Process Learning for Autonomous Process Anomaly Correction

Nolle, Timo

(2020)

PROCESS LEARNING FOR AUTONOMOUS PROCESS ANOMALY CORRECTION

TIMO NOLLE

Kumulative Dissertation
zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr. rer. nat.)

Dissertationsschrift in englischer Sprache von
Timo Nolle, M.Sc.
aus Bensheim, Deutschland
geb. am 17.03.1990 in Ratingen

Erstreferent: Prof. Dr. Max Mühlhäuser
Korreferent: Prof. Dr. Ingo Weber

Tag der Einreichung: 15.09.2020
Tag der Prüfung: 29.10.2020

TECHNISCHE UNIVERSITÄT DARMSTADT

Telecooperation Lab
Fachbereich Informatik
Technische Universität Darmstadt

December 1, 2020 – Version 1.2

Timo Nolle: *Process Learning for Autonomous Process Anomaly Correction*

Darmstadt, Technische Universität Darmstadt
Jahr der Veröffentlichung auf TUPrints: 2020
URN: urn:nbn:de:tuda-tuprints-142579
URI: https://tuprints.ulb.tu-darmstadt.de/id/eprint/14257
Tag der Einreichung: 15.09.2020
Tag der Prüfung: 29.10.2020

ABSTRACT

The automatic detection of divergences from a desired process behavior is a common research topic in the business process management community. An established technique to analyze processes is called *conformance checking*. Given a definition of a process in form of a *process model*, conformance checking can be used to test whether the executions of a process contained in a so-called *event log* data structure are conforming with the process as it was defined. The result is a comparison of the execution traces and their respective correct execution, according to the process model. This technique provides insights into where the divergence has occurred and how the execution must be altered to conform to the process model. However, a problem is that it requires a process model to be available. Process models in the correct format are not always available.

Contrary to conformance checking, *process anomaly detection* aims to find anomalous executions without relying on a predefined process model. A process anomaly detection algorithm derives the process logic from the event log itself and exploits the patterns found within the event log to distinguish normal from anomalous process executions. Though process anomaly detection provides the benefit of not relying on a process model, it typically does not provide the level of detail that conformance checking does. A process execution can either be *normal* or it can be *anomalous*.

This dissertation proposes *process anomaly correction*, a novel approach that combines the benefits of conformance checking and process anomaly detection. Given only an event log, *process anomaly correction* detects anomalous executions, clearly indicates where the anomaly has occurred during the execution and suggests possible corrective measures. The solution presented in this work is based on a new concept to the field of process anomaly detection: *Process learning*. In process learning, the task of understanding the process based on the example data is transformed into a learning problem in which a neural network is trained to predict the very next activity in a running process execution. The resulting machine learning model thus represents an approximation of the real process that created the data.

This cumulative dissertation consists of five contributions to the field of business process management that demonstrate how, starting from a process anomaly detection, *process anomaly correction* is achieved in a series of four steps. (1) Process learning is employed to generate an approximated model of the process logic. (2) The limitation of only distinguishing between *normal* and *anomalous* process execu-

tions is overcome by employing the process learning model which processes the process executions on a finer level of detail than existing approaches. (3) The necessity of providing manual threshold settings, as it is typical for process anomaly detection algorithms, is replaced by an automatic parameterization utilizing the process learning model. (4) The predictive capabilities of the process learning model are exploited to generate possible corrections of detected anomalies.

The resulting *process anomaly correction* approach can be employed in scenarios where classic conformance checking would be infeasible, due to the restriction of relying on a process model. Furthermore, it can be employed alongside classical conformance checking, for it incorporates more information coming from the event log than classical conformance checking (such as employees executing a process step, in which country the process is executed, etc.), and thus provides a new perspective for the process analyst.

ZUSAMMENFASSUNG

Die automatische Erkennung von Abweichungen von einem gewünschten Prozessverhalten ist ein häufiges Forschungsthema auf dem Gebiet des Geschäftsprozessmanagements. Eine gängige Technik zur Analyse von Prozessen wird als *Conformance Checking* bezeichnet. Ausgehend von der Definition eines Prozesses in Form eines *Prozessmodells* kann mit Hilfe von Conformance Checking überprüft werden, ob die Ausführungen eines Prozesses, die in einem so genannten *Event Log* enthalten sind, mit dem definierten Prozess konform sind. Das Ergebnis ist ein Vergleich der tatsächlichen Ausführungen und ihrer jeweils korrekten Ausführung gemäß dem Prozessmodell. Diese Technik gibt Aufschluss darüber, wo die Divergenz aufgetreten ist und wie die Ausführung geändert werden muss, um dem Prozessmodell zu entsprechen. Ein Problem besteht jedoch darin, dass dazu ein Prozessmodell verfügbar sein muss. Prozessmodelle im korrekten Format sind nicht immer verfügbar.

Im Gegensatz zur Konformitätsprüfung zielt *Process Anomaly Detection* darauf ab, anomale Ausführungen zu finden, ohne sich auf ein vordefiniertes Prozessmodell zu stützen. Ein Algorithmus zur Erkennung von Prozessanomalien leitet die Prozesslogik aus dem Event Log selbst ab und nutzt die im Ereignisprotokoll gefundenen Muster aus, um normale von anomalen Prozessausführungen zu unterscheiden. Obwohl die Erkennung von Prozessanomalien den Vorteil bietet, dass sie sich nicht auf ein Prozessmodell stützt, bietet sie in der Regel nicht den Detaillierungsgrad, den *Conformance Checking* bietet. Eine Prozessausführung kann entweder *normal* oder *anomal* sein.

In dieser Dissertation wird *Process Anomaly Correction* vorgeschlagen, ein neuartiger Ansatz, der die Vorteile von Conformance Checking

und Process Anomaly Detection kombiniert. Nur unter Verwendung des Event Logs, erkennt die Process Anomaly Correction anomale Ausführungen, zeigt deutlich, wo die Anomalie während der Ausführung aufgetreten ist, und schlägt mögliche Korrekturmaßnahmen vor. Die in dieser Arbeit vorgestellte Lösung basiert auf einem neuen Konzept für den Bereich der Erkennung von Prozessanomalien: *Process Learning*. Beim Process Learning wird die Aufgabe, den Prozess auf der Grundlage der Beispieldaten zu verstehen, in ein Lernproblem umgewandelt, bei dem ein neuronales Netz darauf trainiert wird, die nächste Aktivität in einer laufenden Prozessausführung vorherzusagen. Das resultierende Modell des maschinellen Lernens stellt somit eine Annäherung an den realen Prozess dar, der die Daten erzeugt hat.

Diese kumulative Dissertation besteht aus fünf Beiträgen aus dem Gebiet des Geschäftsprozessmanagements, die zeigen, wie, ausgehend von Process Anomaly Detection, *Process Anomaly Correction* in einer Reihe von vier Schritten erreicht wird. (1) Process Learning wird eingesetzt, um ein approximiertes Modell der Prozesslogik zu erzeugen. (2) Die Beschränkung, nur zwischen normalen und anomalen Prozessausführungen zu unterscheiden, wird durch die Anwendung des Process Learning Modells überwunden, welches die Prozessausführungen auf einer feineren Detailebene verarbeitet als bestehende Ansätze. (3) Die Notwendigkeit der Bereitstellung manueller Schwellenwerteinstellungen, wie sie für Algorithmen zur Erkennung von Prozessanomalien typisch sind, wird durch eine automatische Parametrisierung unter Verwendung des Process Learning Modells ersetzt. (4) Die prädiktiven Fähigkeiten des Process Learning Modells werden ausgenutzt, um mögliche Korrekturen der erkannten Anomalien zu generieren.

Der sich daraus ergebende Ansatz zur Korrektur von Prozessanomalien kann in Szenarien eingesetzt werden, in denen klassisches Conformance Checking aufgrund der Einschränkung, von einem Prozessmodell abhängig zu sein, nicht durchführbar wäre. Darüber hinaus kann dieser Ansatz zusätzlich zu klassischem Conformance Checking eingesetzt werden, da er mehr Informationen aus dem Event Log einbezieht als klassisches Conformance Checking (z. B. Mitarbeiter, die einen Prozessschritt ausführen, in welchem Land der Prozess ausgeführt wird usw.) und somit eine neue Perspektive für den Prozessanalysten bietet.

ACKNOWLEDGMENTS

Without the wonderful support of my amazing advisors, colleagues, friends, and my loving family, this dissertation would not have been possible. Many thanks to each and every one of you, for making this time of my life as amazing as it was.

First and foremost there is Max, my doctoral advisor, mentor, and friend. It was an honor for me to work so closely with you over the last five of years. You always supported me in my work and gave me the freedom and the space I needed to be creative. Without exception, you amazed me with your direct and profound feedback on my ideas. Thank you for everything you have done.

I would also like to thank Ingo Weber for serving as the co-referee for my dissertation. Your detailed feedback and constructive criticism helped me in the final phase of writing and in the preparation for the disputation. I am grateful for the support you have given me.

The kindness with which the Telecooperation group has welcomed me is second to none. It has been a pleasure to work with all of you. A special thanks goes to my colleagues of the ISY group, Alex, Chris, and Sebastian. Without the countless hours we spent on presentations, paper ideas, and general discussions, I would not be who I am today.

During the five years, I had the pleasure of working at an incredible startup from Darmstadt, PAF. I am very grateful for Tobias allowing me to follow my academic ambitions while simultaneously working for his company. Without your support, I would never have succeeded. I also want to thank Stefan for his willingness to listen to me and to serve as a motivator. Lastly, I want to thank Susanne, who, in countless evening sessions, listened to my research ideas and guided me through the last years of my dissertation.

I want to thank my family, Gaby, Thomas, and Lars, for their continuous support not only throughout this dissertation but throughout my Bachelor's and Master's degrees as well. You never stopped believing in me and this is what motivated me every day to keep going.

Finally, my deepest gratitude goes to my girlfriend, Ricarda. Thank you for five years of unequivocal support and infinite patience. You helped me become the best version of myself. I am looking forward to a wonderful future together with you and the two cutest little dogs in the world. You are the best!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Part I

SYNOPSIS

1 INTRODUCTION

When striving for a career in academia, it is quite unavoidable to learn the process of writing a scientific paper. Picking up such a new skill can be very challenging. Where do we start? What general plan should we follow? How do we know what is right and what is wrong? Who to ask? Fortunately, some of us have had the privilege of having a mentor, guiding us through the initial hardship, pointing us in the right direction, and explaining to us what we were doing wrong and—more importantly—how to do it right.

Let us consider the following example: We are writing our first scientific publication and we plan to work in the following order: *Identify Problem, Develop Method, Experiment, Conduct Study, Research Related Work, Conclude, Submit*. How do we know we are on the right track? At this point, we would probably ask our mentors for advice, and they would point out the mistakes we made: "You should research the related work earlier!". But how are they doing it?

They will base their assessment of our plan on their theoretical knowledge *and* their practical experience, having mentored many students over the years in the same process. In their mind, they built an abstract model of how the process of scientific writing works (for them), learning from the different scenarios they have experienced throughout the years. Such a process model [15] could look as depicted in Figure 1.1. Whenever a student approaches them with a new question, they can utilize this model to check the plan of the student for its conformance with their mind's model of the process.

CONFORMANCE CHECKING WITH ALIGNMENTS A more general statement of the problem above is: Given a *process model* and an *event log*, provide an *alignment* between the *cases* of the event log and the process model. To understand this statement, we have to define the emphasized terms. In the business process management (BPM) research community, process data is typically stored in a special data structure called *event log* [1, 2]. An event log consists of *cases*, which in turn are sequences of *events* (an event in its simplest form: an activity a happened in case c at time t) that happened during the execution of a process. A *process model* is an abstract concept that holds information about the process logic, for example, in which order the events have to occur to be compliant with the process. An *alignment* is a projection of a single case onto the process model, which highlights where they overlap and also where they diverge. The technique of *checking* if the

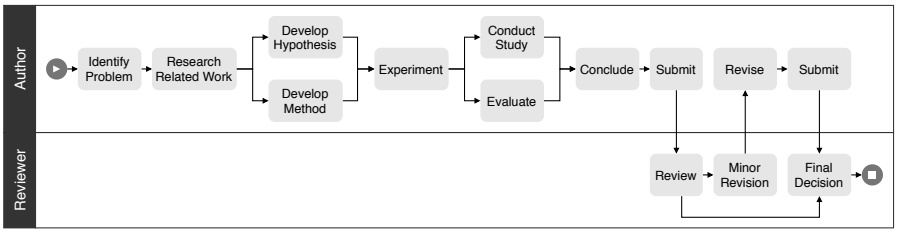

Figure 1.1: A paper writing and reviewing process in form of a process model

Table 1.1: An example alignment of a case from an event log (top row) and the closest path through the process model from Figure 1.1 (bottom row); ≫ indicates that a necessary event has been skipped (when appearing in the top row) or that an event is not allowed by the process (when appearing in the bottom row)

Identify Problem	≫	Develop Method	Experiment	Conduct Study	Research Related Work	Conclude	Submit
Identify Problem	Research Related Work	Develop Method	Experiment	Conduct Study	≫	Conclude	Submit

cases of an event log are *conforming* with the logic as defined in the process model is thus called *conformance checking* [43].

In the context of the mentor example from earlier, the hint to "[...] research the related work earlier!" from the mentor can be visualized as an alignment as shown in Table 1.1.

While conformance checking can provide alignments which offer a detailed explanation of what is right and what is wrong in a case, it does have a downside: It relies on the existence of a predefined process model. In the mentor example introduced above, this process model is created over time and shaped by the experience of the mentor. In real-life scenarios, however, the process logic has to be manually transferred to a digital process model so it can be used for conformance checking. Such a digital process model cannot be assumed to always be available, and if it is, it might be outdated or even wrong.

PROCESS ANOMALY DETECTION Process anomaly detection [3] is a technique that can be used—without a process model—to automatically identify anomalous cases in an event log. Instead of relying on a process model as input, process anomaly detection infers the process logic directly from an event log. With increasing digitization, process data is much more readily available. It is generated as a side-effect of businesses running their processes guided by process-aware information systems. Thus, process anomaly detection can be a feasible alternative to conformance checking if no process model is available.

However, most process anomaly detection algorithms focus on the identification of anomalous *cases*, and therefore provide a binary clas-

sification as output. A case is either anomalous or it is normal. All we know is if an execution is deemed to be *right* or *wrong*. *Where* an anomalous execution has diverged and how the mistake can be corrected remains to be investigated. In the context of the aforementioned mentor example, imagine our mentor simply proclaiming "Right!" or "Wrong!" whenever we asked them for guidance in a particularly tricky situation. This advice would not have been very helpful to us.

While process anomaly detection provides the benefit of not relying on a predefined process model, it is lacking in the quality of its output compared to conformance checking. To provide a solution that is as easy to use as it is to ask your mentor for advice, the benefits of process anomaly detection and conformance checking have to be combined. Since process anomaly detection is applied without being manually engineered to model the process logic, it has to provide explanations to the analyst so its detection can be interpreted by someone unfamiliar with the process. The concept of alignments provides these explanations, but it is tightly coupled to the idea of a process model serving as the definition of the normative behavior.

Inferring the normative behavior from the event log is beneficial for several reasons: The issue of a process model possibly being outdated is addressed by continuously updating the learned process logic based on the incoming event data. Furthermore, the a priori knowledge necessary to analyze a process is being reduced because the only input to the algorithm is the event log. However, basing the analysis solely on the event log also has a downside: Since no process model can be utilized to explain detected anomalies, the algorithm now has to provide both the anomaly detection *and* an explanation as to why this anomaly has been detected.

Though alignments can be used to provide the explanations, they were not compatible with process anomaly detection algorithms. Hence, a new approach was necessary.

PROCESS ANOMALY CORRECTION In this thesis, a new approach is proposed that combines the benefits of process anomaly detection and conformance checking. It uses as input only the event log itself but produces an output akin to alignments in conformance checking. The technique is called *process anomaly correction*. Note that we use the term correction in the sense of *corrective suggestions* and not in the sense of triggering *corrective actions* since the process anomaly correction approach generally works independent from the system running the process. It is based on the application of a machine learning technique that exploits the event log data structure to learn the underlying business process, which in this thesis will be referred to as *process learning*. Similar to how *process mining* can be seen as process-aware *data mining*, *process learning* is process-aware *machine learning*. Process learning operates on *event logs* and can exploit the assumption that

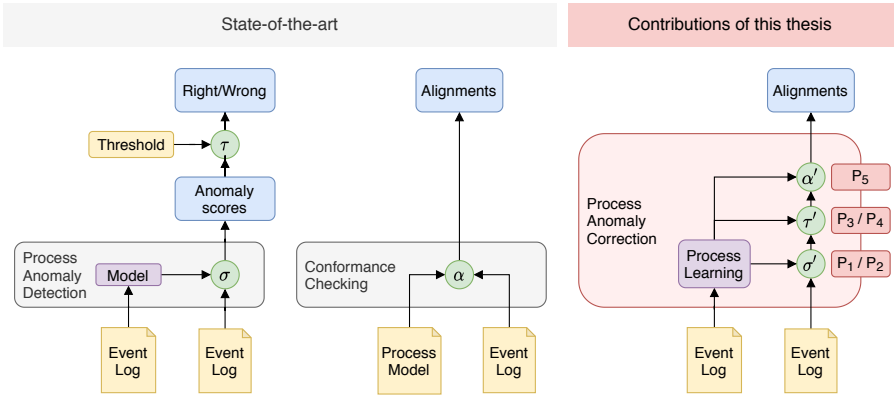

Figure 1.2: Comparison of *process anomaly detection*, *conformance checking*, and the proposed solution, *process anomaly correction*

the data found in the event log models an underlying process. Just as the mentors who developed their mastery of the scientific paper writing process by experiencing various scenarios, *process learning* can be used to infer the process logic from an event log.

Figure 1.2 compares the two state-of-the-art approaches, *process anomaly detection* and *conformance checking* on the left, with the proposed approach *process anomaly correction* on the right. *Conformance checking*, on the one hand, is based on an alignment function, α, that compares an event log and a process model and produces the desired alignments. *Process anomaly detection*, on the other hand, first infers an internal *model* from the event log and then uses a scoring function, σ, to assign an anomaly score to every case in the event log based on its internal model. Secondly, the anomaly scores that result from the scoring function have to be passed through a threshold function, τ, that maps the anomaly scores to either 0 or 1. Note that τ usually requires an input parameter setting the threshold value to operate. This is another downside of most process anomaly detection algorithms since the users need to provide the threshold themselves. Setting the threshold to an appropriate value is challenging, especially for someone not familiar with the process. Hence, a manual setting of the threshold is best avoided.

Process anomaly correction is achieved in four steps. (1) The modeling phase of the process anomaly detection algorithm is replaced by a tailored *process learning* neural network architecture that is trained on the event log to derive the process logic. The resulting *process learning model* is subsequently used to parameterize the three functions σ', τ', and α', which removes the necessity of human intervention. (2) The scoring function σ is replaced by σ' to not only allow the detection of anomalous cases (*right* vs. *wrong*), but also the detection of anomalous events and the detection of anomalous event attributes (e.g., the resource executing an event). (3) The threshold function τ is replaced by τ', a novel heuristic that is parameterized by the

process learning model. (4) The alignment algorithm α is replaced by a specialized function, α', that utilizes the *process learning model* to analyze the event log and produce an alignment as the output.

RESULTS This cumulative dissertation contains five contributions to the field of BPM that were published in conference proceedings and scientific journals between 2016 and 2020. The publications build upon each other and, as a collection, serve as evidence that *process learning* is not only a viable base for state-of-the-art process anomaly detection but further that fully autonomous *process anomaly correction* is possible. Each individual publication addressed specific shortcomings of the state-of-the-art at the time. The proposed solutions were evaluated on both synthetic and real-world event logs. The results of these evaluations demonstrated that process learning based solutions were able to outperform the respective state-of-the-art approaches known at the time. The individual contributions are positioned in Figure 1.2 (P1 to P5 in the red boxes) according to their scientific focus.

STRUCTURE OF THIS DOCUMENT This dissertation is structured into two parts: Part I, the synopsis; and Part II, the cumulative part of this thesis. Part I first introduces necessary concepts and terminology and gives an overview of the development of the state-of-the-art (excluding the contributions presented in Part II) in Chapter 2. Chapter 3 elaborates on the five individual contributions of this thesis and how they relate to each other. Part I closes with an overall conclusion as well as an outlook on future work in Chapter 4. Part II is the cumulative part of this thesis and contains the five publications in chronological order and in their original published form (Chapters P1–P5).

2 STATE-OF-THE-ART

The concepts of *process anomaly detection*, *conformance checking*, and *process learning* have already been introduced in the preceding chapter. This chapter gives a summary of the essential contributions to the fields of *process anomaly detection* and *conformance checking*. Furthermore, it outlines contributions to the field of BPM that base their solutions on the concept of *process learning* and its related areas.

Note that this overview of the state-of-the-art omits the contributions of this dissertation. The contributions of this thesis and how they chronologically fit into the state-of-the-art will be presented in detail in Chapter 3 and the individual publication chapters in Part II.

2.1 PRELIMINARIES

As already discussed in the introduction, the predominant data structure in process mining is the *event log*. It is also the basis for process anomaly detection algorithms. An event log is a set of ordered sequences of *events*, called *cases*. The events in a case are ordered by a *timestamp* that indicates *when* an event has been executed. *What* has happened is described by the *activity name* of an event. In real-life examples, event logs typically hold more auxiliary information about the process, the cases, and the events. An example event log following the paper writing and reviewing process from Figure 1.1 is shown in Table 2.1.

In the process mining literature [2], the concept of different process perspectives has been introduced. In [2], the authors distinguish between four different process perspectives: Control-flow perspective, organizational perspective, time perspective, and data perspective.

The *control-flow perspective* is utilized to analyze the order in which activities are executed. An incorrect order of activities is thus something that a process anomaly detection algorithm should detect. The initial example of executing the activity *Research Related Work* too late is an example of a control-flow anomaly.

The *organizational perspective* focuses on the structure of an organization running the process. Typically, this perspective is used to analyze how different *resources* are utilized during the execution of a process. The term *resource* can refer to employees of a company, automated agents, or other intelligent systems. As a concrete example in the scientific writing process from earlier, only a reviewer resource is allowed to review a paper. If the author of a paper were to accept his or her own paper, this would constitute an organizational anomaly.

Table 2.1: Example event log of the paper writing and reviewing process

Case Identifier	Timestamp	Activity	Resource	Conference
P3	2018-01-10 17:03	Identify Problem	Author	BPM'18
P3	2018-01-15 14:04	Research Related Work	Author	BPM'18
P3	2018-02-15 08:05	Develop Method	Author	BPM'18
P3	2018-03-01 02:09	Experiment	Author	BPM'18
P3	2018-03-15 18:08	Evaluate	Author	BPM'18
P3	2018-03-17 13:37	Conclude	Author	BPM'18
P3	2018-03-18 18:49	Submit	Author	BPM'18
P3	2018-05-01 10:00	Review	Reviewer	BPM'18
P3	2018-05-14 21:44	Final Decision	Reviewer	BPM'18
P4	2018-11-19 17:03	Identify Problem	Author	ISJ'19
P4	2018-11-24 14:04	Research Related Work	Author	ISJ'19
P4	2018-12-21 08:05	Develop Method	Author	ISJ'19
P4	2019-01-14 02:09	Experiment	Author	ISJ'19
P4	2019-01-25 18:08	Evaluate	Author	ISJ'19
P4	2019-02-01 13:37	Conclude	Author	ISJ'19
P4	2019-02-03 23:53	Submit	Author	ISJ'19
P4	2019-04-08 14:48	Review	Reviewer	ISJ'19
P4	2019-05-01 10:18	Minor Revision	Reviewer	ISJ'19
P4	2019-06-20 14:00	Revise	Author	ISJ'19
P4	2019-06-27 21:27	Submit	Author	ISJ'19
P4	2019-10-21 11:00	Final Decision	Reviewer	ISJ'19

The *time perspective* is adopted for analysis related to the time dimension of a process, such as lead times between two specific activities, execution durations of single activities, or deadlines that must not be exceeded. For example, any paper that is submitted after the deadline should be rejected immediately.

Lastly, the *data perspective* is used to inspect miscellaneous event and case attributes that do not fit the other perspectives. For example, a paper can be published at a certain conference. This information can be included in the event log as a case attribute since its value will not change for the different events within this case. Conversely, the type of study that has been conducted can be included as an event attribute, since it only relates to the *Conduct Study* activity. An anomaly connected to the data perspective is, for instance, if a study paper is accepted at a conference that only accepts proof papers.

As we have seen, anomalies can occur in all of these perspectives, and hence it is important for a process anomaly detection algorithm to support the different perspectives. A process anomaly detection algorithm that can detect anomalies across multiple perspectives is thus referred to be a *multi-perspective* algorithm.

In *process anomaly detection* research, the *organizational* and the *time* perspective are often regarded to be part of the *data perspective* since they indeed are just event attributes, albeit special ones. Most process anomaly detection approaches are *not* opinionated with respect to the

event attributes, and thus no special meaning is connected to these attributes. Instead, this meaning is inferred from the patterns in the data. Hence, we typically include these attributes as part of the *data perspective*, without specifically referring to the *organizational* or the *time* perspective.

An important aspect of dealing with real-life event logs is that they cannot be expected to be free of anomalies. After all, the anomalies we are trying to detect occur during the real execution of the process, and therefore will be contained in the logs. An event log containing anomalies is typically called a *noisy* event log.

2.2 PROCESS ANOMALY DETECTION

Process anomaly detection was introduced by van der Aalst et al. in 2005 [3]. Interestingly, the authors also proposed the idea of conformance checking in their paper from 2005 [3]. Back then, the two approaches were not clearly distinguishable yet. Though both approaches have their roots in the same place, they developed in different directions over time. Let us first take a look at the original publication that started both disciplines.

WORKFLOW NETS The concept of detecting anomalous process executions was first coined by van der Aalst and Medeiros in 2005 [3]. In their paper, they propose the concept of modeling the logic of a process using workflow nets, a special version of Petri nets. A *workflow net* is a directed graph, consisting of transitions and places (both are nodes in the graph), in which a transition can only be connected to places, and a place can only be connected to transitions. Transitions can only be fired if all incoming places are active, indicated by at least one *token* occupying the place. Given an initial *marking* (mapping of tokens to places in the workflow net), a correct sequence of transitions will result in a final marking where only a defined set of places contains tokens. Any incorrect sequence will lead to a deadlock in the workflow net, that is, no transition can fire but the final marking has not been reached.

To detect an anomaly in a sequence of events, the sequence is transformed into a sequence of transitions in the workflow net, and the transitions are successively fired. If the workflow net does not reach the final marking, the sequence is regarded as anomalous. They refer to this process as "playing the token-game". Van der Aalst and Medeiros base their solution on the existence of such a workflow net. If no workflow net is available, they suggest using *process discovery* algorithms which, given an event log, can produce a process model in the form of a workflow net. However, the discovery algorithm they suggested in their paper (at the time) required a noise-free event log

to generate an appropriate process model, which cannot be assumed to be available.

HANDLING NOISY EVENT LOGS Three years later, in 2008, process anomaly detection was picked up again by Bezerra and Wainer. They had identified the issue of relying on a noise-free event log and proposed three different solutions throughout 2008 and 2009 [6–9] that did not rely on a noise-free event log. The three methods are based on a discovery algorithm that can handle noisy event logs and a metric to quantify how different an anomalous execution is from the behavior in the discovered process model (conformance score). The three approaches differ in the way the event log is used in the process discovery algorithm.

In their *Threshold* algorithm, the authors iterate over all cases in the log and if a case has a frequency of less than 2 percent in the log, they remove the case from the log. After each removal of an infrequent case, they apply the discovery algorithm on all remaining cases to generate a process model. If the conformance score between the filtered log and the resulting process model is below a conformance threshold, the case is regarded to be anomalous. The iteration over all cases is then continued until all cases have been processed. In their *Iterative* algorithm, instead of removing all infrequent cases in one iteration, they opt to only remove the case with the lowest conformance score in each iteration. They still only consider cases that have a frequency of less than 2 percent as potential anomalies. Lastly, in their *Sampling* algorithm, for each case that is below the 2 percent threshold, they sample a fixed percentage of cases from the whole log and discover a process model based on the sample. If the case in question is not an instance of the resulting process model, the case is regarded to be anomalous. The rationale behind the sampling approach is that if the case under inspection is normal, it should also be included in a representative sample of the entire log since anomalies are assumed to be rare.

All three approaches have in common that they rely on the definition of a threshold to indicate when a conformance score is low enough to indicate an anomaly. This threshold depends on the event log itself and must be fine-tuned to ensure proper operation. In their papers, they have set this threshold to a fixed value across all their experiments. Additionally, only the control-flow perspective is being addressed by Bezerra and Wainer. All three approaches will detect anomalies on the case level. The whole case is classified as either normal or anomalous. Localization of the event where the case diverged is thus not possible. Nevertheless, process anomaly detection had now, for the first time, been able to deal with noisy event logs.

LIKELIHOOD GRAPHS In 2016, Böhmer and Rinderle-Ma proposed the first solution incorporating multiple perspectives (control-flow, and data perspective) [10]. The key idea behind their algorithm is a probabilistic model which they refer to as a likelihood graph. A likelihood graph is computed by calculating pairwise probabilities between all possible combinations of activities based on their frequency in the event log. Afterwards, the likelihood graph is extended by calculating the probabilities of event attribute values co-occurring with a specific activity and annotating the activities with the respective probabilities. The extended likelihood is then used to identify anomalies by computing the probability of a new case under the likelihood graph. Any case that has a lower probability than any of the cases in the original log, is regarded as anomalous. A novel aspect of Böhmer's and Rinderle-Ma's work is how they deal with unseen data (event sequences that were not part of the original dataset used to calculate the likelihood graph). Instead of regarding the appearance of a new event sequence as anomalous, they estimate its probability by mapping it to the closest known sequence through the process, which reduces the number of false-positive alarms in the later detection phase.

Every case in the original log is considered to be normal by definition, and hence this approach is not applicable to noisy event logs. Again, like the approaches of Bezerra and Waidner, the approach of Böhmer and Rinderle-Ma detects anomalies based on the case level, which does not allow for localizing the point of divergence. This was the first publication on multi-perspective process anomaly detection.

ASSOCIATION RULES Böhmer and Rinderle-Ma have proposed a different approach in 2018 [11, 13] that allows to analyze the detected anomalies further. Instead of relying on a likelihood graph, they opt for use of association rule mining to model the process behavior. Association rule mining is used to identify interesting relations between different events and their attributes. In a first step, a set of association rules is generated from the event log such that each case in the log is supported by these rules. To identify anomalies, new cases are compared to all cases in the original log and the most similar case with respect to the control-flow is returned. The rules associated with the most similar case are applied to the case under inspection and if the support for the case under inspection (i.e., the number of rules supporting the case) is lower than the support for the most similar case, it is regarded as anomalous.

This approach has the benefit of providing explanations of why a certain case is anomalous since the rules that are not supported by the case provide an indication. However, the rules still have to be analyzed by someone familiar with the process to find the point of divergence. The detection itself only indicates whether a case is anomalous or

not. Similar to their approach from 2016 [10], this method relies on a noise-free event log to generate the association rules.

DYNAMIC BAYESIAN NETWORKS In 2019, Pauwels and Calders proposed their solution based on extended dynamic Bayesian networks [40, 41]. Bayesian networks provide the benefit of modeling the probability of unseen data through Bayesian logic while providing the benefit of being applicable to noisy event logs. Though dynamic Bayesian networks existed before, they were not applicable to sequential data. Pauwels and Calders addressed this problem by introducing the concept of a k-contextlog. A k-contextlog is similar to an event log, but each event holds information (context) about the k events that preceded it. By incorporating the context, Pauwels and Calders were able to extend the dynamic Bayesian network to model the time dimension.

Böhmer's and Rinderle-Ma's approach also deals with unseen values but it relies on a noise-free dataset to achieve it. Pauwels' and Calders' approach removes this restriction and is applicable to noisy event logs, while still providing the benefit of handling unseen values in the event log. However, though their solution relies on the setting of a threshold, they provide no solution to this problem.

SHORTCOMINGS All aforementioned methods have in common that they rely on the definition of a threshold to distinguish anomalous from normal behavior. This means that the user has to define the anomaly, which requires knowledge about the process in the first place. Automatically defining the threshold based on the data itself, removing the need for manual intervention, would thus reduce the amount of a priori knowledge necessary to use the algorithm.

Although some of the approaches utilize multiple perspectives to model the process behavior, they do not incorporate them in the presentation of the results. Yet, information about where the divergence occurred, whether it is connected to the control-flow or the data perspective, as well as, what was the expected behavior, is important to identify the source of the anomalies, to isolate them, and to initiate timely countermeasures.

If someone was confronted with the picture of a case and a label deeming it to be incorrect, a natural question that arises is "Where exactly is the problem?". Even if they were provided with a clear indication of where the anomaly occurred, someone unfamiliar with the process will likely ask "Ok, but why is it anomalous?" or "What should have been done instead?". Answering these questions with current process anomaly detection algorithms is still challenging.

2.3 CONFORMANCE CHECKING

The term conformance checking was used multiple times already. This section shall give a brief introduction to the core concepts and how they relate to process anomaly detection.

Conformance checking can be utilized to relate the behavior found inside an event log to the behavior as defined by a predefined process model [43]. Single cases of an event log can be checked for their conformance with the process model. Similarly, the conformance of a process model with an entire event log can be calculated. While the initial focus of conformance checking had been to calculate the overall conformance between event logs and process models, the focus has later shifted towards a more detailed analysis of single cases.

ALIGNMENTS After van der Aalst and Medeiros had pioneered the concepts of process anomaly detection and conformance checking in 2005 [3], conformance checking has seen a series of improvements. Especially, the concept of alignments has stood the test of time. In 2010, Bose and van der Aalst proposed the use of sequence alignment algorithms to relate the behavior of a single case to the behavior of an entire event log. The idea of alignments was borrowed from the field of bioinformatics where different DNA sequences are being compared following the same principle. An alignment is a mapping of one sequence onto another, indicating which parts of the first sequence have to be altered to align it with the second one. Alignments have the benefit that they provide the necessary context to understand where a sequence is diverging, why it is diverging, and how a corresponding correct sequence ought to look.

Later, in 2013, Adriansyah, van Dongen, and van der Aalst expanded this idea to relate single cases to process models [4]. Now, this concept could be applied to visualize exactly which parts of a case were in alignment with the process and which parts were not. Furthermore, the resulting alignment relates the case to the most similar conforming path through the process model, thereby correcting it. However, this approach considers only the control-flow perspective when calculating the most similar path through the process.

MULTI-PERSPECTIVE CONFORMANCE CHECKING De Leoni and van der Aalst addressed this caveat later that year [28, 29], demonstrating that a data-aware conformance checking algorithm based on multi-perspective alignments is possible, albeit under the assumption that the cardinality of the event attributes is very low. Mannhardt et al. have rectified this shortcoming in 2015 [31], demonstrating that the requirement of low cardinality event attributes can be eliminated and, indeed, large-scale multi-perspective conformance checking is feasible.

To achieve multi-perspective conformance checking, the user is required to manually encode the process logic regarding the other perspectives (control-flow is already included in the process model) in forms of a ruleset, so that the conformance checking algorithm can check the event logs for conformance with the respective rules. To create this ruleset, a deep understanding of the process logic is required. Not every user of such algorithms can be expected to be an expert in the process they analyze.

Another approach to multi-perspective conformance checking is to encode the information of the data perspective in the names of the events, e.g., by clustering them. In 2014, Weber et al. proposed Process Oriented Dependability (POD) Discovery [49], a process discovery technique that incorporates information from the data perspective through a clustering algorithm. It was specifically targeted at discovering process models from low-level system log files coming from cloud applications.

Since the resulting process model holds the information about the data perspective in the event names, classical conformance checking can be used to identify errors in the system logs, as shown by Xu et al. in 2015 [47, 50, 51]. The result of classical conformance checking only provides information about the control-flow, so the authors investigate the root causes of a detected error by assertion evaluation of predefined rules. Xu et al. later, in 2016, showed that similar techniques can be utilized to trigger corrective actions to counteract detected errors [22]. In the same context of system error diagnosis, Farshchi et al. used statistical regression analysis on these process models to detect correlations between different activities, and hence provide root cause analysis for system errors [20, 21]. However, this technique only considers pairwise correlations between activities and not the full context of a sequence of events.

ONLINE CONFORMANCE CHECKING In 2015, Weber et al. introduced the concept of online conformance checking to the BPM community [48]. As opposed to classical conformance checking which is typically an a posteriori analysis, errors in sporadic operations on cloud applications have to be detected timely to counteract downtimes of critical services. Instead of analyzing event logs, they propose the use of event streams. Another contribution of this work is the introduction of automated post-processing steps after the conformance checking to detect numerical invariants (based on a predefined specification) and time anomalies (inferring anomaly intervals directly from the event stream). Though time anomaly intervals are derived automatically, numerical invariants still have to be specified manually.

SHORTCOMINGS Conformance checking, as proposed in the works discussed, has two downsides. It relies on the existence of a predefined

process model that captures the control-flow perspective of the process, and it relies on the manual definition of a ruleset (e.g., reviewer must take final decision) to account for the data perspective. Both the process model and the ruleset require diligent maintenance throughout the lifetime of a process, which often is too expensive to uphold. Nevertheless, conformance checking provides the benefit of generating explanations for divergences. The level of detail with which cases can be aligned to a process model, as well as the possibility of correcting the case, seem to be characteristics that process anomaly detection could benefit from.

2.4 TOWARDS PROCESS LEARNING

The goal of the proposed *process anomaly correction* approach is to provide insights to divergences from the process without the need to manually define the process logic in the form of process model or similar structures. In essence, the machine needs to learn the concept behind the process itself by exploiting patterns in the data. Machine learning, and especially deep learning [27], has become an integral part of many areas where similar problems were faced, such as natural language processing. It has been shown that deep learning could consistently reach state-of-the-art performance in language tasks, at times even surpassing it, without being specifically programmed [5, 16, 42, 46].

The structure of natural language holds some similarities to the structure of event logs. Just as natural language consists of sentences and words, following a certain grammar to make sense, an event log consists of cases and events, following the process logic to make sense. Naturally, deep neural networks should be an excellent choice to learn the underlying business process from the examples in an event log.

In 2018, Klinkmüller, van Beest, and Weber showed that predictive process monitoring algorithms that disregard the sequentiality of process data, which they call *local* algorithms, tend to make unjustified predictions [26]. Instead, *global* algorithms should be applied that consider the event sequences as a whole. The deep neural networks employed in natural language processing consider the sentences as sequences of words, and do not operate on individual words in isolation. Hence, they can be seen as global algorithms (according to the definition from [26]).

The first publication applying deep learning techniques to event logs is from 2016 [18]. Evermann, Rehse, and Fettke demonstrate that long short-term memory networks [24], a special kind of recurrent neural networks, can be utilized to predict the final state of a running business process, its remaining time to completion, as well as the most likely next event. The same authors corroborated the applicability of their approach presented in [18] with an elaborate evaluation in

2017 [19]. Tax et al. followed their example, also in 2017 [45], providing an elaborate evaluation of deep neural networks on the task of next event prediction. Di Francescomarino et al., later in 2017, expanded this idea by incorporating domain knowledge to enhance the quality of the predictions [17]. The task of next event prediction has since been researched rather frequently [14, 30, 32, 33, 39].

As we have seen, important work has been published on the task of next event prediction. Yet, to the best of our knowledge, none of the concepts has been applied to process anomaly detection, despite the tasks being connected. To be able to accurately predict the next event, the neural network requires a deep understanding of the underlying process. If the actual next event in a case does not match the predictions of a next event prediction model, it can be regarded as anomalous. This concept serves as the basis for the work on BINet in Publications P3 and P4.

2.5 SHORTCOMINGS OF PROCESS ANOMALY DETECTION

Process anomaly detection, unlike conformance checking, is lacking the ability to provide explanations for identified anomalies. Namely the detection resolution of a process anomaly detection algorithm should cover not only the case dimension but also the event and attribute dimensions of an event log. Lower-level detection is an essential aspect of being able to suggest a possible correction of an anomalous case.

A key requirement for process anomaly detection is that no a priori knowledge about the process is necessary for it to operate. The understanding of the process behavior must be inferred from an event log. Furthermore, the event log cannot be assumed to only contain normal process behavior. Anomalies are naturally contained in it. An event log holds data about the control-flow perspective of a process, as well as, information about the data perspective. Attributes from the data perspective are usually either categorical or numerical in nature. To properly model the underlying process, an anomaly detection algorithm should cover these perspectives.

The requirement of not relying on a priori knowledge about the process can be extended to include the user of the algorithm itself. No assumption should be made about the expertise of the user with regard to the process. Hence, auto-parameterization is essential. Process anomaly detection lends itself to autonomous operation, but this requires an automatic parameterization of the thresholds that the algorithms rely on.

Table 2.2 summarizes the state-of-the-art related to process anomaly detection as described before and highlights the novelties of the individual contributions. Note that the last five columns of Table 2.2 do not contain any check marks and indicate five areas for improvement. We

Table 2.2: Summary of the state-of-the-art with respect to desirable features; novelties to the state-of-the-art (omitting the contributions of this work) are highlighted in red

	Aalst 2005 [3]	Bezerra 2009 [6–9]	Böhmer 2016 [10]	Böhmer 2018 [11, 13]	Pauwels 2019 [40, 41]
Method	Process Discovery	Process Discovery	Probabilistic Model	Association Rules	Dynamic Bayesian Networks
No a priori knowledge		✓	✓	✓	✓
Noisy event logs		✓			✓
Control-flow perspective	✓	✓	✓	✓	✓
Data perspective (cat.)			✓	✓	✓
Data perspective (num.)				✓	✓
Case-level detection	✓	✓	✓	✓	✓
Event-level detection					
Attribute-level detection					
Auto-parameterization					
Classification					
Correction					

can see that none of these algorithms supports multiple perspectives in the detection result. Furthermore, none of the approaches provides an auto-parameterization solution. In every method, parameters have to be defined and fine-tuned to ensure good results. These are significant shortcomings of process anomaly detection that, so far, have not been sufficiently addressed. Lastly, the concept of alignments and corrective functionality has not yet been incorporated into process anomaly detection, as it has been done for conformance checking.

Conformance checking has addressed these issues by incorporating a predefined process model. To provide the same functionality without relying on such a predefined process model, these five issues needed to be addressed. The contributions of this dissertation demonstrate how process anomaly detection can gradually be transformed into *process anomaly correction*, by iteratively addressing the five aforementioned issues.

These five open issues are connected to the four steps from the introduction section (see Figure 1.2) in the following way: (1) *Process anomaly correction* employs the concept of *process learning* to learn the business process from the event log. (2) This model is the basis to provide event and case level detection capabilities, for *process learning* does not regard a case of an event log as a single unit, but rather as a sequence of events and their corresponding attributes . (3) The first step is also essential to automatically parameterize the necessary threshold function because the process learning model, being a probabilistic model, encapsulates information about the distribution of normal and anomalous cases in the event log. (4) Lastly, classification and correction can be realized by exploiting the predictive capabilities of the process learning model.

3 CONTRIBUTIONS

In Chapter 2 we saw that to combine the benefits of *conformance checking* and *process anomaly detection*, a new method was necessary. We identified five areas of improvement in state-of-the-art process anomaly detection. To improve upon these areas, this thesis addresses three research questions (RQ1–RQ3). Figure 3.1 shows the current state-of-the-art, as outlined in the introduction, and positions the respective research questions. The three research questions are formulated as follows:

RQ1 *How can process anomaly detection provide more detailed results that allow pinpointing anomalies according to the different process perspectives?*

To combine the benefits of conformance checking and process anomaly detection, it is necessary that the anomalies can be detected on a more granular level. If the detection is based on entire cases of an event log, the detected anomaly cannot be corrected on a lower level. Hence, event-level and attribute-level detection is necessary.

Publication P1 (conference paper) and P2 (extended journal article) address RQ1.

RQ2 *How can process anomaly detection be automatically parameterized, without relying on external input?*

Most conformance checking approaches do not rely on external input to operate. To provide the same experience with a process anomaly detection algorithm, automatic parameterization is necessary.

Publication P3 (conference paper) and P4 (extended journal article) address RQ2.

RQ3 *How can the concept of alignments from conformance checking be transferred to process anomaly detection in order to provide process anomaly correction?*

To transfer the concept of alignments from conformance checking, where cases of an event log are aligned with a process model, a new method is necessary that aligns cases of event logs with a process learning model.

Publication P5 (conference paper) addresses RQ3.

The following sections are dedicated to the three research questions and how the five individual publications relate to them. In Section 3.1,

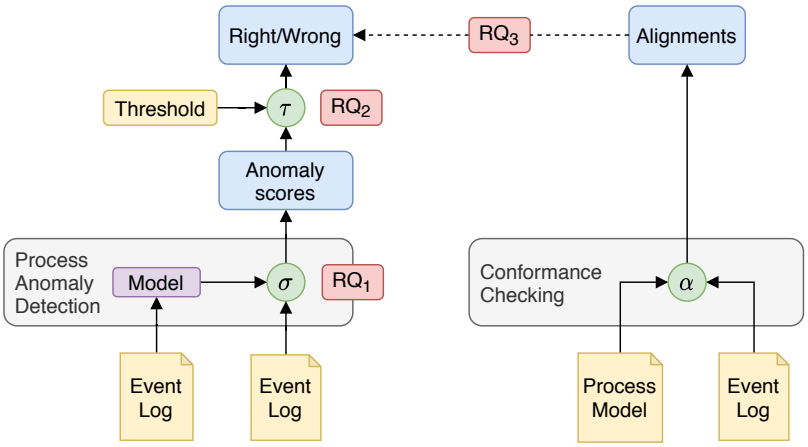

Figure 3.1: Current state-of-the-art (left part of Figure 1.2 from the introduction) and how the three research questions relate to it

two publications on the application of denoising autoencoders are presented. It is demonstrated how they can be employed to first learn processes from event logs, and secondly how multi-perspective detection can be realized. Section 3.2 summarizes our two publications on recurrent neural networks and how their predictive capabilities can be utilized for automatic parameterization, as well as anomaly classification. Section 3.3 is dedicated to alignments and how they can be obtained from the recurrent neural networks. This chapter closes with a short summary in Section 3.4.

3.1 RQ1: PROVIDING MULTI-PERSPECTIVE DETECTION

With the first two publications, P1 and P2, we focused on RQ1. To work towards RQ1, the way in which process anomaly detection algorithms had been modeling the process based on the event log had to be improved. Denoising autoencoders (DAE) are commonly utilized for classic anomaly detection tasks (see [23, 25]).

An autoencoder is a neural network that is trained to predict its own input. Since the input and the expected output of the autoencoder are the same, an autoencoder could simply learn the identity function and thus would not learn anything reasonable. Two techniques are commonly employed to guarantee that the autoencoder does not learn the identity function. The first one is to limit the number of available neurons within a central layer of the neural network. This limitation of capacity creates a bottleneck that forces the autoencoder to *encode* (as in lossy compression) the input so that the original sequence can still be reproduced (decoded) from it.

The second technique to suppress the learning of the identity function is to not give the autoencoder access to the original input but, instead, to an altered version of it. A typical alteration is the addition of Gaussian noise to the input. Since the input now differs from the

expected output, the autoencoder cannot learn the identity function. The addition of noise to the input of an autoencoder transforms the task into a *denoising task*, and hence these autoencoders were given the name *denoising autoencoders*.

Based on the assumption that anomalous cases in the event log are outnumbered by normal cases, the hypothesis is that the autoencoder will learn to correctly reproduce the normal cases while ignoring the anomalous ones. Any case where the mean absolute error between the original input of the autoencoder (without noise) and its output exceeds a threshold is regarded to be anomalous.

However, autoencoders were not directly applicable to sequential process data. Moreover, as most anomaly detection methods did, they treated the task as a binary classification problem (normal vs. anomalous). To address RQ1, we had to extend the idea of denoising autoencoders to fit the sequential nature of process data.

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

In this work, we demonstrated that denoising autoencoders can be extended to learn sequences of events coming from a noisy event log by transforming the event log into a 2-dimensional tensor. Even if not all cases have the same length, they can be fit into a 2-dimensional tensor by padding shorter sequences with zeros.

We also showed that both the input and the output of the autoencoder can be split up along the time dimension of the original sequence, and hence the error can be computed for individual events of a case, rather than the whole case at once. This novelty to autoencoders addressed RQ1 regarding the control-flow perspective.

> **Publication:** This section summarized our work on denoising autoencoders from 2016 [36]. The complete publication can be found in Part II as Publication P1.
>
> **Contribution statement:** I led the idea generation, implemented the prototype, performed the data evaluation, and formulated the manuscript. Alexander Seeliger and Max Mühlhäuser contributed to the conceptual design and the writing process.

3.1.2 *Analyzing Business Process Anomalies Using Autoencoders*

The previous work on denoising autoencoders had demonstrated that event level process anomaly detection was possible. In this paper, we extend this idea to provide detection on event attribute level, incorporating the data perspective. Instead of splitting the inputs and outputs of the autoencoder only along the time dimension, additional dimensions can be added to facilitate the inclusion of the event attributes.

This novel way of detecting the anomalies allowed for faster insights on which attribute might be driving the anomaly. Altogether, we provided attribute-level process anomaly detection, answering RQ1.

To demonstrate the significance in detection quality, we conducted a comprehensive evaluation of state-of-the-art anomaly detection techniques for discrete sequences, including all process anomaly detection algorithms known to us at the time. By evaluating the performance of all approaches on an elaborate data corpus of 600 synthetic and 100 real-life event logs, we corroborated our preliminary results from 2016. The denoising autoencoder approach outperformed all other methods.

As a contribution to the community, we published the source code of all algorithms, the generation algorithm, as well as all datasets used in the evaluation to serve as a process anomaly detection benchmark to future research.

> **Publication:** This section summarized our extension to the previous publication in 2018 [34]. The complete publication can be found in Part II as Publication P2.
>
> **Contribution statement:** I led the idea generation, implemented the prototype, performed the data evaluation, and formulated the manuscript. Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser contributed to the conceptual design and the writing process.

3.1.3 *Discussion*

Our work on denoising autoencoders for process anomaly detection addressed RQ1. To provide event and attribute-level detection, our solution relies on a new thresholding function that allows to set thresholds for events and attributes separately. This new thresholding function provides the base to work towards RQ2 because it can be automatically parameterized by the process learning model, as we will see in the next section. Figure 3.2 shows the process anomaly detection architecture after the two contributions, P1 and P2. The underlying process is approximated through process learning. The resulting process learning model is used by the scoring function σ' to analyze event logs on the event and attribute levels. Thus, RQ1 is answered.

3.2 RQ2: PROVIDING AUTOMATIC PARAMETERIZATION

The following two publications, P3 and P4, focused on RQ2, and partly on RQ3. Our research on denoising autoencoders had shown that a global threshold is not sufficient to accurately detect anomalies on event and attribute level. To automatically parameterize the threshold function and address RQ2, the predictive capabilities of the process

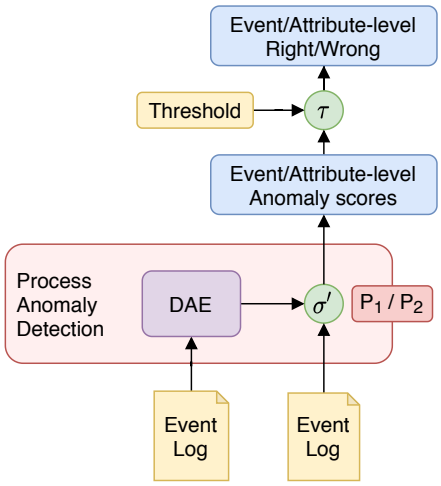

Figure 3.2: Process anomaly detection architecture including the contributions of P1 and P2

learning model can be utilized. Therefore, no global threshold is used but instead, a dynamic threshold is derived for each case in the event log based on the predictions (and their corresponding probabilities) of the process learning model. To provide the necessary predictive capabilities, the denoising autoencoder approach was improved by introducing a recurrent neural network architecture.

We had already hinted at the possibility of basing process anomaly detection on a recurrent neural network in the discussion of our paper from 2016. The reasoning behind this idea is that a recurrent neural network is better suited to process sequential data since it is equipped with an internal memory that allows it to remember important events while ignoring others. Coincidentally, the work on next event prediction [18, 19, 45], as outlined in Chapter 2, had already demonstrated the benefits of utilizing recurrent neural networks in the context of BPM.

The problem of next event prediction can be restated to fit our definition of a process anomaly detection algorithm since divergence from the predicted next event can be seen as a strong indicator for an anomaly. Not only do recurrent neural networks aid the auto-parameterization of the threshold, but they also bring us one step closer to answering RQ3 because they provide the necessary predictive capabilities to correct a detected anomaly.

3.2.1 *BINet: Multivariate Business Process Anomaly Detection Using Deep Learning*

In this paper, we proposed a novel neural network architecture that was tailored towards the structure of event logs. The neural network architecture was named BINet (business intelligence network). As

indicated before, it was based on a long short-term memory [24] network and was trained on the task of next event prediction. In comparison to existing next event prediction solutions, BINet featured a specialized structure to incorporate the data perspective in the learning process. To the best of our knowledge, BINet was the first contribution to utilize recurrent neural networks for the purposes of process anomaly detection.

To detect anomalies, BINet assigns anomaly scores to every attribute in every event, based on the probability of the respective attribute occurring in the next event in the case. Anomaly scores are normal practice when it comes to anomaly detection algorithms. However, one must still define a threshold to define whether an anomaly score is high enough to indicate an anomaly. Within this paper, we proposed the use of the *elbow heuristic* to mimic the human intuition when assigning a threshold manually (e.g., by moving slider in a graphical user interface). We demonstrated that this heuristic outperforms existing state-of-the-art approaches. Furthermore, we demonstrated that the same heuristic can be applied to other process anomaly detection algorithms, namely Bezerra's and Böhmer's approaches. Replacing the existing threshold heuristics in these two methods resulted in much better performance of the algorithms compared to their non-optimized versions.

Finally, we evaluated BINet against state-of-the-art approaches, including the optimized versions of Böhmer and Bezerra, and our own denoising autoencoder approach. The results showed that BINet outperformed all other approaches. Especially compared to our denoising autoencoder solution, BINet reached significantly better results by exploiting the time dependencies in the data.

> **Publication:** This section summarized our work on the application of recurrent neural networks for process anomaly detection, called BINet, from 2018 [37]. The complete publication can be found in Part II as Publication P3.
>
> **Contribution statement:** I led the idea generation, implemented the prototype, performed the data evaluation, and formulated the manuscript. Alexander Seeliger and Max Mühlhäuser contributed to the conceptual design and the writing process.

3.2.2 *BINet: Multi-perspective Business Process Anomaly Classification*

In this paper, we extended our work on BINet from 2018. We demonstrated that utilizing the predictive capabilities of BINet, a simple rule-based classifier can be constructed to identify different types of anomalies, such as rework, late execution, or early execution. Moreover, we optimized the BINet neural architecture with respect to computational efficiency and proposed three different versions of it. These

versions differ in the data dependencies they can model, namely: no dependencies (only the control-flow is used), dependencies between the event activity and event attributes (e.g., a review must be done by a reviewer), dependencies between event attributes (e.g., reviewer 1 always works Sundays). Further, we refined the threshold heuristics from the previous publication, and introduced the *lowest-plateau heuristic* to more closely mimic human intuition.

This publication addresses a part of RQ3 since it provides anomaly classifications. To our knowledge, no other process anomaly detection algorithm had incorporated anomaly classification.

> **Publication:** This section summarized our extension on the work on BINet from 2019 [35]. The complete publication can be found in Part II as Publication P4.
>
> **Contribution statement:** I led the idea generation, implemented the prototype, performed the data evaluation, and formulated the manuscript. Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser contributed to the conceptual design and the writing process.

3.2.3 *Discussion*

The BINet architecture had significantly improved the quality of the results compared to other process anomaly detection algorithms. Additionally, the predictive capabilities of the approach proved to be a solid foundation for an anomaly classifier. The introduction of the elbow heuristic and its extension, the lowest plateau heuristic, allowed for anomaly score based anomaly detection algorithms to be transformed into auto-parameterized algorithms. With our work on BINet, we answered RQ2 and made significant progress towards RQ3. Figure 3.3 shows how P3 and P4 contributed to the process anomaly detection architecture, now providing process anomaly classification.

3.3 RQ3: BRINGING ALIGNMENTS TO PROCESS ANOMALY DETECTION

To be able to correct detected anomalies and thus answer RQ3, the learned process model, as approximated by BINet as explained above, had to be made accessible. Similar to how alignments in conformance checking indicate skipped and incorrect events, BINet had to be utilized to alter sequences of events by removing unnecessary events and adding skipped events.

In natural language processing, recurrent neural networks are often utilized to generate parts of or even full sentences. Given the beginning of a sentence, a language model, approximated by a recurrent neural network, can generate, word for word, a meaningful contin-

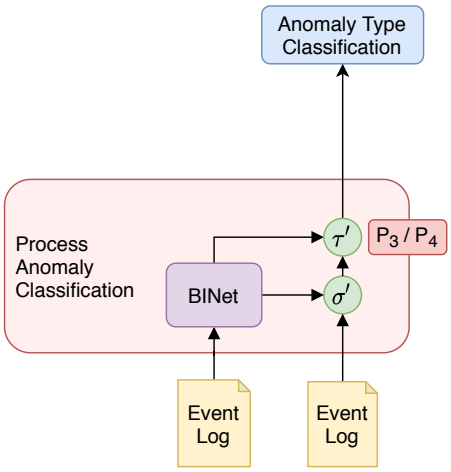

Figure 3.3: Process anomaly classification architecture after the addition of BINet from P3 and P4

uation of the sentence. Furthermore, a special case of this problem had been addressed by Sun et al. in 2017 [44]. Instead of generating the continuation until the end of a sentence, they proposed a problem in which a gap in an existing sentence had to be filled with a meaningful continuation, satisfying both the beginning and the end of the sentence. They based their solution on two separate neural networks, one reading sentences from the left, and another reading sentences from the right. A bidirectional beam search can then be employed to generate sentences that fit the gaps from both sides.

To correct an anomalous case in which certain events are missing, this method seemed promising. However, we had to generalize the problem from one single gap to an arbitrary number of gaps in the sequence. Moreover, we also needed to be able to remove incorrect events when encountering them, which was not part of the original solution in [44].

To answer the final research question RQ3, we had to develop a new method, following the example of Sun et al. [44], that supports the BINet recurrent architecture, incorporates the data perspective which does not exist in natural language, and alters the algorithm to allow for arbitrary numbers of gaps, as well as the removal of incorrect events from the sequence.

3.3.1 *DeepAlign: Alignment-based Process Anomaly Correction Using Recurrent Neural Networks*

In this paper, we introduced the concept of aligning a case with a process model, approximated by two separate BINet models, one reading cases left-to-right, and one reading right-to-left, to the field of process anomaly detection. Although bidirectional beam search

was an existing technique, the application to generate alignments in process anomaly detection was novel to the field.

DeepAlign, as we named it, is based on two identical but separate BINet models. One is processing the cases from the left (forward), while the other is processing them from the right (backward). Both of them are trained on the task of next event prediction. BINet can be utilized to calculate the probability of an event given a sequence of preceding events. By iteratively calculating probabilities, the individual probability for each event in a sequence can be computed. These probabilities are computed both for the forward BINet and the backward BINet and are subsequently combined into a joint probability for each event in the case.

To find the best alterations to a sequence, all possible single operation alterations are calculated, namely deletion, insertion (of any activity at any position), or leaving the sequence as is. The altered sequences are ranked by their respective probability according to both BINet models. Only the top-k sequences are selected and used for the next iteration. This procedure is repeated until convergence, that is, when no alteration to the top-k resulting sequences would yield a higher probability. We demonstrated that the resulting sequence of alterations to the original case can be transformed into a valid alignment as defined in [12].

Comparing the performance of the DeepAlign algorithm to state-of-the-art conformance checking methods based on alignments showed that DeepAlign is able to outperform existing approaches both in the control-flow perspective as well as in the data perspective.

Lastly, we showed that the DeepAlign algorithm can be utilized to generate cases according to the rules of the process from scratch by starting from an empty sequence. Depending on the case attribute values, different sequences resulted from the algorithm. In the context of the scientific paper writing and reviewing process from the introduction, setting a case attribute *publication type* to *conference* would result in a different sequence than setting it to *journal* because the review process for a journal submission is different from the review process for a conference submission. This fact suggests that BINet had successfully learned the impact of the case attributes on the corresponding control-flow sequences.

> **Publication:** This section summarized our work on the application of bidirectional beam search to produce alignments for process anomaly correction from 2020 [38]. The complete publication can be found in Part II as Publication P5.
>
> **Contribution statement:** I led the idea generation, implemented the prototype, performed the data evaluation, and formulated the manuscript. Alexander Seeliger, Nils Thoma, and Max Mühl-

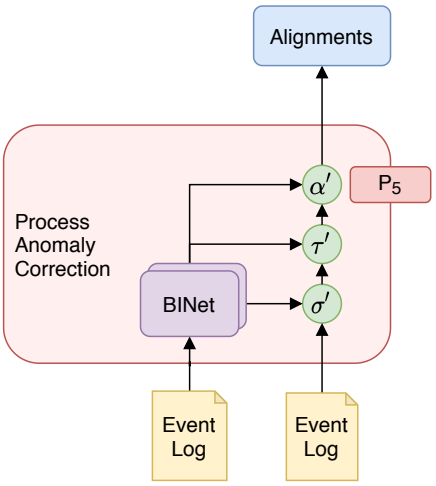

Figure 3.4: Process anomaly correction architecture after the addition of DeepAlign from P5; this architecture corresponds to the right side of Figure 1.2 from the introduction

> häuser contributed to the conceptual design and the writing process.

3.3.2 *Discussion*

The addition of the DeepAlign algorithm to the architecture achieved *process anomaly correction*. Figure 3.4 shows the final *process anomaly correction* architecture. In Figure P5.1, α' corresponds to the DeepAlign algorithm. *Process anomaly correction* now combines the benefits of process anomaly detection (not relying on a process model) and conformance checking (alignments), while providing additional benefits, such as noisy event log handling, auto-parameterization, and memory efficiency. Detected anomalies could be corrected on the fly and incorrect cases could be aligned with the process as modeled by the two BINets. With the publication on DeepAlign, P5, we have provided an answer to RQ3.

3.4 SUMMARY

Table 3.1 shows the complete history of the state-of-the-art, including the contributions of this work. These five publications included in this thesis, in their entirety, respond to the three research questions set forth in the beginning, coined as RQ1–RQ3. Two particularly notable characteristics of the contributions reported here are these: On the one hand, they successively answered the research questions RQ1–RQ3 which had been identified as major deficiencies in the state-of-the-art. On the other hand, they drew level or even outperformed existing approaches with respect to detection performance.

The resulting *process anomaly correction* approach does not rely on a predefined process model but provides the benefit of alignments known from conformance checking. As such, it can be used as a replacement for classic conformance checking in cases where no predefined process model *is* available. Furthermore, it can be a viable option even in cases where a predefined process model is available since it provides support for multiple perspectives. Compared to classic conformance checking, it incorporates the data perspective without requiring a manual definition of the rules connected to this perspective. The implications of *process anomaly correction* as a new approach and *process learning* as a general concept as well as their general applicability will be discussed in the following chapter.

Table 3.1: Comparison of the solutions presented in this thesis and the state-of-the-art; novelties to the field are highlighted in red

	Aalst 2005 [3]	Bezerra 2009 [6–9]	Böhmer 2016 [10]	Nolle 2016 P1	Nolle 2018 P2	Böhmer 2018 [11, 13]	Nolle 2018 P3	Nolle 2019 P4	Pauwels 2019 [40, 41]	Nolle 2020 P5
Method	Process Discovery	Process Discovery	Probabilistic Model	Process Learning	Process Learning	Association Rules	Process Learning	Process Learning	Dynamic Bayesian Networks	Process Learning
No a priori knowledge		✓	✓	✓	✓	✓	✓	✓	✓	✓
Noisy event logs		✓		✓	✓		✓	✓	✓	✓
Control-flow perspective	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Data perspective (cat.)			✓		✓	✓	✓	✓	✓	✓
Data perspective (num.)						✓			✓	
Case-level detection	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Event-level detection				✓	✓		✓	✓		✓
Attribute-level detection					✓		✓	✓		✓
Auto-parameterization							✓	✓		✓
Classification								✓		✓
Correction										✓

4

CONCLUSION AND OUTLOOK

This dissertation opened with the example of a mentor teaching us the process of writing a scientific paper, by pointing out our mistakes, but also by suggesting possible corrections. The example served as a reminder that advanced process analytics methodology, such as process anomaly detection and conformance checking, could be improved by adopting this principle. Without the expectation of a priori knowledge, these methods should provide easily interpretable, and actionable results.

While conformance checking provides error detection as well as correction, it requires a predefined process model, which in most circumstances can only be provided by an expert in the process itself. So, we would first have to teach the mentors in order to get answers from them. Process anomaly detection, on the other hand, does not require a priori knowledge, but provides results in the form of *case is normal* or *case is anomalous*. A mentor that only ever answers questions with "Right!" or "Wrong!" is not a good mentor. After all, the students want to know *why* something is right or wrong.

4.1 SUMMARY OF ACHIEVEMENTS

The contribution of this dissertation is the combination of the benefits from both conformance checking and process anomaly detection to create a new method, *process anomaly correction*. Without relying on a priori knowledge about the process, *process anomaly correction* can provide comparable, if not better, results than classical conformance checking. By lifting the restriction of conformance checking relying on predefined process models, *process anomaly correction* can provide the same quality of analysis across a variety of different scenarios, which until now, required a process model to be created.

This dissertation further serves as an example that *process learning* as a general concept can be utilized to infer complex process logic from event logs, without being specifically programmed. The use case of *process anomaly correction* has shown that *process learning* is able to model dependencies between the different process perspectives. *Process learning* significantly reduces the effort of adding new concepts to a process model. As long as the event log contains a sufficient amount of examples of the concept, *process learning* can pick up on the emerging patterns. *Process learning* is mature enough to be applied in various other scenarios apart from *process anomaly correction*. It offers a solid foundation for promising future research in the field of BPM.

4.2 FUTURE WORK

This section shall give an overview of possible continuations of this work regarding *process anomaly correction* as well as novel applications for *process learning*.

GENERATIVE CAPABILITIES Process learning is a promising concept and its application areas are manifold. For instance, it is conceivable to create a process discovery algorithm by exploiting the generative capabilities of deep neural networks. They are also commonly used to translate sentences from one language to another. Similar ideas could be applied to, perhaps, translate one process into another, as in migrating a process from one system to another.

EXPLAINABLE MACHINE LEARNING While deep learning techniques can produce remarkable results, the research community around them still struggles to understand how they accomplish these feats. It is important to not blindly trust such a system. No guarantees can be made (yet) that their understanding of the data is what it is intended to be. While the presented *process anomaly correction* technique proved to be successful in the tasks it was evaluated on, the success relies on the assumption that the process to be learned is adequately resembled by the event log.

Various research exists around the idea of making deep neural networks explainable. For instance, by adding a special attention layer that forces a network during the training to select only the strongest bits of information from its input, thereby creating the notion of focus. The attention of the network can then be visualized to demonstrate how the network likely perceives an input. Possibly, insights about how a neural network models the underlying process could be derived.

NUMERICAL ATTRIBUTES This work focused on categorical attributes since most attributes coming from the control-flow (*Research Related Work* vs. *Conclude*) and the organizational perspective (reviewer vs. student) are categorical. However, numerical attributes (cost, duration, etc.) are important to fully support the data perspective. The challenge of learning relations between numerical and categorical attributes simultaneously, which requires a synchronization of two separate loss functions, is an interesting opportunity for future research.

DEEP LEARNING LIMITATIONS The proposed solution is based on process learning and hence requires a lengthy training period prior to the actual detection of anomalies. Time spent waiting for the training procedure to finish might be gained in the analysis of the results. However, it remains an open question. Moreover, deep

learning approaches require the use of hardware accelerators like graphics processing units (GPUs) or special tensor processing units (TPUs) to train faster. Memory on GPUs and TPUs is still limited and thus training on big event logs is still challenging.

DEALING WITH CONCEPT DRIFT The assumption that the process is static does not always hold. Event logs are ever-changing since they hold information about running, real-life processes. To offer robust performance, concept drift needs to be accounted for. A possible approach is to assign unseen values to an *other group* and to treat them all the same. Another solution is to run the training continuously so that new concepts in the process are learned while they are appearing.

BIBLIOGRAPHY

[1] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Vol. 2. Springer, 2011 (cit. on p. 3).

[2] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016 (cit. on pp. 3, 9).

[3] Wil M. P. van der Aalst and Ana Karla A de Medeiros. "Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance." In: *Electronic Notes in Theoretical Computer Science* 121 (2005), pp. 3–21 (cit. on pp. 4, 11, 15, 19, 32).

[4] Arya Adriansyah, Boudewijn F van Dongen, and Wil M. P. van der Aalst. "Memory-efficient alignment of observed and modeled behavior." In: *BPM Center Report* 3 (2013) (cit. on p. 15).

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on p. 17).

[6] Fábio Bezerra and Jacques Wainer. "Anomaly Detection Algorithms in Logs of Process Aware Systems." In: *Proceedings of the 23rd Annual ACM Symposium on Applied Computing – SAC'08*. 2008, pp. 951–952 (cit. on pp. 12, 19, 32).

[7] Fábio Bezerra and Jacques Wainer. "Anomaly detection algorithms in business process logs." In: *Proceedings of the 10th International Conference on Enterprise Information Systems – ICEIS'08*. 2008, pp. 11–18 (cit. on pp. 12, 19, 32).

[8] Fábio Bezerra and Jacques Wainer. "Algorithms for Anomaly Detection of Traces in Logs of Process Aware Information Systems." In: *Information Systems* 38.1 (2013), pp. 33–44 (cit. on pp. 12, 19, 32).

[9] Fábio Bezerra, Jacques Wainer, and Wil M. P. van der Aalst. "Anomaly Detection Using Process Mining." In: *Proceedings of the 10th International Workshop on Enterprise, Business-Process and Information Systems Modeling – BPMDS'09*. Springer, 2009, pp. 149–161 (cit. on pp. 12, 19, 32).

[10] Kristof Böhmer and Stefanie Rinderle-Ma. "Multi-perspective Anomaly Detection in Business Process Execution Events." In: *Proceedings of On the Move to Meaningful Internet Systems – OTM'16*. Springer. 2016, pp. 80–98 (cit. on pp. 13, 14, 19, 32).

[11] Kristof Böhmer and Stefanie Rinderle-Ma. "Association rules for anomaly detection and root cause analysis in process executions." In: *Proceedings of the 30th International Conference on Advanced Information Systems Engineering – CAiSE'18*. Springer. 2018, pp. 3–18 (cit. on pp. 13, 19, 32).

[12] RP Jagadeesh Chandra Bose and Wil M. P. van der Aalst. "Trace alignment in process mining: opportunities for process diagnostics." In: *Proceedings of the 8th International Conference on Business Process Management – BPM'10*. Springer, 2010, pp. 227–242 (cit. on p. 29).

[13] Kristof Böhmer and Stefanie Rinderle-Ma. "Mining association rules for anomaly detection in dynamic process runtime behavior and explaining the root cause to users." In: *Information Systems* 90 (2020), p. 101438. ISSN: 0306-4379 (cit. on pp. 13, 19, 32).

[14] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. "Learning accurate LSTM models of business processes." In: *Proceedings of the 17th International Conference on Business Process Management – BPM'19*. 2019, pp. 286–302 (cit. on p. 18).

[15] Bill Curtis, Marc I. Kellner, and Jim Over. "Process modeling." In: *Communications of the ACM* 35.9 (1992), pp. 75–90 (cit. on p. 3).

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018) (cit. on p. 17).

[17] Chiara Di Francescomarino, Chiara Ghidini, Fabrizio Maria Maggi, Giulio Petrucci, and Anton Yeshchenko. "An eye into the future: leveraging a-priori knowledge in predictive business process monitoring." In: *Proceedings of the 15th International Conference on Business Process Management – BPM'17*. 2017, pp. 252–268 (cit. on p. 18).

[18] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "A Deep Learning Approach for Predicting Process Behaviour at Runtime." In: *Proceedings of the 14th International Conference on Business Process Management – BPM'16*. Springer. 2016, pp. 327–338 (cit. on pp. 17, 25).

[19] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "Predicting Process Behaviour Using Deep Learning." In: *Decision Support Systems* 100 (2017), pp. 129–140 (cit. on pp. 18, 25).

[20] Mostafa Farshchi, Jean-Guy Schneider, Ingo Weber, and John Grundy. "Metric selection and anomaly detection for cloud operations using log and metric correlation analysis." In: *Journal of Systems and Software* 137 (2018), pp. 531–549 (cit. on p. 16).

[21] Mostafa Farshchi et al. "Contextual Anomaly Detection for a Critical Industrial System Based on Logs and Metrics." In: *Proceedings of the 14th European Dependable Computing Conference – EDCC'18*. 2018, pp. 140–143 (cit. on p. 16).

[22] Min Fu et al. "Process-Oriented Non-intrusive Recovery for Sporadic Operations on Cloud." In: *Proceedings of the 46th International Conference on Dependable Systems and Networks – DSN'16*. 2016, pp. 85–96 (cit. on p. 16).

[23] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. "Outlier detection using replicator neural networks." In: *Data warehousing and knowledge discovery*. Springer, 2002, pp. 170–180 (cit. on p. 22).

[24] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on pp. 17, 26).

[25] Nathalie Japkowicz. "Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks." In: *Machine Learning* 42.1 (2001), pp. 97–122 (cit. on p. 22).

[26] Christopher Klinkmüller, Nick R. T. P. van Beest, and Ingo Weber. "Towards reliable predictive process monitoring." In: *Proceedings of the 30th International Conference on Advanced Information Systems Engineering – CAiSE'18*. Springer. 2018, pp. 163–181 (cit. on p. 17).

[27] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on p. 17).

[28] Massimiliano de Leoni and Wil M. P. van der Aalst. "Aligning Event Logs and Process Models for Multi-perspective Conformance Checking: An Approach Based on Integer Linear Programming." In: *Proceedings of the 11th International Conference on Business Process Management – BPM'13*. 2013, pp. 113–129 (cit. on p. 15).

[29] Massimiliano de Leoni and Wil M. P. van der Aalst. "Data-Aware Process Mining: Discovering Decisions in Processes Using Alignments." In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing – SAC'13*. 2013, pp. 1454–1461 (cit. on p. 15).

[30] Li Lin, Lijie Wen, and Jianmin Wang. "MM-Pred: a deep predictive model for multi-attribute event sequence." In: *Proceedings of the 19th International Conference on Data Mining – SIAM'19*. 2019, pp. 118–126 (cit. on p. 18).

[31] Felix Mannhardt, Massimiliano De Leoni, Hajo A Reijers, and Wil M. P. van der Aalst. "Balanced multi-perspective checking of process conformance." In: *Computing* 98.4 (2016), pp. 407–437 (cit. on p. 15).

[32] Nijat Mehdiyev, Joerg Evermann, and Peter Fettke. "A novel business process prediction model using a deep learning method." In: *Business & Information Systems Engineering* 62 (2020), pp. 143–757 (cit. on p. 18).

[33] Andreas Metzger, Johannes Franke, and Thomas Jansen. "Data-driven Deep Learning for Proactive Terminal Process Management." In: (2019), pp. 196–211 (cit. on p. 18).

[34] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. "Analyzing Business Process Anomalies Using Autoencoders." In: *Machine Learning* 107.11 (2018), pp. 1875–1893 (cit. on p. 24).

[35] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. "BINet: Multi-perspective Business Process Anomaly Classification." In: *Information Systems* (2019), p. 101458. ISSN: 0306-4379 (cit. on p. 27).

[36] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders." In: *Proceedings of the 19th International Conference on Discovery Science – DS'16*. Springer. 2016, pp. 442–456 (cit. on p. 23).

[37] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "BINet: Multivariate Business Process Anomaly Detection Using Deep Learning." In: *Proceedings of the 16th International Conference on Business Process Management – BPM'18*. 2018, pp. 271–287 (cit. on p. 26).

[38] Timo Nolle, Nils Thoma, Alexander Seeliger, and Max Mühlhäuser. "DeepAlign: Alignment-based Process Anomaly Correction using Recurrent Neural Networks." In: *Proceedings of the 32nd International Conference on Advanced Information Systems Engineering – CAiSE'20*. 2020, pp. 319–333 (cit. on p. 29).

[39] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, and Donato Malerba. "Using Convolutional Neural Networks for Predictive Process Analytics." In: *Proceedings of the 1st International Conference on Process Mining – ICPM'19*. 2019, pp. 129–136 (cit. on p. 18).

[40] Stephen Pauwels and Toon Calders. "An anomaly detection technique for business processes based on extended dynamic bayesian networks." In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM. 2019, pp. 494–501 (cit. on pp. 14, 19, 32).

[41] Stephen Pauwels and Toon Calders. "Detecting anomalies in hybrid business process logs." In: *ACM SIGAPP Applied Computing Review* 19.2 (2019), pp. 18–30 (cit. on pp. 14, 19, 32).

[42] Alec Radford et al. "Language models are unsupervised multitask learners." In: *OpenAI Blog* 1.8 (2019), p. 9 (cit. on p. 17).

[43] Anne Rozinat and Wil MP van der Aalst. "Conformance checking of processes based on monitoring real behavior." In: *Information Systems* 33.1 (2008), pp. 64–95 (cit. on pp. 4, 15).

[44] Qing Sun, Stefan Lee, and Dhruv Batra. "Bidirectional beam search: Forward-backward inference in neural sequence models for fill-in-the-blank image captioning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition – CVPR'17*. 2017, pp. 6961–6969 (cit. on p. 28).

[45] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. "Predictive Business Process Monitoring with LSTM Neural Networks." In: *Proceedings of the 29th International Conference on Advanced Information Systems Engineering – CAiSE'17*. Springer. 2017, pp. 477–492 (cit. on pp. 18, 25).

[46] Ashish Vaswani et al. "Attention is all you need." In: *Proceedings of Advances in neural information processing systems 30 – NeurIPS'17*. 2017, pp. 5998–6008 (cit. on p. 17).

[47] Ingo Weber, Chao Li, Len Bass, Xiwei Xu, and Liming Zhu. "Discovering and Visualizing Operations Processes with POD-Discovery and POD-Viz." In: *Proceedings of the 45th International Conference on Dependable Systems and Networks – DSN'15*. 2015, pp. 537–544 (cit. on p. 16).

[48] Ingo Weber, Andreas Rogge-Solti, Chao Li, and Jan Mendling. "CCaaS: Online Conformance Checking as a Service." In: *Proceedings of the BPM Demo Session 2015. Co-located with the 13th International Conference on Business Process Management – BPM'15* 1418 (2015), pp. 45–49 (cit. on p. 16).

[49] Xiwei Xu, Liming Zhu, Ingo Weber, Len Bass, and Daniel Sun. "POD-Diagnosis: Error Diagnosis of Sporadic Operations on Cloud Applications." In: *Proceedings of the 44th International Conference on Dependable Systems and Networks – DSN'14*. 2014, pp. 252–263 (cit. on p. 16).

[50] Xiwei Xu et al. "Crying wolf and meaning it: Reducing false alarms in monitoring of sporadic operations through POD-Monitor." In: *Proceedings of the 1st International Workshop on Complex Faults and Failures in Large Software Systems – COUFLESS'15*. 2015, pp. 69–75 (cit. on p. 16).

[51] Xiwei Xu et al. "Error diagnosis of cloud application operation using bayesian networks and online optimisation." In: *Proceedings of the 11th European Dependable Computing Conference – EDCC'15*. IEEE. 2015, pp. 37–48 (cit. on p. 16).

Part II

PUBLICATIONS

P1

UNSUPERVISED ANOMALY DETECTION IN NOISY BUSINESS PROCESS EVENT LOGS USING DENOISING AUTOENCODERS

Timo Nolle, Alexander Seeliger, and Max Mühlhäuser
In: *Proceedings of the 19th International Conference on Discovery Science – DS'16*. 2016, pp. 442–456.

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 percent F1-score in detecting anomalous traces in the event log, and 95.6 percent 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 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 refer-

ence 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,

Table P1.1: 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

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 P1.1 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 P1.2: 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

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 P1.2 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 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 percent for all the three mentioned mutations. Thus, the resulting event log will contain roughly ten percent anomalous traces, as shown in Table P1.2. 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 P1.2.

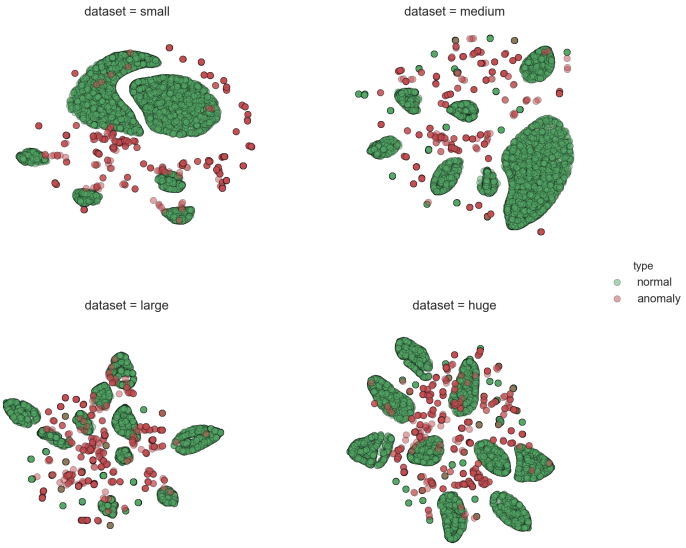

Figure P1.1: t-SNE visualization of the randomly generated datasets from Table P1.2

Figure P1.1 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. P1.2. 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. 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

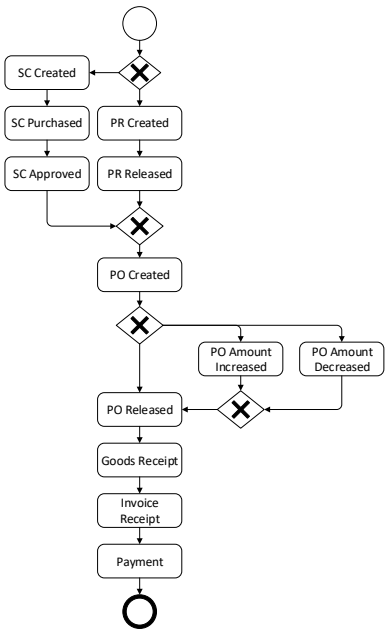

Figure P1.2: BPMN model of a simplified purchase to pay process

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

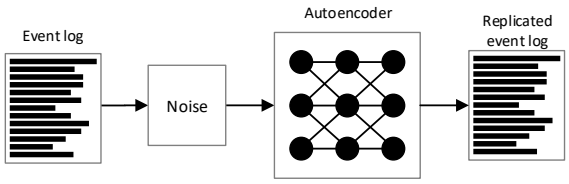

Figure P1.3: Autoencoder is trained to replicate the traces in the event log after the addition of gaussian noise

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 P1.3 shows a simplified version of the architecture. Notice that Fig P1.3 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 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 P1.3.

Table P1.3: 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

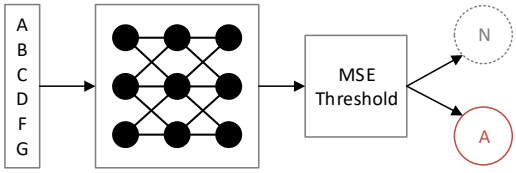

Figure P1.4: Threshold classifier based on the mean squared error between the input vector and the output of the autoencoder

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 P1.4 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. P1.5 in Sec. 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 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 P1.5 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 P1.5 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 P1.4 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 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 the P2P dataset autoencoder. Figure P1.6 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

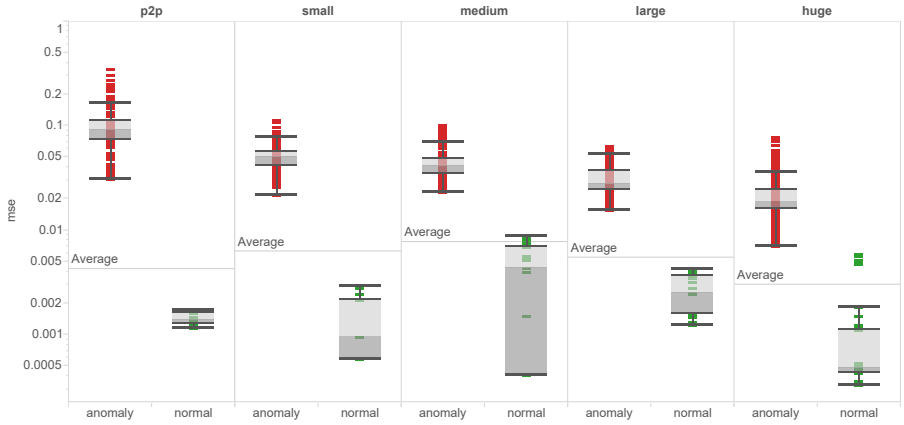

Figure P1.5: The autoencoder succeeds in perfectly splitting the dataset into normal and anomalous traces solely based on the reproduction error

the range from zero to the maximum reproduction error in the event log. When comparing the traces to the reference procurement model from Fig. P1.2, 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 P1.5 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 P1.5 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

Table P1.4: 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

Table P1.5: 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		

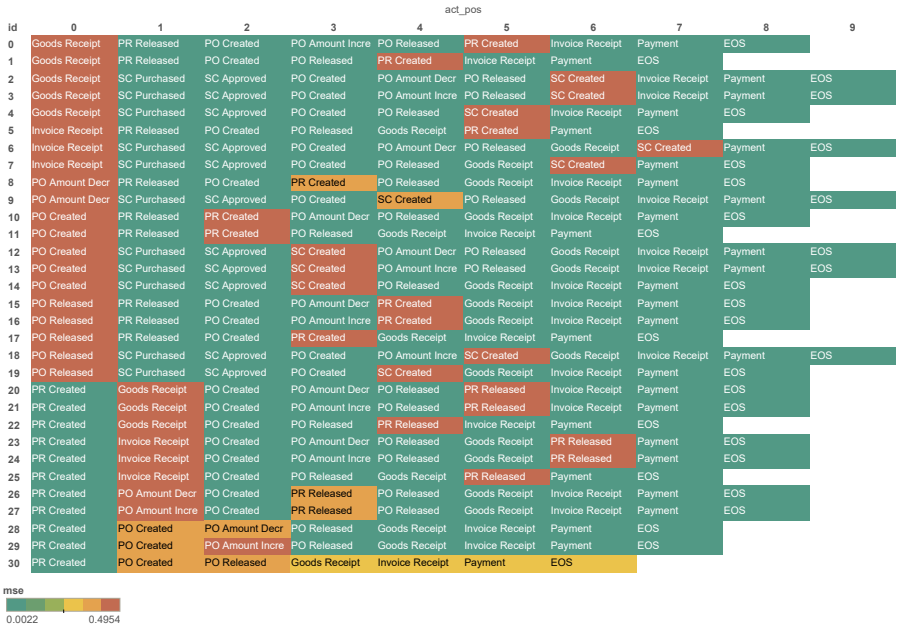

Figure P1.6: Conformance check on a sample of the P2P dataset

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 percent 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-Verbund-vorhaben (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] Wil Van der Aalst, Ton Weijters, and Laura Maruster. "Workflow mining: Discovering process models from event logs." In: *Knowledge and Data Engineering, IEEE Transactions on* 16.9 (2004), pp. 1128–1142 (cit. on p. 46).

[2] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. "Enhancing one-class support vector machines for unsupervised anomaly detection." In: *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*. ACM. 2013, pp. 8–15 (cit. on p. 48).

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on p. 51).

[4] Fábio Bezerra, Jacques Wainer, and Wil MP van der Aalst. "Anomaly detection using process mining." In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, pp. 149–161 (cit. on p. 47).

[5] Andrea Burattin. "PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings." In: *CoRR* abs/1506.0 (2015) (cit. on p. 49).

[6] Corinna Cortes and Vladimir Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on p. 48).

[7] Marlon Dumas, Wil M Van der Aalst, and Arthur H Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005 (cit. on p. 45).

[8] Eleazar Eskin. "Anomaly detection over noisy data using learned probability distributions." In: *In Proceedings of the International Conference on Machine Learning*. Citeseer. 2000 (cit. on p. 48).

[9] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. "Outlier detection using replicator neural networks." In: *Data warehousing and knowledge discovery*. Springer, 2002, pp. 170–180 (cit. on p. 47).

[10] Robert Hecht-Nielsen. "Replicator neural networks for universal optimal source coding." In: *Science* 269.5232 (1995), p. 1861 (cit. on p. 47).

[11] Geoffrey E Hinton. "Connectionist learning procedures." In: *Artificial intelligence* 40.1 (1989), pp. 185–234 (cit. on p. 47).

[12] Nathalie Japkowicz. "Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks." In: *Machine Learning* 42.1 (2001), pp. 97–122 (cit. on p. 47).

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 51).

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on p. 51).

[15] Laurens Van Der Maaten and Geoffrey E. Hinton. "Visualizing Data using t-SNE." In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605 (cit. on p. 50).

[16] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814 (cit. on p. 51).

[17] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. "A review of novelty detection." In: *Signal Processing* 99 (2014), pp. 215–249 (cit. on p. 47).

[18] Anne Rozinat and Wil MP van der Aalst. "Conformance checking of processes based on monitoring real behavior." In: *Information Systems* 33.1 (2008), pp. 64–95 (cit. on p. 46).

[19] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *Cognitive modeling* 5.3 (1988), p. 1 (cit. on p. 52).

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on p. 53).

[21] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning." In: *Proceedings of the 30th international conference on machine learning (ICML-13)*. 2013, pp. 1139–1147 (cit. on p. 53).

[22] Wil Van Der Aalst et al. "Process mining manifesto." In: *Business process management workshops*. Springer. 2011, pp. 169–194 (cit. on pp. 46, 47).

P2

ANALYZING BUSINESS PROCESS ANOMALIES USING AUTOENCODERS

Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser
In: *Machine Learning 107.11* (Nov. 2018), pp. 1875–1893.

ABSTRACT: Businesses are naturally interested in detecting anomalies in their internal processes, because these can be indicators for fraud and inefficiencies. Within the domain of business intelligence, classic anomaly detection is not very frequently researched. In this paper, we propose a method, using autoencoders, for detecting and analyzing anomalies occurring in the execution of a business process. Our method does not rely on any prior knowledge about the process and can be trained on a noisy dataset already containing the anomalies. We demonstrate its effectiveness by evaluating it on 700 different datasets and testing its performance against three state-of-the-art anomaly detection methods. This paper is an extension of our previous work from 2016 [30]. Compared to the original publication we have further refined the approach in terms of performance and conducted an elaborate evaluation on more sophisticated datasets including real-life event logs from the Business Process Intelligence Challenges of 2012 and 2017. In our experiments our approach reached an F_1 score of 0.87, whereas the best unaltered state-of-the-art approach reached an F_1 score of 0.72. Furthermore, our approach can be used to analyze the detected anomalies in terms of which event within one execution of the process causes the anomaly.

KEYWORDS: Deep Learning, Autoencoder, Anomaly Detection, Process Mining, Business Intelligence

1 INTRODUCTION

Anomaly detection is becoming an integral part of business intelligence. Businesses are naturally interested in detecting anomalies in their processes, because these can be indicators for inefficiencies in their process, badly trained employees, or even fraudulent behavior. Consequently, being able to detect such anomalies is of great value, for they can have enormous impact on the economic well-being of the businesses.

More and more companies rely on process-aware information systems (PAISs) [13] to improve their processes. This increasing number of PAISs has generated a lot of interest in the data these systems are gathering. The log files these systems are storing can be used to extract the events executed in the process, and thereby create so called event log files. Event logs are comprised of activities (and other miscellaneous information) that occurred during the execution of the process. These event logs enable process analysts to explore the underlying process. In other words, the event log consists of footprints of the process. Consequently, it is 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 [42].

Process mining provides methodologies to detect anomalies in the execution of a process; e.g., by discovering the as-is process model from the event log [1] using discovery algorithms and then comparing the discovered model to a reference model. This is known as *conformance checking* [35]. Another way of detecting anomalies is to compare the event log to the reference model. However, this approach requires the existence of such a reference model.

If no reference model is available, process mining relies on discovering a reference model from the event log itself [3, 4]. These methods make use of a threshold to deal with infrequent behavior in the log, so that the discovered model is a good representation of the normal behavior of the process. Hence, this model can be used as a reference model for the conformance check.

A key assumption in anomaly detection is that the anomalous executions occur less frequent than normal executions. This skewed distribution can be taken advantage of when applying anomaly detection techniques.

In this paper, we propose a method for detecting anomalies in business process data. Our method works under the following assumptions.

- No prior knowledge about the process
- Training data already contains anomalies
- No reference model needed
- No labels needed (i.e., no knowledge about anomalies)
- The algorithm must detect the exact activity at which the anomaly occurred

The system must deduce the difference between normal and anomalous executions 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.

The main contribution of this work is the application of an autoencoder to analyze the detected anomalies in terms of which event within a sequence is anomalous as opposed to the whole sequence at once. This can be refined further by analyzing which characteristic of the event (e.g., the executing user) is anomalous, not just the event itself. We demonstrate that, using this approach, we can accurately identify activities that have been executed in the wrong order, skipped, or unnecessarily reworked. Furthermore, we can detect when unauthorized users have illegally executed an activity.

To demonstrate the feasibility of our approach we compare its performance to seven state-of-the-art methods for anomaly detection. In addition to these six methods, we also present an adaptation of one of the methods. All methods were applied to a comprehensive set of 600 different artificial event logs featuring authentic business process anomalies as well as 100 real-life event logs coming from the Business Process Intelligence Challenge (BPIC).

In summary, the contributions of this paper are as follows.

1. Novel application of autoencoders to automatically analyze anomalies in the domain of business process intelligence.
2. Adaptation of the t-STIDE anomaly detection method from [43] to work with event attributes.
3. Comprehensive evaluation of state-of-the-art anomaly detection methods in the domain of business process intelligence.
4. Provision of a representative, labelled, set of artificial process event logs containing authentic anomalies.

2 RELATED WORK

In the field of process mining [42] anomaly detection is not very frequently researched. Most proposed methods work by using discovery algorithms to mine a reference model from the event log [4] and then using it for conformance checking to detect anomalous behavior. The bigger part of these methods relies on a clean dataset to work correctly. Unfortunately, this violates our assumptions, as the data coming from the PAISs will naturally contain anomalies.

Recently there has been some research on approaches that can deal with noisy event logs. Through the use of special discovery algorithms, that can deal with noise and infrequent behavior in the process, the approach from [4] can be refined to work with noisy logs [3]. The authors in [3] give three different algorithms in their paper. Within this work we will compare our approach to two of the proposed approaches.

A more recent publication proposes the use of likelihood graphs to analyze business process behavior [5]. Specifically, the authors

describe a method to extend the likelihood graph to include event attributes. This method works both on noisy event logs and includes important characteristics of the process itself by including the event attributes. We will also compare our method to the method from [5] in the evaluation section.

A review of classic anomaly detection methodology can be found in [32]. Here, the authors describe and compare many methods that have been proposed over the last decades. Another elaborate summary on anomaly detection in discrete sequences is given by Chandola in [7]. 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 can detect anomalies as they were sampled from a different distribution. In speech recognition [23], hidden Markov models (HMMs) [33, 34] are a popular choice for modeling sequential data. HMMs can also be used for anomaly detection as shown in [43] and [21], where they are used successfully for system intrusion detection. However, as Chandola pointed out in [8], the performance of such HMMs strongly depends on the fact that the raw data can be sufficiently modeled by a Markov process.

Another important probabilistic technique is the sliding window approach as proposed in [15], where it is used for intrusion detection. In window based anomaly detection, every window of a sequence is assigned an anomaly score. Then the anomaly score of the sequence can be inferred by aggregating the window anomaly scores. Recently, Wressnegger et al. used this approach for intrusion detection and give an elaborate evaluation in [45]. While being inexpensive and easy to implement, sliding window approaches show a robust performance in finding anomalies in sequential data, especially within short regions of the data [7].

Distance-based novelty detection 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. A popular distance-based approach is the one-class support vector machine (OC-SVM). Schölkopf et al. [37] first used support vector machines [9] for anomaly detection. Tax, in his PhD thesis [40], gives a sophisticated overview over one-class classification methods, also mentioning the OC-SVM. OC-SVMs have shown to be successful in the field of intrusion detection as demonstrated by [19].

Reconstruction-based novelty detection (e.g., neural networks) is similar to the aforementioned approaches in [17, 22]. 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 [14].

Domain-based novelty detection requires domain knowledge, which violates our assumption of no prior knowledge about the process. 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.

The approach within this paper is highly influenced by the works in [12, 17, 22], in which they propose the use of replicator neural networks [18] for anomaly detection, i.e., networks that reproduce their input, which are based on the idea of autoencoders from [20]. Autoassociative neural network encoders use a similar concept and have been used to model the nominal behavior of complex systems [41]. They have also been used for residual generation in [11], demonstrating that these models can also model behavior not directly observed in the training data, which increases generalization. A comprehensive study of replicator neural networks for outlier detection can be found in [44]. The approaches from [11, 12, 17, 22, 41], however, do not work well with variable length input. In our approach, we address this problem by using a padding technique. We opted to use a neural network based approach, for recent achievements in machine translation and natural language processing indicate that neural networks are an excellent choice when modeling sequential data [10, 26].

The main distinction between all other methods and the proposed approach is that it can be used to identify which exact event and furthermore which attribute characteristic is the cause of the anomaly. The only other approach that can deal with event attributes is the method from [5]. However, it can not deal with long-term dependencies, because it works on a general likelihood graph, which disregards the past events when calculating the probability of an event occurring at a specific point in the process. Our approach can deal both with the attributes and with non-local dependencies in the logs.

3 DATASET

PAISs keep a record of almost everything that happened during the execution of a business process. This information can be extracted from the systems in form of an event log. Event logs are the most common data structure when working with process data from PAISs, especially in the field of process mining.

3.1 *Event logs*

An event log consists of traces, each consisting of the activities that have been executed. Table P2.1 shows an excerpt of such an event log. In this case, it is representative for the execution of a procurement process. Notice that an event log must consist of at least three

Table P2.1: Example event log of a procurement process

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

columns: a 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. Optionally, the event log can contain so called event attributes. In the example event log from Tab. P2.1, the user column is such an event attribute, indicating which user has executed the respective activity.

3.2 *Process model generation*

To create a test setting for our approach we randomly generated process models and then sampled event logs from them. The process models were generated using PLG2 [6], a process simulation and randomization tool. Each process model has a different complexity, with regard to the number of possible activities and the branching factors (i.e., out-degrees). The complexity of a process model can also be measured by the number of possible variants. A variant is a valid path through the complete process model from a valid start activity to a valid end activity. Table P2.2 shows the process models with their corresponding complexities. Note that the *Wide* process model was specifically generated to evaluate the approach on a dataset that has low complexity in terms of the number of variants, but a high branching factor.

Now, we generated authentic event logs from these process models by randomly sampling variants of the process with replacement. In real process models these variants are not equally distributed. Therefore, we randomly generated a distribution for the variants each time we were sampling an event log. These probabilities were sampled from a normal distribution with $\mu = 1$ and $\sigma = 0.2$, and then normalized so they sum up to 1. Furthermore, we randomly generated a set of users in the process (between 10 and 30 different users per process). Then

Table P2.2: Overview over the 5 different randomly generated process models and the P2P process

Model	#Nodes	#Edges	#Variants	Max length	Out-degree
P2P	14	16	6	9	1.14
Small	22	26	6	10	1.18
Medium	34	48	25	8	1.41
Large	44	56	28	12	1.27
Huge	56	75	39	11	1.34
Wide	36	53	19	7	1.47

we sampled subsets of the user set for each activity, denoting which users are permitted to execute the activity. The number of possible users per activity lies between 1 and 5. After computing all variants, we also introduced a long-term dependency for the user variable in each variant at random. Therefore, we randomly chose two activities in each variant that must be executed by the same user.

3.3 *Example process*

In addition to the five randomly generated models, we also used a simplified version of a purchase to pay (P2P) process model as is depicted by the BPMN model in Fig. P2.1. This model was mainly created for purposes of evaluation, as it features interpretable activity names unlike the randomly generated models. 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. Notice the possible users for each activity as indicated by the italic names in Fig. P2.1.

3.4 *Anomalies*

To introduce noise into the event logs we randomly applied mutations to a fixed percentage of the traces in the event log. These mutations represent the anomalies in the data. Each trace can be affected by one of the following five anomalies (we will use their respective names from now on):

1. *Skipping*: A necessary activity has not been executed,
2. *Switching*: Two consecutive events have been executed in the wrong order,
3. *Reworking*: An activity has been executed twice in a row,

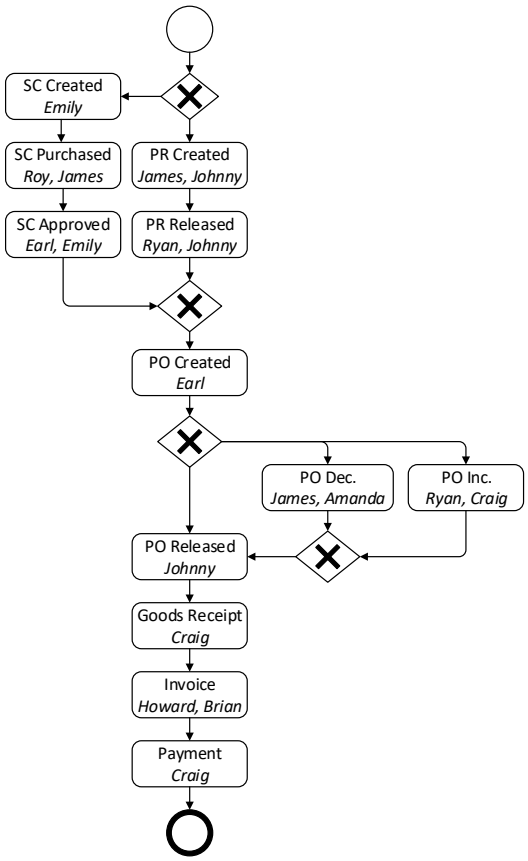

Figure P2.1: BPMN model of a simplified purchase to pay process; the italic names represent the users allowed to execute that activity

4. *Incorrect user*: A user has executed an activity to which he was not permitted,

5. *Incorrect LTD*: The wrong user has executed the long-term dependent activity.

Compared to our work in [30], we have added two more anomalies that we found occur very frequently in real-life scenarios. A classic problem in real-life business processes is the segregation of duty. For example, a user that approves a purchase order must not be the same user that has initially created it. Many anomalies in real-life processes are related to the users executing the events, which is why we included this event attribute here.

Our way of generating the artificial event logs is very similar to the methods of Bezerra [3] and Böhmer [5]. One difference is, that we also introduce anomalies affecting event attributes. We will make these datasets, the generation algorithm, and our implementation of the algorithm publicly available. For more information on this, please consider contacting the corresponding author.

For each process model, we randomly generated a set of permitted users for each activity. We did this ten times, resulting in 60 different

process models. For each of these 60 process models, we then generated 10 event logs, each featuring a different percentage of anomalies and a random variant distribution. The percentage of anomalous traces in the training log ranged from 10%, 20%, up to 100%. That is, we generated training logs containing 10% anomalies and 90% normal traces, as well as logs with 80% anomalies and 20% normal traces, and so on up to a log which entirely consists of anomalies, i.e., 100%. In total, we generated 600 different artificial event logs. Each event log consisted of 12 500 traces. For each event log we created a separate test event log containing 2 500 traces featuring the same variant distribution and users.

3.5 *Real-life event logs*

In addition to the artificial event logs we also generated training and test event logs from the public datasets of the Business Process Intelligence Challenge 2012[1] and 2017[2], which we will refer to as BPIC12 and BPIC17 respectively. BPIC17 is an updated version of BPIC12, representing the same loan application process. However, BPIC17 contains data from the last 5 years, after the company has introduced a new workflow system.

Similarly to the artificial logs, we used the event logs as a basis and randomly applied anomalies to a fixed percentage of traces in the logs. As these logs did not feature a user attribute we did not include the *Incorrect user* and *Incorrect LTD* anomalies. For BPIC12 and BPIC17 we generated training sets featuring between 10% and 100% anomalies, as was done for the artificial logs. We also generated separate test sets for both logs, resulting in 100 real-life training event logs with artificial anomalies.

4 METHOD

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 [25] or machine translation [2]. Before we introduce our method, we first want to give a brief overview over the neural network architecture we employed.

A feed-forward 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. 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

1 http://www.win.tue.nl/bpi/doku.php?id=2012:challenge
2 http://www.win.tue.nl/bpi/doku.php?id=2017:challenge

$f(x) = \max(0, x)$ [29]) to the sum over all outputs of the neurons in the previous layer times their respective connection weights. The initialization of these weights is important, as pointed out in [16], for no two weights within one layer must be initialized to the same value. Then, the back-propagation algorithm [36] is used to iteratively tune the weights, so that the neural network produces the desired output, or a close enough approximation of it.

In a classification setting, the desired output of the neural network is the class label. However, a neural network can also be trained without the use of class labels. One such type of neural network is called an autoencoder. Instead of using class labels, we are using the original input as the target output when training the autoencoder. 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 capacity limitation is needed. This can be done by forcing one of the autoencoder's hidden layers to be narrow (i.e., narrower than the input dimension), thereby not allowing the autoencoder to learn the identity function. Another common way of limiting the capacity 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. We use a combination of both these strategies for our method.

4.1 *Setup*

To train an autoencoder on the generated event logs, we first must transform them. The first step is to encode each activity and user using a one-hot encoding. Each activity is encoded by an n-dimensional vector, where n is the number of different activities encountered in the event log. To encode one activity, we simply set the corresponding dimension of the one-hot vector to a fixed value of one, while setting all the other dimensions to zero. We use the same method to encode the user event attribute. This results in a one-hot vector for the activity and another for the user for each event in a trace. Now we combine these vectors by concatenating them into one vector. If the activity vectors are $a_1, a_2, ..., a_n$ and the respective user vectors are $u_1, u_2, ..., u_n$, the resulting vector will be $a_1 \| u_1 \| a_2 \| u_2 \| ... \| a_n \| u_n$, where $\|$ denotes concatenation.

Note that another option of dealing with variable size traces is dividing the traces into subsequences of equal size (n-grams). However, using n-grams of events loses the connection between distant events, if the n-gram size is too narrow. Consequently, the system is unable accurately model long-term dependencies between events. Therefore, we chose to use the one-hot encoding method.

Because feed-forward neural networks have a fixed size input, we must apply one more step of pre-processing. To force all encoded trace

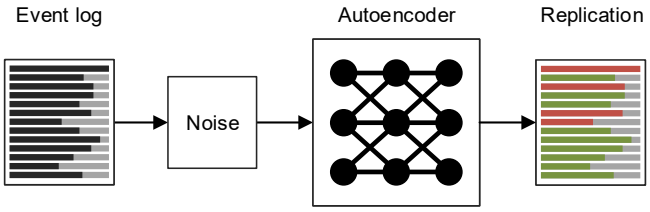

Figure P2.2: Autoencoder is trained to replicate the traces in the event log after the addition of Gaussian noise

vectors to have the same size we pad all vectors with zeros, so each vector has the same size as the longest vector (i.e., the longest trace) in the event log.

Suppose an event log consists of 10 different activities, 20 different users, and the maximum length of all traces in the event log is 12. The longest trace within the event log will have a size of $(10+20)\cdot 12 = 360$. Therefore, we must pad all shorter vectors with zeros so they reach size 360.

Using the one-hot encoded event log we can train the autoencoder with the back-propagation algorithm [36], using the event log both as the input and the label. Figure P2.2 shows a simplified version of the architecture. The special noise layer adds Gaussian noise before feeding the input into the autoencoder. This layer is only active 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 on mini batches of size 50 for 200 epochs, allowing early stopping when the loss on the validation set did not decrease within the last 10 epochs. We used the Adam optimizer [24], which utilizes the momentum technique [39]. We set the optimizer parameters to $\beta_1 = 0.9$, $\beta_2 = 0.99$ and $\epsilon = 10^{-8}$. The learning rate was set to 0.001 initially, and was scaled by a factor of 0.1 when the validation loss did not improve within the last 5 epochs. Additionally, we used a dropout of 0.5 between all layers, as suggested in [38]; the additive noise applied to the input was sampled from a Gaussian distribution with $\mu = 0$ and $\sigma = 0.1$. Each autoencoder consists of an input and an output layer with linear units, and 2 hidden layers 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, i.e., the number of neurons in the hidden layer was set to be half the size of the input layer. For the real-life BPIC event logs we only used 1 hidden layer.

4.2 *Classifying traces*

After training the autoencoder, it can be used to reproduce the traces in the test event logs, 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 τ, where if the reproduction error of a trace succeeds this threshold τ, we consider it an anomaly. To set the threshold we use the mean reproduction error over the training dataset and apply a scaling factor α. We define the threshold as in Equation P2.1, where e_i is the reproduction error for trace i, and n the number of traces in the dataset.

$$\tau = \frac{\alpha}{n} \sum_{i=1}^{n} e_i \tag{P2.1}$$

4.3 *Classifying events and attributes*

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 which event in the trace influences the reproduction error the most. Hence, we must change our calculation of the reproduction error from trace based to event 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 event in the sequence separately. Furthermore, we can also compute the error for each activity and user separately.

Let us consider the example input vector i from Equation P2.2. We can divide the vector into the corresponding subvectors, as indicated by the curly braces. This gives us $a_1, u_1, a_2, u_2, ..., a_n, u_n$. Now we can split the reproduced version of i (i.e., the output vector) identically, obtaining $\hat{a}_1, \hat{u}_1, \hat{a}_2, \hat{u}_2, ..., \hat{a}_3, \hat{u}_3$.

$$i = [\underbrace{00001}_{a_1}\ \underbrace{0100}_{u_1}\ \underbrace{10000}_{a_2}\ \underbrace{0010}_{u_2}\ ...\ \underbrace{01000}_{a_n}\ \underbrace{0100}_{u_n}] \tag{P2.2}$$

The error E for an activity vector a_i is then given by the mean squared error between a_i and $\hat{a}_i$. For a user vector u_i the method works analogously. Thus, we can compute the error for all activity vectors and all user vectors over the whole dataset. Notice that this works for any number of event attributes.

The benefit is that we can distinguish between activity related anomalies and user related anomalies. We will elaborate on this in the evaluation section below.

5 EVALUATION

We evaluated the autoencoder approach (DAE) on all 700 event logs and compared it to state-of-the-art anomaly detection methods mentioned [7]. Namely: a sliding window approach named t-STIDE [43]; the one-class SVM approach (OC-SVM); and the Markovian approach using a hidden Markov model (HMM) [43]. In addition to that, we also compared our approach to two approaches proposed in [3], the Naive algorithm and the Sampling algorithm. Lastly, we compared our approach to the most recent approach proposed in [5], using an extended likelihood graph (Likelihood). As a baseline we provide the results of a random classifier.

For the OC-SVM we relied on the implementation of the scikit-learn package for Python [31] using an RBF kernel of degree 3 and a $\nu = 0.6$. The HMM approach was implemented using the hmmlearn package for Python. We implemented the t-STIDE algorithm ourselves using a window size $k = 4$. The hyperparameters for both approaches were optimized using grid search. The Naive, Sampling, and Likelihood methods were implemented as described in the original papers.

At last, we used our own implementation of the t-STIDE method which we will refer to as t-STIDE+. The classic t-STIDE approach only takes into account the activities of an event log, but not the attributes. To make use of the attributes we must adapt the original method.

A window of size k is a tuple of k events, where each event consists of a tuple of the activity name a and the corresponding user u. Let us consider an example window size of three. A window w is defined as $w = \{(a_1, u_1), (a_2, u_2), (a_3, u_3)\}$. The approach works by employing a frequency analysis over all windows of size k in the training set, and then comparing the relative frequencies of all windows in the test set to the corresponding ones from the training set. Whenever a window's relative frequency is significantly lower than its frequency in the training set, the trace containing this window is considered an anomaly. We evaluated the t-STIDE and the t-STIDE+ approach on all datasets and for all feasible choices of k, i.e., k was chosen to lie between 2 and the maximum trace length in the dataset. The evaluation showed that $k = 4$ performed the best for both approaches.

We used the threshold technique from Equation P2.1 for all approaches except the OC-SVM, for the scikit-learn implementation automatically optimizes the threshold. For the other approaches, we optimized the scaling factor α by an exhaustive grid search. One requirement when setting α was that α must be the same for all event logs, i.e., we strive for a general setting of α.

We also considered isolation forests [27, 28] for our experiments; however, this approach relies on setting a contamination parameter indicating the noise level of the data, rendering the approach unusable as we assume no prior knowledge about the noise level.

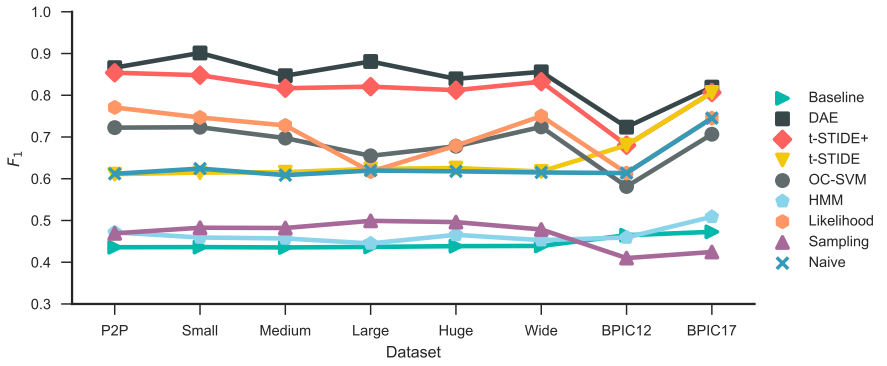

Figure P2.3: F_1 score by process model and method

Table P2.3: Results of the experiments for all evaluated methods for each process model; best results are shown in bold typeface

	P2P	Small	Medium	Large	Huge	Wide
Baseline	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01	0.44 ± 0.01
HMM [43]	0.47 ± 0.02	0.46 ± 0.01	0.46 ± 0.01	0.45 ± 0.02	0.47 ± 0.01	0.45 ± 0.02
OC-SVM [37]	0.72 ± 0.06	0.72 ± 0.05	0.70 ± 0.05	0.65 ± 0.04	0.68 ± 0.05	0.72 ± 0.05
Naive [3]	0.61 ± 0.01	0.62 ± 0.01	0.61 ± 0.03	0.62 ± 0.01	0.62 ± 0.02	0.62 ± 0.02
Sampling [3]	0.47 ± 0.03	0.48 ± 0.05	0.48 ± 0.06	0.50 ± 0.06	0.50 ± 0.06	0.48 ± 0.05
t-STIDE [43]	0.61 ± 0.01	0.61 ± 0.03	0.62 ± 0.02	0.62 ± 0.01	0.63 ± 0.01	0.62 ± 0.02
Likelihood [5]	0.77 ± 0.17	0.75 ± 0.14	0.73 ± 0.15	0.62 ± 0.10	0.68 ± 0.11	0.75 ± 0.17
t-STIDE+	0.85 ± 0.09	0.85 ± 0.08	0.82 ± 0.09	0.82 ± 0.07	0.81 ± 0.05	0.83 ± 0.11
DAE	**0.87 ± 0.09**	**0.90 ± 0.07**	**0.85 ± 0.08**	**0.88 ± 0.07**	**0.84 ± 0.08**	**0.86 ± 0.08**

We evaluated the 9 methods on all 600 artificial, as well as the 100 real-life event logs. In total, we evaluated 6 300 models.

5.1 *Experiment results*

Figure P2.3 shows the F_1 score of all methods for each process model. The F_1 score per model was calculated using the macro average for each model. Then all F_1 scores were averaged over all models for the corresponding process model. A more detailed evaluation is given in Tab. P2.3 and Table P2.4, which show the F_1 scores and their standard deviation for each process model, best results being shown in bold typeset. Notice that the DAE approach performs best in all settings, closely followed by t-STIDE+, whereas the other approaches perform significantly worse. Another interesting point is that the HMM approach performs no better than the random baseline, which supports Chandola's claim that HMMs are not a good method for anomaly detection in sequential data [8]. Also the Sampling approach performs only slightly better than chance. However, this is due to the fact that we average all results over all training sets including training event logs with higher share of anomalies. In Fig. P2.4 we can see that the Sampling method works only for low noise levels.

Table P2.4: Results on the BPIC event logs; best results are shown in bold typeface

	BPIC12	BPIC17
Baseline	0.46 ± 0.01	0.47 ± 0.01
HMM [43]	0.46 ± 0.00	0.51 ± 0.00
OC-SVM [37]	0.58 ± 0.07	0.71 ± 0.04
Naive [3]	0.61 ± 0.12	0.75 ± 0.12
Sampling [3]	0.41 ± 0.00	0.42 ± 0.00
t-STIDE [43]	0.68 ± 0.14	0.81 ± 0.02
Likelihood [5]	0.61 ± 0.12	0.75 ± 0.12
t-STIDE+	0.68 ± 0.14	0.81 ± 0.02
DAE	**0.72** ± **0.08**	**0.82** ± **0.05**

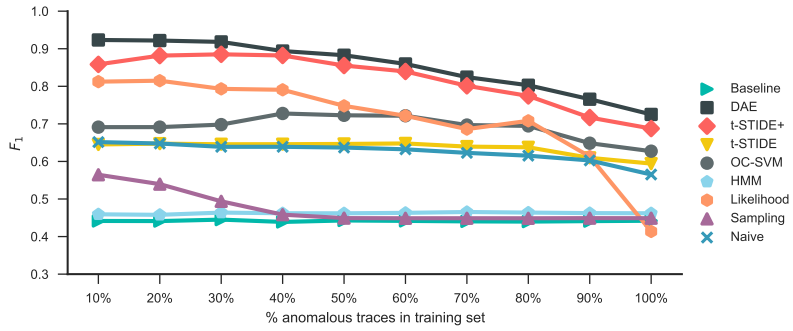

Figure P2.4: F_1 score by percentage of anomalous traces in the training set

Overall, we can conclude that the DAE performs better than the state-of-the-art methods in all of our test settings.

5.2 *The impact of the noise level*

As described before, we used different noise levels when generating the datasets, by generating training sets which included between 10% and 100% anomalous traces. We use the word noise to refer to the share of traces in the training set which are anomalous. Notice that an anomalous trace still contains normal subsequences of events. Only a small part of the trace is affected by the anomaly in our test settings. Hence, there is still normal behavior present in parts of each trace, even when each trace has been affected by an anomaly, as in the 100% case. We specifically included these harsh noise levels to test the different approaches on their ability to generalize. We want to point out, however, that noise levels greater than 50% are extremely unlikely in real-world settings.

One can also argue that a noise level greater than 50% is illogical, because the classification task just gets inverted; hence, the anomaly class becomes the normal class. This is not true for the same reason as before. As we are dealing with sequential data and many different events in sequence (i.e., a trace) are assigned one label, there are still events that carry information about the normal behavior of the process. And in most cases the normal events in an anomalous trace, still overpower the anomalous ones. Hence, a noise level of 60% is not the same as a noise level of 40% with classes inverted.

Figure P2.4 shows the F_1 score for all methods for the different noise levels. Again, we find that the DAE outperforms the other approach at all noise levels, again closely followed by t-STIDE+.

Notice that the DAE still performs remarkably well, even when trained on the 100% training set. This is due to its ability to generalize over multiple traces. The t-STIDE approaches can also generalize over multiple traces, because they classify based on windows; and the windows itself can contain a completely valid sequence of events. These approaches can learn what a normal trace ought to look like, by combining the knowledge they gathered of normal subsequences over multiple traces. For the t-STIDE approaches this is obvious, as the window size is usually smaller than the trace is long; hence, it is only trained on subsequences in the first place. The DAE, on the other hand, is trained on the whole trace at once, which makes this level of generalization much more remarkable and unique among all the approaches.

5.3 *Interpreting the anomalies*

An interesting feature of the DAE approach is that it can be used to detect not only anomalous traces, but also which event or which event attribute has influenced the reproduction error the most. This can be done by computing the reproduction error for each event attribute separately, as described earlier. Figure P2.5 shows 12 example traces of the P2P test dataset for a DAE trained on a training set with 10% anomalous traces. For clarity, we only show the first 6 events omitting the remaining events. The cells are colored according to the reproduction error; the higher the error the darker the color.

As we can clearly see, it is never the whole trace that leads to a high reproduction error. The DAE succeeds to reproduce the normal parts of the traces quite well, whereas it fails to reproduce the anomalous parts. For example, the first two *Normal* traces are reproduced with almost no error at all, which is exactly what we expected. Let us now look at the four examples at the bottom (*Incorrect user* and *Incorrect LTD*). The DAE is remarkably good at detecting incorrect users. Neither Craig, nor Earl, are permitted to execute the activity *PR Created* (cf. Fig. P2.1). Detecting *Incorrect LTD* works just as fine.

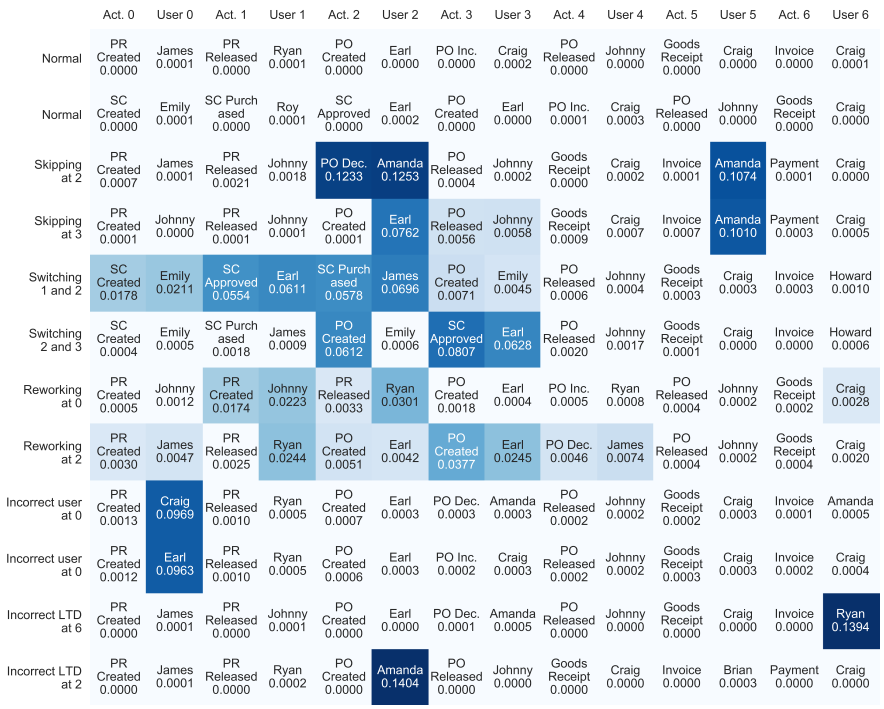

Figure P2.5: DAE error heatmap, trained on a P2P event log with 10% anomalous traces

Moving to the three anomalies from the original paper, we want to recall one problem that we have observed during the evaluation in [30]. Whenever an activity is skipped or reworked, the remaining subsequence is shifted by one to the left, or the right respectively. In [30] this led to the effect that all activities after the initial skipped (or reworked) activity had high reproduction error. This phenomenon does not occur as severely in the extended approach, but it is still noticeable. We assume that the additional hidden layers provide enough abstraction so the DAE can adapt to this problem.

Overall, we can see that the approach is very precise in narrowing down the exact cause of the anomaly. In fact, this approach can be used to perform an automatic root cause analysis on the detected anomalies, without the need of an extra processing step. Most other anomaly detection algorithms can only be used to divide the normal examples from the anomalies, but then an additional algorithm has to be used to, for instance, cluster the anomalies. Another important point about this is that it allows to follow what the DAE has learned as well as to interpret it. Usually, not being interpretable is a notorious problem for neural network based approaches. Not in this case.

5.4 *Discussion*

At last we want to point out some interesting observations. You might notice that in the two *Skipping* examples the user at index 5, Amanda, produces a high reproduction error. This is due to the fact that this event has a long-term dependency to an earlier event. In the first case the event is connected to the *PO Decreased* event, in the second one the connected event is the event that has been skipped. Now that the trace has, in part, been shifted due to the skipping, the original event from index 6 is now at index 5. Essentially, we have detected a fluke anomaly, that was not supposed to be there, yet the DAE approach has found it, demonstrating the feasibility of the approach.

This indicates, as already mentioned in [30], that the DAE is sensitive towards the actual position of an event within a trace, which also becomes apparent in the second *Skipping* example. The event *PO Created* is wrong, yet the DAE reproduces it correctly. This is due to the fact the *PO Created* can be correct here when the trace starts with the *SC Created* event.

Furthermore, we still observe some cross-talk between adjacent events. If we inspect the first *Switching* example, we notice that *SC Approved* and *SC Purchased* have been switched, as correctly identified by the DAE. However, the first event also produces a high reproduction error, albeit being correct at that location. This error, compared to the error at indices 2 and 3, is significantly lower.

Table P2.5 provides the average F_1 score of the approaches when classifying traces, events, and attributes respectively. When classifying attributes we classify the activity and the user separately, whereas when classifying events we do not separate the attributes. Therefore, we also produced labels indicating anomalous and normal event attributes, when generating the datasets. Any event attribute that had not been affected by any of the anomalies, has been labeled normal, whereas all other attributes have been labeled as anomalous. An event is an anomalous when any of its attributes is anomalous and similarly a trace is anomalous when any of its events is anomalous. Consequently, we can now calculate the performance of the DAE based on single events.

Sampling, t-STIDE, and t-STIDE+ can all also naturally be used to classify events. Apart from DAE, only t-STIDE+, due to our adaption of the algorithm, can also be used to classify single attributes. This can be done, for instance, by assigning the specific window anomaly scores to the respective last event or attribute in the window. Sampling relies on a conformance check, which per definition gives a per event resolution. Table P2.5 shows that the DAE outperforms the other approaches in all three categories.

We can conclude that this approach can discover the special characteristics of anomalies in an otherwise unknown process, while still

Table P2.5: Results of the experiments for the anomalous event classifier per label and process model; best results are shown in bold typeface

Resolution	Method	Average	Normal	Anomaly
Traces	Baseline	0.44 ± 0.01	0.25 ± 0.01	0.62 ± 0.01
	HMM [43]	0.46 ± 0.02	0.17 ± 0.06	0.75 ± 0.05
	OC-SVM [37]	0.70 ± 0.06	0.51 ± 0.09	0.89 ± 0.05
	Naive [3]	0.62 ± 0.02	0.49 ± 0.02	0.74 ± 0.02
	Sampling [3]	0.48 ± 0.05	0.11 ± 0.18	0.86 ± 0.09
	t-STIDE [43]	0.62 ± 0.02	0.49 ± 0.02	0.74 ± 0.02
	Likelihood [5]	0.72 ± 0.15	0.52 ± 0.26	0.91 ± 0.07
	t-STIDE+	0.83 ± 0.08	0.73 ± 0.12	0.93 ± 0.05
	DAE	**0.87 ± 0.08**	**0.78 ± 0.13**	**0.95 ± 0.03**
Events	Sampling [3]	0.44 ± 0.17	0.64 ± 0.20	0.25 ± 0.14
	t-STIDE [43]	0.66 ± 0.03	0.90 ± 0.02	0.42 ± 0.04
	t-STIDE+	0.61 ± 0.03	0.86 ± 0.03	0.36 ± 0.05
	DAE	**0.72 ± 0.02**	**0.92 ± 0.02**	**0.53 ± 0.04**
Attributes	t-STIDE+	0.59 ± 0.03	0.87 ± 0.03	0.31 ± 0.06
	DAE	**0.70 ± 0.03**	**0.94 ± 0.01**	**0.47 ± 0.05**

being able to correctly identify normal behavior. All together, we can say that the DAE approach is the most versatile out of all the approaches, as it works well in all of our test settings.

6 CONCLUSION

We have presented a novel application of denoising autoencoders to detect anomalies in business process data. Our approach does not rely on any prior knowledge about the process itself. Also, we do not rely on a clean dataset for the training; our approach is trained on a noisy dataset already containing the anomalies. Furthermore, we have demonstrated that the autoencoder can also be used to easily identify the anomalous event(s) or event attribute(s), making results interpretable with regards to why an anomaly has been classified as such. Even though we showed that this approach works for business process data, it can be applied just as easily to other domains with discrete sequential data.

We conducted a comprehensive evaluation using representative artificial and real-life event logs. These event logs featured a range of different anomalies, different complexities in terms of the process model, variable variant probabilities, random user sets for each activity, and different shares of anomalous traces, ranging from 10% to 100%. We compared the autoencoder approach to 7 other state-of-the-art

anomaly detection methods, as described in [3, 5, 7, 43], showing that our approach outperforms all other methods in all of the test settings, reaching an F_1 score of 0.87 on average, whereas the second-best approach, our own adaption of the t-STIDE algorithm reached 0.83. The next best unaltered anomaly detection algorithm, using an extended likelihood graph, reached an F_1 score of 0.72. To our knowledge, this is the most sophisticated evaluation and comparison of anomaly detection methodology within the domain of process intelligence to date.

The biggest advantage of the autoencoder approach over the other methods is that it allows to analyze the detected anomalies even further. Computing the anomaly score for each event attribute individually, the approach indicates the anomalous attribute very convincingly. To our knowledge, this method of analyzing the anomalies is novel to the field of discovery science, as well as business intelligence and process mining.

The presented approach is an extended version of the approach from [30]. In the original paper, we postulated that the approach is susceptible to anomalous behavior in the event log that is very frequent. However, by showing that the approach works well for all noise levels, especially the higher noise levels where the exact same anomaly can occur many times, we have shown this not to be the case. We also showed, by using skewed variant distributions, that the autoencoder is robust towards process models with unequally distributed variants, that is, some variants (i.e., one valid path through the process model) are more likely than others. By including the user as an event attribute, we demonstrated that more dimensions can be added easily to the approach, without a significant loss of accuracy.

As an inspiration for future work on the matter we want to give a few remarks. Note that for the DAE approach to work in a real-time setting the trace length of all future traces must be conform with the input size of the neural network. If traces surpass the input size, they cannot be fed into the autoencoder. There are some strategies to compensate for this problem. For example, the autoencoder can be set up with spare padding input units. Instead of padding all traces to match the maximum length encountered in the training set, we pad all traces to an arbitrary length greater than the maximum length. If we do want to reuse the already trained autoencoder, we can use another strategy. Every trace that is too long to feed into the autoencoder is divided into subsequences of exactly the size of the input. For example, if an autoencoder has input size 10 and a trace has size 12, we would first feed the sequence starting from the first event until the tenth event, then the sequence from the second until the eleventh, and so on. Then we can average the anomaly scores over all subsequences. Another solution to the problem is the use of

recurrent neural networks, which can be used to consume sequences of arbitrary length.

Another problem arises if one of the attributes is set to a value not encountered during training. Consequently, there will be no dimension allocated in the one-hot encoding for it. A simple solution to this problem is to add one extra dimension to the encoding vector which is used to encode all unknown attribute characteristics. Nevertheless, the autoencoder should be retrained regularly to counteract concept drift.

With t-STIDE+ and DAE we have presented two approaches to detect anomalies in business process data. It is quite costly to train a neural network on big datasets, because the dataset needs to be iterated many times. The t-STIDE+ approach has the advantage that it can be trained with just one iteration.

However, due to the nature of the algorithm, it has some drawbacks; it cannot capture long-term dependencies if the window size is too small and if the windows size is too big the accuracy decreases. Furthermore, it is not trivial to assign an anomaly score to a single attribute of an event, because anomaly scores are based on windows. Lastly, it cannot deal with numerical event attributes (e.g., prices), without resorting to binning or grouping, which is not obvious.

The DAE approach does not have these drawbacks. Numerical data can easily be modelled by using a single linear input and output neuron for real-valued numbers. Certainly, it does require more training time, but with the introduction of evermore powerful GPUs and lately TPUs the trade-off between accuracy and efficiency is not as severe.

Overall, the results presented in this paper suggest that a denoising autoencoder is a reliable and versatile method for detecting—and interpreting—anomalies in unknown business processes.

ACKNOWLEDGEMENTS

This project (HA project no. 522/17-04) is funded in the framework of Hessen ModellProjekte, financed with funds of LOEWE – Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbund-vorhaben (State Offensive for the Development of Scientific and Economic Excellence).

REFERENCES

[1] Wil Van der Aalst, Ton Weijters, and Laura Maruster. "Workflow mining: Discovering process models from event logs." In: *Knowledge and Data Engineering, IEEE Transactions on* 16.9 (2004), pp. 1128–1142 (cit. on p. 64).

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." In: *arXiv preprint arXiv:1409.0473* (2014) (cit. on p. 71).

[3] Fábio Bezerra and Jacques Wainer. "Algorithms for anomaly detection of traces in logs of process aware information systems." In: *Information Systems* 38.1 (2013), pp. 33–44 (cit. on pp. 64, 65, 70, 75–77, 81, 82).

[4] Fábio Bezerra, Jacques Wainer, and Wil MP van der Aalst. "Anomaly detection using process mining." In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, pp. 149–161 (cit. on pp. 64, 65).

[5] Kristof Böhmer and Stefanie Rinderle-Ma. "Multi-perspective Anomaly Detection in Business Process Execution Events." In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2016, pp. 80–98 (cit. on pp. 65–67, 70, 75–77, 81, 82).

[6] Andrea Burattin. "PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings." In: *CoRR* abs/1506.0 (2015) (cit. on p. 68).

[7] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly Detection for Discrete Sequences: A Survey." In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 823–839 (cit. on pp. 66, 75, 82).

[8] V. Chandola, V. Mithal, and V. Kumar. "Comparative Evaluation of Anomaly Detection Techniques for Sequence Data." In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 743–748 (cit. on pp. 66, 76).

[9] Corinna Cortes and Vladimir Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on p. 66).

[10] Andrew M Dai and Quoc V Le. "Semi-supervised sequence learning." In: *Advances in Neural Information Processing Systems*. 2015, pp. 3079–3087 (cit. on p. 67).

[11] Ignacio Diaz and Jaakko Hollmén. "Residual generation and visualization for understanding novel process conditions." In: *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*. Vol. 3. IEEE. 2002, pp. 2070–2075 (cit. on p. 67).

[12] Yue Dong and Nathalie Japkowicz. "Threaded ensembles of supervised and unsupervised neural networks for stream learning." In: *Canadian Conference on Artificial Intelligence*. Springer. 2016, pp. 304–315 (cit. on p. 67).

[13] Marlon Dumas, Wil M Van der Aalst, and Arthur H Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. John Wiley & Sons, 2005 (cit. on p. 64).

[14] Eleazar Eskin. "Anomaly detection over noisy data using learned probability distributions." In: *In Proceedings of the International Conference on Machine Learning*. Citeseer. 2000 (cit. on p. 66).

[15] Stephanie Forrest, Steven A Hofmeyr, Anil Somayaji, and Thomas A Longstaff. "A sense of self for unix processes." In: *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*. IEEE. 1996, pp. 120–128 (cit. on p. 66).

[16] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Aistats*. Vol. 9. 2010, pp. 249–256 (cit. on p. 72).

[17] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. "Outlier detection using replicator neural networks." In: *Data warehousing and knowledge discovery*. Springer, 2002, pp. 170–180 (cit. on pp. 66, 67).

[18] Robert Hecht-Nielsen. "Replicator neural networks for universal optimal source coding." In: *Science* 269.5232 (1995), p. 1861 (cit. on p. 67).

[19] Katherine A. Heller, Krysta M. Svore, Angelos D. Keromytis, and Salvatore J. Stolfo. "One Class Support Vector Machines for Detecting Anomalous Windows Registry Accesses." In: *In Proc. of the workshop on Data Mining for Computer Security*. 2003 (cit. on p. 66).

[20] Geoffrey E Hinton. "Connectionist learning procedures." In: *Artificial intelligence* 40.1 (1989), pp. 185–234 (cit. on p. 67).

[21] R. Jain and N. S. Abouzakhar. "Hidden Markov Model based anomaly intrusion detection." In: *2012 International Conference for Internet Technology and Secured Transactions*. 2012, pp. 528–533 (cit. on p. 66).

[22] Nathalie Japkowicz. "Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks." In: *Machine Learning* 42.1 (2001), pp. 97–122 (cit. on pp. 66, 67).

[23] Biing Hwang Juang and Laurence R Rabiner. "Hidden Markov models for speech recognition." In: *Technometrics* 33.3 (1991), pp. 251–272 (cit. on p. 66).

[24] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 73).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 71).

[26] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning." In: *Nature* 521.7553 (2015), pp. 436–444 (cit. on p. 67).

[27] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation forest." In: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE. 2008, pp. 413–422 (cit. on p. 75).

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation-based anomaly detection." In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6.1 (2012), p. 3 (cit. on p. 75).

[29] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 807–814 (cit. on p. 72).

[30] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders." In: *International Conference on Discovery Science*. Springer. 2016, pp. 442–456 (cit. on pp. 63, 70, 79, 80, 82).

[31] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python." In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 75).

[32] Marco AF Pimentel, David A Clifton, Lei Clifton, and Lionel Tarassenko. "A review of novelty detection." In: *Signal Processing* 99 (2014), pp. 215–249 (cit. on p. 66).

[33] Lawrence R Rabiner. "A tutorial on hidden Markov models and selected applications in speech recognition." In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286 (cit. on p. 66).

[34] Lawrence Rabiner and B Juang. "An introduction to hidden Markov models." In: *IEEE ASSP magazine* 3.1 (1986), pp. 4–16 (cit. on p. 66).

[35] Anne Rozinat and Wil MP van der Aalst. "Conformance checking of processes based on monitoring real behavior." In: *Information Systems* 33.1 (2008), pp. 64–95 (cit. on p. 64).

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors." In: *Cognitive modeling* 5.3 (1988), p. 1 (cit. on pp. 72, 73).

[37] Bernhard Schölkopf, Robert C Williamson, Alexander J Smola, John Shawe-Taylor, John C Platt, et al. "Support vector method for novelty detection." In: *NIPS*. Vol. 12. 1999, pp. 582–588 (cit. on pp. 66, 76, 77, 81).

[38] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on p. 73).

[39] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. "On the importance of initialization and momentum in deep learning." In: *Proceedings of the 30th international conference on machine learning (ICML-13)*. 2013, pp. 1139–1147 (cit. on p. 73).

[40] David Martinus Johannes Tax. "One-class classification: Concept learning in the absence of counter-examples." PhD thesis. Technische Universiteit Delft, 2001 (cit. on p. 66).

[41] Benjamin Berry Thompson et al. "Implicit learning in autoencoder novelty assessment." In: *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*. Vol. 3. IEEE. 2002, pp. 2878–2883 (cit. on p. 67).

[42] Wil Van Der Aalst et al. "Process mining manifesto." In: *Business process management workshops*. Springer. 2011, pp. 169–194 (cit. on pp. 64, 65).

[43] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. "Detecting intrusions using system calls: Alternative data models." In: *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE. 1999, pp. 133–145 (cit. on pp. 65, 66, 75–77, 81, 82).

[44] Graham Williams, Rohan Baxter, Hongxing He, Simon Hawkins, and Lifang Gu. "A comparative study of RNN for outlier detection in data mining." In: *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE. 2002, pp. 709–712 (cit. on p. 67).

[45] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. "A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification." In: *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*. AISec '13. ACM, 2013, pp. 67–76 (cit. on p. 66).

P3

BINET: MULTIVARIATE BUSINESS PROCESS ANOMALY DETECTION USING DEEP LEARNING

Timo Nolle, Alexander Seeliger, and Max Mühlhäuser
In: *Proceedings of the 16th International Conference on Business Process Management – BPM'18*. 2018, pp. 271–287.

ABSTRACT: In this paper, we propose BINet, a neural network architecture for real-time multivariate anomaly detection in business process event logs. BINet has been designed to handle both the control flow and the data perspective of a business process. Additionally, we propose a heuristic for setting the threshold of an anomaly detection algorithm automatically. We demonstrate that BINet can be used to detect anomalies in event logs not only on a case level, but also on event attribute level. We compare BINet to 6 other state-of-the-art anomaly detection algorithms and evaluate their performance on an elaborate data corpus of 60 synthetic and 21 real life event logs using artificial anomalies. BINet reached an average F_1 score over all detection levels of 0.83, whereas the next best approach, a denoising autoencoder, reached only 0.74. This F_1 score is calculated over two different levels of detection, namely case and attribute level. BINet reached 0.84 on case and 0.82 on attribute level, whereas the next best approach reached 0.78 and 0.71 respectively.

KEYWORDS: Business Process Management, Anomaly Detection, Artificial Process Intelligence, Deep Learning, Recurrent Neural Networks

1 INTRODUCTION

Anomaly detection is an important topic for today's businesses because its application areas are so manifold. Fraud detection, intrusion detection, and outlier detection are only a few examples. However, anomaly detection can also be applied to business process executions, for example to clean up datasets for more robust predictive analytics and robotic process automation (RPA). Especially in RPA, anomaly detection is an integral part because the robotic agents must recognize tasks they are unable to execute to not halt the process. Naturally, businesses are interested in anomalies within their processes, as these can be indicators for inefficiencies, insufficiently trained employees, or even fraudulent activities. Consequently, being able to detect such

anomalies is of great value, for they can have an enormous impact on the economic well-being of the business.

In today's digital world, companies rely more and more on process-aware information systems (PAISs) to accelerate their processes. A byproduct of such PAISs is an enormous data base that often remains unused. The log files these systems are storing can be used to extract valuable information about a process. One key data structure is an event log, which contains information about what activities have been executed in a process, who executed it, at which time, etc. These event logs are a great source of information and are frequently used for different data mining techniques, such as process mining.

In this paper, we propose BINet (Business Intelligence Network), a novel neural network architecture that allows to detect anomalies on attribute level. Often, the actual cause of an anomaly is only captured by the value of a single attribute. For example, a user has executed an activity without permission. This anomaly is only represented by the user attribute of exactly this event. Anomaly detection algorithms must work on the lowest (attribute) level, to provide the greatest benefit. BINet has been designed to process both the control flow (sequence of activities) and the data flow (see [1]).

Due to the nature of the architecture of BINet it can be used for ex-post analysis, but can also be deployed in a real-time setting to detect anomalies at runtime. Being able to detect anomalies at runtime is important because otherwise no counter measures can be undertaken in time. BINet can be trained during the execution of the process and therefore can adapt to concept drift. If unseen attribute values occur during the training, the network can be altered and retrained on the historic data to include the new attribute value in the future. Dealing with concept drift is also important as most business processes are flexible systems. BINet is a recurrent neural network architecture and therefore can detect point anomalies as well as contextual anomalies (see [12]). BINet works under the following assumptions.

- No domain knowledge about the process
- No clean dataset (i.e., dataset contains anomalous examples)
- No reference model
- No labels (i.e., no knowledge about anomalies)

In the context of business processes an anomaly is defined as a deviation from a defined behavior, i.e., the business process. An anomaly is an event that does not typically occur as a consequence of preceding events, specifically their order and combination of attributes. Anomalies that are attributed to the order of activities (e.g., two activities are executed in the wrong order) are called control flow anomalies. Anomalies that are attributed to the attributes (e.g., a user that is

not part of a certain security group has illicitly executed an event) of events are called data flow anomalies.

Many anomaly detection algorithms rely on the manual setting of a threshold value to determine anomalies. We propose an unsupervised method for automatically setting the threshold using a heuristic.

We compare BINet to 6 state-of-the art anomaly detection methods and evaluate on a comprehensive dataset of 60 synthetic logs and 20 real-life logs, using artificial anomalies. This work contains four main contributions.

1. BINet neural network architecture[1]
2. Automatic threshold heuristic
3. Comprehensive evaluation of state-of-the-art methods

2 RELATED WORK

In the field of process mining [1], it is popular to use discovery algorithms to mine a process model from an event log and then use conformance checking to detect anomalous behavior [2, 4, 27]. However, the proposed methods do not utilize the event attributes, and therefore cannot be used to detect anomalies on attribute level.

A more recent publication proposes the use of likelihood graphs to analyze business process behavior [5]. Specifically, the authors describe a method to extend the likelihood graph to include event attributes. This method works on noisy event logs and includes important characteristics of the process itself by including the event attributes. A drawback of this method is that the attributes are checked in a specific order, thereby introducing a bias towards certain attributes.

A review of classic anomaly detection methodology can be found in [22]. Here, the authors describe and compare many methods that have been proposed over the last decades. Another elaborate summary on anomaly detection in discrete sequences is given by Chandola in [7]. 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 estimate the probability distribution of the normal class, and thus can detect anomalies as they come from a different distribution. An important probabilistic technique is the sliding window approach [26]. In window-based anomaly detection, an anomaly score is assigned to each window in a sequence. Then the anomaly score of the sequence can be inferred by aggregating the window anomaly scores. Recently, Wressnegger et al. used this approach for intrusion detection and gave an elaborate evaluation in [28]. While being inexpensive and easy to implement, sliding window approaches

1 https://github.com/tnolle/binet

show a robust performance in finding anomalies in sequential data, especially within short regions [7].

Distance-based novelty detection does not require a clean dataset, yet it is only partly applicable for process cases, as anomalous cases are usually very similar to normal ones. A popular distance-based approach is the one-class support vector machine (OC-SVM). Schölkopf et al. [23] first used support vector machines [9] for anomaly detection.

Reconstruction-based novelty detection (e.g., neural networks) is based on the idea to train a model that can reconstruct normal behavior but will fail to do so with anomalous behavior. Therefore, the reconstruction error can be used to detect anomalies [15]. This approach has successfully been used for the detection of control flow anomalies [21] as well as data flow anomalies [20] in event logs of PAISs.

Domain-based novelty detection requires domain knowledge, which violates our assumption of no domain knowledge about the process. Information-theoretic novelty detection defines anomalies as the examples that influence an information measure (e.g., entropy) on the whole dataset the most. Iteratively removing the data with the highest impact will yield a cleaned dataset, and thus a set of anomalies.

The core of BINet is a recurrent neural network, trained to predict the next event and its attributes. The architecture is influenced by the works of Evermann [10, 11] and Tax [24], who utilized long short-term memory [13] (LSTM) networks for next event prediction, demonstrating their utility. LSTMs have been used for anomaly detection in different contexts like acoustic novelty detection [18] and predictive maintenance [17]. These applications mainly focus on the detection of anomalies in time series and not, like BINet, on multivariate anomaly detection in discrete sequences of events.

The novelty of BINet lies in the tailored architecture for business processes, including the control and data flow, the scoring system to assign anomaly scores, and the automatic threshold heuristic.

3 DATASETS

As a basis for the understanding of the following sections, we first need to define the terms case, event, log, and attribute. A log consists of cases, each of which consists of events executed within a process. Each event is defined by an activity name and its attributes, e.g., a user who executed the event. We use a nomenclature adapted from [1].

Definition P3.1. *Case, Event, Log, Attribute. Let $\mathcal{C}$ be the set of all cases and $\mathcal{E}$ be the set of all events. The event sequence of a case $c \in \mathcal{C}$, denoted by $\hat{c}$, is defined as $\hat{c} \in \mathcal{E}^*$, where $\mathcal{E}^*$ is the set of all sequences over $\mathcal{E}$. An event log is a set of cases $\mathcal{L} \subseteq \mathcal{C}$. Let $\mathcal{A}$ be a set of attributes and $\mathcal{V}$ be a set of attribute values, where $\mathcal{V}_a$ is the set of possible values for the attribute*

$a \in \mathcal{A}$. *Note that* $|\hat{c}|$ *is the number of events in case* c, $|\mathcal{L}|$ *is the number of cases in* $\mathcal{L}$, *and* $|\mathcal{A}|$ *is the number of event attributes.*

To evaluate our method, we generated synthetic event logs from random process models of different complexities. We used PLG2 [6] to generate five process models: Small, Medium, Large, Huge, and Wide. The complexity of the models varies in number of activities, breadth, and width; for Small to Huge, activities, breadth, and width increase uniformly, whereas Wide features a much larger breadth than width. Wide was designed as a challenge because it features a high branching factor, thereby making it hard to predict the next activity or attribute. We also use a handmade procurement process model called P2P for demonstrative purposes because it features human readable activity names.

Now, we randomly generate logs from these process models, following the control flow and generating attributes for each event. Each possible sequence of activities was assigned a random probability sampled from a normal distribution with $\mu = 1$ and $\sigma = 0.2$, so that not all sequences appear equally likely.

To generate the attributes, we first create a set of possible values $\mathcal{V}_a$ for each attribute a, with set sizes ranging from 20 to 100. Then we assign random subsets of $\mathcal{V}_a$ to each activity, ranging from 5 to 40 in size. When generating a sequence from the process model, we also sample one possible value for each attribute in each event. While sampling we enforce long term dependencies between attributes. For example, 2 of the 10 attribute values of one activity always occur when 1 of the attribute values for a different event occurred earlier in the sequence; hence, we model causal relationships between attributes within sequences.

In addition to the synthetic logs we also use the event logs from the Business Process Intelligence Challenge (BPIC): BPIC12[2], BPIC13[3], BPIC15[4] and BPIC17[5]. Furthermore, we evaluate our method on 10 real-life event logs of procurement processes, made available to us by a consulting company. Refer to Tab. P3.1 for information about the datasets.

Like Bezerra [3] and Böhmer [5], we apply artificial anomalies to the event logs, altering 30 percent of all cases. In addition to the three anomaly types used in [3, 5], we introduced a new, attribute-based anomaly. The anomalies are defined as follows: *Skip*, a necessary activity has not been executed; *Switch*, two events have been executed in the wrong order; *Rework*, an activity has been executed too many times; *Attribute*, an incorrect attribute value is set (e.g., a user does not have the necessary security level).

2 http://www.win.tue.nl/bpi/doku.php?id=2012:challenge
3 http://www.win.tue.nl/bpi/doku.php?id=2013:challenge
4 http://www.win.tue.nl/bpi/doku.php?id=2015:challenge
5 http://www.win.tue.nl/bpi/doku.php?id=2017:challenge

Table P3.1: Overview showing dataset information

Name.	#Logs	#Activities	#Cases	#Events	#Attributes
P2P	10	12	12.5K	102K	0–5
Small	10	20	12.5K	111K	0–5
Medium	10	32	12.5K	73K	0–5
Large	10	42	12.5K	138K	0–5
Huge	10	54	12.5K	100K	0–5
Wide	10	34	12.5K	75K	0–5
BPIC12	1	36	13K	262K	0
BPIC13	3	5–13	0.8K–7.5K	2.4K–66K	2–4
BPIC15	5	355–410	0.8K–1.4K	44K–60K	2–3
BPIC17	2	8–26	31K–43K	194K–1.2M	1
Comp	10	7–18	0.9K–56K	4K–180K	1

Notice that we do apply the artificial anomalies to the real-life event logs as well, which very likely already contain natural anomalies. Thereby, we can measure the performance of the algorithms on the real-life logs to demonstrate feasibility while using the synthetic logs to evaluate accuracy.

When applying the artificial anomalies, we also gather a ground truth dataset. Whenever the anomaly is of type *Skip*, *Rework*, or *Switch*, it is a control flow anomaly, and hence the activity name attribute is marked as anomalous for affected events. If the anomaly is of type *Attribute*, the affected attribute is marked as anomalous. Hence, we obtain ground truth data on attribute level. The ground truth data can easily be adapted to case level by the following rule: A case is anomalous if any of the attributes in its events are anomalous.

We generated 10 flavors of each synthetic process model with different numbers of attributes (0, 1, 2, 3, and 5) and different sizes for $\mathcal{V}_a$, resulting in 60 synthetic logs. Together with BPIC12 (1 log), BPIC13 (3 logs), BPIC15 (5 logs), and BPIC17 (2 logs), and the 10 procurement event logs (Comp), the corpus consists of 81 event logs.

4 METHOD

In this section we will describe the BINet architecture and all necessary steps for the implementation.

4.1 *Preprocessing*

Due to the mathematical nature of neural networks, we must transform the logs into a numerical representation. To accomplish this, we encode all string attribute values. Multiple options are available, such as

integer encoding or one-hot encoding. We chose to use an integer encoding, which is a mapping $\mathcal{I}_a : \mathcal{V}_a \to \mathbb{N}$, mapping all possible attribute values for an attribute a to a unique positive integer. The integer encoding is applied to all attributes of the log, including the activity name.

We will represent the event logs as third-order tensors. Each event e is a first-order tensor $\boldsymbol{e} \in \mathbb{R}^A$, with $A = |\mathcal{A}|$, the first attribute always being the activity name, representing the control flow. Hence, an event is defined by its activity name and the event attributes. Each case is then represented as a second-order tensor $\boldsymbol{C} \in \mathbb{R}^{E \times A}$, with $E = \max_{c \in \mathcal{L}} |\hat{c}|$, being the maximum case length of all cases in the log $\mathcal{L}$. To force all cases to have the same size, we pad all shorter cases with event tensors only containing zeros, which we call padding events (these will be ignored by the neural network). The log $\mathcal{L}$ can now be represented as a third-order tensor $\boldsymbol{L} \in \mathbb{R}^{C \times E \times A}$, with $C = |\mathcal{L}|$, the number of cases in log $\mathcal{L}$. Using matrix index notation, we can now obtain the second attribute of the third event in the ninth case with $\boldsymbol{L}_{9,3,2}$. Now we can define a preprocessor as follows:

Definition P3.2. *Preprocessor. Let* C, E, *and* A *be defined as above, then a preprocessor is a mapping* $\mathcal{P} : \mathcal{L} \to \mathbb{R}^{C \times E \times A}$.

The BINet preprocessor $\mathcal{P}$ will encode all attribute values and then transform the log $\mathcal{L}$ into its tensor representation. In the following, we will refer to the preprocessed log $\mathcal{L}$ by $\boldsymbol{F}$ (features), with $\boldsymbol{F} = \mathcal{P}(\mathcal{L})$.

4.2 *BINet Architecture*

BINet is based on a neural network architecture that is trained to predict the next event, including all its attributes. To model the sequential nature of event log data, the core of BINet is a recurrent neural network, using a Gated Recurrent Unit (GRU) [8], an alternative for the popular long short-term memory (LSTM) [13].

We must distinguish between the control flow and the data flow aspect of event log data. Therefore, BINet is composed of two parts: CFNet (control flow) and DataNet (data flow). CFNet is responsible for predicting the next activity name of the next event, while DataNet is responsible for predicting all attributes of the next event.

Figure P3.1 shows the internal architecture of BINet. CFNet retrieves as input all attributes of an event $e^{(t-1)}$ and is trained to predict the activity name of event $e^{(t)}$, where t is the discrete time step. In the figure we use f_{act} for the activity name feature and $f_{supervisor}$ and f_{user} as examples for attribute features. Note that the architecture will grow automatically if more event attributes are present. The output layer of CFNet is one single softmax layer that outputs a probability distribution $\boldsymbol{p}_{act}^{(t)}$ over all possible activity names.

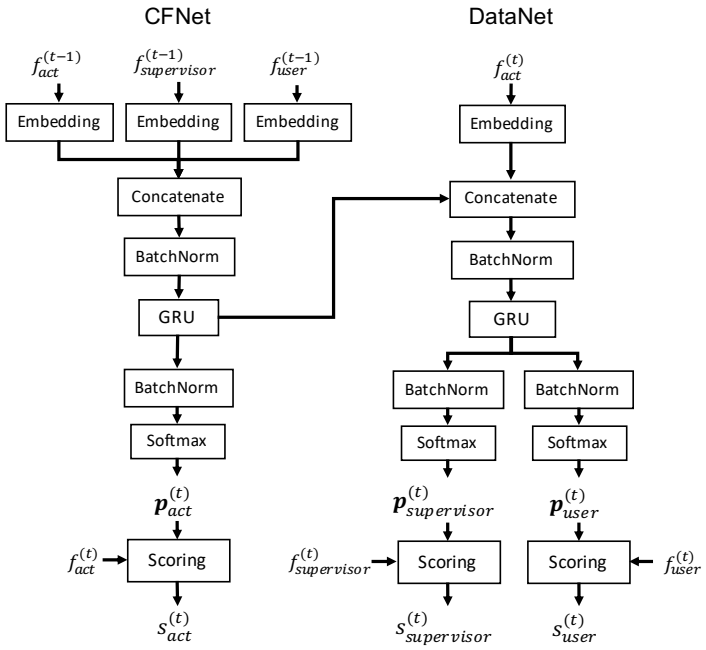

Figure P3.1: BINet architecture for a log with two event attributes, *supervisor* and *user*

DataNet retrieves as input the activity name of $e^{(t)}$ and the internal state of the CFNet GRU and is trained to predict all attributes. DataNet will have a separate softmax layer for each event attribute. Note that DataNet is being fed the activity name at time t, which is the same time it is predicting the attributes for. This is crucial because the attributes of an event strongly depend on the activity. Without this information, DataNet will predict the attributes for the most likely next activity (based on the internal state of the CFNet GRU), which can be different from the actual next activity. Because DataNet is only predicting the attributes of an event and not its activity, using $f_{act}^{(t)}$ is legitimate.

BINet is trained on the event log to predict the next event and all its attributes. After the initial training phase, BINet can now be used for anomaly detection. This is based on the assumption that an anomalous attribute will be assigned a lower probability by BINet than a normal attribute.

The last step of the anomaly detection process is the scoring of the events. Therefore, we use a scoring function in the last layer of the architecture. This scoring function receives as input the probability distribution $\boldsymbol{p}_a^{(t)}$ for an attribute a and the actual value of the attribute $f_a^{(t)}$.

A softmax layer outputs a probability distribution over all possible values. When an event log features 5 different activity names, $\boldsymbol{p}_{act}^{(t)}$ will be a first-order tensor of size 5. Each of the dimensions of $\boldsymbol{p}_{act}^{(t)}$

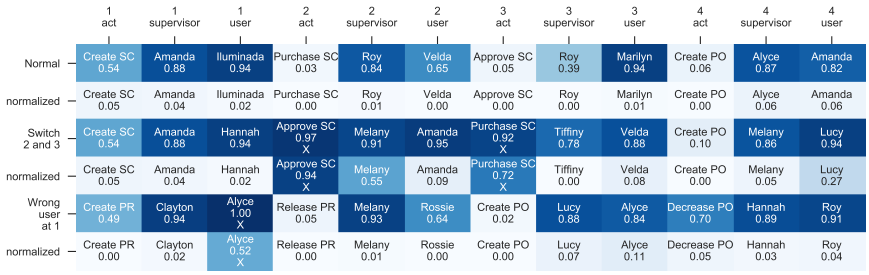

Figure P3.2: Effect of confidence normalization on BINet anomaly scores (high scores indicate anomalies); anomalies are marked with X

holds the probability that the softmax layer assigns to one of the 5 possible activity values.

We can now define the scoring function σ as the difference between the probability of the most likely attribute (according to BINet) and the probability of the attribute value encountered $f_a^{(t)}$, where $\boldsymbol{p}_a$ is the probability tensor for attribute a and an event e, and i is the corresponding index of $f_a^{(t)}$ in $\boldsymbol{p}_a$.

$$\sigma(\boldsymbol{p}_a, i) = \max_{p \in \boldsymbol{p}_a} p - \boldsymbol{p}_{a_i}$$

The effect of normalization is demonstrated by Fig. P3.2, which shows three example cases of a P2P dataset with 2 attributes with and without normalization. Without normalization, the scoring function is defined as $\sigma(\boldsymbol{p}_a, i) = 1 - \boldsymbol{p}_{a_i}$, i.e., the inverse probability. We can observe that BINet is able to accurately predict the activities after the first activity, however, the anomaly scores for the user attribute are close to 1 because multiple users are permitted to execute an activity. We can counteract this effect by applying the normalization by confidence as demonstrated in Fig. P3.2.

The complete BINet architecture is then comprised of CFNet, DataNet and the scoring function σ applied to each softmax layer. We can obtain the anomaly scores tensor S by applying BINet to the feature tensor $\boldsymbol{F}$

$$\boldsymbol{S} = (s_{ijk}) \in \mathbb{R}^{C \times E \times A} = \text{BINet}(\boldsymbol{F}),$$

mapping an anomaly score to each attribute in each event in each trace. The anomaly score for attributes of padding events will always be 0.

4.3 *Training*

BINet is trained without the scoring function. The GRU units are trained in sequence to sequence fashion. With each event that is fed in, the network is trained to predict the attributes of the next event.

We train BINet with a GRU size of $2E$ (two times the maximum case length), on mini batches of size 100 for 50 epochs using the Adam [16] optimizer using the parameters stated in the original paper. We use batch normalization [14] between all layers to counteract overfitting. Every feature is passed through a separate embedding layer (see [19]) to reduce the input dimension.

4.4 *Detection*

An anomaly detector only outputs anomaly scores. We need to define a function that maps anomaly scores to a label $l \in \{0, 1\}$, 0 indicating normal and 1 indicating anomalous, by applying a threshold t. Whenever an anomaly score for an attribute is greater than t, this attribute is flagged as anomalous. To obtain a separate threshold for each anomaly score in $\boldsymbol{S}$, we define a threshold tensor $\boldsymbol{T} = \alpha \cdot \boldsymbol{\tau}$, where $\boldsymbol{\tau} \in \mathbb{R}^{C \times E \times A}$ is a baseline threshold tensor and $\alpha \in \mathbb{R}$ is a scaling factor. Now we can define the function θ, with inputs $\boldsymbol{S}$, α, and $\boldsymbol{\tau} = (\tau_{ijk})$, as

$$\theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) = (p_{ijk}) = \begin{cases} 1 & \text{if } (s_{ijk}) > \alpha \cdot (\tau_{ijk}) \\ 0 & \text{otherwise} \end{cases}.$$

To obtain a baseline threshold $\boldsymbol{\tau} \in \mathbb{R}^{C \times E \times A}$, we propose four different strategies τ_0, τ_e, τ_a, and τ_{ea}. In the following we will use $N = \sum_{c \in \mathcal{L}} |\hat{c}|$ to denote the number of non-padding events. The first baseline threshold function, τ_0, is defined by the average anomaly score over all cases, events, and attributes.

$$\tau_0(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{NA} \sum_{a}^{C} \sum_{b}^{E} \sum_{c}^{A} (s_{abc})$$

It is sensible to use a separate threshold for each event position in a case because the branching factor can vary for different points in a process model. Therefore, we define the second baseline threshold function, τ_e, based on the position of an event e in a case c.

$$\tau_e(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{CA} \sum_{a}^{C} \sum_{c}^{A} (s_{ajc})$$

It is also sensible to use a separate threshold for each attribute because $\mathcal{V}_a$ has a different size for each event e and attribute a. Thus, we define the third baseline threshold function, τ_a, to output a separate threshold for each event attribute.

$$\tau_a(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{N} \sum_{a}^{C} \sum_{b}^{E} (s_{abk})$$

The fourth baseline threshold function, τ_{ea} is a combination of τ_e and τ_a, and outputs a threshold for each event position and attribute separately.

$$\tau_{ea}(\boldsymbol{S}) = (\tau_{ijk}) = \frac{1}{C} \sum_{a}^{C} \left(s_{ajk}\right)$$

Note that we use matrix index notation to broadcast $\boldsymbol{\tau}$ to the right dimensionality to conform with the definition of θ from before. Utilizing the automatic broadcasting functionality in modern numerical computing libraries, such as TensorFlow[6] or NumPy[7], this can be implemented very efficiently. Remember that anomaly scores for padding events are set to 0, and hence they do not influence the sums. By normalizing with N we calculate the average based only on non-padding events.

4.5 *Threshold Heuristic*

Most anomaly detection algorithms rely on a manual setting for the threshold. We have proposed four different methods of obtaining a baseline threshold tensor $\boldsymbol{\tau}$ from the anomaly scores tensor $\boldsymbol{S}$. We still need to set the scaling factor α manually. To overcome this, we need to introduce a heuristic to set α automatically.

Because anomaly detection is an unsupervised task and no labels are available at runtime, we cannot optimize α based on the detection F_1 score. However, using the F_1 score function we can define the heuristic h_{best} that computes the best possible α for a given anomaly score tensor $\boldsymbol{S}$, a baseline threshold $\boldsymbol{\tau}$, and the ground truth label tensor $\boldsymbol{L}$ as

$$h_{best}(\boldsymbol{L}, \boldsymbol{S}, \boldsymbol{\tau}) = \arg\max_{\alpha} F_1(\boldsymbol{L}, \theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau})).$$

As we do not have access to L at runtime, we cannot use h_{best}. We propose a new heuristic that works like the elbow method, commonly used to find an optimal number of clusters for clustering algorithms (see [25]). As we cannot rely on the F_1 score as our metric, we propose the use the anomaly ratio r, which can be defined as, with $\theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) = (p_{ijk})$.

$$r(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) = \frac{1}{CEA} \sum_{i}^{C} \sum_{j}^{E} \sum_{k}^{A} (p_{ijk})$$

The optimal α must lie between α_{low} and α_{high}, where $r(\boldsymbol{S}, \alpha_{low}, \boldsymbol{\tau}) = 1$ and $r(\boldsymbol{S}, \alpha_{high}, \boldsymbol{\tau}) = 0$. We can reduce our search space to this in-

6 https://tensorflow.org
7 http://numpy.org

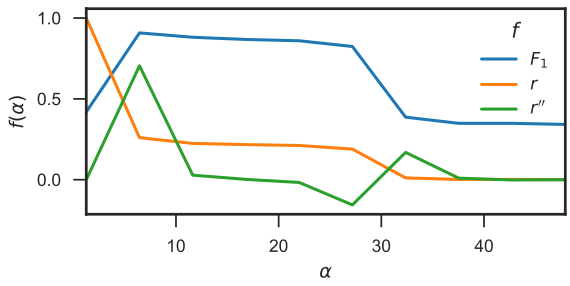

Figure P3.3: F_1 score, anomaly ratio r, and second order derivative r'' (scaled for clarity) by α for BINet on a dataset with 5 attributes using τ_a as the baseline threshold

terval. Now we span a grid G of size s between α_{low} and α_{high} to define our candidates for α.

$$G = \left\{ \alpha_{low} + \frac{1}{s}(\alpha_{high} - \alpha_{low}), \ldots, \alpha_{low} + \frac{s}{s}(\alpha_{high} - \alpha_{low}) \right\}$$

In our experiments we found that $s = 20$ is a good choice for s, however, any reasonable choice of $s \in \{5, \ldots, 100\}$ generally works.

Because r is a discrete function, we use the central difference approximation to obtain the second order derivative r'' of r.

$$r''(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) \approx \frac{r(\boldsymbol{S}, \alpha - s, \boldsymbol{\tau}) - 2r(\boldsymbol{S}, \alpha, \boldsymbol{\tau}) + r(\boldsymbol{S}, \alpha + s, \boldsymbol{\tau})}{s^2}$$

Now we can define the elbow heuristic

$$h_{elbow}(\boldsymbol{S}, \boldsymbol{\tau}) = \arg\max_{\alpha \in G} r''(\boldsymbol{S}, \alpha, \boldsymbol{\tau}).$$

h_{elbow} mimics the way a human would set α manually. When given a user interface with a heatmap visualization (like in Fig. P3.2) for $\theta(\boldsymbol{S}, \alpha, \boldsymbol{\tau})$ and control over the value of α, a human would start with a value of α where all attributes are marked as anomalous (i.e., the heatmap shows only blue and no white), and then gradually decrease α until the point where the heatmap switches from mostly showing blue to mostly showing white.

Figure P3.3 shows the F_1 score and the corresponding anomaly ratio r for a model of BINet, trained on a P2P dataset and using τ_a as the baseline threshold. We can see that the highest possible F_1 score correlates with the maximum of r'', and hence with the "elbow" of r. Interestingly, we found that h_{elbow} works just as well for other anomaly detection algorithms, such as t-STIDE [26], Naive [3], and DAE [21].

Evaluating h_{elbow} for BINet over all synthetic datasets and all baseline threshold strategies, we can see in Fig. P3.4 that the best baseline threshold is τ_a. We also find, that the h_{elbow} works remarkably well over all strategies, for the performance of h_{elbow} is very close to h_{best}.

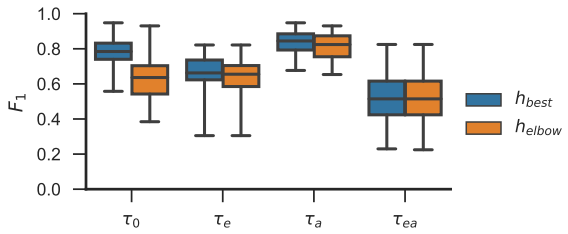

Figure P3.4: F_1 score by strategy and heuristic for BINet on the P2P dataset

5 EVALUATION

We evaluated BINet on all 81 event logs and compared it to two methods from [7]: a sliding window approach (t-STIDE+) [26]; and the one-class SVM (OC-SVM). Additionally, we compared BINet to two approaches from [3]: the Naive algorithm and the Sampling algorithm. Furthermore, we provide the results of the denoising autoencoder (DAE) approach from [20]. Lastly, we compared BINet to the approach from [5], which utilizes an extended likelihood graph (Likelihood). As a baseline, we provide the results of a random classifier.

For the OC-SVM, we relied on the implementation of scikit-learn[8] using an RBF kernel of degree 3 and $\nu = 0.5$. The Naive, Sampling, Likelihood, and DAE methods were implemented as described in the original papers. t-STIDE+ is an implementation of the t-STIDE method from [26], which we adapted to work with event attributes (see [20]).

Sampling, Likelihood, Baseline, and the OC-SVM do not rely on a manual setting of the threshold and were unaltered. For the remaining algorithms we used h_{elbow} and chose the following baseline threshold strategies following a grid search: τ_0 for Naive, τ_{ea} for t-STIDE+, τ_a for DAE, and τ_a for BINet.

Figure P3.5 shows the F_1 score distribution for all methods over all datasets and for the two detection levels. F_1 score is calculated as the macro average F_1 score over the normal and the anomalous class. BINet outperforms all other methods on both detection levels. The more important detection level is the attribute level, as this measure demonstrates how accurately an anomaly detection algorithm can detect the actual attribute that caused an anomaly.

Expectedly, methods without attribute resolution, like Naive and OC-SVM, perform poorly on attribute level. t-STIDE+, Likelihood, and DAE support detection on attribute level, but show a significantly lower performance than BINet. DAE and BINet are both neural network based. The main advantage of BINet over DAE is that BINet makes use of the time dimension in the sequential data, whereas DAE does not.

8 http://scikit-learn.org

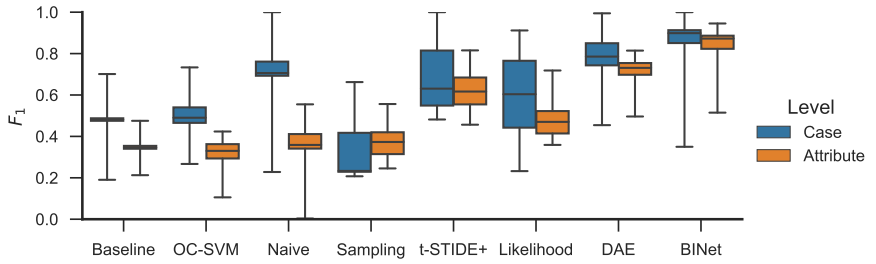

Figure P3.5: F_1 score by method and detection level using h_{elbow} where applicable

Table P3.2: Results showing F_1 score over all datasets by detection level and method; best results are shown in bold typeface

Level	Method	P2P	Small	Medium	Large	Huge	Wide	BPIC12	BPIC13	BPIC15	BPIC17	Comp
Case	Baseline	0.47	0.50	0.47	0.48	0.48	0.50	0.55	0.50	0.49	0.47	0.45
	OC-SVM [23]	0.49	0.50	0.51	0.53	0.52	0.52	0.42	0.52	0.47	0.60	0.51
	Naive [3]	**0.91**	0.80	0.71	0.71	0.76	0.72	0.58	0.47	0.24	0.53	0.63
	Sampling [3]	0.23	0.23	0.44	0.34	0.23	0.23	0.45	0.26	0.22	0.22	0.57
	t-STIDE+ [26]	0.72	0.71	0.73	0.68	0.69	0.67	**0.81**	0.57	**0.51**	0.68	0.68
	Likelihood [5]	0.64	0.65	0.62	0.61	0.65	0.60	0.65	0.29	0.30	0.53	**0.73**
	DAE [20]	0.86	0.81	0.78	0.89	0.80	0.75	0.76	0.52	0.45	**0.75**	0.68
	BINet	**0.91**	**0.92**	**0.92**	**0.92**	**0.92**	**0.92**	0.58	**0.58**	0.49	0.58	0.66
Attribute	Baseline	0.34	0.36	0.35	0.35	0.35	0.36	0.35	0.35	0.34	0.35	0.36
	OC-SVM [23]	0.33	0.34	0.32	0.35	0.34	0.32	0.11	0.36	0.32	0.37	0.26
	Naive [3]	0.50	0.41	0.35	0.35	0.39	0.37	0.09	0.14	0.00	0.24	0.36
	Sampling [3]	0.28	0.32	0.40	0.46	0.40	0.36	0.37	0.30	0.29	0.32	0.48
	t-STIDE+ [26]	0.66	0.66	0.68	0.64	0.64	0.65	0.67	0.56	0.51	0.62	0.49
	Likelihood [5]	0.50	0.50	0.51	0.48	0.50	0.50	0.47	0.40	0.36	0.47	0.52
	DAE [20]	0.77	0.72	0.72	0.75	0.75	0.71	**0.72**	0.52	0.51	**0.73**	0.61
	BINet	**0.85**	**0.89**	**0.87**	**0.89**	**0.88**	**0.88**	0.59	**0.57**	**0.54**	0.64	**0.68**

Table P3.2 contains the detailed results for each method by dataset and detection level. BINet performs best across levels on the synthetic logs. On the real event logs, BINet performs best on attribute level, whereas on case level, the field is mixed. An accurate prediction on attribute level is to be favored over an accurate prediction on case level because only the attribute level allows to identify the exact cause of an anomaly.

Fig. P3.6 shows a heatmap of BINet anomaly scores for a P2P dataset with 2 attributes. Two example cases are chosen for each type of anomaly and the normal class to demonstrate how BINet detects anomalies based on attribute level. No threshold has been applied to the anomaly scores. This way, the severity of an anomaly can be illustrated by the colors in the heatmap. Overall, we find that BINet detects control flow anomalies very effectively. For example, a shopping cart (SC) cannot be approved before it has been purchased (first Switch). Similarly, a purchase requisition (PR) cannot be released before it has been created (second Skip). Rework and Attribute anomalies are also detected accurately. In the case of Attribute, only the incorrect attribute is assigned a significantly high anomaly score. In the two examples,

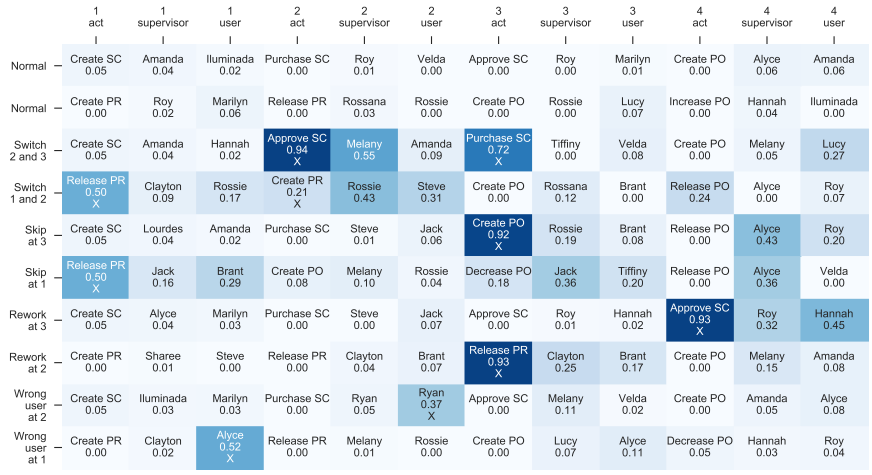

Figure P3.6: Anomaly score heatmap for BINet trained on P2P with 2 attributes (supervisor and user); anomalies are marked by X

Ryan cannot be his own supervisor and Alyce is not permitted to create a PR.

6 CONCLUSION

In this paper we presented BINet, a neural network architecture for multivariate anomaly detection in business process event logs. Additionally, we proposed a heuristic for setting the threshold of an anomaly detection algorithm automatically, based on the anomaly ratio function.

BINet is a recurrent neural network, and can therefore be used for real-time anomaly detection, since it does not require a completed case for detection. BINet does not rely on any information about the process modeled by an event log, nor does it depend on a clean dataset. Utilizing the elbow heuristic, BINet's internal threshold can be set automatically, reducing manual workload. It can be used to find point anomalies as well as contextual anomalies because it models the time dimension in event sequences and utilizes both the control flow and the data flow information. Furthermore, BINet can cope with concept drift, as it can be setup to continuously train on new cases in real-time.

Based on the empirical evidence obtained in the evaluation, BINet is a promising method for anomaly detection, especially in business process event logs. BINet outperformed the opposition on all detection levels (case, event, and attribute level). Specifically, on the synthetic datasets BINet's performance surpasses those of other methods by an order of magnitude.

For an accurate detection of an anomaly it is essential that an anomaly detection algorithm processes event logs on attribute level; otherwise, a control flow anomaly cannot be distinguished from a

data flow anomaly. To allow easy analysis of an event log, the attribute level is the most important detection level. On attribute level, BINet performs significantly better than the other methods.

Overall, the results presented in this paper suggest that BINet is a reliable and versatile method for detecting attribute anomalies in business process logs.

ACKNOWLEDGMENTS

This project [522/17-04] is funded in the framework of Hessen ModellProjekte, financed with funds of LOEWE, Förderlinie 3: KMU-Verbundvorhaben (State Offensive for the Development of Scientific and Economic Excellence), and by the German Federal Ministry of Education and Research (BMBF) Software Campus project "AI-PM" [01IS17050].

REFERENCES

[1] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016 (cit. on pp. 90–92).

[2] Fábio Bezerra and Jacques Wainer. "Anomaly detection algorithms in logs of process aware systems." In: *Proceedings of the 2008 ACM symposium on Applied computing*. ACM. 2008, pp. 951–952 (cit. on p. 91).

[3] Fábio Bezerra and Jacques Wainer. "Algorithms for anomaly detection of traces in logs of process aware information systems." In: *Information Systems* 38.1 (2013), pp. 33–44 (cit. on pp. 93, 100–102).

[4] Fábio Bezerra, Jacques Wainer, and Wil M. P. van der Aalst. "Anomaly detection using process mining." In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, 2009, pp. 149–161 (cit. on p. 91).

[5] Kristof Böhmer and Stefanie Rinderle-Ma. "Multi-perspective Anomaly Detection in Business Process Execution Events." In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer. 2016, pp. 80–98 (cit. on pp. 91, 93, 101, 102).

[6] Andrea Burattin. "PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings." In: *arXiv preprint arXiv:1506.08415* (2015) (cit. on p. 93).

[7] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly Detection for Discrete Sequences: A Survey." In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 823–839 (cit. on pp. 91, 92, 101).

[8] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on p. 95).

[9] Corinna Cortes and Vladimir Vapnik. "Support-vector networks." In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on p. 92).

[10] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "A deep learning approach for predicting process behaviour at runtime." In: *International Conference on Business Process Management*. Springer. 2016, pp. 327–338 (cit. on p. 92).

[11] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "Predicting process behaviour using deep learning." In: *Decision Support Systems* 100 (2017), pp. 129–140 (cit. on p. 92).

[12] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011 (cit. on p. 90).

[13] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory." In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on pp. 92, 95).

[14] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *International conference on machine learning*. 2015, pp. 448–456 (cit. on p. 98).

[15] Nathalie Japkowicz. "Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks." In: *Machine Learning* 42.1 (2001), pp. 97–122 (cit. on p. 92).

[16] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 98).

[17] Pankaj Malhotra et al. "LSTM-based encoder-decoder for multi-sensor anomaly detection." In: *arXiv preprint arXiv:1607.00148* (2016) (cit. on p. 92).

[18] Erik Marchi, Fabio Vesperini, Florian Eyben, Stefano Squartini, and Björn Schuller. "A Novel Approach for Automatic Acoustic Novelty Detection Using a Denoising Autoencoder with Bidirectional LSTM Neural Networks." In: (Apr. 2015) (cit. on p. 92).

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." In: *arXiv preprint arXiv:1301.3781* (2013) (cit. on p. 98).

[20] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. "Analyzing Business Process Anomalies Using Autoencoders." In: *arXiv preprint arXiv:1803.01092* (2018) (cit. on pp. 92, 101, 102).

[21] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders." In: *International Conference on Discovery Science*. Springer. 2016, pp. 442–456 (cit. on pp. 92, 100).

[22] Marco A. F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. "A review of novelty detection." In: *Signal Processing* 99 (2014), pp. 215–249 (cit. on p. 91).

[23] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, John C. Platt, et al. "Support vector method for novelty detection." In: *NIPS*. Vol. 12. 1999, pp. 582–588 (cit. on pp. 92, 102).

[24] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. "Predictive business process monitoring with LSTM neural networks." In: *International Conference on Advanced Information Systems Engineering*. Springer. 2017, pp. 477–492 (cit. on p. 92).

[25] Robert Tibshirani, Guenther Walther, and Trevor Hastie. "Estimating the number of clusters in a data set via the gap statistic." In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63.2 (2001), pp. 411–423 (cit. on p. 99).

[26] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. "Detecting intrusions using system calls: Alternative data models." In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. IEEE. 1999, pp. 133–145 (cit. on pp. 91, 100–102).

[27] Lijie Wen, Wil M. P. van der Aalst, Jianmin Wang, and Jiaguang Sun. "Mining process models with non-free-choice constructs." In: *Data Mining and Knowledge Discovery* 15.2 (2007), pp. 145–180 (cit. on p. 91).

[28] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. "A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification." In: *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security*. AISec '13. ACM, 2013, pp. 67–76 (cit. on p. 91).

P4

BINET: MULTI-PERSPECTIVE BUSINESS PROCESS ANOMALY CLASSIFICATION

Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser
In: *Information Systems* (2019), p. 101458

ABSTRACT: In this paper, we introduce BINet, a neural network architecture for real-time multi-perspective anomaly detection in business process event logs. BINet is designed to handle both the control flow and the data perspective of a business process. Additionally, we propose a set of heuristics for setting the threshold of an anomaly detection algorithm automatically. We demonstrate that BINet can be used to detect anomalies in event logs not only at a case level but also at event attribute level. Finally, we demonstrate that a simple set of rules can be used to utilize the output of BINet for anomaly classification. We compare BINet to eight other state-of-the-art anomaly detection algorithms and evaluate their performance on an elaborate data corpus of 29 synthetic and 15 real-life event logs. BINet outperforms all other methods both on the synthetic as well as on the real-life datasets.

KEYWORDS: Business Process Management, Anomaly Detection, Artificial Process Intelligence, Deep Learning, Recurrent Neural Networks

1 INTRODUCTION

Corporations are relying on business processes to ensure a smooth operation of their business. Deviations from these processes can have a significant impact on the economic well-being of a company. Naturally, businesses are interested in finding and, subsequently, reducing these deviations. Analyzing the business processes and their performance has traditionally been a human task, and therefore a manual effort. However, in the digital age, every business process leaves a digital footprint, allowing for a completely data-driven analysis of business processes.

Anomaly detection is a popular technique to automatically find these deviations based on the data the businesses are generating. An anomaly, in this context, is a deviation from the behavior defined by the business process. This can be related to the order of activities (control flow perspective), e.g., two activities are executed in the wrong

order. Furthermore, it can be related to certain data attributes (data perspective), e.g., an employee has executed an activity without permission or violations against segregation of duty regulations. These two perspectives are inherent to how business process data is saved (see [1]) and must be accounted for by a business process anomaly detection algorithm.

The basis for an accurate anomaly detection is an accurate model of the normal behavior, i.e., the normal business process. Any behavior not captured by this model can be regarded as anomalous. Some methods rely on a human-defined process model as the basis for normal behavior, called a reference model. However, such a reference model is not always available and it might not be economically justifiable to create such a reference model. In those cases, a method that does not require such a reference model is desired.

Autonomous anomaly detection algorithms have to infer this model directly from the patterns in the data and without any domain knowledge about the underlying process. This can be challenging because the data naturally contains anomalous behavior which the algorithm has to ignore. Some algorithms depend on a clean dataset, that is, a dataset that only consists of normal behavior. However, this requires a dedicated preprocessing step based on domain knowledge. Very few business process anomaly detection algorithms exist that infer the normal model from an uncleaned dataset. In Sec. 3 we give an elaborate summary of existing approaches.

There are further desired properties for a business process anomaly detection algorithm. It is important that an algorithm can be used during the execution of the business process because otherwise no countermeasures can be initiated in time. Furthermore, business processes are dynamic systems, adapting to seasonal effects, market changes, or other external factors. An autonomous anomaly detection algorithm should deal with this dynamic behavior.

Against this background, we propose BINet (Business Intelligence Network), a neural network architecture for real-time multi-perspective business process anomaly detection. BINet has been specifically designed to model both the control flow and the data perspective of business process execution data. Thus, it can capture a variety of different causal dependencies, allowing to detect anomalies related to control flow and data perspective (cf. point and contextual anomalies in [15]). It does not rely on domain knowledge about the business process or a reference model and it does not require prior knowledge about the anomalies such as anomaly labels. BINet can be run continuously during the execution of a business process, adapting to concept drift and enabling early detection at runtime. The only input to the BINet algorithm is the business process data, including anomalous behavior. BINet distinguishes between normal and anomalous behavior by automatically defining a threshold solely based on the input data.

This paper is an extension of our previous work on BINet from 2018 [26]. Compared to the original publication, we have simplified the architecture of BINet and present three versions of BINet with different dependency modeling capabilities. Additionally, we elaborate on the threshold heuristics proposed in [26] and introduce an improved set of heuristics mimicking human intuition. We improved the dataset generation algorithm, which now uses an extended likelihood graph to generate causally dependent activities and event attributes. Finally, we propose a simple rule-based classifier to distinguish different anomaly classes, solely based on the outputs of BINet.

This work contains five main contributions.

- BINet neural network architecture
- Automatic threshold heuristics
- Generation algorithm for synthetic event logs
- Comprehensive evaluation of state-of-the-art methods
- Rule-based anomaly classifier

This paper is structured as follows. We start with background information on process mining and neural networks in Sec. 2, followed by a comprehensive summary of related works and how they relate to BINet in Sec. 3. Then, we elaborate on the generation algorithm for realistic datasets in Sec. 4. In Sec. 5, we describe BINet's preprocessing, neural architecture, detection algorithm, and automatic threshold heuristic. We compare BINet to 8 state-of-the-art anomaly detection methods and evaluate them on a comprehensive dataset of 29 synthetic logs and 15 real-life logs, using artificial anomalies in Sec. 6. Lastly, we demonstrate that BINet can be used to distinguish anomalies in Sec. 7. After a short discussion in Sec. 8, we conclude this paper in Sec. 9.

2 BACKGROUND

In this section, we give an introduction to the main concepts behind business process anomaly detection.

2.1 *Process Mining*

For business process data, the most popular techniques originate from the field of process mining. Process mining is centered around the idea of human-readable representations of business processes called process models. Process models are widely used in business process management as a tool for defining, documenting, and controlling business processes inside companies.

During the execution of a digital business process, each process step is stored in a database. This includes information on when the process

step was executed (timestamp), what process step was executed (activity), and to which business case it belongs (case identifier). These three fundamental bits of event information are the basis for every process mining algorithm and are usually combined into a single data structure called event log.

A log consists of cases, each of which consists of events executed within a process. Each event is defined by an activity name and its attributes (e.g., a user who executed the event). We use a nomenclature adapted from [1].

Definition P4.1. *Case, Event, Log, and Attribute. Let $\mathcal{C}$ be the set of all cases, and $\mathcal{E}$ be the set of all events. The event sequence of a case $c \in \mathcal{C}$, denoted by $\hat{c}$, is defined as $\hat{c} \in \mathcal{E}^*$, where $\mathcal{E}^*$ is the set of all sequences over $\mathcal{E}$. An event log is a set of cases $\mathcal{L} \subseteq \mathcal{C}$. Let $\mathcal{A}$ be a set of attributes and $\mathcal{V}$ be a set of attribute values, where $\mathcal{V}_a$ is the set of possible values for the attribute $a \in \mathcal{A}$. $|\hat{c}|$ is the number of events in case c, $|\mathcal{L}|$ is the number of cases in $\mathcal{L}$, and $|\mathcal{A}|$ is the number of event attributes.*

Based on these event logs, process mining is used to discover the underlying business process and produce a human-readable process model visualization. These algorithms are known as discovery algorithms. An optimal process model should be simple (simplicity), general (generality), precise (precision), and model all behavior found in the event log (fitness). However, these goals are conflicting because, for instance, a process model cannot be both simple and precise, and thus discovery algorithms often emphasize two of the four goals. Simplicity can be emphasized, for example, by ignoring infrequent patterns, since they are unlikely to be part of the real business process.

Some companies document their processes and save them as digital process models. These process models can be used to check whether or not the process is being executed according to it. This is called conformance checking. The result is an annotated event log in which events are marked as conform or non-conform. Non-conform can either indicate that the activity is found in the event log, but is not part of the process, or the activity is found in the model, but not in the event log.

Conformance checking can be used for anomaly detection if a reference model is available. Otherwise, a discovery algorithm can be used to infer such a reference model from the event log. Simplicity and generality are usually preferred over precision and fitness in this setting. However, most discovery and conformance checking algorithms mainly focus on the control flow perspective, and hence they can only partially be applied to the data perspective.z

2.2 *Artificial Neural Networks*

In fields like computer vision, natural language processing, and speech recognition, artificial neural networks (NNs) have become the state-

of-the-art, outperforming other machine learning algorithms by great margins. Key to this success is a concept called representation learning. Aforementioned fields are especially challenging because they require sophisticated feature engineering, namely transforming the raw data into meaningful features to be used as input for the machine learning algorithm. NNs are capable of discovering the necessary representations as part of their training algorithm, reducing, or even removing entirely, the need for feature engineering. This has opened up many areas for the application of machine learning algorithms that were originally considered impractical.

One of these areas is natural language processing, which can be considered particularly relevant to our case since business process executions hold some similarities to natural language, such as the sequential nature. Activities can be considered as words, cases as sentences, and the order of events as well as the event attributes as the grammar of the language. Furthermore, if the same activity is executed by different users, the two are considered distinct. Similarly, if the same activity is preceded by different events, they are also considered distinct.

An NN is a non-linear composite function with multiple degrees of freedom, called weights. These weights can be altered to control the output of the NN, minimizing the error with respect to some desired output. This optimization procedure is referred to as training. During training, the weights are iteratively updated by continuously considering examples of inputs (e.g., a sequence of events) and desired outputs (e.g., the most probable next activity), and changing the weights according to their influence on the output. The goal of the training is to find a set of weights that approximates, as closely as possible, a mapping function from inputs to the respective desired outputs.

Recurrent neural networks (RNNs) have been designed specifically to handle sequential data such as sentences. Instead of processing an event sequence all-at-once, an RNN processes each event individually, recurrently using the same set of weights. To retain information about past events, an RNN uses an internal state (memory), that is updated for each event. This internal state resembles a representation of the event sequence up until that point. The update of the internal state is again controlled by a separate set of weights which are also optimized as part of the training procedure.

A drawback of the design of a classic RNN is that it is forced to update its internal state with each new event, thereby potentially overriding previous information. As a consequence, older events are not as strongly remembered as very recent ones. The RNN quickly forgets about the distant past. Hochreiter and Schmidhuber have addressed this problem with their design of the long short-term memory (LSTM) [16].

Instead of forcing the network to always update its internal state, an LSTM features specific sets of weights that control if the internal state is updated. Thus, through the training procedure, an LSTM can learn when to remember important information, when to retrieve it, and when to forget it. Because an LSTM is not forced to remember everything, it can retain information about past events for arbitrary long sequences, and without loss of memory. Contrary to RNNs, where the internal state at any time is a representation of all events up until that point, the internal state of an LSTM is a representation of important events (i.e., events worth remembering) up until that point.

In the field of natural language processing, LSTMs are used to predict the most probable word to continue an unfinished sentence. This concept can be utilized for anomaly detection in business processes. For an unfinished business case, an LSTM predicts the most probable next event, and if the prediction does not match the actual next event, it can be considered to be anomalous.

3 RELATED WORK

In the field of process mining [1], it is popular to use discovery algorithms to mine a process model from an event log and then use conformance checking to detect anomalous behavior [2, 4, 31]. However, the proposed methods do not utilize the event attributes, and therefore cannot be used to detect anomalies at attribute level.

A more recent publication proposes the use of likelihood graphs to analyze business process behavior [5]. This method includes important characteristics of the process itself by including the event attributes as part of an extended likelihood graph. However, this method relies on a discrete order in which the attributes are connected to the graph, which may introduce a bias towards certain attributes. Furthermore, the same activities are mapped to the same node in the likelihood graph, thereby assigning a single probability distribution to each activity. In other words, control flow dependencies cannot be modeled by the likelihood graph, because the probability distribution of attributes following an activity does not depend on the history of events.

The main drawback of this method is that it uses the initial log to build the likelihood graph, and therefore no case of the original log is classified as anomalous. This is related to the strategy to determine a threshold for the anomaly detection task. We address this caveat in Sec. 6. However, the notion of the likelihood graph inspired the generation method for synthetic event logs in Sec. 4.

A review of traditional anomaly detection methodology can be found in [27]. The authors describe and compare many methods that have been proposed over the last decades. Another elaborate summary of anomaly detection in discrete sequences is given by Chandola et al. in [7]. 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 estimate the probability distribution of the normal class and thus can detect anomalies as they come from a different distribution. An important probabilistic technique is the sliding window approach [30]. In window-based anomaly detection, an anomaly score is assigned to each window in a sequence. Then the anomaly score of the sequence can be inferred by aggregating the window anomaly scores. Recently, Wressnegger et al. used this approach for intrusion detection and gave an elaborate evaluation in [32]. While being inexpensive and easy to implement, sliding window approaches show a robust performance in finding anomalies in sequential data, especially within short regions [7].

Distance-based novelty detection does not require a clean dataset, yet it is only partly applicable to process cases, as anomalous cases are usually very similar to normal ones. A popular distance-based approach is the one-class support vector machine (OC-SVM). Schölkopf et al. [28] first used support vector machines [9] for anomaly detection.

Reconstruction-based novelty detection (e.g., neural networks) is based on the idea to train a model that can reconstruct normal behavior but fails to do so with anomalous behavior. Therefore, the reconstruction error can be used to detect anomalies [18]. This approach has successfully been used for the detection of control flow anomalies [25] as well as data flow anomalies [24] in event logs of PAISs.

Domain-based novelty detection requires domain knowledge, which is something we want to avoid. Information-theoretic novelty detection defines anomalies as the examples that influence an information measure (e.g., entropy) on the whole dataset the most. Iteratively removing the data with the highest impact yields a cleaned dataset and thus a set of anomalies.

The core of BINet is a recurrent neural network, trained to predict the next event and its attributes. The architecture is influenced by the works of Evermann [12, 13] and Tax [29], who utilized long short-term memory (LSTM) [16] networks for next event prediction, demonstrating their utility. LSTMs have been used for anomaly detection in different contexts like acoustic novelty detection [21] and predictive maintenance [20]. These applications mainly focus on the detection of anomalies in time series and not, like BINet, on multi-perspective anomaly detection in discrete sequences of events.

The novelty of BINet lies in the tailored architecture for business processes, including the control flow and data perspective, the scoring function to assign anomaly scores, and the automatic threshold heuristic. It is a universally applicable method for anomaly detection both in the control flow and the data perspective of business process event logs. Lastly, BINet can handle multiple event attributes and model

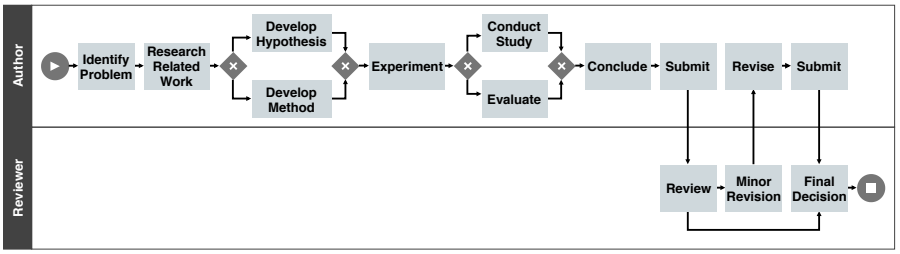

Figure P4.1: A simple paper submission process which is used as an example throughout the paper

causal dependencies between control flow and data perspective, as well dependencies between event attributes. This combination is, to the best of our knowledge, novel to the field.

4 GENERATING REALISTIC EVENT LOGS

To calculate the performance of a business process anomaly detection algorithm, a labeled dataset is necessary. In this section, we will describe a generation algorithm that can be used to generate realistic event logs from random process models of different complexities. Additionally, we will describe how realistic artificial anomalies can be added to these event logs, resulting in a labeled dataset.

We use a simple paper submission process as the running example to illustrate concepts, methods, and results throughout this paper. The process model in Fig. P4.1 describes the creation of a scientific paper. Note that the process includes the peer review process, which is executed by a reviewer, whereas the paper is conceptualized and compiled by an author.

4.1 *Synthetic Event Log Generation*

We used PLG2 [6] to generate six random process models. The models vary in complexity with respect to the number of activities, breadth, and width. Additionally, we use a handmade procurement process model called P2P as in [26], and the paper process shown in Fig. P4.1.

To generate causally dependent event attributes, we adopt the notion of the extended likelihood graph from [5]. A likelihood graph is a directed graph where each node represents a possible activity and the connection between nodes represents the probability of one activity following another. For example, activity B has a 60 percent chance of following A. This idea can be extended to include multiple event attributes. For instance, B has a 35 percent chance of following A if it is Monday or B has a 15 percent chance of following A if it is Monday and user 1 executed the activity. This pattern can be continued to include an arbitrary numbers of event attributes.

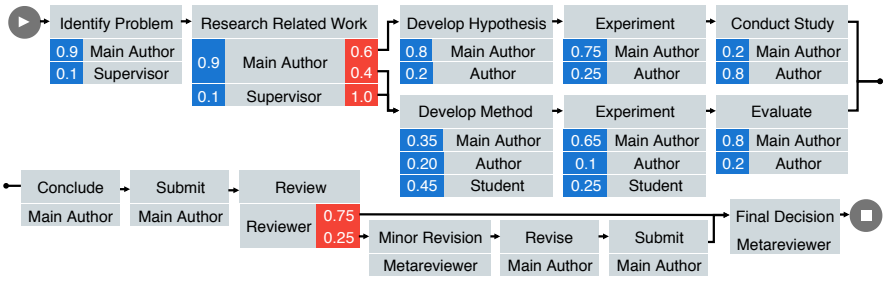

Figure P4.2: A likelihood graph with user attribute; 1.0 probabilities omitted for simplicity

We applied this technique to the process shown in Fig. P4.1. For each activity, we create a group of possible users allowed to execute the activity and assign probabilities to each user. Figure P4.2 demonstrates the final result.

The *Experiment* activity appears twice in this likelihood graph to introduce a long-term control flow dependency. That is, *Conduct Study* always eventually follows *Develop Hypothesis*, and never eventually follows *Develop Method*. Additionally, the user group, as well as the corresponding probabilities, are different.

We can also model causal dependencies between event attributes by including more event attributes as explained before. For example, we could add a weekday attribute to model that *Main Author* only works Mondays and Tuesdays. This is realized by setting the probabilities for the other weekdays to zero.

We have described long-term control flow dependencies as well as data dependencies. By combining the two ideas, data to control flow dependencies are also possible, as *Research Related Work* demonstrates. *Develop Method* always directly follows *Research Related Work* if *Student* is the user.

We can generate event logs by using a random-walk through the likelihood graph, complying with the transition probabilities, and generating activities and attributes along the way. We implemented the generation algorithm so that all these dependencies can be controlled by parameters and the event attributes are automatically generated. Please refer to the code repository[1], and specifically the notebooks section, for a detailed description of the algorithm as well as examples.

In addition to the synthetic logs, we also use the real-life event logs from the Business Process Intelligence Challenge (BPIC): BPIC12[2], BPIC13[3], BPIC15[4], and BPIC17[5]. Furthermore, we use a set of 4 event logs (referred to by Anonymous) from real-life procurement processes provided by a consulting company.

1 https://github.com/tnolle/binet
2 http://www.win.tue.nl/bpi/doku.php?id=2012:challenge
3 http://www.win.tue.nl/bpi/doku.php?id=2013:challenge
4 http://www.win.tue.nl/bpi/doku.php?id=2015:challenge
5 http://www.win.tue.nl/bpi/doku.php?id=2017:challenge

Figure P4.3: Anomalies applied to cases of the paper submission process

4.2 *Artificial Anomalies*

Like Bezerra [3] and Böhmer [5], we apply artificial anomalies to the event logs, altering 30 percent of all cases with exactly one anomaly type. Inspired by the anomaly types (*Skip*, *Insert*, and *Switch*) used in [3, 5], we identified additional, more elaborate anomalies that frequently occur in real business processes. These anomalies are defined as follows.

- *Skip*: A sequence of up to 3 necessary events has been skipped
- *Insert*: Up to 3 random activities have been inserted
- *Rework*: A sequence of up to 3 events has been executed a second time
- *Early*: A sequence of up to 2 events has been executed too early, and hence is skipped later in the case
- *Late*: A sequence of up to 2 events has been executed too late, and hence is skipped earlier in the case
- *Attribute*: An incorrect attribute value has been set in up to 3 events

We apply the artificial anomalies to the real-life event logs as well, knowing that they likely already contain natural anomalies which are not labeled. However, we can measure the performance of the algorithms on the real-life logs to demonstrate feasibility while using the synthetic logs to evaluate accuracy.

As indicated in Fig. P4.3 we can gather a ground truth dataset by marking the attributes with their respective anomaly types. Note that we introduce a *Shift* anomaly type, which is used to indicate the place where an *Early* or *Late* event used to be. This is equivalent to a *Skip*. However, we want to differentiate these two cases.

We insert a random event in case of *Insert*, i.e., the activity name does not come from the original process. This is to prevent a random insert from resembling *Rework*, which is possible when randomly

Table P4.1: Overview showing dataset information

Name.	#Logs	#Activities	#Cases	#Events	#Attr.	#Attr. Values
Paper	1	27	5K	66K	1	13
P2P	4	27	5K	48K–53K	1–4	13–386
Small	4	41	5K	53K–57K	1–4	13–360
Medium	4	65	5K	39K–42K	1–4	13–398
Large	4	85	5K	61K–68K	1–4	13–398
Huge	4	109	5K	47K–53K	1–4	13–420
Gigantic	4	154–157	5K	38K–42K	1–4	13–409
Wide	4	68–69	5K	39K–42K	1–4	13–382
BPIC12	1	73	13K	290K	0	0
BPIC13	3	11–27	0.8K–7.5K	4K–81K	2–4	23–1.8K
BPIC15	5	422–486	0.8K–1.4K	46K–62K	2–3	23–481
BPIC17	2	17–53	31K–43K	284K–1.2M	1	289–299
Anonymous	4	19–37	968–17K	6.9K–82K	1	160–362

choosing an activity from the original process. Attribute values of inserted events are set in the same fashion. When applying an *Attribute* anomaly, we randomly select an attribute value from the likelihood graph that is not a direct successor of the last node.

We created datasets with ground truth data on attribute level. For the anomaly detection task, these labels are mapped to either *Normal* or *Anomaly*, thus creating a binary classification problem. The ground truth data can easily be adapted to case level by the following rule: A case is anomalous if any of the attributes in its events are anomalous.

We generated 4 likelihood graphs for each synthetic process model with different numbers of attributes, different transition probabilities, and dependencies. Then, we sampled logs from these likelihood graphs, resulting in 28 synthetic logs; 29, including the Paper dataset. Together with BPIC12, BPIC13, BPIC15, BPIC17, and Anonymous, the corpus consists of 44 event logs. We refer to the datasets by their names as defined in Table P4.1, which gives a detailed overview of the corpus.

5 METHOD

In this section, we describe the BINet architecture and how it is utilized for anomaly detection.

5.1 *Preprocessing*

Due to the mathematical nature of neural networks, we must transform the logs into a numerical representation. To accomplish this, we use integer encoding over all nominal attributes. An integer encoding is a mapping $\mathcal{I}_a : \mathcal{V}_a \to \mathbb{N}$ of all possible attribute values for an attribute

a to a unique positive integer. The integer encoding is applied to all attributes of the log, including the activity name.

Event logs can be represented as third-order tensors. Each event e is a first-order tensor $\boldsymbol{e} \in \mathbb{R}^A$, with $A = |\mathcal{A}|$, the first attribute always being the activity name, representing the control flow perspective. Hence, an event is defined by its activity name and the event attributes. Each case is then represented as a second-order tensor $\boldsymbol{C} \in \mathbb{R}^{E \times A}$, with $E = \max_{c \in \mathcal{L}} |\hat{c}|$ being the maximum case length of all cases in the log $\mathcal{L}$. For mathematical simplicity and to represent an event log as a tensor, we assume all cases to have the same length. Therefore, we pad all shorter cases with event tensors only containing zeros, which we call padding events. The padding events are ignored by the neural network during training and inference.

The log $\mathcal{L}$ can now be represented as a third-order tensor $\boldsymbol{L} \in \mathbb{R}^{C \times E \times A}$, where $C = |\mathcal{L}|$ is the number of cases in log $\mathcal{L}$. Using matrix index notation, we can now obtain the second attribute of the third event in the ninth case with $\boldsymbol{L}_{9,3,2}$. We can also obtain all the second attributes of the third event using $\boldsymbol{L}_{:,3,2}$, using ":" to denote the cross-section of tensor $\boldsymbol{L}$ along the case axis. Likewise, we can obtain all the second attributes of case nine with $\boldsymbol{L}_{9,:,2}$. Thus, we can define a preprocessor as follows.

Definition P4.2. *Preprocessor. Let* C, E, *and* A *be defined as above, then a preprocessor is a mapping* $\mathcal{P} : \mathcal{L} \to \mathbb{R}^{C \times E \times A}$.

The preprocessor $\mathcal{P}$ encodes all attribute values and then transforms the log $\mathcal{L}$ into its tensor representation. In the following, we refer to the preprocessed log $\mathcal{L}$ as $\boldsymbol{F}$ (features), where $\boldsymbol{F} = \mathcal{P}(\mathcal{L})$.

5.2 *BINet Neural Architecture*

To model the sequential nature of event log data, the core of BINet is a recurrent neural network, based on Gated Recurrent Units (GRUs) [8], an alternative to the popular LSTM. BINet processes the distinct sequence of events for each case. For each event, BINet has to predict the next event based on the history of events in the case.

Figure P4.4 shows the architecture of BINet. We propose three versions of BINet (BINetv1, BINetv2, and BINetv3). These versions differ in their capability of modeling causal dependencies based on the inputs they receive. BINet consists of dedicated encoder GRUs (light green) for each input attribute (light blue) of the last event. These GRUs are responsible for creating a latent representation of the complete history of a single attribute. Each attribute is fed through an embedding layer to reduce the input dimensionality (see [10, 22]).

The counterpart to the encoder GRUs are the decoder GRUs (dark green) which receive as input a concatenation of attribute representations. These GRUs are responsible for combining the control flow

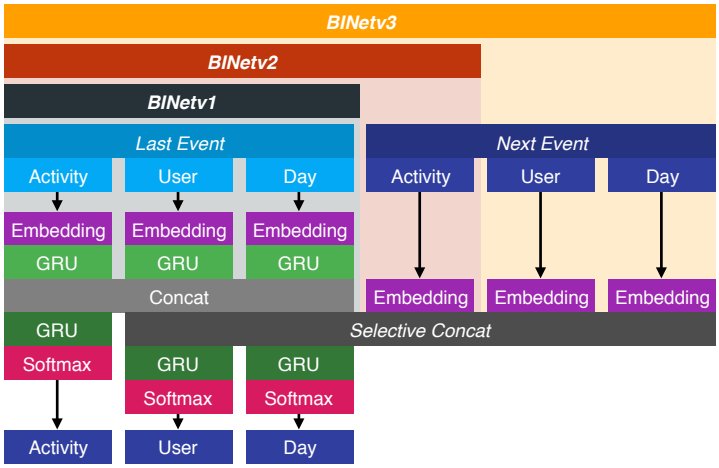

Figure P4.4: BINet architectures for a log with two event attributes, *User* and *Day*; the three versions of BINet differ only in the inputs they receive

and the data perspective, and hence to create a latent representation of the complete history of events that is meaningful to the respective next attribute prediction in the next layer. For prediction, BINet uses a softmax output layer (pink) for each next attribute. A softmax layer outputs a probability distribution over all possible values of the respective attribute, that is, all outputs will sum to one.

In its simplest form, BINet predicts the next activity and all next attributes solely based on past events. This architecture is called BINetv1 (black). However, there likely exist causal dependencies between the activity and the corresponding attribute values for that activity. To model this type of dependency, we propose a second architecture, BINetv2 (red), which in addition to the input of BINetv1, gets access to the activity of the next event. This conditions the attribute decoders onto the actual next activity, as opposed to inferring the next activity from the states of the encoders. Using the BINetv2 architecture, we can model control flow to data dependencies.

There also likely exist dependencies between attributes (i.e., certain users only work certain days of the week). With BINetv1 or BINetv2, these dependencies are not modeled because the attributes are treated as though they were independent. To address this, we propose BINetv3 (orange) which gets access to the complete next event. Hence, BINetv3 can model data to data dependencies.

The selective concatenation layer (dark grey) does not allow information to flow from one of the next event inputs to the respective next attribute decoder GRU. Otherwise, a decoder GRU would gain direct access to the information they are trained to predict, resulting in no learning at all because the decoder can simply copy the input.

To elaborate on the differences in the BINet versions we refer back to the paper submission process from Fig. P4.2. Suppose the last activity

input to BINet is *Research Related Work* and the user was *Main Author*. The activity output should now give a probability of approximately 60 percent to *Develop Hypothesis*. In case of BINetv1, however, the user output does not match the 80 percent for *Main Author* and the 20 percent for *Author*, because the respective decoders also have to take into account the other 40 percent of not going to *Develop Hypothesis*, and hence output a higher probability for *Student*. BINetv2 does not suffer from this problem, because it is certain that *Develop Hypothesis* is the next activity (because of the next activity input), and thus can learn the probabilities appropriately.

To demonstrate the advantage of BINetv3, we have to imagine a third weekday attribute as part of the extended likelihood graph. Suppose for a given activity *Main Author* works only on Fridays, and *Author* works from Monday until Thursday. BINetv3 can correctly predict that if the weekday is Friday, the user must be *Main Author*, whereas BINetv2 can not. Likewise, BINetv3 can infer that if the user is *Main Author*, the day must be Friday.

For BINetv1, activity, user, and day are entirely independent, for BINetv2, user and day are dependent on the activity, and for BINetv3, user is dependent on activity and day, whereas day is dependent on activity and user. Our implementation of BINet theoretically allows for any number of events and attributes.

BINetv2 is equivalent to the original BINet architecture from [26].

5.3 *Calculating Anomaly Scores*

After the initial training phase, BINet can be used for anomaly detection. This is based on the idea that BINet assigns a lower probability to an anomalous attribute than a normal attribute.

The last step of the anomaly detection process is the scoring of the events. Therefor, we use a scoring function in the last layer of the architecture. This scoring function for an attribute a receives as input the output of the softmax layer for a, that is, a probability distribution $\boldsymbol{p}_a$, and the actual value of the attribute, v.

Using the example above (the last activity being *Research Related Work* and the user being *Main Author*), the output of the activity softmax layer might look as depicted in Fig. P4.5. The probability p for *Develop Hypothesis* is 0.55, and the probability for *Develop Method* is 0.30. Note that BINet gives a high probability for the two correct next activities with respect to the paper process.

We can now define the anomaly score for a possible attribute value v as the sum of all probabilities p of the probability distribution tensor $\boldsymbol{p}$ greater than the probability assigned to v, p_v. The scoring function σ is therefore defined as follows, with $\boldsymbol{p}_i$ being i-th probability.

$$\sigma(\boldsymbol{p}, p_v) = \sum_{\boldsymbol{p}_i > p_v} \boldsymbol{p}_i$$

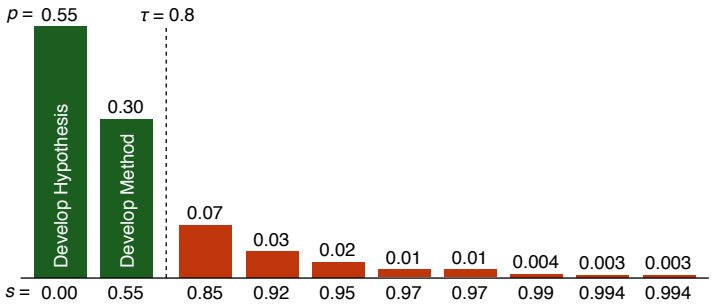

Figure P4.5: Output of the activity softmax layer after reading activity *Research Related Work* and user *Main Author*

Figure P4.5 also shows the resulting anomaly scores, s, for each possible activity. Intuitively, an anomaly score of 0.55 indicates that there is a 55 percent chance that the attribute value is anomalous. Thus, we can set a threshold as indicated in Fig. P4.5 ($\tau = 0.8$), to flag all values as normal where the anomaly score is less than 0.8.

The scoring function σ is applied to each softmax output of BINet, transforming the probability distribution tensor into a scalar anomaly score. We can now obtain the anomaly scores tensor S by applying BINet to the feature tensor $\boldsymbol{F}$,

$$\boldsymbol{S} = (s_{ijk}) \in \mathbb{R}^{C \times E \times A} = \text{BINet}(\boldsymbol{F}),$$

mapping an anomaly score to each attribute in each event in each case. The anomaly score for attributes of padding events is always 0.

5.4 *Training*

BINet is trained without the scoring function. The GRU units are trained in sequence to sequence fashion. With each event that is fed in, the network is trained to predict the attributes of the next event. We train BINet with a GRU size of $2E$ (two times the maximum case length), on mini batches of size 500 for 20 epochs using the Adam optimizer with parameters as stated in the original paper [19]. Additionally, we use batch normalization [17] after each GRU to counteract overfitting.

5.5 *Detection*

An anomaly detector only outputs anomaly scores. We need to define a function that maps anomaly scores to a label $l \in \{0, 1\}$, 0 indicating normal and 1 indicating anomalous, by applying a threshold τ. Whenever an anomaly score for an attribute is greater than τ, this attribute

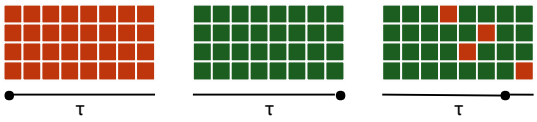

Figure P4.6: Example of how an anomaly detection visualization changes with different threshold settings; the rightmost setting corresponds to how a user would likely set the slider manually

is flagged as anomalous. Therefore, we define a threshold function θ, with inputs $\boldsymbol{S}$ and $\tau \in \mathbb{R}$.

$$\theta(\boldsymbol{S}, \tau) = \boldsymbol{Y}_{ijk} = \begin{cases} 1 & \text{if } \boldsymbol{S}_{ijk} > \tau \\ 0 & \text{otherwise} \end{cases}$$

In the example from Fig. P4.5, setting $\tau = 0.8$ results in *Develop Hypothesis* and *Develop Method* being flagged as normal, whereas all other activities are flagged as anomalous.

5.6 *Threshold Heuristic*

Most anomaly detection algorithms rely on the user manually setting a threshold or define the threshold as a constant. To determine a threshold automatically, we propose a new heuristic that mimics how a human would set a threshold manually.

Let us consider the following example. A user is presented with a visualization of an anomaly detection result, for instance, a simple case overview showing all events and their attributes as depicted in Fig. P4.6. Anomalous attributes are shown in red and normal attributes are shown in green. The user is asked to set the threshold manually using a slider. Most people start with the slider either set to the maximum (all attributes are normal, all green) or the minimum (all attributes are anomalous, all red) and then move the slider while observing the change of colors in the visualization. Intuitively, most users fix the slider within a region where the number of shown anomalies is stable, that is, even when moving the slider to the left and right, the visualization stays the same. Furthermore, users likely prefer a threshold setting that shows significantly less anomalous than normal attributes, which corresponds to a slider setting closer to the maximum (the right side). In other words, a setting that produces less false positives.

This behavior of a human setting the threshold can be modeled based on the anomaly ratio r, which can be defined as follows, with $N = \sum_{c \in \mathcal{L}} |\hat{c}|$ denoting the number of non-padding events and $\boldsymbol{Y} = \theta(\boldsymbol{S}, \tau)$.

$$r(\theta, \boldsymbol{S}, \tau) = \frac{1}{NA} \sum_{i}^{C} \sum_{j}^{E} \sum_{k}^{A} \boldsymbol{Y}_{ijk}$$

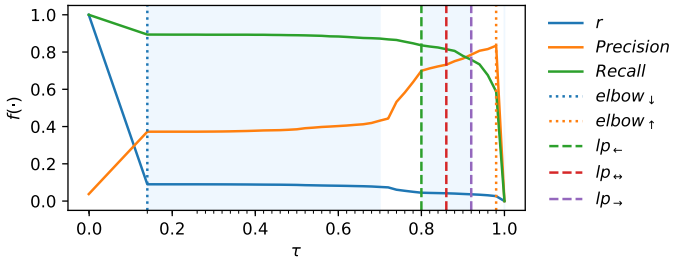

Figure P4.7: Thresholds as defined by the heuristics in relation to the anomaly ratio r and its plateaus (blue intervals)

By dividing by N we calculate the average based only on non-padding events.

Figure P4.7 shows r for a run of BINetv1 on the Paper dataset. In addition to r, the figure shows the values for `Precision` and `Recall` for the anomaly class. Note that r is a discrete function that we sample for each reasonable threshold. Reasonable candidate thresholds are the distinct anomaly scores encountered in $\boldsymbol{S}$; other values can be disregarded. The candidate thresholds $\mathcal{T}$ are indicated by the minor ticks in Fig. P4.7.

To define the heuristics in the following, we first have to define the first and second order derivatives of the discrete function r. They can be retrieved using the central difference approximation. Let $\tau_i \in \mathcal{T}$, then the derivatives are approximated by

$$r'(\theta, \boldsymbol{S}, \tau_i) \approx \frac{r(\theta, \boldsymbol{S}, \tau_{i+1}) - r(\theta, \boldsymbol{S}, \tau_i)}{\tau_{i+1} - \tau_i},$$
$$r''(\theta, \boldsymbol{S}, \tau_i) \approx \frac{r(\theta, \boldsymbol{S}, \tau_{i-1}) - 2\, r(\theta, \boldsymbol{S}, \tau_i) + r(\theta, \boldsymbol{S}, \tau_{i-1})}{(\tau_{i-1} - \tau_i)(\tau_i - \tau_{i+1})}.$$

To mimic the human intuition, we have to consider regions of r where the slope is close to zero. These are regions where $|r'(\theta, \boldsymbol{S}, \tau)| < \varepsilon$ (we chose ε to be two times the average slope of r), which we refer to as plateaus (blue regions in Fig. P4.7). Based on these plateaus we can now define the lowest plateau heuristic lp as follows. Let $\mathrm{LP} = \langle \tau_0, \tau_1, \ldots, \tau_n \rangle$ be the sequence of candidate thresholds that lie within the lowest plateau, then

$$\mathrm{lp}_{\leftarrow} = \tau_0, \quad \mathrm{lp}_{\leftrightarrow} = \frac{1}{n} \sum_{i=0}^{n} \tau_i, \quad \mathrm{lp}_{\rightarrow} = \tau_n,$$

corresponding to the left-most ($\mathrm{lp}_{\leftarrow}$), the right-most ($\mathrm{lp}_{\rightarrow}$), and the mean-centered ($\mathrm{lp}_{\leftrightarrow}$) threshold inside the lowest plateau.

In [26], we proposed the elbow heuristic to mimic the same behavior. The definition of the elbow heuristics are given by

$$\mathrm{elbow}_{\downarrow} = \arg\max_{\tau \in \mathcal{T}} r''(\theta, \boldsymbol{S}, \tau), \quad \mathrm{elbow}_{\uparrow} = \arg\min_{\tau \in \mathcal{T}} r''(\theta, \boldsymbol{S}, \tau).$$

So, $elbow_{\downarrow}$ is the threshold where the rate of change of r is maximized, whereas $elbow_{\uparrow}$ is where it is minimized. With respect to r these thresholds are the points where either a steep drop ends in a plateau ($elbow_{\downarrow}$) or a plateau ends in a steep drop ($elbow_{\uparrow}$). Although $elbow_{\downarrow}$ and $elbow_{\uparrow}$ can indicate the beginning and the end of a plateau, these are not necessarily the thresholds a human would naturally pick.

To compare our results to the best possible threshold, we define the $best$ heuristic by use of the F_1 score metric. The $best$ heuristic is defined as follows, where $\boldsymbol{L}$ is the set of ground truth labels.

$$best = \arg\max_{\tau \in \mathcal{T}} F_1(\boldsymbol{L}, \theta(\boldsymbol{S}, \tau)).$$

It is important to understand that $best$ can only be used if the labels are available at runtime. However, in most cases, anomaly detection is an unsupervised problem, and hence no labels are available.

It might be beneficial to apply different thresholds to different dimensions of $\boldsymbol{S}$. For example, it might be sensible to set a different threshold for the user attribute than the activity because the inherent probability distribution can be different. This is possible by using ":" to apply heuristics on cross-sections of $\boldsymbol{S}$ by using index notation. Let $h \in \{lp_{\leftarrow}, lp_{\leftrightarrow}, lp_{\rightarrow}, elbow_{\downarrow}, elbow_{\uparrow}, best\}$ then we can define the following threshold strategies

$$\begin{aligned} h &= h(\boldsymbol{S}), \\ h^{(a)} &= (\tau_i) = h(\boldsymbol{S}_{:,:,i}), \\ h^{(e)} &= (\tau_i) = h(\boldsymbol{S}_{:,i,:}), \\ h^{(ea)} &= (\tau_{ij}) = h(\boldsymbol{S}_{:,i,j}). \end{aligned}$$

We only explicitly show the parameter $\boldsymbol{S}$ for clarity, other parameters are set according to the definition of the chosen heuristic.

Thus, $h^{(a)} \in \mathbb{R}^A$ returns a tensor that holds one threshold for each attribute in an event, whereas $h^{(e)} \in \mathbb{R}^E$ holds a threshold for each event position in a case. Lastly, $h^{(ea)} \in \mathbb{R}^{E \times A}$ combines the two ideas and gives a threshold for each combination of event position and attribute. In other words, instead of applying the threshold heuristic h once for all dimensions of $\boldsymbol{S}$, we apply it multiple times for different cross-sections of $\boldsymbol{S}$, obtaining multiple different thresholds.

6 EVALUATION

We evaluated BINet on all 44 event logs and compared it to eight state-of-the-art methods. Two methods from [7]: a sliding window approach (t-STIDE+) [30]; and the one-class SVM (OC-SVM). Two methods from [3]: the Naive algorithm and the Sampling algorithm. Furthermore, we provide the results of the denoising autoencoder

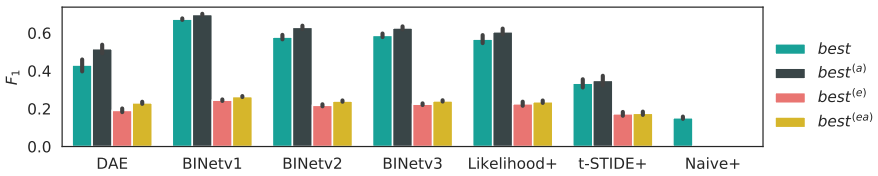

Figure P4.8: Average F_1 score by method and strategy over all synthetic datasets, using $best$ as the heuristic

(DAE) approach from [24]. Lastly, we compared BINet to the approach from [5] (Likelihood). Naive and Likelihood set the threshold statically, so we extended the approaches to support the use of our external threshold heuristics. These extensions are referred to by Naive+ and Likelihood+. For all non-deterministic methods (i.e., DAE, BINet, and Sampling), we executed five independent runs to account for randomness.

For the OC-SVM, we relied on the implementation of scikit-learn[6] using an RBF kernel of degree 3 and $\nu = 0.5$. The Naive, Sampling, Likelihood, and DAE methods were implemented as described in the original papers. Sampling, Likelihood, Baseline, and the OC-SVM do not rely on a manual setting of the threshold and were unaltered. t-STIDE+ is an implementation of the t-STIDE method from [30], which we adapted to support the data perspective (see [24]). Naive+ is an implementation of Naive that removes the fixed threshold of 0.02 and sets the threshold according to the heuristic. Likelihood+ implements the first part of Likelihood (the generation of the extended likelihood graph from the log) and replaces the threshold algorithm with the aforementioned heuristics.

In the last section, we described the intuition of setting separate thresholds using different strategies (e.g., one threshold per attribute). To decide on the best strategy, we evaluate the four strategies (h, $h^{(e)}$, $h^{(a)}$, and $h^{(ea)}$) for all synthetic datasets and all methods that support the heuristics, with $h = best$. The results of the experiments in Fig. P4.8 indicate that, indeed, it is sensible to set separate thresholds for individual attributes. Interestingly, we also find that setting a single threshold yields similar results. Setting a threshold per event or per event and attribute does perform significantly worse.

Next, we repeated the same experiment for all of the aforementioned heuristics and using $h^{(a)}$ as the strategy. The results can be seen in Fig. P4.9. Intriguingly, the lowest plateau heuristics perform best for all methods except the DAE. Furthermore, it seems to work best to choose the mean-centered threshold within the lowest plateau ($lp_{\leftrightarrow}$).

Based on the results of the preliminary experiments, we set $h = lp_{\leftrightarrow}^{(a)}$ as the heuristic for the following experiments for all methods apart from the DAE, for which we set $h = elbow_{\uparrow}^{(a)}$. For Likelihood,

6 http://scikit-learn.org

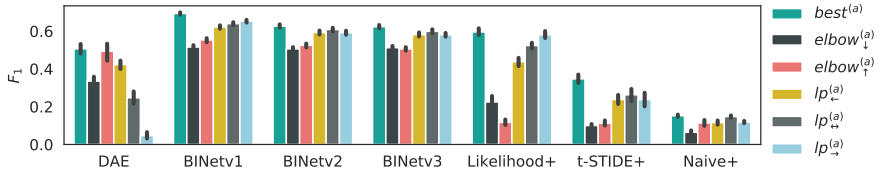

Figure P4.9: Average F_1 score by method and heuristic over all synthetic datasets, using $h^{(a)}$ as the strategy

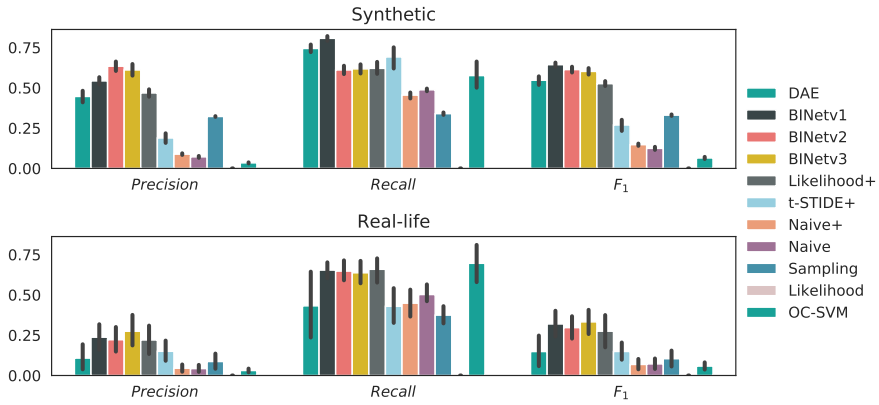

Figure P4.10: Average Precision, Recall, and F_1 by dataset type over all datasets; error bars indicate variance over datasets with different numbers of attributes and multiple runs

Sampling, Naive, and OC-SVM we use the internal threshold heuristics.

The overall results are shown in Fig. P4.10. For the real-life datasets, we do not have complete information, and hence the F_1 score is not a good representation of the quality of the detection algorithms. However, because we compare all methods on the same basis, the results are still meaningful. Furthermore, we only know about the artificial anomalies inside the real-life datasets, and therefore we expect a high recall of the artificial anomalies, whereas we expect a low precision because the dataset likely contains natural anomalies which are not labeled.

This theory is confirmed by the results in Fig. P4.10. Note also that the recall scores for both the synthetic and the real-life datasets are very similar, indicating comparable performance (for artificial anomalies) on both dataset types.

Finally, we find that BINetv1 works best for the synthetic datasets, whereas the field is mixed for the real-life datasets. However, all three BINet versions perform better than the other methods. DAE performs significantly worse on real-life data because it ran out of memory on some of the bigger datasets. Therefore, DAE has been penalized defining precision and recall to be zero for these runs.

Table P4.2: F_1 score over all datasets by detection level and method; best results (before rounding) are shown in bold typeface

Level	Method	Paper	P2P	Small	Medium	Large	Huge	Gigantic	Wide	BPIC12	BPIC13	BPIC15	BPIC17	Anonymous
Case	OC-SVM [30]	0.49	0.27	0.25	0.29	0.24	0.23	0.29	0.31	0.55	0.24	0.26	0.35	0.10
	Naive [4]	0.50	0.48	0.49	0.39	0.41	0.40	0.34	0.44	0.55	0.21	0.17	0.31	0.16
	Sampling [4]	0.50	0.49	0.49	0.47	0.49	0.49	0.45	0.49	0.55	0.21	0.17	0.32	0.23
	Likelihood [5].	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Naive+	0.50	0.48	0.49	0.44	0.49	0.45	0.38	0.47	0.55	0.21	0.17	0.28	0.15
	t-STIDE+ [25]	0.40	0.51	0.53	0.43	0.45	0.45	0.41	0.47	**0.68**	0.32	0.29	0.32	0.22
	Likelihood+	**0.85**	0.74	0.76	0.72	0.73	0.73	0.73	0.73	0.62	0.44	0.33	0.45	0.51
	DAE [25]	0.46	0.71	0.72	0.71	0.71	0.70	0.63	0.70	0.60	0.21	0.00	0.30	0.35
	BINetv1	0.74	**0.77**	**0.78**	**0.75**	**0.75**	**0.75**	**0.74**	**0.76**	0.62	0.41	0.37	**0.51**	**0.51**
	BINetv2 [26]	0.76	0.77	0.77	0.72	0.71	0.70	0.68	0.73	0.61	0.40	**0.38**	0.43	0.45
	BINetv3	0.79	0.77	0.76	0.71	0.69	0.69	0.66	0.74	0.66	**0.45**	0.36	0.49	0.50
Attribute	OC-SVM [30]	0.09	0.06	0.05	0.08	0.04	0.05	0.07	0.09	0.05	0.06	0.01	0.09	0.30
	Naive [4]	0.13	0.15	0.14	0.12	0.09	0.11	0.09	0.16	0.05	0.05	0.01	0.10	0.39
	Sampling [4]	0.33	0.33	0.34	0.32	0.34	0.34	0.31	0.32	0.08	0.07	0.01	0.14	0.39
	Likelihood [5]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Naive+	0.13	0.16	0.15	0.16	0.13	0.13	0.13	0.18	0.05	0.05	0.01	0.09	0.33
	t-STIDE+ [25]	0.28	0.33	0.32	0.25	0.25	0.26	0.19	0.28	0.40	0.12	0.05	0.17	0.39
	Likelihood+	**0.74**	0.61	0.63	0.58	0.57	0.58	0.55	0.57	0.35	0.29	0.07	0.28	0.63
	DAE [25]	0.25	0.61	0.61	0.56	0.56	0.55	0.46	0.56	0.06	0.09	0.00	0.24	0.52
	BINetv1	0.64	**0.68**	**0.69**	**0.65**	**0.65**	**0.65**	**0.64**	**0.65**	0.42	0.28	**0.21**	**0.38**	0.60
	BINetv2 [26]	0.67	0.65	0.67	0.60	0.59	0.60	0.56	0.60	0.34	0.25	0.19	0.29	0.55
	BINetv3	0.67	0.65	0.66	0.59	0.57	0.59	0.54	0.61	**0.48**	**0.29**	0.19	0.35	**0.63**

Detailed results can be found in Tab. P4.2, which also gives results for case level (i.e., only anomalous cases have to be detected, not the attributes). An interesting observation is that t-STIDE+ performs best on BPIC12 when evaluating on case level. This might be attributed to BPIC12 being a dataset without event attributes (the only one in the corpus). On attribute level, Likelihood+ is marginally better than BINet on BPIC13. For all other datasets, BINet shows the best performance.

All results are given using the heuristics described above. Labels were not used in the process. Additional material (e.g., evaluation per perspective, per dataset, runtime) can be found in the code repository.

To validate the significance of the results, we apply the non-parametric Friedman test [14] on average ranks of all methods based on F_1 score for all synthetic datasets. Then, we apply the Nemenyi post-hoc test [23], as demonstrated in [11], to calculate pairwise significance.

Figure P4.11 shows a critical difference diagram, as proposed in [11], to visualize the results with a confidence interval of 95 percent. Based on the critical difference, we recognize that BINetv1 performs significantly better than all other methods, except BINetv2 and BINetv3. That is, all three BINet versions lie in the same significance group with respect to the critical difference. DAE lies in the same group as BINetv2 and BINetv3, and Likelihood+ in the same as DAE and BINetv3. All other methods lie more than the critical difference away from the three BINets.

7 CLASSIFYING ANOMALIES

Until now, we have not utilized the predictive capabilities of BINet. Using the probability distribution output of the softmax layers in conjunction with the binarized anomaly scores, we can define simple rules to infer the type of anomaly.

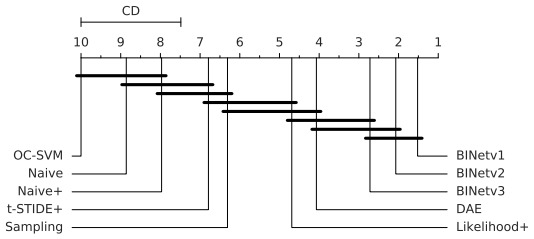

Figure P4.11: Critical difference diagram for all methods on all synthetic datasets; groups of methods that are not significantly different (at $p = 0.05$) are connected (cf. [11])

We use the term predictions to denote all possible attribute values with an anomaly score above the threshold. Here, the *Shift* class becomes relevant because it indicates the place where an early or late execution would belong. For each anomalous attribute (according to BINet), we apply the following rules in order.

1. *Skip*: If all predictions do not appear in the case
2. *Insert*: If one of the predictions appears in the case and that occurrence has not been flagged as anomalous
3. *Rework*: If the same activity appears earlier in the case and is not flagged as anomalous
4. *Shift*: If one of the predictions appears earlier or later in the case and is flagged as anomalous
5. *Late*: If the activity appears earlier in the predictions and is flagged as anomalous
6. *Early*: If the activity appears later in the predictions and is flagged as anomalous
7. *Attribute*: Trivially, all anomalous attributes in the data perspective are of type *Attribute*

The result of the classification is visualized in Fig. P4.12. Interestingly, this set of simple rules performs remarkably well. Anomaly classes inferred by the rules are indicated by the color of the cells, whereas ground truth labels are shown as text in the cells (we omit *Normal* for clarity). Incidentally, this visualization also depicts the binarized anomaly scores according to the threshold since each classified attribute is also an anomalous attribute.

We also included some examples where the classification is incorrect. An interesting case is the second *Skip* example because *Evaluate* has also been marked as anomalous ①. As we have defined in Fig. P4.2, *Evaluate* is an activity that always eventually follows *Develop Method*. However, *Develop Method* was skipped. Therefore, BINet is

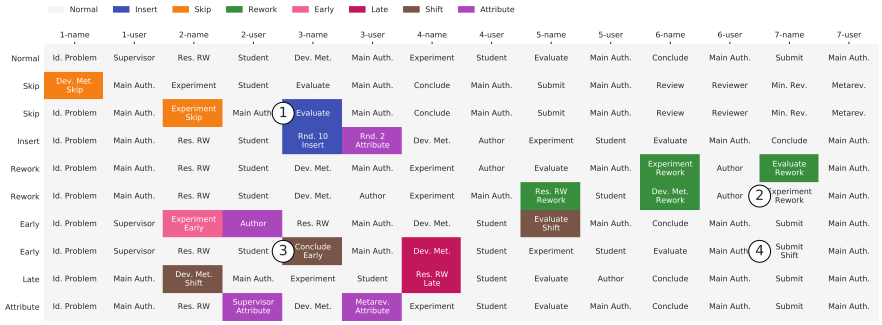

Figure P4.12: Classification of anomalies on the Paper dataset based on anomaly scores from BINetv1 using $h = lp^{(a)}_{\rightarrow}$; colors indicate the prediction of the classifier (see legend) and actual classes are shown as text within the cells

never presented with the causing activity, and hence regards *Evaluate* as anomalous.

A different example is that BINet misses the third *Rework* activity in the second example ②. We observed many of these errors, and they are related to the fact that BINet is conditioned on the last input activity and forgets the history of the case (forgetting problem). Under these conditions, *Develop Method* indeed directly follows *Research Related Work*, and hence BINet misses it. This forgetting problem is something we want to address in the future.

The most interesting case is the second *Early* example ③. Here, BINet misclassifies the *Early* activity as *Shift*. Upon closer inspection, we realize that this is indeed a way of explaining the anomaly, albeit not the one the labels indicate. With respect to *Develop Method*, *Conclude* indeed occurs too early in the case. Nevertheless, BINet fails to detect the actual Shift point ④, and thus the rules do not match the pattern correctly.

Using this simple set of rules, we ran the classifier on all synthetic datasets, using BINetv1 as the anomaly detection method and $h = lp^{(a)}_{\rightarrow}$ (the best heuristic for BINetv1). Figure P4.13 shows the results in a confusion matrix. Note that the classifier uses as input the anomaly detection result of BINet, and hence can never distinguish normal from anomalous examples. Thus, the errors for the normal class are based on the errors BINet commits in the anomaly detection task. Disregarding these errors, this results in a macro average F_1 score of 0.83 over all datasets for the classification task. Since BINetv1 reaches an average F_1 score of 0.64 on the detection task, this result is truly impressive, considering the simplicity of the rules. For the joint task (detection and classification), BINetv1 reaches an average F_1 score of 0.70.

In Fig. P4.13, we notice that BINet errs especially often for *Rework*, *Early*, *Late*, and *Shift*. This is connected to the forgetting problem mentioned earlier. Remember that *Rework*, *Early*, and *Late* anomalies

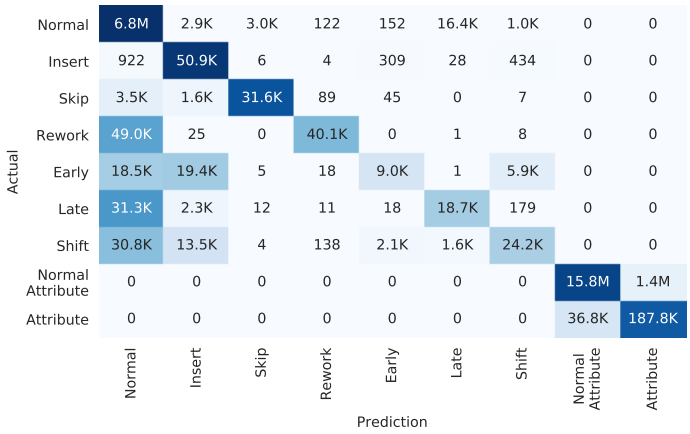

Figure P4.13: Confusion matrix for all runs of BINetv1 on synthetic datasets with $h = \mathrm{lp}^{(a)}_{\rightarrow}$; color indicates distribution of actual class

are affecting sequences of events, that is, up to 3 events can be part of a rework anomaly, and up to 2 events can be executed early or late. In the case of Rework, we have already seen an example in Fig. P4.12, where BINet misclassifies because of forgetting. Figure P4.13 confirms that this error occurs quite often since more than 50 percent of all *Rework* anomalies are misclassified as *Normal*. All of these misclassifications happen in cases where a sequence of more than one event has been executed again.

Not every repetition of an event is classified as a *Rework*, only the events identified to be anomalous are. Hence, a repeated event (a loop in the process) is classified as *Normal*, if BINet has learned that it can occur multiple times in a case. In the Paper process, this is demonstrated by the second *Submit* event, which can naturally occur multiple times in a case. In Fig. P4.13 we can see that BINet very rarely classifies a *Normal* activity as *Rework* (never in the Paper datasets); thus, we can conclude that BINet has learned to model the loop in the Paper process correctly.

As with *Rework*, we can explain the errors for *Early* and *Late* by the same argument. However, these two classes are also often misclassified as *Insert* or *Shift*. The latter goes back to the second *Early* example of Fig. P4.12 and the ambiguity of labels. The *Insert* errors are of a different kind. They occur because the rule set is not taking into account that multiple events can be executed early or late. We expect to find an early execution somewhere later in the case as the prediction; however, this can only be true for the first event of an early sequence. The same argument can be made for late executions.

The *Shift* errors are related to the fact that the random process models often allow skipping of events. When a *Shift* anomaly is applied to an optional event, BINet, or any other method, has no means of

finding the anomaly. This could be accounted for by altering the generation algorithm.

Nevertheless, the results indicate that a simple set of rules can be used to classify the anomalies types we have introduced before. Note that this is a white-box approach and a human user can easily interpret the resulting classification. Even though the different classes are only a subset of all anomaly types, they do cover many of the anomalies encountered in real-life business processes. Importantly, it is quite easy to define new rules for new types of anomalies based on the predictive capabilities of BINet.

8 DISCUSSION

As part of the discussion, we would like to address three matters: The performance on the real-life datasets, the differences between the BINet versions, and the difference in datasets compared to the original publication [26].

Firstly, the difference in performance when comparing synthetic and real-life datasets seems substantial. Yet, the real-life datasets contain natural anomalies that are not labeled and therefore incorrectly represent normal behavior. If BINet correctly detects these anomalies, this is considered to be wrong because the labels indicate otherwise. The results indicate that BINet detects most of the artificial anomalies, which leads to high recall scores. These recall scores are similar to the recall scores reached on the synthetic sets, suggesting that BINet performs equally good on both the synthetic and the real-life datasets. Conversely, we can observe low precision scores on the real-life data. BINet detects natural anomalies that are labeled as normal, thereby affecting the precision. Based on our domain knowledge about the BPIC datasets, we believe most of these misclassified anomalies to be actual anomalies that BINet correctly found, but the lack of labels compromises the precision score.

Secondly, it appears that BINetv1 outperforms BINetv2 and BINetv3. We would have expected a different outcome because BINetv2 and BINetv3 can model causal dependencies. BINetv1 performs particularly well on the synthetic data. We suspect that this is attributed to the generation algorithm and the number of event attributes in the datasets. The design of BINetv2 and BINetv3 is targeted at improving the predictions of event attributes. This design can only be utilized if more than one event attribute exists. In some datasets, this was not the case. However, for the real-life datasets, where more event attributes are available, BINetv3 performs best. Additionally, it appears that for the synthetic datasets the historical information of past events is sufficient to accurately predict the next event. This is not the case for the real-life datasets. In the future, we intend to address this by altering the generation algorithm to generate stronger causal dependencies.

Thirdly, the total number of datasets used in this work is different from the number used in [26]. In [26] we generated multiple redundant logs from the same randomly generated process model. However, we found that a higher number of datasets did not affect the overall evaluation results. Instead, we decided to include more sophisticated anomalies, resulting in a more challenging, diverse, and realistic corpus, albeit a smaller one.

9 CONCLUSION

In this paper, we presented three versions of BINet, a neural network architecture for multi-perspective anomaly classification in business process event logs. Additionally, we proposed a set of heuristics for setting the threshold of an anomaly detection algorithm automatically, based on the anomaly ratio function. Finally, we demonstrated that a simple set of rules could be used for classification of anomaly types, solely based on the output of BINet.

BINet is a recurrent neural network, and can, therefore, be used for real-time anomaly detection, since it does not require a completed case for detection. BINet does not rely on any information about the process, nor does it depend on a clean dataset. Utilizing the lowest plateau heuristic, BINet's internal threshold can be set automatically, reducing manual workload and allowing fully autonomous operation. It utilizes both the control flow and the data perspective. Furthermore, BINet can cope with concept drift, for it can be set up to train on new cases in real-time continuously.

Based on the empirical evidence obtained in the evaluation, BINet is a promising method for anomaly detection, especially in business process event logs. BINet outperformed the opposition on all detection levels. Specifically, on the synthetic datasets, BINet's performance surpasses those of other methods by an order of magnitude. We demonstrated that BINet also performs well on the real-life datasets because BINet shows high recall of the artificial anomalies introduced to the original real-life logs.

Although the results look very promising, there is still room for improvement. For example, BINet suffers from forgetting when sequences of events are repeated in a case. This issue can be addressed in future work, for example, by using a special attention layer. An interesting option is the use of a bidirectional encoder-decoder structure to read in cases both from left to right and from right to left. Hereby, sequences of repeated events can be identified from two sides, as opposed to just one.

Overall, the results presented in this paper suggest that BINet is a reliable and versatile method for detecting—and classifying—anomalies in business process event logs.

ACKNOWLEDGMENTS

This project [522/17-04] is funded in the framework of Hessen ModellProjekte, financed with funds of LOEWE, Förderlinie 3: KMU-Verbundvorhaben (State Offensive for the Development of Scientific and Economic Excellence), and by the German Federal Ministry of Education and Research (BMBF) Software Campus project "R2PA" [01IS17050].

REFERENCES

[1] Wil M. P. van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016 (cit. on pp. 108, 110, 112).

[2] Fábio Bezerra and Jacques Wainer. "Anomaly Detection Algorithms in Logs of Process Aware Systems." In: *Proceedings of the 23rd Annual ACM Symposium on Applied Computing – SAC '08*. 2008, pp. 951–952 (cit. on p. 112).

[3] Fábio Bezerra and Jacques Wainer. "Algorithms for Anomaly Detection of Traces in Logs of Process Aware Information Systems." In: *Information Systems* 38.1 (2013), pp. 33–44 (cit. on pp. 116, 124).

[4] Fábio Bezerra, Jacques Wainer, and Wil M. P. van der Aalst. "Anomaly Detection Using Process Mining." In: *Proceedings of the 10th International Workshop on Enterprise, Business-Process and Information Systems Modeling – BPMDS'09*. Springer, 2009, pp. 149–161 (cit. on pp. 112, 127).

[5] Kristof Böhmer and Stefanie Rinderle-Ma. "Multi-perspective Anomaly Detection in Business Process Execution Events." In: *Proceedings of On the Move to Meaningful Internet Systems, OTM'16*. Springer. 2016, pp. 80–98 (cit. on pp. 112, 114, 116, 125, 127).

[6] Andrea Burattin. "PLG2: Multiperspective Processes Randomization and Simulation for Online and Offline Settings." In: *arXiv preprint arXiv:1506.08415* (2015) (cit. on p. 114).

[7] V. Chandola, A. Banerjee, and V. Kumar. "Anomaly Detection for Discrete Sequences: A Survey." In: *IEEE Transactions on Knowledge and Data Engineering* 24.5 (2012), pp. 823–839 (cit. on pp. 112, 113, 124).

[8] Kyunghyun Cho et al. "Learning Phrase Representations Using RNN Encoder-decoder for Statistical Machine Translation." In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on p. 118).

[9] Corinna Cortes and Vladimir Vapnik. "Support-vector Networks." In: *Machine learning* 20.3 (1995), pp. 273–297 (cit. on p. 113).

[10] Pieter De Koninck, Seppe vanden Broucke, and Jochen De Weerdt. "act2vec, trace2vec, log2vec, and model2vec: Representation Learning for Business Processes." In: *Proceedings of the 16th International Conference on Business Process Management – BPM'18*. Springer, 2018, pp. 305–321. ISBN: 978-3-319-98648-7 (cit. on p. 118).

[11] Janez Demšar. "Statistical Comparisons of Classifiers Over Multiple Data Sets." In: *Journal of Machine Learning Research* 7.Jan (2006), pp. 1–30 (cit. on pp. 127, 128).

[12] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "A Deep Learning Approach for Predicting Process Behaviour at Runtime." In: *Proceedings of the 14th International Conference on Business Process Management – BPM'16*. Springer. 2016, pp. 327–338 (cit. on p. 113).

[13] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. "Predicting Process Behaviour Using Deep Learning." In: *Decision Support Systems* 100 (2017), pp. 129–140 (cit. on p. 113).

[14] Milton Friedman. "The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance." In: *Journal of the American Statistical Association* 32.200 (1937), pp. 675–701 (cit. on p. 127).

[15] Jiawei Han, Jian Pei, and Micheline Kamber. *Data Mining: Concepts and Techniques*. Elsevier, 2011 (cit. on p. 108).

[16] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory." In: *Neural Computation* 9.8 (1997), pp. 1735–1780 (cit. on pp. 111, 113).

[17] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *Proceedings of the 32nd International Conference on Machine Learning – ICML'15*. 2015, pp. 448–456 (cit. on p. 121).

[18] Nathalie Japkowicz. "Supervised Versus Unsupervised Binary-Learning by Feedforward Neural Networks." In: *Machine Learning* 42.1 (2001), pp. 97–122 (cit. on p. 113).

[19] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 121).

[20] Pankaj Malhotra et al. "LSTM-based encoder-decoder for multi-sensor anomaly detection." In: *arXiv preprint arXiv:1607.00148* (2016) (cit. on p. 113).

[21] Erik Marchi, Fabio Vesperini, Florian Eyben, Stefano Squartini, and Björn Schuller. "A Novel Approach for Automatic Acoustic Novelty Detection Using a Denoising Autoencoder with Bidirectional LSTM Neural Networks." In: (Apr. 2015) (cit. on p. 113).

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient Estimation of Word Representations in Vector Space." In: *arXiv preprint arXiv:1301.3781* (2013) (cit. on p. 118).

[23] Peter Nemenyi. "Distribution-Free Multiple Comparisons." PhD thesis. Princeton University, 1963 (cit. on p. 127).

[24] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. "Analyzing Business Process Anomalies Using Autoencoders." In: *Machine Learning* 107.11 (Nov. 2018), pp. 1875–1893 (cit. on pp. 113, 125).

[25] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "Unsupervised Anomaly Detection in Noisy Business Process Event Logs Using Denoising Autoencoders." In: *Proceedings of the 19th International Conference on Discovery Science – DS'16*. Springer. 2016, pp. 442–456 (cit. on pp. 113, 127).

[26] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "BINet: Multivariate Business Process Anomaly Detection Using Deep Learning." In: *Proceedings of the 16th International Conference on Business Process Management – BPM'18*. 2018, pp. 271–287 (cit. on pp. 109, 114, 120, 123, 127, 131, 132).

[27] Marco A. F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. "A Review of Novelty Detection." In: *Signal Processing* 99 (2014), pp. 215–249 (cit. on p. 112).

[28] Bernhard Schölkopf, Robert C. Williamson, Alexander J. Smola, John Shawe-Taylor, John C. Platt, et al. "Support Vector Method for Novelty Detection." In: *Proceedings of the 12th International Conference on Neural Information Processing Systems – NIPS'99*. Vol. 12. 1999, pp. 582–588 (cit. on p. 113).

[29] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. "Predictive Business Process Monitoring with LSTM Neural Networks." In: *Proceedings of the 29th International Conference on Advanced Information Systems Engineering – CAiSE'17*. Springer. 2017, pp. 477–492 (cit. on p. 113).

[30] Christina Warrender, Stephanie Forrest, and Barak Pearlmutter. "Detecting Intrusions Using System Calls: Alternative Data Models." In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy – SP'99*. 1999, pp. 133–145 (cit. on pp. 113, 124, 125, 127).

[31] Lijie Wen, Wil M. P. van der Aalst, Jianmin Wang, and Jiaguang Sun. "Mining Process Models with Non-free-choice Constructs." In: *Data Mining and Knowledge Discovery* 15.2 (2007), pp. 145–180 (cit. on p. 112).

[32] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. "A Close Look on N-grams in Intrusion Detection: Anomaly Detection vs. Classification." In: *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security – AISec'13*. 2013, pp. 67–76 (cit. on p. 113).

P5

DEEPALIGN: ALIGNMENT-BASED PROCESS ANOMALY CORRECTION USING RECURRENT NEURAL NETWORKS

Timo Nolle, Alexander Seeliger, Nils Thoma, and Max Mühlhäuser
In: *Proceedings of the 32nd International Conference on Advanced Information Systems Engineering – CAiSE'20*. 2020, pp. 319–333.

ABSTRACT: In this paper, we propose DeepAlign, a novel approach to multi-perspective process anomaly correction, based on recurrent neural networks and bidirectional beam search. At the core of the DeepAlign algorithm are two recurrent neural networks trained to predict the next event. One is reading sequences of process executions from left to right, while the other is reading the sequences from right to left. By combining the predictive capabilities of both neural networks, we show that it is possible to calculate sequence alignments, which are used to detect and correct anomalies. DeepAlign utilizes the case-level and event-level attributes to closely model the decisions within a process. We evaluate the performance of our approach on an elaborate data corpus of 252 realistic synthetic event logs and compare it to three state-of-the-art conformance checking methods. DeepAlign produces better corrections than the rest of the field reaching an overall F_1 score of 0.9572 across all datasets, whereas the best comparable state-of-the-art method reaches 0.6411.

KEYWORDS: Business Process Management, Anomaly Detection, Deep Learning, Sequence Alignments

1 INTRODUCTION

Process anomaly detection can be used to automatically detect deviations in process execution data. This technique infers the process solely based on distributions of the execution data, without relying on an abstract definition of the process itself. While these approaches can accurately pinpoint an anomaly in a process, they do not provide information about what should have been done instead. Although, the knowledge about the occurrence of an anomaly is valuable, much more value lies in the knowledge of what was supposed to happen and how to avoid this behavior in the future.

Process mining techniques are centered around the notion of a process model that describes the correct behavior of a process. Conformance checking techniques can be utilized to analyze process executions for their conformance with a process model. This method has the benefit of not only detecting deviations from the defined process but also of providing the closest conforming path through the process, thereby correcting it.

The correctness of the conformance checking result depends on the quality of the process model. Furthermore, a correct execution of a process is not necessarily defined by a correct order of process steps but can depend on a variety of other parameters. For example, it might not be allowed that the same person executes two consecutive process steps or a process might differ depending on the country it is being executed in. All these possibilities have to be accounted for both in the process model and the conformance checking algorithm to ensure a correct result. If no process model is available, conformance checking cannot be used and the creation of a good reference model is a time-consuming task.

An automatic process anomaly correction is therefore desirable, combining the autonomy of an anomaly detection algorithm with the descriptive results from conformance checking. Against this background, we propose the DeepAlign[1] algorithm, which combines these two benefits. It borrows from the field of anomaly detection and employs two recurrent neural networks (RNN), trained on the task of next event prediction, as an approximate process model [18]. Inspired by the alignment concept from conformance checking, we show that a bidirectional beam search [17] can be used to align a process execution with the process model as approximated by the two RNNs.

DeepAlign can not only detect that process steps have been skipped, but it can also predict which process steps should have been executed instead. Furthermore, it does not rely on a reference model of the process, nor any prior knowledge about it. It can be used to automatically detect anomalies and to automatically correct them.

2 BACKGROUND

Before we describe the DeepAlign algorithm, we must first introduce some concepts from the field of process mining and deep learning.

2.1 *Process Mining*

Process mining is centered around the idea of human-readable representations of processes called process models. Process models are widely used in business process management as a tool for defining, documenting, and controlling business processes inside companies.

1 Available on GitHub https://github.com/tnolle/deepalign

During the execution of a digital business process, each process step is stored in a database. This includes information on when the process step was executed (timestamp), what process step was executed (activity), and to which business case it belongs (case identifier). These three fundamental bits of event information are the basis for every process mining algorithm and are usually combined into a single data structure called event log.

A log consists of cases, each of which consists of events executed within a process, and some attributes connected to the case (case attributes). Each event is defined by an activity name and its attributes (e.g., a user who executed the event).

Definition P5.1. *Case, Event, and Log. Let $\mathcal{E}$ be the set of all events. A case is a sequence of events $c \in \mathcal{E}^*$, where $\mathcal{E}^*$ is the set of all sequences over $\mathcal{E}$ Let $\mathcal{C}$ be the set of all cases. An event log is a set of cases $\mathcal{L} \subseteq \mathcal{C}$.*

Event logs can be used to automatically discover a process model. Discovery algorithms analyze the event logs for process patterns and aim to produce a human-readable process model that likely produced the event log. Multiple discovery algorithms exist, such as the Heuristics Miner [20] and the Inductive Visual Miner [9].

2.2 *Alignments*

In process analytics, it is desirable to relate the behavior observed in an event log to the behavior defined in a process model. This discipline is called conformance checking. The goal of conformance checking is to find an alignment between an event log and a reference process model. The reference model can be manually designed or be discovered by a process discovery algorithm.

Definition P5.2. *Alignment. An alignment [5] is a bidirectional mapping of an event sequence σ_l from the event log to a possible execution sequence σ_m of the process model. It is represented by a sequence of tuples $(s_l, s_m) \in (\mathcal{E}^{\gg} \times \mathcal{E}^{\gg}) \setminus \{(\gg, \gg)\}$, where $\gg$ is an empty move and $\mathcal{E}^{\gg} = \mathcal{E} \cup \{\gg\}$. We say that a tuple represents a synchronous move if $s_l \in \mathcal{E}$ and $s_m \in \mathcal{E}$, a model move if $s_l = \gg$ and $s_m \in \mathcal{E}$, and a log move if $s_l \in \mathcal{E}$ and $s_m = \gg$. An alignment is optimal if the number of empty moves is minimal.*

For $\sigma_l = \langle a, b, c, x, e \rangle$ and $\sigma_m = \langle a, b, c, d, e \rangle$, the two optimal alignments are

a	b	c	x	$\gg$	e
a	b	c	$\gg$	d	e

and

a	b	c	$\gg$	x	e
a	b	c	d	$\gg$	e

where the top row corresponds to σ_l and the bottom row corresponds to σ_m, mapping moves in the log to moves in the model and vice versa.

2.3 *Recurrent Neural Network (RNN)*

Recurrent neural networks (RNN) have been designed to handle sequential data such as sentences. An RNN is a special kind of neural network that makes use of an internal state (memory) to retain information about already seen words in a sentence. It is processing a sentence word for word, and with each new word, it will approximate the probability distribution over all possible next words. Neural networks can be efficiently trained using a gradient descent learning procedure, minimizing the error in the prediction by tuning its internal parameters (weights). The error can be computed as the difference between the output of the neural network and the desired output.

After the training procedure, the neural network can approximate the probability distribution over all possible next words, given an arbitrary length input sequence. With slight alterations, RNNs can be applied to event logs, which we will explain further in Sec. 3.

2.4 *Beam Search*

In natural language processing, it is common to search for the best continuation of a sentence under a given RNN model. To find the most probable continuation, every possible combination of words has to be considered which, for a length of L and a vocabulary size of V, amounts to V^L possible combinations. Due to the exponential growth of the search space, this problem is NP-hard.

Instead, a greedy approach can be taken, producing always the most likely next word given a start of a sentence, based on probability under the RNN. However, this approach does not yield good results because it approximates the total probability of the sentence continuation based only on the probability of the next word. A more probable sentence might be found when expanding the search to the second most probable next word, or the third, and so on.

Beam search (BS) is a greedy algorithm that finds a trade-off between traversing all possible combinations and only the most probable next word. For every prediction, the BS algorithm expands only the K most probable sentence continuations (beams). In the next step, the best K probable continuations over all K beams from the previous step are chosen, and so on. For $K = 1$, BS is equivalent to the greedy 1-best approach explained above. BS has the advantage of pruning the search space to feasible sizes, while still traversing a sufficient part of the search space to produce a good approximation of the most likely sentence continuation.

The BS algorithm is iteratively applied, inserting new words with each step, until convergence, i.e., the end of a sentence is reached, indicated by the end of sentence symbol.

2.5 *Bidirectional Beam Search*

The BS algorithm continues a sentence until a special end of sentence symbol is predicted. However, if the sentence has a defined beginning and end, this approach cannot be used because a unidirectional RNN only knows about the beginning of the sentence and not the end. This has been demonstrated and been addressed in [17] with a novel bidirectional beam search (BiBS) approach. Instead of using a single unidirectional RNN, the authors propose to use two separate unidirectional RNNs, one reading the input sentences forwards, and one reading them backwards.

The problem that arises with a gap in the middle of a sentence is that the probability of the resulting sentence, after the insertion of a new word, cannot be computed by a single RNN without recomputation of the remainder of the sentence. In BiBS, this probability is approximated by the product of the probability of the beginning of the sentence (by the forward RNN), the end of the sentence (by the backward RNN), and the joint probability of inserting the new word (according to both RNNs). The original BS algorithm is extended to expand the search space based on this joint probability, ensuring a proper fit both for the beginning and the end of the sentence.

The BiBS algorithm is iteratively applied to the original sentence, updating it with each step, until convergence, i.e., no insertions would yield a higher probability in any of the K beams.

3 DEEPALIGN

In this section we describe the DeepAlign algorithm and all its components. An overview of the algorithm is shown in Fig. P5.1. Two neural networks are trained to predict the next event, one reading cases from left to right (forwards), the other reading them from right to left (backwards). An extended BiBS is then used to transform the input case to the most probable case under the two RNN models. Lastly, an alignment is calculated based on the search history of the algorithm.

3.1 *Next Event Prediction*

Next event prediction aims to accurately model the decisions being made in a process. These decisions are based on multiple parameters, such as the history of a case, the attributes connected to past events, and the case level attributes. To succeed, a machine learning model must take into account all of these parameters.

In this paper, we propose a new neural architecture for next event prediction. It has been designed to model the sequence of activities (control-flow), the attributes connected to these activities (event

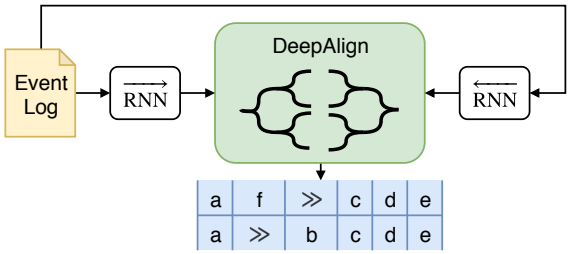

Figure P5.1: The DeepAlign algorithm makes use of two next event prediction RNNs and an extended bidirectional beam search (green) to produce alignments

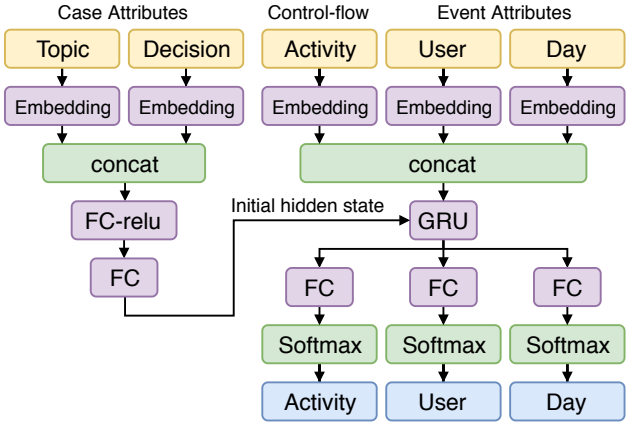

Figure P5.2: RNN architecture for an event log with two case attributes (*Topic* and *Decision*) and two event attributes (*User* and *Day*)

attributes), and the global attributes connected to the case (case attributes). Figure P5.2 shows the architecture in detail.

At the heart of the network is a Gated Recurrent Unit (GRU) [7], a type of RNN. This GRU is iteratively fed an event, consisting of its activity and its event attributes, and must predict the corresponding next event. Each categorical attribute is fed through an embedding layer to map the values into a lower-dimensional embedding space. To include the case attributes, we make use of the internal state of the GRU. Instead of initializing the state with zeros (the default), we initialize it based on a representation of the case attributes. All case attributes are transformed by a case attribute network, consisting of two fully-connected layers (FC), to produce a real-valued representation of the case attributes. In other words, we initialize the next event prediction with a representation of the case attributes, thereby conditioning it to predict events according to these case attributes. Finally, the GRU output is fed into separate FC layers with Softmax activations to produce a probability distribution over all possible attributes of the next event (i.e., the prediction of the next event).

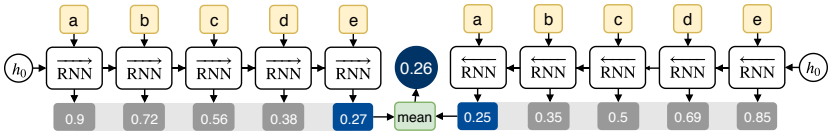

Figure P5.3: The probability of a case $c = \langle a, b, c, d, e \rangle$ is computed by the average probability of the case under both the forward and the backward RNN

We train the networks with a GRU size equal to two times the maximum case length on mini-batches of size 100 for 50 epochs using the Adam optimizer with standard parameters [8]. The first layer of the case attribute network has an output size of the GRU size divided by 8 and the second layer output is equal to the hidden state size of the GRU. These parameters were chosen following an exhaustive grid search, however, we found that any reasonable setting generally worked.

3.2 *The DeepAlign Algorithm*

In the context of processes, the sentences of words from above will become the cases of events from the event log. By replacing the next word prediction RNNs with next event prediction RNNs in the BiBS algorithm we can apply it to event logs. Instead of only predicting the next word, the RNNs will predict the next event, including the most likely event attributes.

Our goal is to utilize the two RNNs as the reference model for conformance checking and produce an alignment between log and the RNNs. Alignments can be interpreted as a sequence of *skip* (synchronous move), *insertion* (model move), or *deletion* (log move) operations. The BiBS algorithm already covers the first two operations, but not the last. To allow for deletions, we have to extend the BiBS algorithm.

Let $\overrightarrow{\text{RNN}}$ be the forward event prediction RNN and $\overleftarrow{\text{RNN}}$ be the backward RNN. Let further $\text{RNN}(h, c)$ be the probability of case c under RNN, initialized with the hidden state h.

The probability of a case c under the two RNNs can be computed by

$$P(c) = \frac{1}{2}\left(\overrightarrow{\text{RNN}}(h_0, c) + \overleftarrow{\text{RNN}}(h_0, c)\right),$$

where h_0 is the output of the case attribute network. If no case attributes are available, the initial state is set to zeros. An example is shown in Fig. P5.3.

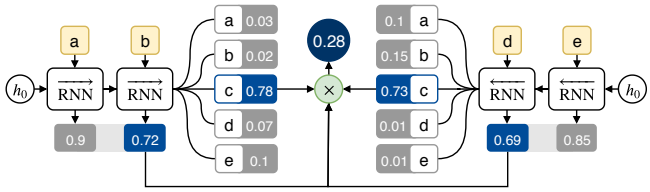

Figure P5.4: The probability of a case $c = \langle a, b, d, e \rangle$ after the insertion of an event c after b is computed by the joint probability $\langle a, b \rangle$ under the forward RNN, $\langle d, e \rangle$ under the backward RNN, and the probabilities of continuing the case with c under both RNNs

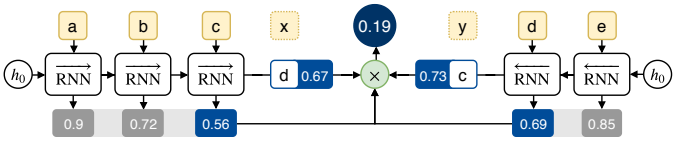

Figure P5.5: The probability of a case $c = \langle a, b, c, x, y, d, e \rangle$ after the deletion of x and y is computed by the joint probability of $\langle a, b, c \rangle$ under the forward RNN, $\langle d, e \rangle$ under the backward RNN, and the probabilities of continuing the case with d and c under the forward and backward RNN, respectively

For an insertion of an event e at time t in a case c, the probability under the two RNNs can be approximated by

$$P_{ins}(c, e, t) = \overrightarrow{RNN}\left(h_0, c_{[1:t]}\right) \cdot \overrightarrow{RNN}\left(\overrightarrow{h}_t, e\right) \cdot \overleftarrow{RNN}\left(\overleftarrow{h}_{t+1}, e\right) \cdot \overleftarrow{RNN}\left(h_0, c_{[t+1:T]}\right),$$

where T is the total case length, $c_{[1:t]}$ is the index notation to retrieve all events from c until time t, and $\overrightarrow{h}_t$ is the hidden state of $\overrightarrow{RNN}$ after reading $c_{[1:t]}$. Similarly, $\overleftarrow{h}_{t+1}$ is the hidden state of $\overleftarrow{RNN}$ after reading $c_{[t+1:T]}$. An example is shown in Fig. P5.4.

The probability of deleting n events at time t in a case c can be approximated by

$$P_{del}(c, n, t) = \overrightarrow{RNN}\left(h_0, c_{[1:t]}\right) \cdot \overrightarrow{RNN}\left(\overrightarrow{h}_t, c_{[t+n]}\right) \cdot \overleftarrow{RNN}\left(\overleftarrow{h}_{t+n}, c_{[t]}\right) \cdot \overleftarrow{RNN}\left(h_0, c_{[t+n:T]}\right).$$

An example is shown in Fig. P5.5.

Algorithm P5.1 shows the full DeepAlign process of aligning a case c with the two RNNs. The algorithm is initialized with an initial set of beams $B = \{c\}$, i.e., the original case. For each possible operation, the probabilities are computed using the aforementioned equations, and the top-K beams are returned. For simplicity, we assume that top-K always returns the updated cases according to the operations with the highest probability. The number of events that can be deleted in one step can be controlled with the parameter N. This is necessary because

successively deleting single events does not necessarily generate higher probabilities than removing multiple events at once.

Algorithm P5.1: DeepAlign algorithm

Data: Given a set of beams B, maximum number of beams K, and a maximum deletion size N
while *not converged* **do**
  $B' = \emptyset$;
  for $b \in B$ **do**
    $B' = B' \cup P(b)$;
    for $t = 1, ..., T$ **do**
      $B' = B' \cup \{P_{del}(b, n, t) \mid n \in 1, ..., N\} \cup \{P_{ins}(b, e, t) \mid e \in \mathcal{E}\}$
    end
  end
  $B = \text{top-K}(B')$
end
Result: B, the top-K beams after convergence

Algorithm P5.1 does not yet return alignments, but the top-K updated cases. By keeping a history of the top-K operations (*skip*, *deletion*, and *insertion*) in every iteration, we can obtain the alignment directly from the history of operations. A *deletion* corresponds to an empty move on the model, whereas an *insertion* corresponds to an empty move in the log.

The top-K selection in Alg. P5.1 will select the top K beams based on the probability under the RNN models. In case of ties, we break the tie by choosing the beam with fewer empty moves (insertions and deletions).

4 EXPERIMENTS

We evaluate the DeepAlign algorithm for the task of anomaly correction. Given an event log consisting of normal and anomalous cases, an anomaly correction algorithm is expected to align each case in the event log with a correct activity sequence according to the process (without anomalies) that produced the event log.

We use a simple paper submission process as a running example throughout the remainder of this paper. The process model in Fig. P5.6 describes the creation of a scientific paper. It includes the peer review process, which is executed by a reviewer, whereas the paper is written by an author.

To evaluate the accuracy of the corrections, we generated six random process models using PLG2 [6]. The models vary in complexity with respect to the number of activities, breadth, and width. Additionally, we use a handmade procurement process model called P2P as in [15].

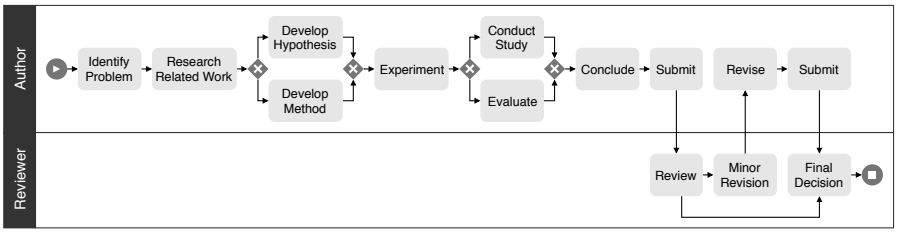

Figure P5.6: A simple paper submission process which is used as an example in the evaluation

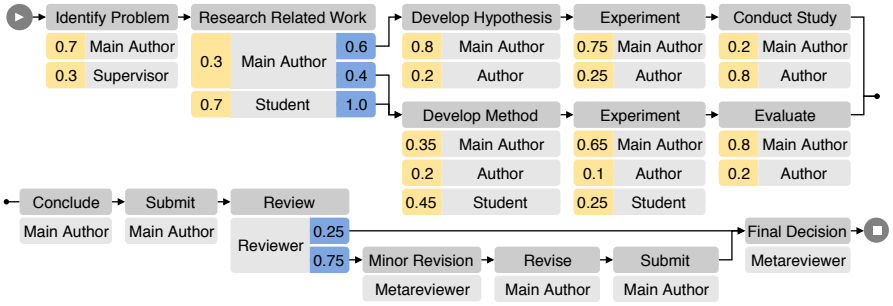

Figure P5.7: A likelihood graph with user attribute; 1.0 probabilities omitted for simplicity

To generate event attributes, we create a likelihood graph [4] from the process models which includes probabilities for event attributes connected to each step in the process. This method has been proposed in [14]. A likelihood graph for the paper process from Fig. P5.6 is shown in Fig. P5.7.

For each process step, the probability of the resource executing it is shown in yellow. Depending on the resource, the probabilities of the next process steps are shown in blue. Note that there is a long-term dependency between the steps *Develop Hypothesis* and *Conduct Study*, and, similarly, between *Develop Method* and *Evaluate*. That is, *Conduct Study* never eventually follows *Develop Method*, and, likewise, *Evaluate* never eventually follows *Develop Hypothesis*.

We can generate event logs by using a random-walk through the likelihood graph, complying with the transition probabilities, and generating activities and attributes along the way. In addition to the event attributes, we also generate case attributes, as well as, dependencies between the case attributes and the output probabilities in the likelihood graph. For the paper process, we generate two case attributes, *Decision* and *Topic*.

If the topic is *Theory*, this implies that *Develop Hypothesis* will occur in a case, whereas if the topic is *Engineering*, it implies *Develop Method* will occur. The decision can be *Accept*, *Weak Accept*, *Borderline*, *Weak Reject*, or *Reject*. For simplicity, we define that there will only be a *Minor Revision* if the *Decision* is either *Accept* or *Weak Accept*. There will be no *Minor Revision* otherwise. We have generated an event log that follows

these rules that we use as an example throughout the remainder of the paper. The paper process was not used in the evaluation because of its simplicity.

For each of the 7 other process models, we generate 4 random event logs with varying numbers of event and case attributes. Additionally, we introduce noise to the event logs by randomly applying one of 7 anomalies to a fixed percentage of the cases in the event log. We generate datasets for noise levels between 10% and 90% with a step size of 10% (9 in total). We gather a ground truth dataset for the anomaly correction problem by retaining the original cases before alteration. The 7 anomalies are defined as follows.

- *Skip*: A sequence of up to 2 necessary events has been skipped
- *Insert*: Up to 2 random activities have been inserted
- *Rework*: A sequence of up to 3 events has been executed a second time
- *Early*: A sequence of up to 2 events has been executed too early, and hence is skipped later in the case
- *Late*: A sequence of up to 2 events has been executed too late, and hence is skipped earlier in the case
- *Attribute*: An incorrect attribute value has been set in up to 3 events

To analyze the impact of the case and event attributes, we evaluate four different implementations of DeepAlign: one that does not use any attributes (DeepAlign$\emptyset$), one that only uses case attributes (DeepAlign$\emptyset$), one that only uses event attributes (DeepAlignE), and one that uses both (DeepAlignCE).

Additionally, we evaluate baseline approaches that first discover a process model using a discovery algorithm and then calculate the alignments [1]. We chose the Heuristics Miner [20] and the Inductive Miner [9] using the implementations of the PM4Py library [2]. For completeness, we also evaluate the conformance checking algorithm using a perfect Reference Model, i.e., the one used to generate the event logs.

We run the DeepAlign algorithm for a maximum number of 10 iterations with a beam size of $K = 5$ and a maximum deletion size of $N = 3$, and consider the top-1 beam for the evaluation. The Inductive Miner and the Heuristics Miner are used as implemented in PM4Py. For the Heuristics Miner, we use a dependency threshold of 0.99, and for the Inductive Miner, we use a noise threshold of 0.2.

Table P5.1: Correction accuracy, average error for incorrect alignments (based on the Levenshtein distance), and alignment optimality for correct alignments; best results are shown in bold typeface

	CF	CA	EA	F_1^N	F_1^A	F_1	Error	Optimal
Reference Model	✓	-	-	0.9011	0.9331	0.9171	**1.46**	-
Heuristics Miner	✓	-	-	0.6678	0.6144	0.6411	3.33	-
Inductive Miner	✓	-	-	0.6007	0.2438	0.4222	2.18	-
DeepAlign∅	✓	-	-	0.7950	0.8111	0.8030	2.52	99.8%
DeepAlignC	✓	✓	-	0.8918	0.9290	0.9104	2.41	**99.9%**
DeepAlignE	✓	-	✓	0.9261	0.9582	0.9421	1.65	86.9%
DeepAlignCE	✓	✓	✓	**0.9442**	**0.9702**	**0.9572**	1.84	86.6%

5 EVALUATION

The overall results are shown in Tab. P5.1. For each dataset we run the algorithms and evaluate the correction accuracy, that is, an alignment is regarded as correct if the model sequence is exactly equal to the ground truth sequence. For correct alignments, we calculate the optimality of the alignment (i.e., if the number of empty moves is minimal). For incorrect alignments, we calculate the distance from the ground truth sequence with Levenshtein's algorithm. Accuracy is measured as the macro average F_1 score of normal (F_1^N) and anomalous (F_1^A) cases across all datasets and noise levels.

Interestingly, DeepAlignE, and DeepAlignCE both outperform the perfect Reference Model approach. This is because the Reference Model does not contain any information about the case and event attributes. The Heuristics Miner yields much better results in the anomaly correction task than the Inductive Miner, however, DeepAlign∅ outperforms both, without relying on case or event attributes.

Reference Model, Heuristics Miner, and Inductive Miner all produce optimal alignments because the alignment algorithm guarantees it. The DeepAlign algorithm shows a significant drop in alignment optimality when including the event attributes. The drop in optimality can be attributed to the fact that we always predict the top-1 attribute value for inserted events in the DeepAlign algorithm. Furthermore, it might be connected to the attribute level anomalies that we introduced as part of the generation. The best results are achieved when including both the case and the event attributes. Figure P5.8 shows the F_1 score for each algorithm per noise level and per dataset. DeepAlignCE always performs better than the Reference Model, and significantly better than the two mining approaches.

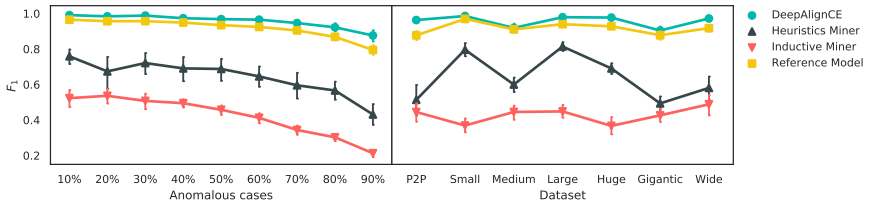

Figure P5.8: F_1 score for each algorithm per noise ratio (left) and per dataset (right); error bars indicate variance across all runs

We want to finish the evaluation with examples from the paper dataset to illustrate the results of the DeepAlign algorithm. This is the resulting alignment for a case with a *Skip* anomaly,

Identify Problem	≫	≫	Experiment	Evaluate	Conclude	Submit	Review	...
Identify Problem	Research Related Work	Develop Method	Experiment	Evaluate	Conclude	Submit	Review	...

this is the result for a case with a *Late* anomaly,

Identify Problem	≫	≫	Experiment	Research Related Work	Develop Method	Evaluate	Conclude	Submit	...
Identify Problem	Research Related Work	Develop Method	Experiment	≫	≫	Evaluate	Conclude	Submit	...

and this is the result for a case with an *Insert anomaly*.

Identify Problem	Research Related Work	Random activity 10	Develop Method	Experiment	Evaluate	Conclude	Random activity 12	Submit	...
Identify Problem	Research Related Work	≫	Develop Method	Experiment	Evaluate	Conclude	≫	Submit	...

The DeepAlign method can also be utilized to generate sequences from nothing, that is, to align the empty case with the most likely case according to the model. Depending on the case attributes that are used to initialize the RNNs, the results will be different.

For Decision = Reject and Topic = Engineering the resulting sequence is ⟨ Identify Problem, Research Related Work, Develop Method, Experiment, Evaluate, Conclude, Submit, Review, Final Decision ⟩, whereas if we set Topic = Theory the resulting sequence is ⟨ Identify Problem, Research Related Work, Develop Hypothesis, Experiment, Conduct Study, Conclude, Submit, Review, Final Decision ⟩. The DeepAlign algorithm correctly generates a sequence including the *Develop Method* and *Develop Hypothesis* activities according to the setting of the *Topic* case attribute. It also does not generate the *Minor Revision* activity because the *Decision* is *Reject*. When setting Decision = Accept, DeepAlign will generate the sequence including the *Minor Revision* branch. A similar effect can be observed when altering the event attributes.

This demonstrates that the RNNs are indeed capable of learning the rules behind the decisions in the paper process (cf. [18]). Although the paper dataset contains unambiguous dependencies between the case attributes and the resulting correct sequences, the overall results on the randomly generated datasets indicate that case and event attributes ought not to be neglected.

6 RELATED WORK

Anomaly detection in business processes is frequently researched. Many approaches exist that aim to detect anomalies in a noisy event log (i.e., an event log that contains anomalous cases).

Bezerra et al. have proposed multiple approaches utilizing discovery algorithms to mine a process model and then use conformance checking to infer the anomalies [3]. Böhmer et al. proposed a technique based on an extended likelihood graph that is utilizing event-level attributes to further enhance the detection [4]. The approach from [4] requires a clean event log (i.e., no anomalies in the log), but it has been shown that the same technique can be applied to noisy logs as well [14]. Recently, Pauwels et al. presented an approach based on Bayesian Networks [16]. Deep learning based approaches are presented in [13] and [14]. However, none of these approaches can be utilized to correct an anomalous case or to produce an alignment.

Since Bezerra et al. presented their approach based on discovery algorithms in 2013, Mannhardt et al. have proposed both a data-aware discovery algorithm [11] and a data-aware conformance checking algorithm [12]. The conformance checking algorithm relies on a configurable cost function for alignments that must be manually defined to include the case and event attributes. Our approach does not rely on a manual definition of the cost function, it traverses the search space based on learned probabilities instead.

Although alignments represent the current state-of-the-art in conformance checking [1], they often pose a significant challenge because they are computationally expensive. Van Dongen et al. address this issue in [19], compromising between computational complexity and quality of the alignments. Very recently, Leemans et al. have presented a stochastic approach to conformance checking [10], which can speed up the computation.

All of these approaches either rely on a non-data-aware discovery technique, require a manual effort to create a proper cost function, or they cannot generate alignments. To the best of our knowledge, DeepAlign is the first fully autonomous anomaly correction method.

7 CONCLUSION

We have demonstrated a novel approach to calculate alignments based on the DeepAlign algorithm. When no reference model is available, two recurrent neural networks can be used to approximate the underlying process based on execution data, including case and event attributes. The empirical results obtained in the experiments indicate that RNNs are indeed capable of modeling the behavior of a process solely based on an event log event if it contains anomalous behavior.

To the best of our knowledge, this is the first time that deep learning has been employed to calculate alignments in the field of process mining. Although we evaluate DeepAlign in the context of anomaly correction, many other applications are conceivable. For example, instead of training on a log that contains anomalies, a clean log could be used. Furthermore, a clean log can be obtained from an existing reference model, and DeepAlign could be used to find alignments. In other words, it might be possible to convert a manually created process model into a DeepAlign model. A discovery algorithm based on DeepAlign is also imaginable since DeepAlign can also be utilized to generate sequences from scratch. Depending on the case attributes the resulting predicted sequences will be different. We think that this idea lends itself to further research.

We further believe that the DeepAlign algorithm could be employed to reduce the memory consumption of an alignment algorithm since the search space is efficiently pruned during the bidirectional beam search. However, on the downside, DeepAlign does not guarantee optimal alignments. This weakness can be addressed by employing an optimal alignment algorithm between the input sequence and the corrected sequence, albeit at the expense of efficiency.

In summary, DeepAlign is a novel and flexible approach with great application potential in many research areas within the field of process mining.

ACKNOWLEDGMENTS

This work is funded by the German Federal Ministry of Education and Research (BMBF) Software Campus project “R2PA” [01IS17050], Software Campus project “KADet” [01IS17050], and the research project “KI.RPA” [01IS18022D].

REFERENCES

[1] Arya Adriansyah, Boudewijn F van Dongen, and Wil MP van der Aalst. “Memory-efficient alignment of observed and modeled behavior.” In: *BPM Center Report* 3 (2013) (cit. on pp. 147, 150).

[2] Alessandro Berti, Sebastiaan J van Zelst, and Wil van der Aalst. "Process Mining for Python (PM4Py): Bridging the Gap Between Process-and Data Science." In: *ICPM'19 (Demos)*. 2019, pp. 13–16 (cit. on p. 147).

[3] Fábio Bezerra and Jacques Wainer. "Algorithms for Anomaly Detection of Traces in Logs of Process Aware Information Systems." In: *Information Systems* 38.1 (2013), pp. 33–44 (cit. on p. 150).

[4] Kristof Böhmer and Stefanie Rinderle-Ma. "Multi-perspective Anomaly Detection in Business Process Execution Events." In: *Proceedings of On the Move to Meaningful Internet Systems, OTM'16*. Springer. 2016, pp. 80–98 (cit. on pp. 146, 150).

[5] RP Jagadeesh Chandra Bose and Wil van der Aalst. "Trace alignment in process mining: opportunities for process diagnostics." In: *Proceedings of the 8th International Conference on Business Process Management – BPIC'10*. Springer. 2010, pp. 227–242 (cit. on p. 139).

[6] Andrea Burattin. "PLG2: Multiperspective Process Randomization with Online and Offline Simulations." In: *BPM'16 (Demos)*. 2016, pp. 1–6 (cit. on p. 145).

[7] Kyunghyun Cho et al. "Learning Phrase Representations Using RNN Encoder-decoder for Statistical Machine Translation." In: *arXiv preprint arXiv:1406.1078* (2014) (cit. on p. 142).

[8] Diederik Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on p. 143).

[9] Sander JJ Leemans, Dirk Fahland, and Wil MP Van Der Aalst. "Process and Deviation Exploration with Inductive Visual Miner." In: *BPM'14 (Demos)* 1295.46 (2014), p. 8 (cit. on pp. 139, 147).

[10] Sander JJ Leemans, Anja F Syring, and Wil MP van der Aalst. "Earth Movers' Stochastic Conformance Checking." In: *Proceedings of the Business Process Management Forum 2019 – BPM'19*. Springer. 2019, pp. 127–143 (cit. on p. 150).

[11] Felix Mannhardt, Massimiliano De Leoni, and Hajo A Reijers. "The Multi-perspective Process Explorer." In: *BPM'15 (Demos)* 1418 (2015), pp. 130–134 (cit. on p. 150).

[12] Felix Mannhardt, Massimiliano De Leoni, Hajo A Reijers, and Wil MP Van Der Aalst. "Balanced multi-perspective checking of process conformance." In: *Computing* 98.4 (2016), pp. 407–437 (cit. on p. 150).

[13] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. "Analyzing Business Process Anomalies Using Autoencoders." In: *Machine Learning* 107.11 (Nov. 2018), pp. 1875–1893 (cit. on p. 150).

[14] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. "BINet: Multi-perspective Business Process Anomaly Classification." In: *Information Systems* (2019), p. 101458. ISSN: 0306-4379 (cit. on pp. 146, 150).

[15] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. "BINet: Multivariate Business Process Anomaly Detection Using Deep Learning." In: *Proceedings of the 16th International Conference on Business Process Management – BPM'18*. 2018, pp. 271–287 (cit. on p. 145).

[16] Stephen Pauwels and Toon Calders. "An anomaly detection technique for business processes based on extended dynamic bayesian networks." In: *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*. ACM. 2019, pp. 494–501 (cit. on p. 150).

[17] Qing Sun, Stefan Lee, and Dhruv Batra. "Bidirectional beam search: Forward-backward inference in neural sequence models for fill-in-the-blank image captioning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition – CVPR'17*. 2017, pp. 6961–6969 (cit. on pp. 138, 141).

[18] Niek Tax, Sebastiaan J van Zelst, and Irene Teinemaa. "An experimental evaluation of the generalizing capabilities of process discovery techniques and black-box sequence models." In: *Proceedings of the 19th International Conference on Enterprise, Business-Process and Information Systems Modeling – BPMDS'18*. Springer, 2018, pp. 165–180 (cit. on pp. 138, 150).

[19] Boudewijn Van Dongen, Josep Carmona, Thomas Chatain, and Farbod Taymouri. "Aligning modeled and observed behavior: a compromise between computation complexity and quality." In: *Proceedings of the 29th International Conference on Advanced Information Systems Engineering – CAiSE'17*. Springer. 2017, pp. 94–109 (cit. on p. 150).

[20] AJMM Weijters and JTS Ribeiro. "Flexible heuristics miner (FHM)." In: *Proceedings of the 2011 IEEE Symposium on Computational Intelligence and Data Mining – CIDM'11*. IEEE. 2011, pp. 310–317 (cit. on pp. 139, 147).

ERKLÄRUNG

Hiermit erkläre ich, die vorgelegte Arbeit mit dem Titel

Process Learning for Autonomous Process Anomaly Correction

selbstständig und ausschließlich unter der Verwendung der angegebenen Hilfsmittel erstellt zu haben.

Hiermit erkläre ich weiterhin, dass von mir bisher kein Promotionsversuch unternommen wurde.

Bensheim, den 15.09.2020

Timo Nolle