach described in Section 2.4, the threshold can be dynamically selected using a probability distribution on the data (e.g. a normal distribution of the non-anomalous data) as made by [1].

Even though typically autoencoders are used for static data, they can also be adapted and used for streaming data, as done by [1]. The model is trained in batches (not completely online learning, where updates are made to the model after each new data point arrives, but every number of observations instead) since the number of parameters of an LSTM neural network make it virtually impossible to be retrained that often. Notwithstanding, [1] claim that an Autoencoder is able to keep track with contextual and point anomalies, and also keep up with concept drifts accordingly, which makes them very attractive for streaming data applications.

In the context of anomaly detection, autoencoders have, just like HTMs the advantage of being unsupervised models, i.e. no human labeling or interaction is explicitly required. Because of the similar construction (neural network based) autoencoders can, just like LSTMs handle multidimensional data easily are more flexibly in comparison to HTMs, without needing to make further assumptions on the data distributions.

3.3 Evaluation metrics

In order to compare learning models and perform model selection for a particular application, one or several performance metrics are to be implemented. Table 1 shows the most common performance metrics, that apply to all the anomaly detection paradigms hereby considered [5].

A high recall (also often called sensitivity), as explained in Table 1 implies that a high number of the actual anomalies are being detected by the algorithm, but it disregards the amount of false positives (not actual anomalies detected as such), contrary to what Precision does i.e. if there are too many false positives after evaluating a model, the precision may be too low, even if the recall is of a 100%. The F1 Score weights both measures into one, being therefore affected both by false positives and false negatives (anomalies that are not detected as such). Depending on the application, the recall and the precision might have a different relevance. For instance, in medical applications it might be paramount that an algorithm does not skip any anomalies (i.e. does not have or have minimal false negatives) whereby a high recall is mandatory. On the other side, in finantial applications, where anomalies can mean e.g. fraudulent transactions, it is also really important to keep the precision as high as possible, since otherwise many transactions are detected as fraud, when they are actually legitimate, which causes both trouble for customers as high costs to the implicated companies. Thus, choosing the evaluating metrics is also a key aspect of anomaly detection analysis and model selection.

3.4 Experimental results

In order to discuss the advantages and disadvantages of one or another anomaly detection paradigm, aspects such as the architechture and model itself are relevant, but naturally, performance is also a key factor. [4] compared HTMs and LSTMs architectures for anomaly detetion in both artifitially modified datasets (where noise was introduced with different levels of intensity to the data) and real world datasets. Figure 8 shows a comparison of these two paradigms in an artificial dataset, where the metrics are represented as a function of the level of noise introduced to the dataset. It is noteworthy that both HTM and HTM* models represent the same model, but the metrics for the HTM* are calculated only using the test data (as well as for the LSTM model), whereas the HTM uses the whole time series, since it is updated at each data point (unlike the LSTM).

The results obtained by [4] in both artificial and real-world datasets point towards an important remark: The HTM algorithm shows a higher recall than the LSTM (i.e. most of the actual anomalies

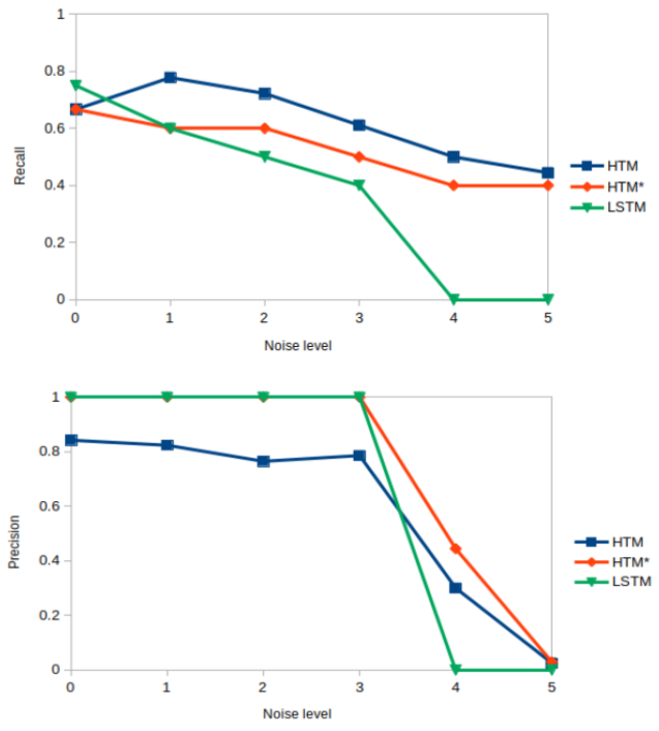

Fig. 8. Comparison LSTM and HTM in artificial datasets with different levels of introduced noise. Taken from: [4]

are detected) particularly as the level of noise is higher, which is tipically the case for e.g. IoT applications. This high recall can be benefitial when it is too sensitive to miss an anomalous behavior, which would happen if the recall is not close to 1. On the ther side, LSTMs are in general more precise, i.e. they have less false positives. Although [4] did not include Autoencoder architectures in their comparison, [8] compared several Neural network architectures, including a forecast-LSTM and Autoencoder LSTM. Therein, the Yahoo Webscope S5 dataset was used for fitting and comparison of the evaluation metrics. The Autoencoder variation showed some improvements with respect to the original LSTM, as shown in Table 2. It is noteworthy, that both approaches have a similar inner architechture, wehre each cell in the neural network is an LSTM cell, but the LSTM approach forecasts the observations, whereas the autoencoder reconstructs the series.

4 COMPARISON AND DISCUSSION

As was mentioned before, LSTMs fall under the forecasting category within the algorithms for anomaly detection in time series, just as HTMs. However, there are several discrepancies between the two. LSTMs are supervised models, which means they are trained on labeled data, which is in general more expensive and technically harder. Besides that, LSTMs are trained on batches, i.e. they are not well-suited for online learning or being updated at every new data

Table 1. Performance metrics, anomaly detection. Adapted from [5]

Metric	Definition
Precision	Number of actual anomalies detected in relation to all detected anomalies
Recall	Number of detected anomalies in relation to all actual anomalies
F1-Score	Measures the quantity of any false detections (based on both precision and recall)
ROC	A curve that visualizes the ratio of correctly detected anomalies against incorrectly detected anomalies for varying thresholds
Area under curve (AUC)	Integral under the ROC. A high value represents a model with high recall and low false positive rate

Table 2. Comparison LSTM and Autoencoders (LSTM-based and Convolutional LSTM-based autoencoders) in real world datasets. Taken from:[6]

Network	Recall	Precision	F1 Score
LSTM	0.88	0.93	0.90
LSTM Autoencoder	0.94	0.97	0.96
Convolutional LSTM Autoencoder	0.97	0.99	0.97

point coming from sensors or any data source. It is a decision of the modeler, after how many new data points the model should be retrained, and this is in turn dependent on many factors such as the speed of data streaming and the time it takes for the model to retrain. Since typically, prediction in time series tends to have a smaller precision whenever the forecast period is further away from the current time, a trade off must be made while choosing the batch size used for training, so as not to lose forecast precision. The fact that LSTMs need to wait until the batch sized for retraining is achieved, makes them less adaptive to concept drifts in the data (or very computationally expensive if trained too often).

As was already mentioned, HTM has the capability to update itself online, i.e. upon receiving new data points, whereas the two other types of models perform retraining by batches of data, due to their much bigger model construction and parameter count. One big disadvantage of LSTMs is that they need labeled data, making them supervised in nature. Because of the high rated of incoming data in a streaming data monitoring application, acquiring such labeled data might not be achievable or realistic. However, for data whose measuring rate is not extremely fast, human labeling might be an option and LSTMs would be a suitable solution to implement. Autoencoders do not suffer from this, since they are unsupervised by nature. However, as stated by [5], in order to further highlight the possible anomalies while calculating the reconstruction error from an autoencoder, preprocessing or feature extraction techniques are applied ro the data before feeding it to the autoencoder, which implies more work in the data pipeline. One important advantage of LSTMs and autoencoders with repect to HTMs is that they can handle multidimensional data in a much more simple and flexible way, since their model architechtures are built to accept any kind of incoming data dimensionality, and their model mechanics can model complex non linear relationships in the data, whereas in HTMs multidimensional data must be handled in a more manual way, stacking several HTMs together to join the prediction errors of each separate measure as was described in Section 2.5.

According to [4], while deploying HTMs in comparison to LSTMs for real world time series data, HTMs had comparatively more false positives but better recognition of point anomalies i.e. a higher recall. On the other side LSTMs showed a better precision, at the expense letting go some actual anomalies undetected. As was discussed earlier in this text, such a model would make a better fit for e.g. finantial operations, where a low false positive rate is desired to minimize operational costs and trouble, whereas for health related applications it would be more unacceptable to have a high number of undetected anomalies, which could potentially be life threatening signals for an individual. Furthermore, since both HTMs and Autoencoders work in an unsupervised manner, they should be preferred for applications in manufacturing and industry 4.0, where many sensors are to be monitored simultaneously and automatically, without the aid of labeling from an expert, since it would be timely impossible to keep track with. The choice between HTMs and Autoencoders would then depend on how critical or often concept drifts might occur, or how many and how interrelated the variables measured by sensors are. If concept drifts are critical and fast adaptability to them is paramount, then HTMs are likely better than autoencoders, because of their online learning capabilites. On the other side, if there are many potential interrelations between several sensors sending streaming data, autoencoders might potentially be a more robust solution, since they can handle multidimensional data and more complex dynamics within it in a more comprehensive way and with less human intervention as compared to HTMs.

5 SUMMARY

Anomaly detection for streaming data is paramount for many applications, and increasing with the rise of data automation in industry, where IoT and online data processing is becoming the rule. This paper reviewed, compared and discussed three methods used for the processing of streaming data in order to detect anomalies.

The First model involved was the HTM (Hierarchical Temporal Memory) model, which stems from the forecasting paradigm of anomaly detection, i.e. the model first predicts the upcoming observation and the anomaly scoring and likelihood are calculated as a function of the distance between this forecast and the actual observed value. HTMs encodes each temporal data observation of a univariate time series into a sparse binary representation and uses a sequence memory scheme to capture the temporal relationships and context (relevant information from previous observations) from the data in order to make predictions. The comparison between the forecast and the actual observation allow the model to estimate the likelihood or probability of an observation being anomalous. Since the HTM scheme is constructed for unidimensional data, it makes an independent distribution assumption on the data in order to handle anomaly detection from various sources simultaneously.

The LSTM (LOng Short Term Memory) model was also briefly discussed. This is a Neural Network based model, which also works under the forecast paradigm. LSTMs have the advantage over HTMs of handling by construction multidimensional data, since they can handle virtually any sequential input data. However, they learn much more slowly and possibly do not have the capacity to perform online learning for many applications because of their very robust architecture, with many thousands of parameters to estimate. Autoencoders are similar to LSTMs in their model architecture and flexibility to handle multidimensional data. Autoencoders work however under the reconstruction paradigm, attempting to rebuilt a window of the time series data, and then computing the reconstruction error by overlapping the reconstruction with the original data. The anomaly score is in this case a function of the reconstruction error.

While comparing the performance of the three approaches, it turns out that HTMs are probably most suitable for online learning applications, especially when the inflow of data is particularly fast, as might often happen in IoT related applications. On the other side LSTMs or Autoencoders might be preferable when a higher precision in the detection is relevant, or when it is desired to handle simultaneously several sources of data that might be interdependent. In such case, LSTMs or Autoencoders can capture or model complicated non-linear dependencies within the data that HTMs typically cannot model as easily or without human intervention. Some approaches in the literature are attempting to further improve Autoencoders, in order to make them almost comparable to HTMs in the sense of their capacity to process and learn from the data in an online manner, which is its current main shortcoming for anomaly detection of streaming data.

REFERENCES

[1] Raghav Agarwal, Tanishq Nagpal, Dibyajyoti Roy, and Aju D. 2021. A Novel Anomaly Detection for Streaming Data using LSTM Autoencoders. (2021).
[2] Subutay Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing* (2017).
[3] Subutay Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. 2017. Unsupervised real-time anomaly detection for streaming data, Supplementary Material. *Neurocomputing* (2017).
[4] Josef Haddad and Carl Piehl. 2019. *Unsupervised anomaly detection in time series with recurrent neural networks.*
[5] Benjamin Lindemann, Benjamin Maschler, Nada Sahlab, and Michael Weyrich. 2021. Survey on anomaly detection for technical systems using LSTM networks. (2021).
[6] H.D. Nguyen, K.P. Tran, S. Thomassey, and M. Hamad. 2020. Forecasting and Anomaly Detection approaches using LSTM and LSTM Autoencoder techniques with the applications in supply chain management. (2020).
[7] Baron Schwartz and Pritam Jinka. 2015. *Anomaly Detection Monitoring: A Statistical Approach to Time Series Anomaly Detection.*
[8] Chunyong Yin, Sun Zhang, Jin Wang, and Neal N. Xiong. 2022. Anomaly Detection Based on Convolutional Recurrent Autoencoder for IoT Time Series. (2022).