Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders

Timo Nolle(✉), Alexander Seeliger, and Max Mühlhäuser

Technische Universität Darmstadt, Telecooperation Lab, Darmstadt, Germany
{nolle,seeliger,max}@tk.tu-darmstadt.de

Abstract. Business processes are prone to subtle changes over time, as unwanted behavior manifests in the execution over time. This problem is related to anomaly detection, as these subtle changes start of as anomalies at first, and thus it is important to detect them early. However, the necessary process documentation is often outdated, and thus not usable. Moreover, the only way of analyzing a process in execution is the use of event logs coming from process-aware information systems, but these event logs already contain anomalous behavior and other sorts of noise. Classic process anomaly detection algorithms require a dataset that is free of anomalies; thus, they are unable to process the noisy event logs. Within this paper we propose a system, relying on neural network technology, that is able to deal with the noise in the event log and learn a representation of the underlying model, and thus detect anomalous behavior based on this representation. We evaluate our approach on five different event logs, coming from process models with different complexities, and demonstrate that our approach yields remarkable results of 97.2 % F1-score in detecting anomalous traces in the event log, and 95.6 % accuracy in detecting the respective anomalous activities within the traces.

1 Introduction

Anomaly detection, or outlier detection, is an important topic for todays businesses. Companies all over the world are interested in anomalous executions within their process, as these can be indicators for fraud, or inefficiencies in their process.

More and more companies rely on process-aware information systems (PAISs) [7] to assist their employees in the execution. The increasing numbers of PAISs has generated a lot of interest in the data these systems are storing about the execution of a process. PAISs provide data analysts with a huge amount of information in form of log files. These log files can be used to extract the events that happened during the execution of the process, and hence create so called event log files. Event logs are comprised of activities that have occurred during the execution of a process. These event logs enable a process analyst to explore the process by which this event log has been generated. In other words, the event log is the leftover evidence of the process that produced it. Consequently, it is

T. Calders et al. (Eds.): DS 2016, LNAI 9956, pp. 442–456, 2016.
DOI: 10.1007/978-3-319-46307-0_28

possible to recreate the process model by evaluating its event log. This is known as process model discovery and is one of the main ideas in the domain of process mining [22].

Often, process models have been designed by experts some time in the past, but over the years the process has slowly mutated. These mutations can be caused by new employees, working slightly different than their predecessors, or the use of new technology, or simply changes in the business plan. Process changes usually happen in a very subtle way. In a procurement process, for example, an employee stops requesting the required approval of their advisor, as this speeds up the process. At first this will happen rarely, but over time it will start to overwhelm the designated behavior. In other words, these subtle changes start off as anomalies in the process, and therefore it is important to detect them early and take actions, before they can manifest.

Process mining provides methodologies to detect these changes, e.g., by discovering the as-is process model from the event log [1]; that is, generating a valid process model that is capable of producing the same event log. After the discovery of the as-is model, it can be compared to a reference model in order to detect the changes. This is known as conformance checking [18]. However, this approach requires the existence of such a reference model of some form (e.g., BMPN model, petri-net, rule set). Unfortunately, these reference models are often not well maintained by the company, or even non-existent.

The absence of a reference model is a big problem for conformance checking, and especially in the field of anomaly detection, as there is no model that defines what a normal execution of the process ought to look like. Thus, we can not define what an anomalous execution is either. Many of the current anomaly detection algorithms work by first learning what a normal example is, by training on a training set that solely consists of normal examples, and then using this knowledge to detect the anomalies, as they are much different from what they have learned about normal examples during training. However, this is not a valid assumption we can make when considering process event logs from PAISs, as they usually already contain anomalous behavior and other sorts of noise. What is a valid assumption, on the other hand, is that the anomalous executions are highly outnumbered by the normal ones, which we will take advantage of.

In this paper we propose a method to automatically split a noisy event log into normal and anomalous traces. The main contribution of this approach is that it does not require the existence of a reference model, nor prior knowledge about the underlying process. We train our system on the raw input from the event log, including the anomalous traces, and without the use of labels indicating which cases are, in fact, anomalous. The system has to deduce the difference between normal and anomalous traces purely based on the patterns in the raw data. Our approach is based on a special type of neural network, called an autoencoder, that is trained in an unsupervised fashion. We will demonstrate that our system is able to understand the underlying processes of five different event logs. Consequently, it can automatically analyze a given event log and filter out anomalous traces, based on the implicitly inferred model. Not only can

we filter out anomalous traces, but we can also infer which specific activity in a trace is the cause for the anomaly.

2 Related Work

In the field of business processes, and especially process mining [22], anomaly detection is not very frequently researched. The most recent publication [4] describes an approach where a reference model is build through the use of discovery algorithms. Then, this reference model can be used to automatically detect anomalous traces. However, this approach relies on a clean dataset, that is, no anomalous traces must be present in the data set during the discovery. As we have described earlier, this is usually not the case, as the event logs from PAISs most likely already do contain these anomalies.

The approach within this paper is highly influenced by the works in [9,12], where they propose the use of replicator neural networks [10], i.e., networks that reproduce their input, which are based on the idea of autoencoders from [11]. The approaches from [9,12], however, do not work well with variable length input.

A review of novelty detection (i.e., anomaly detection) methodology can be found in [17], where they describe and compare many methods that have been proposed over the last decades. The authors differentiate between five different basic methods for novelty detection: probabilistic, distance-based, reconstruction-based, domain-based, and information-theoretic novelty detection.

Probabilistic approaches try to estimate the probability distribution of the normal class, and thus are able to detect anomalies as they were sampled from a different distribution. However, this approach also requires a clean dataset. Distance-based novelty detection (e.g., nearest neighbor, clustering) does not require a cleaned dataset, yet it is only partly applicable for process traces, as anomalous traces are usually very similar to normal ones.

Reconstruction-based novelty detection (e.g., neural networks) is similar to the aforementioned approaches in [9,12]. However, training a neural network usually also requires a cleaned dataset. Nevertheless, we will show that our approach works on the noisy dataset, by taking advantage of the skewed distribution of normal data and anomalies, as demonstrated in [8].

Domain-based novelty detection requires domain knowledge, which violates our assumption, that we do not require any prior knowledge, only the data. Information-theoretic novelty detection defines anomalies as the examples that most influence an information measure (e.g., entropy) on the whole dataset. Iteratively removing the data with the highest impact will yield a cleaned dataset, and thus a set of anomalies. With the exception of reconstruction-based approaches, this is the only approach that can, to a certain degree, handle noisy datasets.

Within this paper, we opted to use a neural network based approach, as the recent achievements in machine translation and natural language processing indicate that neural networks are an excellent choice when modeling sequential

data. At last, we want to point out that one-class support vector machines (SVMs) [6] are usually very sensitive to outliers in the data [2], which is why we did not apply classic one-class SVMs in this setting.

3 Dataset

PAISs keep a record of almost everything that has happened during the execution of a business process. This information can be extracted in form of an event log. An event log consists of traces, each consisting of the activities that have been executed. Table 1 shows an excerpt of such an event log, in this case it has been generated by a procurement process model. Notice that an event log must consist of at least three columns: trace id, to uniquely assign an executed activity to a trace; a timestamp, to order the activities within a trace; and an activity label, to distinguish the different activities.

Table 1. Example event log of a procurement process

Trace ID	Timestamp	Activity
1	2015-03-21 12:38:39	PR Created
1	2015-03-28 07:09:26	PR Released
1	2015-04-07 22:36:15	PO Created
1	2015-04-08 22:12:08	PO Released
1	2015-04-21 16:59:49	Goods Receipt
2	2015-05-14 11:31:53	SC Created
2	2015-05-21 09:21:26	SC Purchased
2	2015-05-28 18:48:27	SC Approved
2	2015-06-01 04:43:08	PO Created

In order to create a test setting for our approach we randomly generated process models and then sampled event logs from those models. These event logs have been generated by PLG2 [5], a process simulation and randomization tool. PLG2 allows the user to randomly generate a process model and then simulate it to generate genuine event logs. It also comes with a feature to perturb the generated event log by, for example, skipping events, doubling events, or changing the sequence in which events have occurred. PLG2 was specifically designed for process mining researchers' needs, as the amount of publicly available datasets of reasonable sizes is minuscule.

We have used the PLG2 tool to generate random process models of different complexities. Table 2 shows the complexity of these models in terms of the number of distinct activities and the number of gateways in the model. The resulting models were then used to generate noisy event logs, i.e., event logs including anomalous traces. Each trace in the event log has the chance of being

Table 2. Overview over the four different randomly generated process models and the corresponding event logs

Model	Activities	Gateways	Traces	Unique	Anomalous
Small	12	2	10 000	240	978
Medium	32	12		398	973
Large	42	14		621	985
Huge	51	22	50 000	1 044	4 778
P2P	12	8	10 000	232	968

affected by any of the following mutations: skipping an event; swapping two activities, or duplicating an activity so that it appears twice in a row.

The probability that a mutation occurs has been set to 3.3 % for all the three mentioned mutations. Thus, the resulting event log will contain roughly ten percent anomalous traces, as shown in Table 2. When considering a real-life business process, ten percent anomalous traces in the event log is quite high, and thus we have chosen to set this as our upper bound. Notice that our approach ought to yield better results with less anomalous traces in the log, as it becomes easier to generalize to the normal traces. This is why we use a highly noisy event log as compared to real life event logs, where the number of anomalies is usually much smaller. Notice that we used a fixed size of 10 000 traces for the small, medium, and large dataset, and a bigger size of 50 000 traces for the huge dataset. The huge dataset required a bigger sample due to its higher complexity. The increasing complexity with every dataset is also illustrated by the number of unique traces in each log, as shown in Table 2.

Figure 1 shows a t-SNE [15] visualization of the four randomly generated datasets, depicting anomalous traces in red and normal ones in green. We can clearly recognize the clusters that are being formed by the normal traces. However, within these clusters there also lie anomalous traces, which is exactly what we want, as anomalies in real life are typically very similar to normal traces in terms of the sequence of activities.

In addition to the four randomly generated models, we also used a simplified version of a purchase to pay (P2P) process model as is depicted by the BPMN model shown in Fig. 2. This model was mainly created for the evaluation part within this paper, as it features interpretable activity names, unlike the randomly generated ones. The resulting event log for the P2P model was generated in the same fashion as those of the randomly generated models, using the same parameters as mentioned above. A trace in our P2P model can either start with the manual creation of a purchase request (PR) or the creation of a shopping cart (SC). In both cases, after the necessary approval of the SC and the release of the PR, a purchase order (PO) is created. This PO can be altered by increasing or decreasing the order quantity. After the PO has been released the orderer receives the goods, and usually in quick succession also the corresponding invoice.

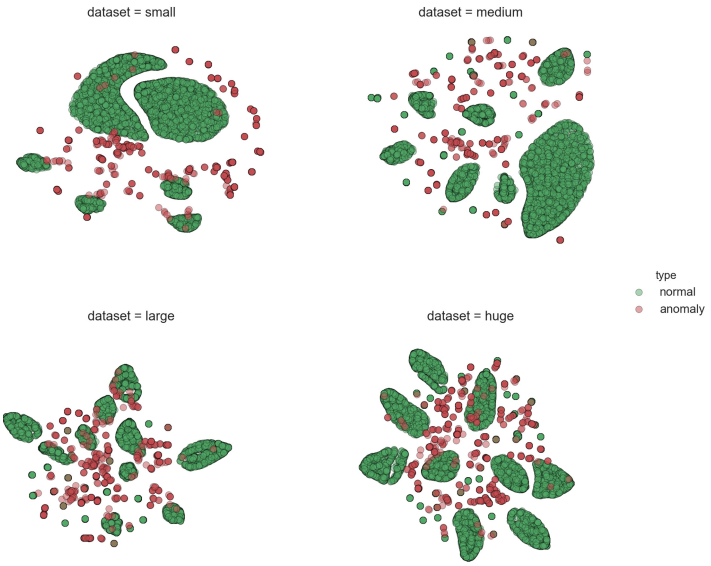

Fig. 1. t-SNE visualization of the randomly generated datasets from Table 2 (Color figure online)

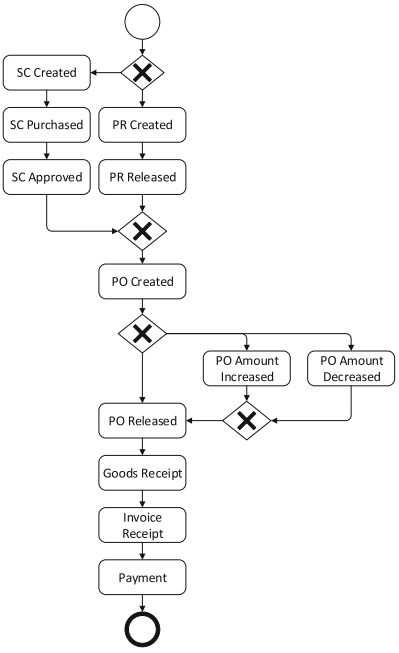

Fig. 2. BPMN model of a simplified purchase to pay process

The orderer ought to settle the invoice if and only if they have already received the goods.

Ultimately, our datasets consist of 10 000 traces (50 000 for the huge dataset), each consisting of a variable number of activities. Notice that we also assume that the event log only contains complete traces, that is, every trace starts and ends with valid activities according to the process model. The only exception to this is if either the start activity or the end activity, or both, are affected by one of the aforementioned mutations.

4 Method

Before we introduce our method, first we want to give a short overview over deep learning. Deep learning is a branch of machine learning that has been inspired by the human brain [14]. That is, deep learning methods try to replicate the way the human brain learns new concepts by connecting neurons with axons in the brain. So called artificial neural networks connect simple processing units with weighted connections to imitate the behavior of the brain. Recently, artificial neural networks have gotten a lot of attention by outclassing the state-of-the-art methods in many domains such as object recognition in images [13], or machine translation [3].

A neural network consists of multiple layers, each containing many neurons. Every neuron in one layer is connected to all neurons in the preceding and succeeding layers (if present). These connections have weights attached to them, which can be used to control the impact a neuron in one layer has on the activation of a neuron in the next layer. To calculate the output of a neuron we apply a non-linear activation function (a popular choice is the rectifier function $f(x) = \max(0, x)$ [16]) to the sum over all outputs of the neurons in the previous layer times their respective connection weights. When training a neural network all the weights are set in a random fashion. Then the backpropagation algorithm [19] can be used to iteratively tune the weights, so that the neural network produces the desired output, or a close enough approximation of it. This is done by measuring how far the output of the neural network differs from the desired output, for example by calculating the mean squared error, and then back-propagating the error to the weights, so that the error gets minimized.

In classic classification tasks the desired output of the neural network will be a class label. However, one can also train a neural network without the use of class labels. This is especially helpful when no labels exist. One type of neural network that does not rely on labels is called an autoencoder, which is what we deployed in our method. Whereas a classic neural network is trained in a supervised fashion, an autoencoder is trained in an unsupervised fashion, as it is trained to reproduce its own input. Obviously, a neural network, if given enough capacity and time, can simply learn the identity function of all examples in the training set. To overcome this issue, some kind of corruption is added to the autoencoder, for example, by forcing one of the hidden layers to be very small, therefore not allowing the autoencoder to simply learn the identity. Another very common

way of adding corruption is to distribute additive gaussian noise over the input vector of the autoencoder. Thus, the autoencoder, even if repeatedly trained on the same trace, will always receive a different input. These types of autoencoders are known as denoising autoencoders, as they are basically producing a noise free version of their input. Our method is based on exactly this kind of autoencoder.

4.1 Setup

An autoencoder has a fixed size input; hence, we have to transform the variable sized traces from the event log. First, we force all traces to have the same length by repeatedly adding a special padding activity to the end until all traces have the same length (this can be set by hand or set to the maximum trace length encountered in the event log). Thereafter, we encoded the activity names using a one-hot encoding. Every activity is encoded by an n-dimensional vector, where n is the number of different activities in the event log, so that every activity is connected to exactly one dimension in the one-hot vector. To encode one activity we simply set the corresponding dimension of the one-hot vector to a fixed value of one, whilst setting all the other dimensions to zero. Notice that, instead of using zero, we opted to use -1 as this results in better distribution of the additive gaussian noise. Because we are using rectified linear units (i.e., units using the aforementioned rectifier function as their activation function), using -1 instead of 0 does not have a huge impact. This is done for every activity in every trace, including the special padding activity.

Consider the following example: let us assume an event log consists of 10 different activities and the maximum length of all traces in the event log is also 10. After the padding, every trace will have a fixed size of 10; and every activity is encoded by a 10-dimensional one-hot vector. Consequently, the resulting one-hot vector for every trace will have a size of 100.

Using the one-hot encoded event log we can train the autoencoder with stochastic gradient descent and backpropagation, using the event log both as the input, and the label. Figure 3 shows a simplified version of the architecture. Notice that Fig. 3 shows the event log with variable size traces for reasons of simplicity. In reality the event log is transformed before being fed into the autoencoder and then decoded afterwards. The special noise layer adds gaussian noise before feeding the input into the autoencoder, this layer is only active

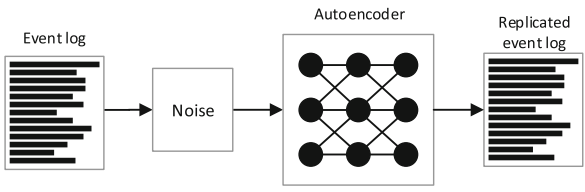

Fig. 3. Autoencoder is trained to replicate the traces in the event log after the addition of gaussian noise

Table 3. Overview of the hidden layer sizes

Model	Input/Output size	Hidden size
Small	264	286
Medium	340	374
Large	616	660
Huge	728	784
P2P	154	168

during training. Now the autoencoder is trained to reproduce its input, that is, to minimize the mean squared error between the input and its output.

We trained the autoencoder for a fixed number of 500 epochs using a mini batch size of 32. As the optimizer we used stochastic gradient descent with a learning rate of 0.01, a learning rate decay of 10^{-5}, and nesterov momentum [21] with a momentum factor of 0.9. Additionally, we used a maxnorm weight constraint of 0.5, as well as a dropout of 0.5, as suggested in [20]; the additive gaussian noise was sampled from a zero centric gaussian distribution with a standard deviation of 0.1. Each autoencoder consisted of an input and an output layer with linear units, and exactly one hidden layer with rectified linear units. These training parameters were used for each of the different event logs, but the size of the hidden layer was adapted depending on the event log. The number of neurons in the hidden layer was set to the size of the input plus one neuron for each possible activity in the event log. This was an arbitrary choice for the hidden layer size, but we found that it worked sufficiently good. However, choosing a hidden layer size smaller than the input layer size did not yield good results. The actual hidden layer sizes can be found in Table 3.

4.2 Anomalous Trace Classifier

After training the autoencoder, it can be used to reproduce the traces in the same event log it was trained on, but without applying the noise. Now, we can measure the mean squared error between the input vector and the output vector to detect anomalies in the event log. Because the distribution of normal traces and anomalous traces in the event log is one sided we can assume that the autoencoder will reproduce the normal traces with less reproduction error than the anomalies. Therefore we can define a threshold t, where if a traces reproduction error succeeds this threshold t we consider it as an anomaly. Figure 4 shows how to transform the trained autoencoder into an anomaly classifier by adding a threshold classifier. We found that using the average reproduction error on the event log is a good general choice for the threshold (cf. Fig. 5 in Sect. 5).

4.3 Anomalous Activity Classifier

We have described how to detect anomalous traces in the event log, now we want to refine this method. Not only can we detect that a trace is anomalous, but also

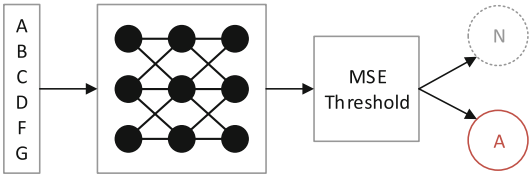

Fig. 4. Threshold classifier based on the mean squared error between the input vector and the output of the autoencoder

what activity in the trace influences the reproduction error the most. Therefore, we have to change our calculation of the reproduction error from trace based to activity based. Up until now, we calculated the reproduction error as the mean squared error between the entire one-hot encoded input and output sequence of the autoencoder. However, we can also consider the mean squared error for every activity in the sequence separately. After a trace has been reproduced by the autoencoder we split the input and the output vectors into subparts, so that every subpart contains the one-hot encoding for one single activity. Now we can compute the mean squared error for each activity separately and then apply the same threshold classifier method as before, only this time the classifier detect anomalous activities as apposed to anomalous traces.

5 Evaluation

We evaluated our approach on four different event logs coming from process models with different complexities, ranging from low to high complexity. In addition to those four event logs we also used an interpretable version with low complexity for demonstrative purposes.

After training one autoencoder for each event log, we evaluated the autoencoders on the same dataset, but without adding gaussian noise, as in the training phase. Therefore, we calculated the mean squared error for every trace in the training set and analyzed the resulting distribution. Figure 5 shows the distribution of the reproduction error for each dataset split into anomalous and normal traces. To indicate the variance of the distribution we used so called box-and-whisker plots. Figure 5 indicates that all five autoencoders are able to perfectly split the normal traces from the anomalous one solely based on the reproduction error. It also shows that the average reproduction error is a good threshold value for the anomaly classifier, albeit one would prefer a more pessimistically set value for real life scenarios, so that the anomaly class precision and the normal class recall are maximized. However, as we do not have the benefit of being provided labels in real life, we stick with the simple solution here. We plan on investigating more sophisticated methods of automatically setting the threshold.

Table 4 shows the respective precision, recall, and F1-score for the 5 different autoencoders in their classification report. Notice that using the average reproduction error as the threshold leads to the low anomaly precision class and

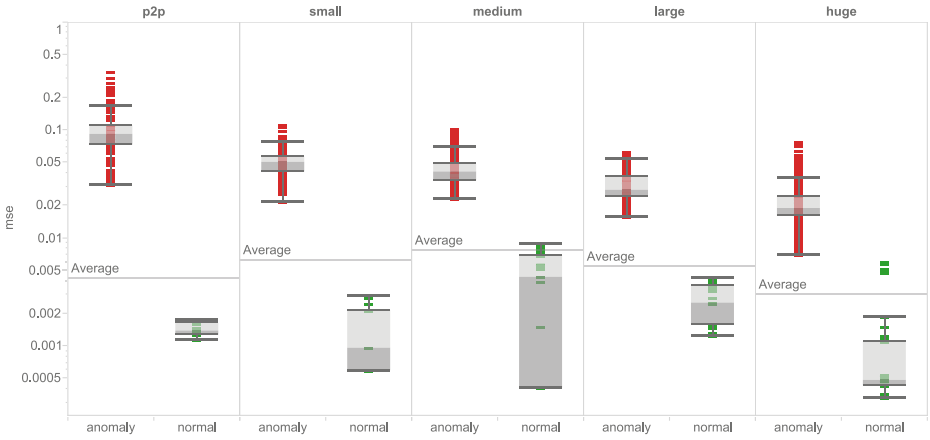

Fig. 5. The autoencoder succeeds in perfectly splitting the dataset into normal and anomalous traces solely based on the reproduction error

Table 4. Classification report for the anomalous traces detector

Dataset	Class	Precision	Recall	F1-score	Support
P2P	normal	1.00	1.00	1.00	9 032
	anomaly	1.00	1.00	1.00	968
	average	1.00	1.00	1.00	10 000
Small	normal	1.00	1.00	1.00	9 022
	anomaly	1.00	1.00	1.00	978
	average	1.00	1.00	1.00	10 000
Medium	normal	1.00	0.95	0.98	9 027
	anomaly	0.70	1.00	0.83	973
	average	0.97	0.96	0.96	10 000
Large	normal	1.00	1.00	1.00	9 015
	anomaly	1.00	1.00	1.00	985
	average	1.00	1.00	1.00	10 000
Huge	normal	1.00	0.87	0.93	45 222
	anomaly	0.46	1.00	0.63	4 778
	average	0.95	0.89	0.90	50 000

normal class recall scores in the medium and huge event log. Even though the overall result is still remarkable, we still have room to improve the automatic adjusting of the threshold, as one can clearly see that setting the threshold optimally is indeed possible.

Next, we want to evaluate the activity based anomaly detection described earlier, but first we want to consider a few examples of classified traces from

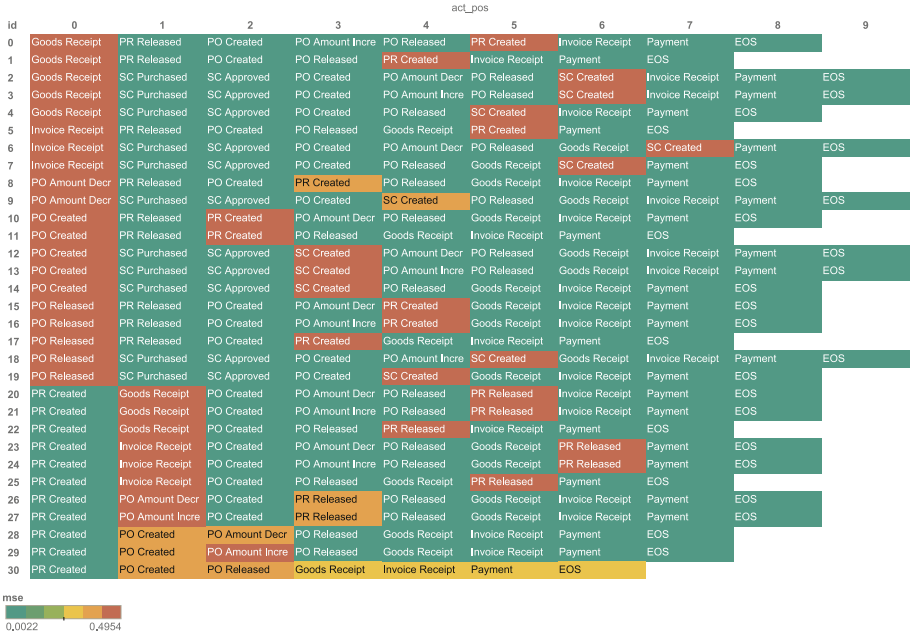

Fig. 6. Conformance check on a sample of the P2P dataset (Color figure online)

the P2P dataset autoencoder. Figure 6 shows a sample of 30 traces of the P2P event log, where every activity has been color coded according to the autoencoders reproduction error. The color coding simply linearly distributes 6 color patches across the range from zero to the maximum reproduction error in the event log. When comparing the traces to the reference procurement model from Fig. 2, we find that the autoencoder detects the anomalous activity in the traces remarkably well. However, we can also see that it has problems when certain activities are left out and the rest of the trace consequently gets shifted by one activity. Trace number 30 demonstrates this phenomenon. We can see that the necessary activity 'PR Released' has been skipped, but the system indicates that all activities after this point are anomalous, albeit the rest of the sequence being valid. The system has problems to handle shifted subsequences. Nevertheless, the autoencoder successfully detects the first activity that does not conform with the underlying model, that is the reproduction error of that activity is significantly high.

We have evaluated two versions of the anomalous activity classifier. The first produces only one position in the sequence by returning the index with the highest reproduction error. The second approach, similar to the anomalous trace classifier before, returns all indices where the reproduction error exceeds the average reproduction error on the whole event log. We shall call the former the argmax approach and the latter the threshold approach. Table 5 shows the classification reports for those two approaches. Notice that we only evaluated

them on the anomalous traces and that we do not give the precision and F1-score for the threshold approach. The threshold approach will always yield a precision of one, hence we concentrate on the recall score, which is equivalent with the accuracy here. In case of the threshold approach, its prediction has been count as correct if the actual index of the first anomalous activity, according to the reference model, produced an above average reproduction error.

Table 5. Classification report for the anomalous activity detector

Dataset	Type	Precision	Recall	F1-score	Support
P2P	argmax	0.55	0.47	0.50	968
	threshold		0.85		
Small	argmax	0.76	0.66	0.68	978
	threshold		0.98		
Medium	argmax	0.68	0.54	0.56	973
	threshold		0.99		
Large	argmax	0.67	0.59	0.61	985
	threshold		1.00		
Huge	argmax	0.70	0.63	0.64	4 778
	threshold		0.96		

Table 5 shows that the argmax approach does not perform well. This is due to the fact that in most cases the reproduction error of the first anomalous activity in the trace is indeed significantly high, yet the overall highest reproduction error is found at a different index. Notice that this is also an effect of the reproduction error being carried through, as mentioned before. However, the threshold approach clearly shows that the anomalous activity almost always produces an above average reproduction error; hence, the autoencoder is capable of detecting them.

6 Conclusion and Future Work

Real-life event logs often contain anomalous traces. We have presented an approach that is capable of automatically filtering out anomalous traces in a noisy event log without any prior knowledge being fed into the system. The system learns to discriminate between normal and anomalous traces only from the present pattern in the data. We have evaluated the system on five different noisy event logs from randomly generated process models (one model was produced manually). Our evaluation has shown that our threshold based anomalous activity classifier is indeed capable of automatically detecting the anomalous activity in a trace with an accuracy of at least 85.0 % and 95.6 on average over all training sets. Especially for the more complex process models this is a remarkable result.

We want to point out that our approach is susceptible to anomalous behavior in the event log that is very frequent, that is the same anomalous trace is found multiple times. This is something we want to investigate in the future. However, as changes in a business process usually happen subtly, anomalous traces with the same sequence should be infrequent at first; thus, our approach will be able to detect them early, so that they do not have time to settle in. We also want to test our approach on a range of different noise levels in the event log (i.e., more anomalous traces), as well as include incomplete traces in the event log. As event logs consist of sequences of activities, it is also sensible to apply recurrent neural networks to the problem. Using recurrent networks could overcome the issue that our system is susceptible to skipped activities, which results in a shifted event sequence that is otherwise valid. Recurrent networks can learn these pattern regardless of where they exactly occur in the sequence, which is something the autoencoder in our approach is unable to do.

Our approach proves that neural networks are applicable within the domain of business processes. Moreover, we have shown that denoising autoencoders are capable of dealing with event logs that do already contain the anomalous traces, as opposed to training them on event logs that only contain normal traces. This approach is especially interesting, as it shows that an autoencoder can capture the underlying process of an event log, without being provided extra knowledge.

Acknowledgments. This project (HA project no. 479/15-21) is funded in the framework of Hessen ModellProjekte, financed with funds of LOEWE – Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbundvorhaben (State Offensive for the Development of Scientific and Economic Excellence) and by the LOEWE initiative (Hessen, Germany) within the NICER project [III L 5-518/81.004].

References

1. 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)
2. Amer, M., Goldstein, M., Abdennadher, S.: Enhancing one-class support vector machines for unsupervised anomaly detection. In: Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description, pp. 8–15. ACM (2013)
3. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate (2014). arXiv preprint: arXiv:1409.0473
4. Bezerra, F., Wainer, J., Aalst, W.M.P.: Anomaly detection using process mining. In: Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Ukor, R. (eds.) BPMDS/EMMSAD 2009. LNBIP, pp. 149–161. Springer, Heidelberg (2009). doi:10.1007/978-3-642-01862-6_13
5. Burattin, A.: PLG2: multiperspective processes randomization and simulation for online and offline settings. CoRR abs/1506.0 (2015)
6. Cortes, C., Vapnik, V.: Support-vector networks. Mach. Learn. **20**(3), 273–297 (1995)
7. Dumas, M., Van der Aalst, W.M., Ter Hofstede, A.H.: Process-Aware Information Systems: Bridging People and Software Through Process Technology. John Wiley & Sons, Hoboken (2005)

8. Eskin, E.: Anomaly detection over noisy data using learned probability distributions. In: Proceedings of the International Conference on Machine Learning. Citeseer (2000)
9. Hawkins, S., He, H., Williams, G., Baxter, R.: Outlier detection using replicator neural networks. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2002. LNCS, pp. 170–180. Springer, Heidelberg (2002). doi:10.1007/3-540-46145-0_17
10. Hecht-Nielsen, R.: Replicator neural networks for universal optimal source coding. Science **269**(5232), 1861 (1995)
11. Hinton, G.E.: Connectionist learning procedures. Artif. Intell. **40**(1), 185–234 (1989)
12. Japkowicz, N.: Supervised versus unsupervised binary-learning by feedforward neural networks. Mach. Learn. **42**(1), 97–122 (2001)
13. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)
14. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
15. Maaten, L.V.D., Hinton, G.E.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**, 2579–2605 (2008)
16. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML 2010), pp. 807–814 (2010)
17. Pimentel, M.A., Clifton, D.A., Clifton, L., Tarassenko, L.: A review of novelty detection. Sign. Process. **99**, 215–249 (2014)
18. Rozinat, A., van der Aalst, W.M.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
19. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cogn. Model. **5**(3), 1 (1988)
20. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(1), 1929–1958 (2014)
21. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th International Conference on Machine Learning (ICML 2013), pp. 1139–1147 (2013)
22. Van Der Aalst, W., Adriansyah, A., de Medeiros, A.K.A., Arcieri, F., Baier, T., Blickle, T., Bose, J.C., van den Brand, P., Brandtjen, R., Buijs, J., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011 Workshops, Part I. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). doi:10.1007/978-3-642-28108-2_19