





Lambda Architecture for Anomaly Detection in Online Process Mining Using Autoencoders

Philippe Krajsic¹  and Bogdan Franczyk^{1,2} 

¹ Leipzig University, Grimmaische Str. 12, 04107 Leipzig, Germany

krajsic@wifa.uni-leipzig.de, bogdan.franczyk@ue.wroc.pl

² Wrocław University of Economics, ul. Komandorska 118-120, 53-345 Wrocław, Poland

Abstract. The analysis of event data in the context of process mining is becoming increasingly important. In particular, the processing of streaming data in the sense of an real-time analysis is gaining in relevance. More and more fields of application are emerging in which an operational support becomes necessary, i.e. in surgery or manufacturing. For proper analysis a cleanup of the streaming data in a pre-processing step is necessary to ensure accurate process mining activities. This paper presents a blueprint of a lambda architecture in which an autoencoder is embedded that is supposed to allow unsupervised anomaly detection in event streams, like incorrect traces, events and attributes, and thus will help to improve results in online process mining.

Keywords: Anomaly detection · Autoencoder · Lambda architecture · Process mining · Streaming data

1 Introduction

In the spectrum of process mining [1] the problem of the analysis of event data is increasingly getting attention. Existing analysis methods typically assume that the input data is completely free of incorrect data and infrequent behavior, which does not usually correspond to reality [2–4]. Incorrect data in the event streams can lead to incorrect results during further processing. For example, the accuracy of drift detection can be negatively affected by stochastic vibrations due to inaccurate event streams [5, 6]. Approaches that have been conducted in this area using filtering techniques to eliminate erroneous events from event data show an improvement in the quality of process mining techniques which leads to an optimization of the analysis of the processes [7–9].

This work presents an architectural blueprint that not only represents the recognition of anomalies as such in a business setting, but also how such mechanisms can be embedded in an online process mining environment.

In order to take full advantage of the possibilities offered by anomaly detection methods, it is necessary to transfer these methods to an online setting. This enables the use of anomaly detection methods and subsequent process mining techniques in operational support. This allows processes to be influenced in real-time due to deviations in process flow. To enable live anomaly detection, the associated technologies must enable the

processing of data streams and, in the process mining context, the processing of event streams. The requirements for the processing of streaming data in a real-time setting must be taken into account in the design of the architecture framework.

The contribution of this work to information system research is as follows: An anomaly detection framework in process mining context for embedding in a process mining environment is developed. In addition, the architecture takes into account the processing of data streams on the basis of suitable infrastructures, such as lambda architecture. The anomaly detection itself is based on an unsupervised autoencoder approach. The unsupervised autoencoder approach works without prior knowledge of the respective process and can therefore be used for different use cases independently of the domain. This enables universal use of the anomaly detection framework. In summary, it will answer the following research question:

- **RQ:** How can an architecture with embedded anomaly detection look like in an online process mining environment?

The remainder of the paper is structured as follows: Sect. 2 presents related work in the area of anomaly detection approaches and architecture concepts for streaming data in process mining context. Section 3 describes the lambda architecture approach. Section 4 subsequently describes the general functionality of unsupervised autoencoders for anomaly detection as well as the process of anomaly detection used in this work. In Sect. 5 the architecture with embedded autoencoder in an online process mining setting is presented. Section 6 shows the results of a preliminary conducted experiment. The final Sect. 7 concludes the paper and presents future work.

2 Related Work

Regarding the approaches to filtering methods of event data in the context of process mining, the current state of research can be described as follows. The related work done in the areas of online process mining and anomaly detection is of particular importance for the work presented in this paper. With regard to detection and filtering of anomalies in event logs, there are some approaches described in the literature. In [8] a reference model is used to detect inappropriate behavior and repair the affected log. The approach proposed in [9] is based on an automaton which is modeled on the frequent process behavior recorded in the logs. Events that cannot be reproduced by the automaton are removed. [11] proposes an approach that uses conditional probabilities between activity sequences to eliminate events that are unlikely in a particular sequence. While existing techniques for filtering anomalies from event-logs show that they help improve process quality, they cannot be applied to the filtering of event streams in an online context. In [10] an approach using autoencoders is proposed that reproduces their input and allows anomalies to be identified in event logs without prior knowledge. The autoencoder can also be trained on a noisy dataset that already contains anomalies.

A stand-alone approach for filtering anomalies in event streams is offered in [12]. It proposes an event processor that allows to effectively filter out unwanted events from an online event stream, based on probabilistic automaton. The basic idea of the approach is

that dominant behavior achieves a higher probability of occurrence in the automaton than unlikely behavior. Thus, this filtering technique improves process discovery in real-time process mining.

On the other Side of the considered topic are architecture models, which enable the processing of streaming data. This area also represents a focus of research. The current interest in big data approaches has led to a renewed interest in software architectures. Most of the existing work in this context focuses on the development of processing strategies for batch architectures. However, the combination of batch and streaming requirements has so far been given little consideration. The best known approaches in this context are the lambda and kappa architectures.

Lambda architectures have aroused great interest in both industry and science [13] and have also shown that they pose a number of challenges [14, 15]. There are many studies on approaches whose basic structures are essentially based on the concept of lambda architectures. In [16–18] various approaches based on a lambda architecture that enable near-real-time processing of process data are proposed. Offline (batch processing) as well as online processing (stream processing) of the data enables efficient further processing of the data as well as scalability for various big data tasks. The possibility of online processing, thus the processing of streaming data, enables the desired interaction with the running process and in the context of this work the filtering of erroneous event data in real-time.

In contrast to a lambda architecture, in which both batch and stream processing is used, a kappa architecture [19] is a concept that only allows the processing of streaming data. A kappa architecture differs from a lambda architecture in such a way that the batch layer is omitted, which in turn simplifies the implementation of the concept. On the other hand, this type of architecture is not applicable if a batch processing engine is required. However, this is the case when using process mining techniques to process historical data [18]. For this reason, the use of lambda architecture as the primary design pattern was chosen within the scope of this work.

3 Lambda Architecture

The design pattern on which this work is built is based on lambda architecture. The concept of lambda architecture was first introduced by Marz and Warren [13] and is a real-time architecture used in big data settings. The idea behind the lambda architecture is to create typically two streams of input data, which are processed separately and then recombined again. There are three layers: batch layer, speed layer and serving layer, as shown in Fig. 1. One of the streams is processed with a real-time framework that performs various calculations to obtain relevant information. This real-time component is referred to as the speed layer, where the main objectives are low latency and high throughput. The other stream is processed with a batch framework. Since the processing is distributed, it is possible to manage a very large amount of data. The consistency of the data is guaranteed by the constant recalculation of the entire data record in real-time. This batch component is called the batch layer. Finally, both streams are combined in a serving layer to query the results of the batch layer and speed layer. The speed layer stores data as views to perform real-time analysis. The batch layer stores the data in a stable

database structure. Real-time data is regularly replaced by batch data. For example, if all data from the previous day has been completely processed in the batch layer, these results replace real-time data that has already been processed.

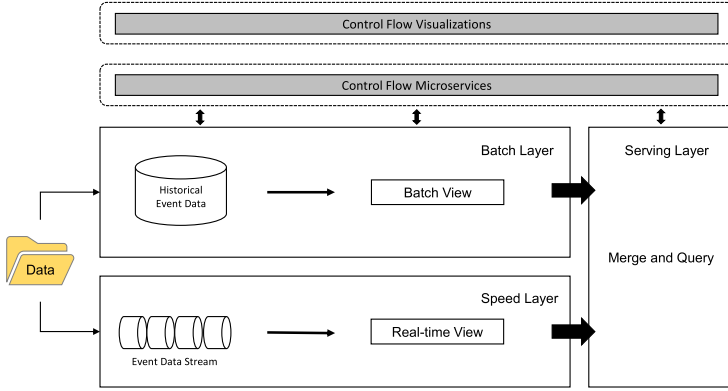


Fig. 1. Lambda architecture with batch layer, speed layer and serving layer

4 Anomaly Detection Using Autoencoders

In the context of this work, an autoencoder in a lambda architecture is used to detect anomalies in a streaming setting. Therefore, the functionality of an autoencoder and the way anomalies are detected in this context are described.

4.1 Autoencoders

An autoencoder basically consists of two components: an *encoder* and a *decoder*. Figure 2 illustrates a conventional autoencoder.

Encoder: The input vectors ($x_i \in R^d$) are compressed into m neurons that form the hidden layer. The activation of the neuron i in the hidden layer is given by:

$$h_i = f_{\theta}(x) = s\left(\sum_{j=1}^n W_{ij}^{in} x_j + b_i^{in}\right) \quad (1)$$

where x represents the input vector, θ represents the parameters (W^{in}, b^{in}), W represents the encoder weight matrix and b represents the bias vector. To achieve a compression of the input data, the input vector is compressed to a lower dimensional vector.

Decoder: The resulting hidden layer h_i is then decoded back into the original input space R^d . The associated mapping function is as follows:

$$x'_i = g_{\theta'}(h) = s\left(\sum_{j=1}^n W_{ij}^{hid} h_j + b_i^{hid}\right) \quad (2)$$

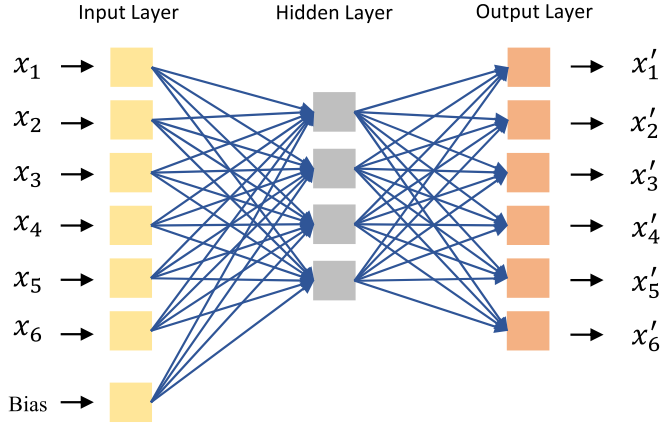


Fig. 2. Architecture of an example autoencoder

The parameter set of the decoder is $\theta' = \{W^{hid}, b^{hid}\}$. The autoencoder is optimized to minimize the average construction error related to θ and θ' :

$$\theta^*, \theta'^* = \arg \min_{\theta, \theta'} \frac{1}{n} \sum_{i=1}^n \varepsilon(x_i, x'_i) \quad (3)$$

$$= \arg \min_{\theta, \theta'} \frac{1}{n} \sum_{i=1}^n \varepsilon(x_i, g_{\theta'}(f_{\theta}(x_i))) \quad (4)$$

where ε is the reconstruction error which is defined by:

$$\varepsilon(x_i, x'_i) = \sum_{j=1}^d (x_i - x'_i)^2 \quad (5)$$

The activation functions f and g should be nonlinear functions to show the nonlinear relationship between the input characteristics. Common activation functions are sigmoid function or rectifier function [20].

The considered autoencoder in this work consists of one input and one output layer with linear units and two hidden layers with rectified linear units.

4.2 General Filter Architecture

Since this work deals with the processing of event streams, a suitable mechanism must be chosen to specify which events are to be passed to the autoencoder for further processing at a certain point in time. Figure 3 illustrates a schematic overview of the anomaly filter architecture. An infinite event stream S is assumed, which contains erroneous data such as incorrect traces, events and other attributes.

Due to the infinity of the stream S , the existence of a finite event window w is assumed, which allows to observe a finite sequence of events at time t . This event window contains new events e that were recently generated. The events contained in the event window w are passed to the autoencoder in the form of batches. These are

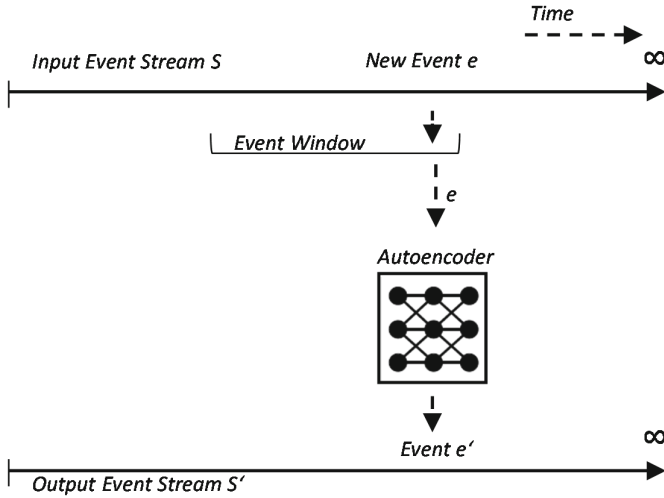


Fig. 3. Schematic architecture of the anomaly filter

then processed and filtered by the autoencoder. In order to decide which events have to leave the event window again, there are a number of stream-based approaches, such as adaptive sliding windows [21] or forward decay methods [22]. For the events received this way, the autoencoder decides which events are correct or incorrect. Incorrect events are discarded and correct events are passed to a filtered event stream S' .

4.3 Method of Anomaly Detection

Since the detection of anomalies is a classification problem, the desired output of a corresponding classification method is a class label. Through the use of neural networks it is possible to get such an output without training the neural network with class labels. A neural network suitable for such a purpose is an autoencoder. Instead of using class labels for training, the original input is used as the target output. To train the autoencoder, the individual activities and users are one-hot encoded in a first step. Each activity is encoded by an n -dimensional vector, where n is the number of individual activities contained in the corresponding training data. This results in a vector for the activities and another vector for the users of each event in a trace. The resulting vectors are then linked to form a single vector. Another way to handle variable size traces is to use n -grams. However, this method complicates the processing of long-term dependencies between events. For such reason one-hot encoding is used for this purpose. To deal with the fixed sized input of a feed-forward network, the vectors have to be pad with zeros to reach the same size as the longest vector in the corresponding training data.

After the mentioned pre-processing steps have been completed the autoencoder can be trained with the backpropagation algorithm [23] using the one-hot encoded event data. Therefore the event data is used both as the input and as the label. In addition, a special noise layer is inserted between the event data and the autoencoder. This noise layer adds Gaussian Noise to the event data. This data is used by the autoencoder as

input. The Autoencoder is trained to reproduce its input in order to minimize the mean squared error between the input and its output.

4.4 Classifying Erroneous Data

After the autoencoder has been trained, it can be used to measure the mean squared error between its input and output and thus detect erroneous data. This erroneous data can be either erroneous traces, events or attributes. Therefore it can be assumed that normal traces are reproduced with a lower reproduction error than incorrect traces. To detect an anomaly this way, a certain threshold value τ can be defined. If a certain reproduction error exceeds this threshold, the associated trace can be classified as an anomaly. To define a threshold value, the mean reproduction error over the training data set can be used. The threshold is defined by:

$$\tau = \frac{1}{n} \sum_{i=1}^n \varepsilon_i \quad (6)$$

where ε_i is the reproduction error for a corresponding trace i and n the number of traces in the data set. This procedure for identifying incorrect traces can be applied analogously to incorrect events and activities by calculating the reproduction error event based.

5 Lambda Architecture-Based Anomaly Filter in an Online Process Mining Setting

After the anomaly detection was described and constructed using an autoencoder, it is now to be adopted into the lambda architecture, as this kind of architecture is well suited for use in an online process mining setting, through its batch and streaming processing. Figure 4 illustrates the embedding of the autoencoder inside the lambda architecture as well as its role in the online process mining context. The figure gives an abstracted overview of the essential architecture components. The batch layer with the provision of historical data and the speed layer with the processing and filtering of streaming data from the real-time process can be recognized. These two layers are merged by the serving layer in which both offline and online process mining take place and lead to the discovery and analysis of process models. The filtered real-time event data used during online process mining is then transferred to the historical data storage which can later be used as historical data for new process mining tasks. The event data used for the process mining activities are generated by human users or digitized and automated processes and extracted and merged from different information systems. After the data has been extracted and collected from the information systems, they can be fed to the corresponding components in a suitable format like the XES format which maintains the compatibility with industry standards. This data serves as raw material for later analyses. The output of the algorithm used in offline process mining (i.e. heuristic miner) is a control flow model. The algorithm used in online process mining produces an update for the process model generated by the offline algorithm. These updates are then passed to the control flow visualization by means of the web sockets technology, that the user is able to see the corresponding updates of the model. In addition, a control flow

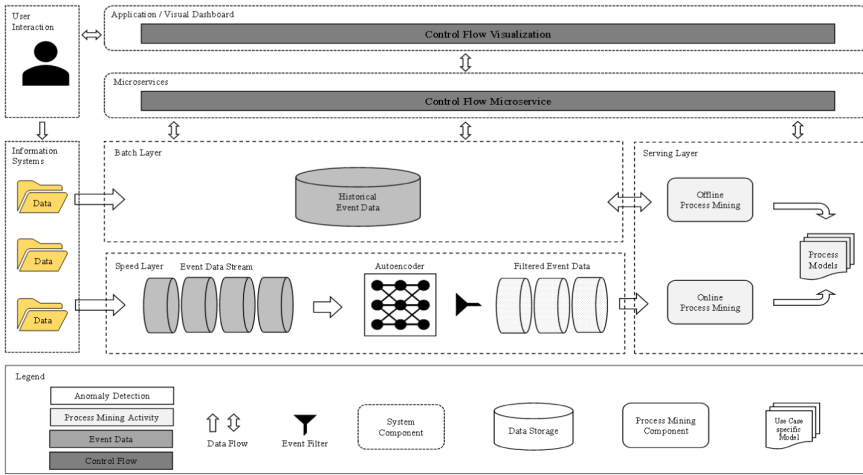


Fig. 4. Integrated autoencoder for filtering anomalies in online process mining

microservice layer is used. This enables a separation of functionalities and simplifies both the implementation and maintenance of the system.

This type of architecture and data processing also poses challenges, in particular by including neural networks, like an autoencoder, in the processing of streaming data. This leads to major problems due to the characteristics of such big data activities [25]. Especially the aspect of velocity brings challenges in a big data streaming environment. Data availability plays a critical role in this environment. Underlying models can change, making the common process of training, testing and prediction more difficult. Furthermore, the use of neural networks in a real-time setting complicates the training of the models. Here it is necessary to reduce the training time drastically due to the short decision times. One more important fact is that an online learning model is not horizontally scalable and cannot be instantiated to any number of instances due to the quickly changing models. This is challenging because this important part, the model, is only vertically scalable, thus a single model instance.

6 Experiment

6.1 Event Data

A preliminary implementation of the autoencoder is evaluated on the public available BPIC19 [26] dataset. BPIC19 consists of over 1.5 million events for purchase orders. The data shows the purchase to pay process.

6.2 Anomalous Event Data

For our experiment erroneous event data was added to the data set. For this purpose, we generated random sequences that correspond to the format of the BPIC19 data set.

6.3 Experimental Setup

Simulating a rapid deployment process, no hyper-parameter tuning was performed. In an first preliminary implementation the autoencoder is trained on batches of size 128 for 100 epochs. This allows an earlier stop if the loss of the validation set did not decrease within the last 10 epochs. Furthermore, the Adam Optimizer [24] was used. The learning rate for the model was initially set and fixed to 0.001. The Autoencoder has two hidden layers with 128 hidden units in the first and 64 hidden units in the second layer.

6.4 Results

Figure 5 shows the confusion matrix of the preliminary conducted experiment. The confusion matrix provides an overview of both the accuracy and the distribution of incorrect predictions.

Predicted	Actual	
	Anomaly	No anomaly
Anomaly	24	11
No anomaly	6	19

Fig. 5. Confusion matrix of the conducted experiment

Table 1. Confusion metrics

Accuracy	Misclassification	Precision	Recall	F1	Specificity
71,6%	28,3%	68,5%	80,0%	73,0%	63,3%

Table 1 lists the corresponding confusion metrics of the experiment. The accuracy of the preliminary model reaches 71,6%. On the other hand, the misclassification amounts to 28.3%. Furthermore, Precision (68.5%), Recall (80%), F1-Measure (73%) and Specificity (63.3%) of the model were measured.

7 Conclusion and Future Work

In this paper a lambda architecture using autoencoder for anomaly detection in an online process mining setting was presented. Therefore the preliminary model of an autoencoder was described that is used to reconstruct its input as output and calculates the mean squared error between input and output to detect incorrect data. The autoencoder as well as the filter architecture are embedded in a lambda architecture to support the processing of streaming data as well as the processing of historical data for process mining purposes. The conducted experiment showed first successful results in detecting anomalies in event data. This construct of technologies is supposed to be used in a real time environment for

operational support to reach a cleansed event stream for better process mining results. Therefore future work should include the further development of the autoencoder for the processing of event streams instead of batches. In particular, the focus is on handling streaming data as well as updating and retraining the model with fast changing data. This includes the technical infrastructure of the implementation, like the usage of the right data processing pipelines, as well as the vertical scalability of an online learning model.

Acknowledgements. This work was funded by the German Federal Ministry of Education and Research within the project Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS14014B).

References

1. van der Aalst, W.: *Process Mining: Data Science in Action*, 2nd edn. Springer, Berlin (2016). <https://doi.org/10.1007/978-3-662-49851-4>
2. van der Aalst, W., Weijters, T., Maruster, L.: Workflow mining: discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.* **16**(9), 1128–1142 (2004)
3. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: Colom, J.-M., Desel, J. (eds.) *PETRI NETS 2013*. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
4. Hassani, M., Sicca, S., Richter, F., Seidl, T.: Efficient process discovery from event streams using sequential pattern matching. In: *IEEE Proceedings of SSCI 2015* (2015)
5. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Detecting sudden and gradual drifts in business processes from execution traces. *IEEE Trans. Knowl. Data Eng.* **29**(10), 2140–2154 (2017)
6. Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A.H.M., van Dongen, B.F.V.: Detecting drift from event streams of unpredictable business processes. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) *ER 2016*. LNCS, vol. 9974, pp. 330–346. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_26
7. Ghionna, L., Greco, G., Guzzo, A., Pontieri, L.: Outlier detection techniques for process mining applications. In: An, A., Matwin, S., Raś, Z.W., Ślęzak, D. (eds.) *ISMIS 2008*. LNCS (LNAI), vol. 4994, pp. 150–159. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68123-6_17
8. Wang, J., Song, S., Lin, X., Zhu, X., Pei, J.: Cleaning structured event logs: a graph repair approach. In: *31st IEEE International Conference on Data Engineering* (2015)
9. Conforti, R., La Rosa, M., ter Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. *IEEE Trans. Knowl. Data Eng.* **29**(2), 300–314 (2017)
10. Nolle, T., Luetgten, S., Seeliger, A., Mühlhäuser, M.: Analyzing business process anomalies using autoencoders. *Mach. Learn.* **107**(11), 1875–1893 (2018). <https://doi.org/10.1007/s10994-018-5702-8>
11. Sani, M.F., van Zelst, S.J., van der Aalst, W.M.P.: Improving process discovery results by filtering outliers using conditional behavioural probabilities. In: Teniente, E., Weidlich, M. (eds.) *BPM 2017*. LNBP, vol. 308, pp. 216–229. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_16

12. van Zelst, S.J., Fani Sani, M., Ostovar, A., Conforti, R., La Rosa, M.: Filtering spurious events from event streams of business processes. In: Krogstie, J., Reijers, H.A. (eds.) CAiSE 2018. LNCS, vol. 10816, pp. 35–52. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91563-0_3
13. Marz, N., Warren, J.: Big Data. Principles and Best Practices of Scalable Realtime Data Systems, Manning (2013)
14. Hoseinyfarahabady, M., Taheri, J., Tari, Z., Zomaya, A.Y.: A dynamic resource controller for a lambda architecture. In: IEEE Proceedings of the International Conference on Parallel Processing (2017)
15. Kiran, M., Murphy, P., Monga, I., Dugan, J., Baveja, S.S.: Lambda architecture for cost-effective batch and speed big data processing. In: IEEE Proceedings International Conference on Big Data (2015)
16. Batyuk, A., Voityshyn, V.: Streaming process discovery for lambda architecture-based process monitoring platform. In: IEEE 8th International Conference on Computer Science and Information Technology (2018)
17. Batyuk, A., Voityshyn, V.: Apache storm based on topology for real-time processing of streaming data from social media. In: IEEE First International Conference on Data Stream Mining & Processing (2016)
18. Batyuk, A., Voityshyn, V., Verhun, V.: Software architecture design of the real-time processes monitoring platform. In: IEEE Second International Conference on Data Stream Mining & Processing (2018)
19. Kreps, J.: Questioning the Lambda Architecture. O'Reilly.com (2014). <https://www.oreilly.com/radar/questioning-the-lambda-architecture>. Accessed 27 Feb 2020
20. Nair, V., Hinton, G.E.: Rectified linear units improve restricted Boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning. Association for Computing Machinery (2010)
21. Bifet, A., Gavalda, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of the SDM. SIAM (2007)
22. Cormode, G., Shkapenyuk, V., Srivastava, D., Xu, B.: Forward decay: a practical time decay model for streaming systems. In: IEEE Proceedings of the ICDE (2009)
23. Rumelhart, D., Hinton, G., Williams, R.: Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986)
24. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint [arXiv:abs/1412.6980](https://arxiv.org/abs/1412.6980) (2014)
25. Augenstein, C., Spangenberg, N., Franczyk, B.: Applying machine learning to big data streams: an overview of challenges. In: 4th IEEE International Conference on Soft Computing and Machine Intelligence (2017)
26. van Dongen, B.F.: Dataset BPI challenge 2019. 4TU. Centre for Research Data (2019). <https://doi.org/10.4121/uuid:d06aff4b-79f0-45e6-8ec8-e19730c248f1>