Predictive Object-Centric Process Monitoring

Bachelor's Thesis

Author : **Timo Rohrer**

Advisors : Anahita Farhang Ghahfarokhi
Mohamed Behery

Examiners : Professor Gerhard Lakemeyer
Professor Wil van der Aalst

Registration date : 2020-11-18

Submission date : 2021-03-18

This work is submitted to the

Knowledge-Based Systems Group (KBSG) at i5

and

Process and Data Science (PADS) Chair at i9

at

RWTH Aachen University

Abstract

The automation and digitalization of business processes has resulted in large amounts of data captured in information systems, which can aid businesses in understanding their processes better, improve workflows, or provide operational support. By making predictions about ongoing processes, bottlenecks can be identified and resources reallocated, as well as insights gained into the state of a process instance (case). Traditionally, data is extracted from systems in the form of an event log with a single identifying case notion, such as an order id for an Order to Cash (O2C) process. However, real processes often have multiple object types, for example, order, item, and package, so a format that forces the use of a single case notion does not reflect the underlying relations in the data. The Object-Centric Event Log (OCEL) format was introduced to correctly capture this information. The state-of-the-art predictive methods have been tailored to only traditional event logs. This thesis shows that a prediction method utilizing Generative Adversarial Networks (GAN), Long Short-Term Memory (LSTM) architectures, and Sequence to Sequence models (Seq2seq), can be augmented with the rich data contained in OCEL. Objects in OCEL can have attributes that are useful in predicting the next event and timestamp, such as a priority class attribute for an object type package indicating slower or faster processing. In the metrics of sequence similarity of predicted remaining events and mean absolute error (MAE) of the timestamp, the approach in this thesis matches or exceeds previous research, depending on whether selected object attributes are useful features for the model. Additionally, this thesis provides a web interface to predict the next sequence of activities from user input.

Contents

Acknowledgments

At first, I would like to express my deep gratitude to my advisers Anahita Farhang Ghahfarokhi and Mohamed Behery for their continual support throughout defining the thesis topic, conducting the research, and writing the thesis. Our regular meetings provided important guidance, excellent feedback, and interesting discussions.

I would futhermore like to thank Professor Gerhard Lakemeyer and Professor Wil van der Aalst for accepting this thesis and taking the position of examiners. I would also like to thank them for enabling this inter-institute thesis, where I was able to gather compelling insights into both KBSG and PADS.

Lastly I am thankful for my family and friends for their encouragement and unfailing support.

Chapter 1

Introduction

In our increasingly digitized world, data is being collected everywhere. Data is not only omnipresent in our personal life but also in production and manufacturing processes, logistics and supply chains, customer service and internal communication systems. From placing an order online, that order being processed at a facility, to it being shipped and delivered, increasingly these activities are all recorded in fine detail [1]. Industry 4.0 and the Internet of Things have greatly accelerated the need for innovative solutions dealing with the data generated and recorded [2]. The Internet of things, referring to the network of objects being fitted with sensors and data reporting capabilities, is a tangible representation of the digital revolution occurring for the last few decades [3]. The tremendous volume, velocity, and variety of data collected poses new challenges but also presents new opportunities. Businesses can extract value from the data by making predictions about future activities and thereby increasing the efficiency of processes [4].

Processes are comprised of a collection of events which can be extracted from information systems that record the data. A single event in a process includes the activity occurring, the timestamp, and additional resources related to the event. Predictive business process monitoring aims to learn from past process data to make predictions about ongoing cases [1]. Predictions can be made for what the next activity will be and when it would occur, or the total time until completion of the case. It is a branch of process mining which covers a collection of techniques supporting process discovery, performance analysis, and conformance checking. Process mining combines the traditional process science of process modeling and analysis with data mining and machine learning methods from data science [1]. Driving the interest for this research discipline is the colossal collection of data in consumer and business contexts as well as the need to remain competitive by improving or rethinking processes with that data. There are multiple types of process mining techniques [1]: process discovery for learning process models from past data, process conformance for comparing a process to its model, and process enhancement where the model is improved based on a process. Several case studies have been done using process mining techniques [5, 6] and some tools have been developed in this area [7**?**]. Additionally, there are techniques that provide operational support, which includes making predictions about ongoing cases.

By exploiting data of past process instances (cases), predictions can be made about ongoing ones. As seen in Figure 1.1, based on event logs a model can, for example, learn to

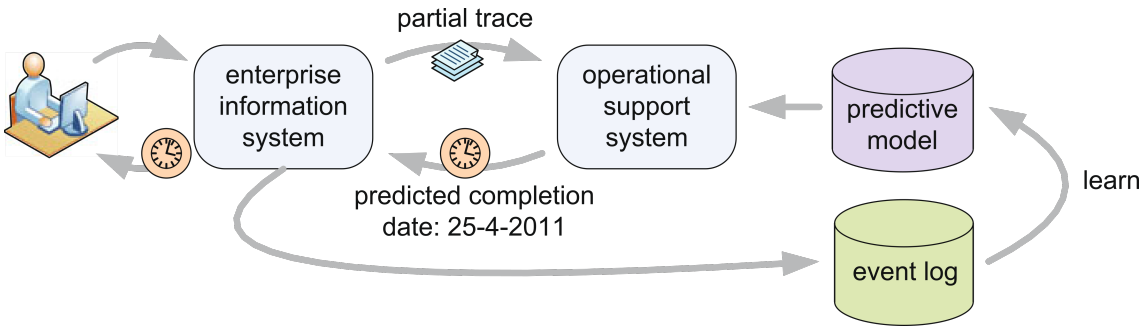


Figure 1.1: Using a model trained on event log data and a partial trace to provide a prediction about the completion date [1].

predict the completion time based on an ongoing case. This thesis focuses on predicting future events, specifically the activity and the timestamp. However, detecting deviations in processes as they are occurring or making direct recommendations with respect to a goal is also possible [1]. Being able to provide such predictions about ongoing cases very powerful. The time left to resolving customer service cases relaionship management (CRM) system would be difficult and time-intensive for employees to predict. In addition to the time to completion, future events themselves can be predicted [8]. For example, a model might predict that based on the current ongoing case, an event will occur where additional information will be requested, or that the case will be resolved shortly. With automatic predictions, customers requesting that information can easily be provided with it, with no additional human intervention or work required. Predicting when an order will be delivered in order to cash (O2C) processes similarly provides this information to customers without additional effort on the side of the company. Aside from customer-facing applications, providing operational support by predicting when production processes will be completed also provides significant value [9]. By giving estimates about remaining manufacturing timelines based on the current progress, bottlenecks can be identified and resources re-allocated to improve efficiency on the fly. By predicting events in a logistics operation, the scheduling of activities can be optimized to improve the efficiency and effectiveness of the operation. With the scale and autonomy of business processes, this form of operational support is vital in the modern interconnected business world [10].

Processes are often captured by information systems, such as Enterprise Resource Planning (ERP) and Supply Chain Management (SCM) systems. The stored data can be extracted as an event log to perform analysis with process mining tools [1]. The event logs that store process data usually contain at minimum information about the type of activity occurring, the timestamp of the event, and a case identifier, which refers to the running case the event is associated with. Traditionally in event log specifications such as the XES standard, a case notion is required to be selected for the event log [11]. The current research on predictive process monitoring has relied on these traditional event log standards for training machine learning models [8, 10, 12–15]. However, in many real information systems capturing process data, there exist multiple objects that relate to events and a single case notion is inadequate to fully capture the process and relations occurring [16].

To illustrate the shortcomings of traditional event logs, consider the following example adapted from [16] ; a customer places an order which is identified with an order id and

which contains multiple different items, each with their own id. The availability of the different items has to be checked and some items are not yet in stock. Shortly after, a second order for the same customer is received. Since all the items of the second order are available, they are packed together with the available items from the first order into a single package and delivered. Later, when the remaining items from the first order come into stock, a second package is loaded and delivered. In total there are multiple different object types, namely customer, order, item, and package. These are logged properly by the information system along with any attributes that exist. But when it comes to extracting the data into an event log, defining a single identifying case notion is difficult. Since there are multiple object types, no one single case notion can capture all the information contained in this process. *Convergence*, which is when one event includes different cases. Additionally, in *divergence* problems can arise there may be multiple instances of the same activity within a case. Traditional event logs make the assumption that there is a single case notion and that each event refers to only one case. Yet as demonstrated by the example, process can include multiple object types. More precisely, there exist one-to-many relations and many-to-many relations between activities and object types that are not correctly modeled by traditional event logs.

To address this problem object-centric event logs were proposed [16, 17]. The shortcomings of traditional event logs are remedied by not forcing the selection of a single case notion and preserving one-to-many and many-to-many relations that exist in many real processes [18–20]. The eXtensible Object-Centric (XOC) format was initially proposed, however, it suffered from complexity and performance problems [21]. To address these concerns, the more efficient OCEL [1] standard was specified to aid the exchange of data between systems and tools. Informally, the data is represented in two tables, one of the events and one for the objects. The event table is organized by event id, and holds information about the activity, timestamp, and other event attributes. Additionally, it includes per event the objects that are related to the event. So in the example above, the place order event would include objects of types customer, order, and item in the OCEL event table. In the object table, the attributes for objects are recorded. There can be attributes such as the size and weight for an object of type item, or the address for an object of type customer.

The challenge remains to integrate the OCEL format of event logs with existing research in prediction methods, as well as using the useful object attributes in predictions.

1.1 Motivation

The motivation for this thesis is to utilize object-centric event logs in predicting succeeding events and timestamps in ongoing cases. Since the related research so far has only used a single case notion format in event logs, predictive methods have been tailored specifically to that structure [8, 10, 12–15]. Without selecting a single case notion, the developed methods can not work. Therefore, in this thesis, the relations between objects and activities in event logs have to be visualized so that appropriate case notions can be selected for the desired prediction task. Additionally, the objects contained in object-centric event logs can have attributes that aid in predictive tasks. In traditional event logs this information is a subclass of case attributes [1] and in OCEL logs it is referred to object attributes. For example, an online order process can contain the object type *package* and for a given

[1] http://ocel-standard.org/

package there can be the object attribute *priority class*. Such data can be a useful feature for predicting the time until a package is delivered. While some past research has utilized the case attributes in traditional event logs [8], the more recent state-of-the-art models [15] forgo using this information in predictions.

1.2 Research Questions

The main goal of the thesis is to leverage object-centric event logs to provide insights for predictions.

To achieve this goal, the questions to be answered are:

1. How can recent improvements in machine learning models be enhanced with object-centric event logs for prediction?
2. How can training and prediction be made more easily accessible?

1.3 Contribution

The principal contribution of this thesis is bridging the gap between recent machine learning models and object-centric event logs. Additionally, the relations between objects and object attributes in OCEL data are used to improve the prediction. To achieve these goals, this thesis:

1. Extracts an OCEL dataset from a factory logistics simulation
2. Augments a state-of-the-art Generative Adversarial Network (GAN) approach with object attributes from OCEL data to improve predictions
3. Develops an interactive web tool for making predictions about ongoing cases deployed using docker and kubernetes

1.4 Thesis Structure

Chapter 2 describes the necessary preliminaries about process mining, traditional and object-centric event logs, and machine learning. Chapter 3 explores recent research and compares it to the proposed methods. Chapter 4 details the data driven prediction model. Chapter 5 describes the implementation of the proposed model and the web interface. Chapter 6 discusses the results of the evaluation. Chapter 7 summarizes the work and a future outlook is given.

Chapter 2

Preliminaries

In this chapter, a short introduction of process mining, the necessary groundwork, and definitions are presented. Formal definitions, as well as intuitive examples of traditional event logs and object-centric event logs are given. Lastly, different neural network architectures are explained.

2.1 Process Mining

Processes are ubiquitous in most companies' day-to-day activities. Common processes include Order to Cash (O2C) and Purchase to Pay (P2P) which record a businesses' workflow regarding order processing. Furthermore, manufacturing and production processes are increasingly recorded as factories move to digitize the assembly line and automate industrial practices [3]. After production, supply chains and logistics organizations have for long recorded receiving and handling of goods in fine detail. Beyond that, internal communications, customer service processes, and essentially any series of activities can and are being recorded for future analysis.

Recording of the data can occur in a variety of ways: Enterprise Resource Planning (ERP), Supply Chain Management (SCM), Customer Relationship Management (CRM), and Business Process Management (BPM) systems record data about events. Process mining extracts the data in the form of event logs and aims to extract useful information. Based on the methods and the final goal, process mining is frequently separated into four types [1]; *Process discovery* deals with discovering process models from event logs, using algorithms to build models such as transition systems or Petri nets. *Conformance checking* deals with detecting abnormalities in an event log as compared to a process model. *Process reengineering* refers to improving an existing process model with data from an event log to better reflect the underlying processes. Lastly, *operational support* is improving process on the fly by making predictions about remaining time or the next event and providing recommendations to decrease delays or bottlenecks [22]. Predictive process mining methods fall in the category of operational support and the developed methods have been tailored to work with traditional event logs which are described in the next section.

2.2 Traditional Event Logs

Typically the process mining methods assume that information systems ingest sequential series of events that are recorded with a variety of information, which can then be extracted as an event log [1]. Table 2.1 gives an example for a traditional event log. Each row in the log shows an event which is represented by the *event id*. Each row is also uniquely tied to a case based on a case notion, which is represented by the *case id*. In addition there is the *activity* and a *timestamp*. While event logs do not necessarily need to be of this general format, for the purposes of prediction this information is important.

Since each event is uniquely tied to a case, it can be represented within a trace. A trace is a finite non-empty sequence of events ordered by time within a case [1]. In Table 2.1 for the case identified by *case id* 73, the activities of the trace are: *place order, check availability, create package, load package, create invoice, failed delivery, receive payment, deliver package*.

An event log as it has been traditionally defined is simply a set of traces, which can also include incomplete or partial traces [1].

Table 2.1: A fragment of a traditional event log example (adapted from [16]).

event id	case id	activity	timestamp
...	...	...	...
9791	73	place order	2020-9-14 11:37
9792	73	check availability	2020-9-14 11:40
9793	73	create package	2020-9-14 15:10
9794	74	place order	2020-9-14 16:01
9795	74	check availability	2020-9-14 16:05
9796	73	load package	2020-9-15 10:15
9797	73	create invoice	2020-9-15 10:16
9798	73	failed delivery	2020-9-15 14:39
9799	74	create package	2020-9-15 16:39
9800	73	receive payment	2020-9-16 09:42
9801	75	place order	2020-9-17 12:03
9802	73	deliver package	2020-9-18 10:39
9803	75	check availability	2020-9-18 11:40
9804	76	place order	2020-9-18 12:03
9805	75	create package	2020-9-18 15:10
9806	76	check availability	2020-9-18 15:40
...	...	...	...

2.3 Object-Centric Event Logs (OCEL)

The event log shown in Table 2.1 can be an inadequate model of real-world processes. In O2C systems like the examples in Tables 2.1 and 2.2, orders can have multiple items, which might be split up into multiple different packages with items from other orders. In Table 2.2, the events that are captured in an object-centric event log are given. Additionally, there is information about the object types contained in the events, presented in Table 2.3. The object types (case notions) are *order, item,* and *package*.

Table 2.2: A fragment of events in an OCEL example (adapted from [16]).

event id	activity	timestamp	order	item	package
...	...	...	...	...	...
9791	place order	2020-9-14 11:37	$\{o_1\}$	$\{i_1, i_2, i_3\}$	$\emptyset$
9792	check availability	2020-9-14 11:40	$\{o_1\}$	$\{i_1\}$	$\emptyset$
9793	pick item	2020-9-14 11:41	$\{o_1\}$	$\{i_1\}$	$\emptyset$
9794	check availability	2020-9-14 11:42	$\{o_1\}$	$\{i_2\}$	$\emptyset$
9795	check availability	2020-9-14 11:43	$\{o_1\}$	$\{i_3\}$	$\emptyset$
9796	pick item	2020-9-14 11:47	$\{o_1\}$	$\{i_2\}$	$\emptyset$
9797	pick item	2020-9-14 11:48	$\{o_1\}$	$\{i_3\}$	$\emptyset$
9798	create package	2020-9-14 15:10	$\emptyset$	$\{i_1, i_2\}$	$\{p_1\}$
9799	place order	2020-9-14 16:01	$\{o_2\}$	$\{i_4, i_5\}$	$\emptyset$
9800	check availability	2020-9-14 16:05	$\{o_2\}$	$\{i_4\}$	$\emptyset$
9801	check availability	2020-9-14 16:06	$\{o_2\}$	$\{i_5\}$	$\emptyset$
9802	pick item	2020-9-15 11:40	$\{o_2\}$	$\{i_1\}$	$\emptyset$
9803	pick item	2020-9-15 11:41	$\{o_2\}$	$\{i_2\}$	$\emptyset$
9804	create package	2020-9-15 16:18	$\emptyset$	$\{i_3, i_4, i_5\}$	$\{p_2\}$
9805	load package	2020-9-16 10:15	$\emptyset$	$\emptyset$	$\{p_1\}$
9806	deliver package	2020-9-16 14:39	$\emptyset$	$\emptyset$	$\{p_1\}$
9807	load package	2020-9-16 16:45	$\emptyset$	$\emptyset$	$\{p_2\}$
9808	send invoice	2020-9-17 10:15	$\{o_1\}$	$\{i_1, i_2, i_3\}$	$\emptyset$
9809	deliver package	2020-9-17 10:26	$\emptyset$	$\emptyset$	$\{p_2\}$
9810	send invoice	2020-9-17 10:45	$\{o_2\}$	$\{i_4, i_5\}$	$\emptyset$
9811	receive payment	2020-9-17 10:49	$\{o_1\}$	$\{i_1, i_2, i_3\}$	$\emptyset$
...	...	...	...	...	...

When approaching the data in Table 2.2 from a traditional event log perspective and trying to fit a single case identifier, events could suffer from convergence or divergence [16]. *Convergence* is when one event is related to multiple cases and *divergence* occurs when within a case a group of activities is independently repeated.

Generally, these issues occur at the point of extracting the data from the systems they were recorded in: different systems such as the SAP ERP software [1] support multiple object types and when extracting the data as a traditional event log, the issues arise. If any one of the object types is picked as the single case identifier, the problems of convergence and divergence result in an inadequate representation of the process, since it is not possible to reduce these complex relationships into a traditional event log [16]. By preserving the complete data in the format of an object-centric event log, an accurate image of the process is obtained, which can enable higher quality insights to be obtained with process mining techniques.

[1] https://www.sap.com/products/erp.html

Table 2.3: A fragment of objects in an OCEL example.

object id	object type	object priority	object size	object product	object weight
...	...	...	...	...	...
o_1	order	low	NaN	NaN	NaN
o_2	order	high	NaN	NaN	NaN
i_1	item	NaN	2mx2mx1m	small box	NaN
i_2	item	NaN	2.4mx2.3mx1.9m	medium box	NaN
i_3	item	NaN	2mx2mx0.5m	small box	NaN
i_4	item	NaN	3mx3mx2m	large box	NaN
p_1	package	NaN	NaN	NaN	50kg
...	...	...	...	...	...

2.3.1 OCEL Standard

To aid in exchanging the data from information systems into formats used in process mining, the Object-Centric Event Logs (OCEL) standard specification was introduced [23]. The universes used and applicable examples from Table 2.2 and 2.3 are given in Definition 2.1.

Definition 2.1 (**Universes**)**.** Below are the universes used:

- U_e is the universe of event identifiers.
  Example: {9791, 9792, ...}
- U_{act} is the universe of activities.
  Example: {place order, create package, ...}
- U_{att} is the universe of attribute names.
- U_{val} is the universe of attribute values.
- U_{typ} is the universe of attribute types.
- U_o is the universe of object identifiers.
  Example: $\{i_1, o_1, p_1, \dots\}$
- U_{ot} is the universe of objects types.
  Example: {package, order, ...}
- U_{timest} is the universe of timestamps.
  Example: {2020-9-16 14:39, ...}

Using the universes, an object-centric event log is defined in Definition 2.2.

Definition 2.2 (**Object-Centric Event Log**)**.** An object-centric event log is a tuple $L = (E, AN, AV, AT, OT, O, \pi_{typ}, \pi_{act}, \pi_{time}, \pi_{vmap}, \pi_{omap}, \pi_{otyp}, \pi_{ovmap}, \leq)$ such that:

- $E \subseteq U_e$ is the set of event identifiers.
- $AN \subseteq U_{att}$ is the set of attributes names.
- $AV \subseteq U_{val}$ is the set of attribute values (with the requirement that $\text{AN} \cap \text{AV} = \emptyset$).
- $AT \subseteq U_{typ}$ is the set of attribute types.

- $OT \subseteq U_{ot}$ is the set of object types.
  Example: the object type *item* exists in Table 2.2
- $O \subseteq U_o$ is the set of object identifiers.
  Example: the object identifier p_1 exists in Table 2.2
- $\pi_{typ} : AN \cup AV \rightarrow AT$ is the function associating an attribute name or value to its corresponding type.
- $\pi_{act} : E \rightarrow U_{act}$ is the function associating an event (identifier) to its activity.
  Example: the first event in Table 2.2 had activity *place order*
- $\pi_{time} : E \rightarrow U_{timest}$ is the function associating an event (identifier) to a timestamp.
  Example: the first event in Table 2.2 had timestamp 2020-9-14 11:37
- $\pi_{vmap} : E \rightarrow (AN \not\rightarrow AV)$ such that

  $$\pi_{typ}(n) = \pi_{typ}(\pi_{vmap}(e)(n)) \quad \forall e \in E \; \forall n \in \text{dom}(\pi_{vmap}(e))$$

  is the function associating an event (identifier) to its attribute value assignments.
- $\pi_{omap} : E \rightarrow \mathcal{P}(O)$ is the function associating an event (identifier) to a set of related object identifiers.
  Example: the first event in Table 2.2 is related to object types *order, item, package*
- $\pi_{otyp} \in O \rightarrow OT$ assigns precisely one object type to each object identifier.
  Example: the first object in Table 2.3 is object type *order*
- $\pi_{ovmap} : O \rightarrow (AN \not\rightarrow AV)$ such that

  $$\pi_{typ}(n) = \pi_{typ}(\pi_{ovmap}(o)(n)) \quad \forall n \in \text{dom}(\pi_{ovmap}(o)) \; \forall o \in O$$

  is the function associating an object to its attribute value assignments.
  Example: the first object in Table 2.3 has attribute value *low*
- $\leq$ is a total order (i.e., it respects the antisymmetry, transitivity, and connexity properties). A possible way to define a total order is to consider the timestamps associated with the events as a pre-order (i.e., assuming some arbitrary, but fixed, order for events having the same timestamp).

The traditional and object-centric event log standards share a far amount of similarities but are crucially different in how they treat case notions and objects. The previous research in predictive methods for event logs discussed in Chapter 3 has only been conducted with traditional event logs. To better understand the components of the most recent research, an overview of deep learning is given next.

2.4 Deep Learning

Machine learning aims to develop methods to make predictions or decisions after learning on training data. Regression analysis [24], support vector machines [4], and extreme gradiant boosting [25] are models that have performed well for many different predictive tasks, but the current state-of-the-art approaches [8, 10, 12–15] utilize neural networks.

2.4.1 Neural Networks (NN)

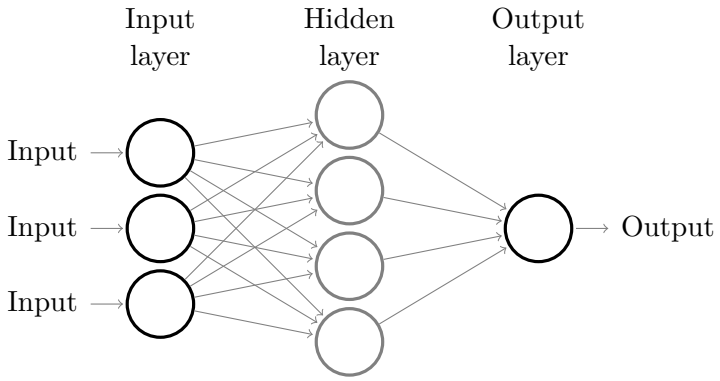


Figure 2.1: An example of a simple feedforward neural network.

A neural network as seen in Figure 2.1 is a network of individual nodes called artificial neurons [26]. Each neuron receives a number of inputs and produces a single output value. Neurons are aggregated into layers and a network usually consists of one input layer, one output layer, and a number of hidden layers in between. The connections between nodes are called edges and have associated weights indicating the relative importance of the connection. The output of an individual neuron is calculated as the weighted sum of all the inputs and passed through an activation function, after which it is fed into a number of neurons. The network is trained on sample data and the weights are adjusted to minimize loss through an algorithm called backpropagation.

Formally, the activation of a neuron is given in Equation 2.1 [27]:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{2.1}$$

Where a_j^l is the activation of the j^{th} neuron in the l^{th} layer for a function σ. The weight matrix w^l consists of the weights connecting to the l^{th} layer. The bias vector b_j^l is the measure of how easy the j^{th} neuron in the l^{th} layer fires.

To update the weights and biases a loss or cost function C is defined. It represents a function used to evaluate the model output. The backpropagation steps include 4 equations which help calculate the error and gradient of the loss function C. The goal is computing the partial derivatives $\partial C/\partial w$ and $\partial C/\partial b$. Where $\odot$ is the elementwise product, the error δ^L of the output layer L is given in Equation 2.2:

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right) \tag{2.2}$$

Where σ is a function and $\nabla_a C$ is a vector comprised of the components of the partial derivative $\partial C/\partial b_j^L$ and z^l is the weighted input to the neurons in layer l. Next the error in terms of the error in the next layer is given in Equation 2.3:

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right) \tag{2.3}$$

The element $(w^{l+1})^T$ is the weight matrix transposed. Now the original goals can be computed; the bias partial derivative as seen in Equation 2.4:

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.4}$$

And the weight partial derivative give in Equation 2.5:

$$\frac{\partial C}{\partial w} = a_k^{l-1}\delta_j^l \tag{2.5}$$

Backpropagation is presented in Algorithm 1 where a mini-batch has m samples and the learning rate is η:

Algorithm 1: Backpropogation (adapted from [27])

```
for Number of epochs do
    for Number of mini-batches do
        foreach Sample x in mini-batch do
            compute a^{x,1};
            foreach l = 2,3, ..., L compute
                z^{x,l} = w^l a^{x,l-1} + b^l;
            end
            a^{x,l} = σ(z^{x,l});
            foreach l = L-1, L-2, ..., 2 compute
                δ^{x,L} = ∇_a C_x ⊙ σ'(z^{x,L});
            end
        end
        foreach l = L, L-1, ..., 2 update
            w^l → w^l − (η/m) Σ_x δ^{x,l} (a^{x,l-1})^T;
            b^l → b^l − (η/m) Σ_x δ^{x,l};
        end
    end
end
```

2.4.2 Recurrent Neural Network (RNN)

As opposed to feedforward neural networks, where edges do not form any cycles, there are cyclic structures used in memory like fashion in an RNN which make them suited well for temporal data [26]. Modeled as a directed graph, they describe a class of networks that

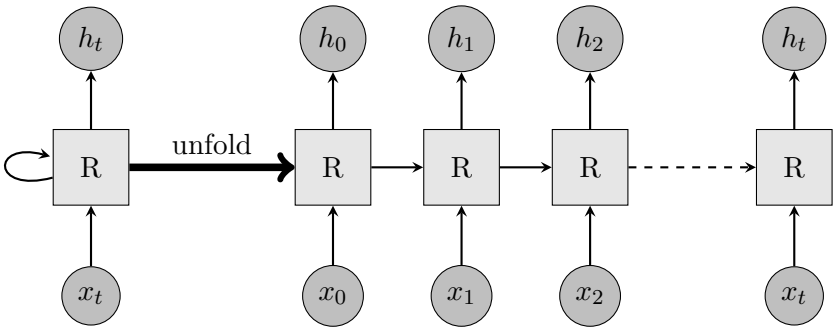


Figure 2.2: Unfolding a recurrent neural network.

can use their internal state to process inputs of any length. This is achieved by unfolding the RNN cell, meaning copies are created for different time steps as seen in Figure 2.2. The input vector into the RNN R is represented as x_t and the output vector as h_t. The output of the cell at t-1 is the input for the cell at time t. This enables the cell to have memory and process inputs of different lengths. However, for long sequences, RNNs can suffer from catastrophic interference where learned patterns are forgotten.

2.4.3 Long Short-Term Memory (LSTM)

To combat the shortcomings of an RNN, the LSTM architecture was introduced [29]. It includes gates that control the flow of data and a memory cell that learns patterns over longer time periods. Intuitively, the input gate determines how much of the new values flow into the cell, the forget gate determines how much of the values will be forgotten, while the output gate controls which values contribute to the output to the next cell. The gates can be seen as the standard neurons discussed before, which compute some activation over the sum of weighted inputs.

As seen in Figure 2.3, the input gate activation vector is i_t, the output gate activation vector o_t, and the forget gate activation vector f_t. The activation functions are σ_g for the sigmoid function and σ_h for the hyperbolic tangent function. The input vector is x_t and h_t represents the hidden state or output vector. The cell state or memory vector is represented by c_t and the cell input activation vector by $\tilde{c}_t$. Finally W and U represent the weight matrices and b any bias that the training step learns. The LSTM cell updates six parameters in each time step as listed in Equation 2.6.

$$\begin{aligned} f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) & \tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\ i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) & c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) & h_t &= o_t \circ \sigma_h(c_t) \end{aligned} \tag{2.6}$$

2.4.4 Generative Adversarial Network (GAN)

While LSTM's were able to demonstrate good results in predicting the next event, they require a lot of labeled training data to be able to generalize well [14]. Most publically available real-life event logs are limited in size, which inhibits the model's ability to perform better. Introduced by Goodfellow et al. in 2014 [30], GANs employ a zero-sum game with two player's, meaning one player's gain is offset by the other player's loss. One

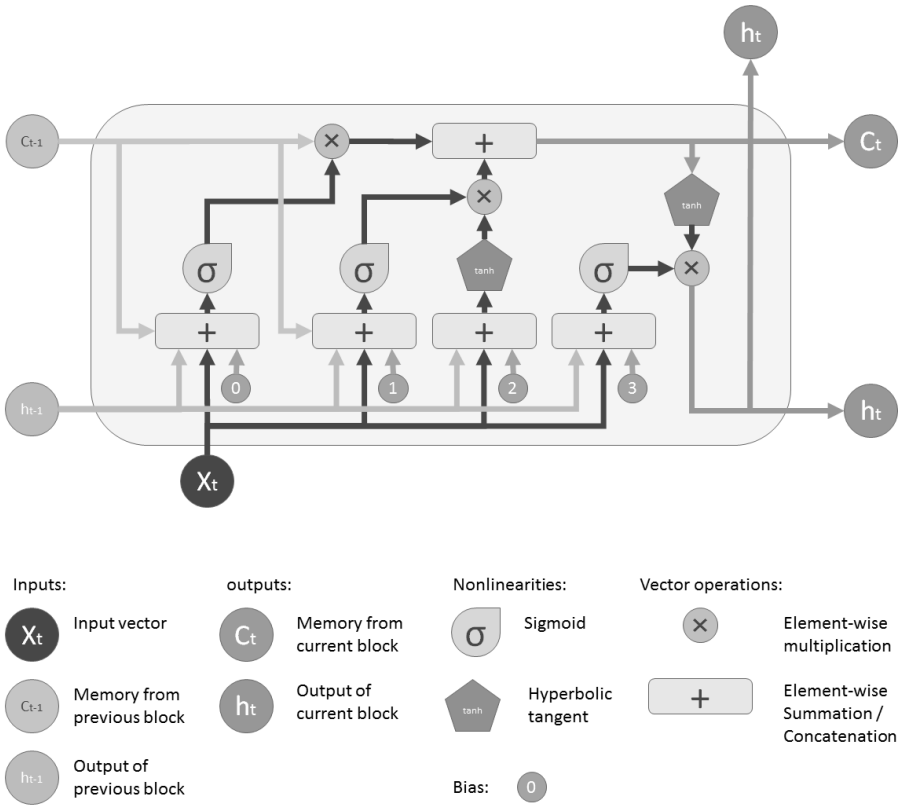


Figure 2.3: LSTM building block [28] (Liscensed under GPL-3.0).

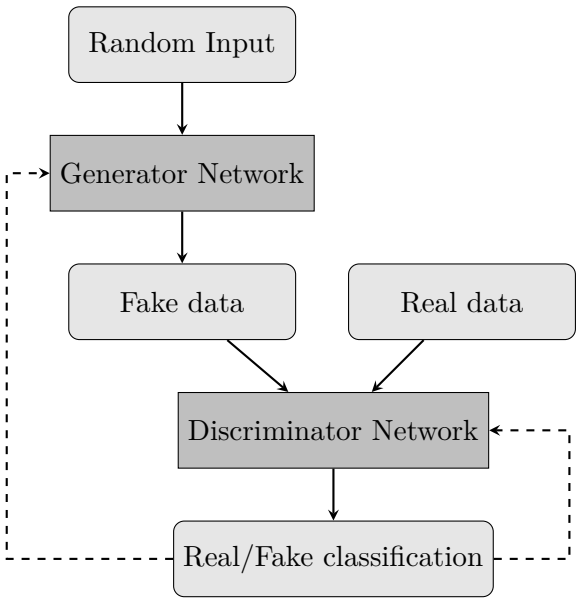


Figure 2.4: Generative adversarial network architecture with random input.

player is called the generator and tries to create new data that is similar to the training data, and the other player is the discriminator who tries to distinguish between real and fake data. Each player tries to maximize their outcome through separate training with backpropagation, which means the generator tries to produce more convincing fake data and the discriminator tries to get better at detecting that fake data. Ideally, this converges to the point where the generator creates near-perfect fakes and the discriminator is no better than 50% at predicting the truth. As seen in Figure 2.4 the generator is fed with random data, so early on in the training, the fakes are poor representations for the real data and the discriminator can easily distinguish them. But with the feedback from the discriminator, the generator can keep improving. Radford et al. [31] standardized deep convolutional generative adversarial networks (DCGANs) which have been used among other things, to create realistic human faces and upsample low-resolution images.

2.4.5 Sequence to Sequence Model (Seq2seq)

Sequence to sequence (Seq2seq) models were first introduced in 2014 [32]. The main problem they address is how to map one sequence to another. This occurs in many domains, for example, machine translation, where a sentence in one language will have a different length in another language. An RNN or an LSTM can not produce variable-length outputs, so Seq2seq was proposed. There is an encoder/decoder architecture central to the model: the encoder is an LSTM that turns the input into a fixed hidden vector and the decoder is another LSTM that transforms the hidden vector into an output.

Chapter 3

Related Work

In this section, an overview is given for past approaches in predictive process mining, starting with process model based methods and ending with the state-of-the-art deep learning models.

3.1 Process Model Based Methods

Process mining is the intersection of data science and process science [1]. Discovering process models to describe past events has been a topic of research since the 1990's, when to better understand the execution of business practices an algorithm was developed to generate a graph model from a log of activities [33]. The proposed algorithm creates process models and can deal with cycles in processes and erroneous activities. Notable improvements were made with the α-miner [34] which can find a Petri net model from an event log. Furthermore, the heuristic miner [35] can better deal with noisy or incomplete data. This represents process discovery, where a process model is generated based on event logs. The model can be visualized to show the workflow between activities.

Apart from performing post-hoc analysis using automatically generated process models, the use case can be extended to provide real time operational support of running cases, such as the prediction of the next activity. The prerequisite for the prediction is a process model obtained from process discovery methods. Le et al.[36] extends higher order Markov models with a sequence alignment technique to predict the next event. With no formal description of the underlying processes, the method utilizes probability matrices for the transition from one event to another. The assumption that similar sequences are likely to produce the same outcome enables better results in terms of prediction accuracy. Lakshmanan et al.[37] also first mines a process model from the event log. Then at each split in the model, a decision tree is learned from execution data and state transition probabilities computed. Based on an ongoing case a Hidden Markov Model is used to predict the likelihood of a future event. Parallel executed tasks are supported and the approach is more accurate as demonstrated on a simulated auto insurance event log. Breuker et al.[10] recognizes the potential biases in models such as Petri nets and proposes using probabilistic finite automaton to predict the next event. Based on research in the field of grammatical interference, a probabilistic model of the event data is learned and used in next event predictions.

All of these approaches deal with discovering process models from logs based on the control

flow of activities where based on an ongoing case the next event is predicted. However, with most information systems saving timestamp data, the models mined from event logs can be enriched with timestamp data. Then the time until the next event can be predicted, or the remaining time for the whole case, increasing the usefulness of the predictions.

The time to completion of a case is predicted using non-parametric regression in [24]. Occurrences of activities in cases, activity durations, and case data are combined in the implementation. Compared to the naive approach of subtracting the elapsed time from the total average case time, the proposed methods perform better in real event logs. The process mining techniques learned from process discovery are applied to time prediction in [22]. A process model is mined from event logs and augmented with time information. The annotated transition system predicts completion time based on an ongoing case, thereby providing operational support. The approach is implemented in the process mining tool ProM [1] and outperforms simple heuristics on both synthetic and real event logs. In order to support dynamic business processes, ad-hoc predictive clustering is used in [9]. By distinguishing patterns in event logs as distinct process variants, clusters are discovered. An ongoing case is estimated to belong to a specific cluster and then the remaining processing time is predicted. The approach outperforms previous methods on a real event log and is able to foresee service level agreement violations. In [38] the authors create Petri net models to predict remaining time. As opposed to state transition systems, the approach can model concurrency and time passed is taken into consideration. The largest gains in predicting accuracy are observed in longer running cases and it matches previous approaches in shorter cases.

A unifying theme to the approaches presented so far is the reliance on some form of process model mined through process discovery. Such methods can not be used when process models are too difficult to obtain. Additionally, process models can be imperfect abstractions of the underlying process, trading off precision for simplicity, therefore, making the predictions only as good as the process model [8]. Authors in [39] completed an empirical evaluation of remaining case time prediction methods as of 2019. They concluded that LSTM Neural Networks achieved the most accurate results in terms of Mean Average Error, with the tradeoff of significantly more computational resources being required to train the models as opposed to transition systems. Recently, machine learning approaches and specifically deep learning methods have been developed which outperform previous work for both next event and time prediction [13, 14]. As opposed to process model based techniques, they do not use an explicit representation of a process model and instead rely on learning features from the ground up to make predictions. The large amounts of trainable parameters and inherent non-linearity in deep neural nets have proven to be advantageous in predictive tasks [8].

3.2 Deep Learning Based Methods

Evermann et al.[8] first used deep learning in process prediction by predicting the next activity. Inspired by research using Recurrent Neural Networks (RNN) in Natural Language Processing, the approach is based on LSTM layers and encodes categorical variables. The approach is tested on multiple real life event logs and is able to surpass previous research [37] [10] in many cases. However, the encoding of attributes in an embedding space limits

[1] https://www.promtools.org/

the approach to event logs with a small number of unique activities and numerical values are not utilized. In [12] sequences of events are mapped into 2D data structures similar to images and used to train Convolutional Neural Networks which predict the next activity. While performing well in terms of precision and recall on multiple real datasets, less frequent activities are not correctly predicted.

Tax et al. [13] combines the challenge of predicting the next event and the time for that event into one architecture utilizing LSTMs with two hidden layers. One-hot encoding was used for categorical data and timestamps were augmented so that business hours were respected in the prediction. The previous research is surpassed in terms of accuracy of the predicted next event and Mean Absolute Error (MAE) of the predicted timestamp on multiple real datasets. Furthermore, by continuously predicting the next event until the end of a case, this approach can also predict the entire remaining case and the total time to completion, where it also exceeds previous approaches. Carmangoe et al. [40] combine the work of [8] and [13] to predict the next event and its timestamp. Categorical attributes are embedded in an n-dimensional space where coordinates correspond to unique categories and numerical attributes normalized to reduce variability. Again two LSTM layers are utilized and different architectures for sharing categorical attributes are explored.

Taymouri et al. [14] first introduced the idea of using Generative Adversarial Nets (GAN) for next event and timestamp prediction in event logs. By having two LSTM networks play a zero-sum game against each other the generator becomes better at creating fake next events over a number of iterations. While normally the generator receives a random vector from a Gaussian distribution as input, the approach uses the ongoing case as the input. The predicted next event from the generator is added on the ongoing case to form the fake prefix and the real next event is added to the ongoing case to form the fake prefix. These are fed into the discriminator which returns a probability of the input being real, which in turn is used as feedback for the generator. The generative approach allows the network to make better generalizations about event logs and requires less training data to produce good results. Activities are one-hot encoded and events are augmented with the elapsed time between the last event. In both predicting the next event and predicting the timestamp of the event, the approach was able to significantly outperform previous approaches on multiple real event logs.

For predicting events further in the future, some of the previously discussed methods simply execute the prediction consecutively, using the output from the previous run as input for the current one. While for shorter average case lengths the results can be good, such methods fare poorly for longer remaining process executions since the errors propagate. In [15] this is addressed by predicting the entire remaining events and timestamps from an ongoing case instead of just the next event. Sequence to Sequence models are integrated into the GAN architecture from [14] to map an input prefix to an output suffix. By utilizing the encoder/decoder structure for the generator, variable length outputs can be achieved. The generator attempts to produce convincing suffixes and the discriminator provides feedback by returning the probability the input is a real suffix for a given prefix. On several real event logs, the approach performed better in terms of suffix similarity and error in the predicted timestamps.

3.3 Object-Centric Event Logs

All the presented methods have worked with traditional event logs, but as [16] points out, real life processes frequently are too complicated to reduce to a single case notion. Instead, there are multiple identifiers that refer to different views of the event log and it is not possible to in general identify a single case identifier. Therefore object-centric event logs were introduced to more accurately capture the relations in real world data. Past research in object-centric event logs has been in formalizing the standard [2] and process discovery [16]. Additionally, there have been techniques proposed in extracting object-centric event logs from information systems [41, 42] and automated event log building [43]. Furthermore, visualizing and modeling the processes has been researched [44–47]. However, the previously discussed prediction methods rely on traditional event logs with a single case notion to calculate a trace for a case and train the predictive models. Since object-centric event logs do not force a single case notion, selecting an appropriate object type is required first. In this thesis, this is made easily understandable by showing the relations between activities and object types as well as statistics about the different object types. Furthermore, the object attributes contained in OCEL can be useful features in predictive tasks. While some past research has utilized the analogous case attributes in traditional event logs [8], the more recent state-of-the-art models [15] forgo using this information in predictions.

Therefore, this thesis augments the state-of-the-art GAN with sequence to sequence models approach [15] with object attributes from object-centric event logs (OCEL). The idea is that certain object attributes can improve the model, for example, an object attribute weight for an object of object type package might indicate faster or slower shipping time. This information can be learned by the model to provide more accurate predictions. Furthermore, few past research makes the predictive models easily accessible, therefore, this thesis designs a web interface for easy training and prediction of OCEL data.

[2] http://ocel-standard.org/

Chapter 4

GAN for OCEL prediction

In this section, the proposed methods are introduced. First, the necessary data preprocessing steps and encoding strategy are explained. Second, the sequence to sequence model architecture for the generator is detailed. Last, the GAN architecture as a whole is discussed.

4.1 Data Preprocessing

For this thesis, object-centric event logs conforming to the OCEL standard [23] are used. The standard specifies data to be stored in the XML or JSON format and provides library support for importing and exporting files in Python. An imported OCEL dataset consists of two tables, one for the event data (Table 2.2) and one for the object data (Table 2.3). In the event dataset, object types have relations with activities. The relations found in the example event data in Table 2.2 are visualized in Figure 4.1.

As seen in Figure 4.1, not every activity is related to every object type. This will vary from dataset to dataset, so it is important to visualize the relationships. For example, as seen in Figure 4.1, only the activities *pack items, load package*, and *deliver package* can be predicted if object type *package* is selected, since those are the only activities that are related to that object type. Therefore, depending on the use case of the prediction, the appropriate object type should be selected. After selection of the object type *package*, the object Table 2.3 contains the object attribute *object weight* which will be used in the prediction.

The event Table 2.2 contains the familiar columns *event id*, *activity* and *timestamp*. There can also be additional event attributes. On the other hand, the object Table 2.3 has the identifier *object id* for the specific object along with the corresponding *object type*. Furthermore, every object attribute is listed. For prediction in the model, only specific data is used: while the *event id* is necessary to organize the cases and create training or testing data, it is not used directly in the prediction. Rather, the *activity*, *timestamp*, and any object attributes are the only values that are used directly in the machine learning model and therefore need data preprocessing. Two types of data occur in the OCEL; numerical data such as the *timestamp* and categorical data such as the *activity*. These require different encoding so that the model can handle the data.

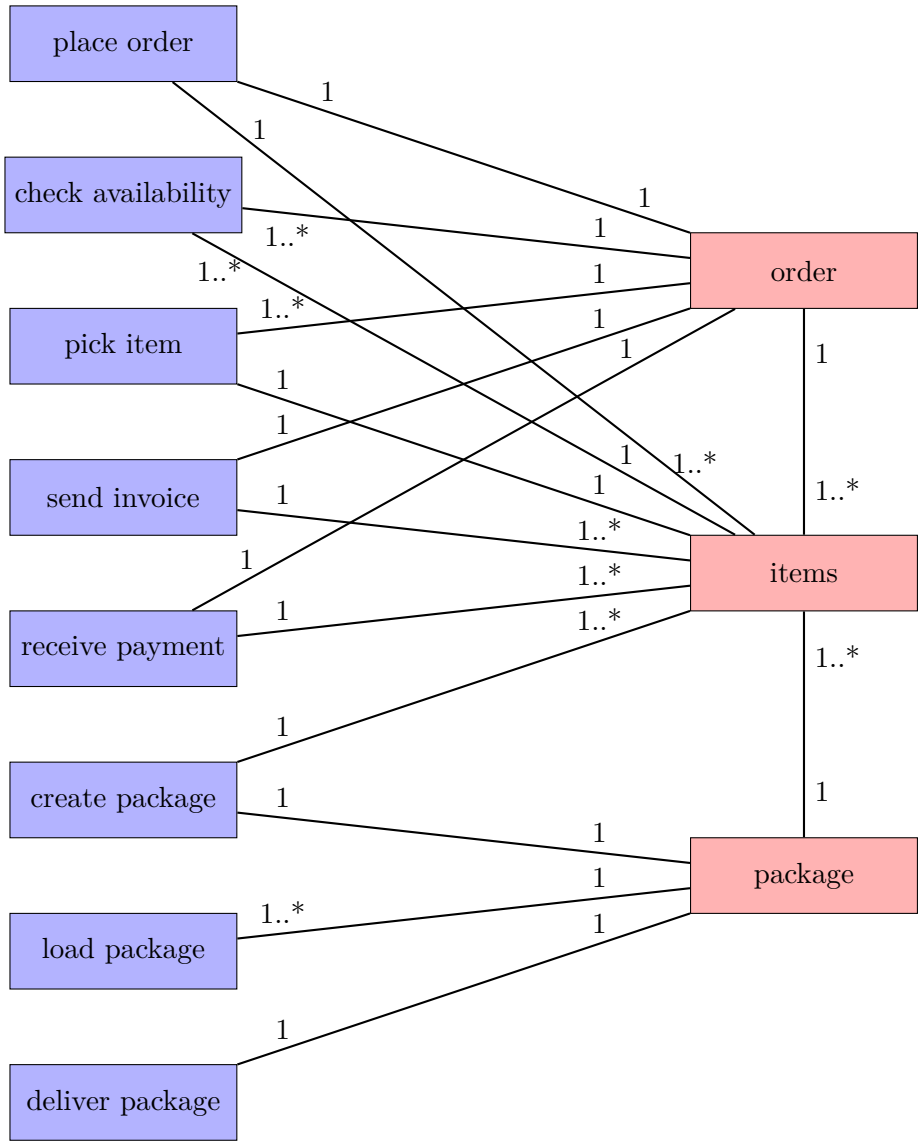


Figure 4.1: Relations in Table 2.2 between activities and object types (adapted from [16]).

Numerical Data

The timestamp of an event occurring is a necessary features for the model, yet needs conversion to be useful. Instead of utilizing absolute dates and times, the relative time elapsed between an event and the preceding event are calculated, as seen in Table 4.1 :

Table 4.1: Example events with calculated elapsed time column.

event id	event timestamp	elapsed (s)
281	2019-05-20 09:07:47	0
282	2019-05-20 09:17:26	579
283	2019-05-20 11:53:12	9346

Additionally, the values are normalized as described in Equation 4.1 so that the model does not learn false relations about the absolute value of elapsed time.

$$\text{normalized value} = \frac{\text{value} - \text{minimum value}}{\text{maximum value} - \text{minimum value}} \tag{4.1}$$

This data can be encoded in other ways, but to show the approach in this thesis performs well independent of a special encoding, the simplest one was chosen. The *event timestamp* needs to be encoded in this manner, as well as any other numerical object attributes, such as *object weight* in Table 2.3.

Categorical Data

Categorical data such as *event activity* needs to undergo multiple transformations to become usable in a machine learning model. First, the data is *label* or *integer* encoded. This refers to converting values to a number, for example:

$$\begin{aligned} \text{place order} &\longrightarrow 1 \\ \text{confirm order} &\longrightarrow 2 \\ \text{pay order} &\longrightarrow 3 \end{aligned}$$

This alone makes categorical data usable in a model, however, the encoding can be misinterpreted. A model might learn a false hierarchy in the encoding from the natural order in the numbers. This can lead to the model learning that 2 must come before 1, or because 2 is larger, it must occur more frequently than 1. These common unintended consequences can be negated by encoding the numbers obtained using *one hot* encoding.

One hot encoding refers to converting the integers to a representation similar to a binary encoding, where each category is represented by a column, as seen in Table 4.2:

Table 4.2: Example activites one-hot encoded.

event activity	place order	confirm order	pay order
place order	1	0	0
confirm order	0	1	0
pay order	0	0	1

The *place order* activity would therefore be encoded as $\langle 1, 0, 0 \rangle$. The *event activity* needs to be one hot encoded as well as any categorical object attributes used in the prediction, such as *object product* in Table 2.3.

Vector Representation

Definition 4.1 (Event vector). Given an event $e_i \in E$ with the activity a_i where $\pi_{act}(e_i) = a_i$ with the timetsamp t_i where $\pi_{time}(e_i) = t_i$, and categorical attributes $C_i \subseteq ATT$ where $\forall c_i \in C_i \; \pi_{typ}(c_i) = string$ and numerical attributes $N_i \subseteq ATT$ where $\forall n_i \in N_i$ $\pi_{typ}(n_i) = float$:

- categorical data a_i and $c_i \in C_i$ is one-hot encoded as $enc(a_i)$ and $enc(c_i)$

- elapsed time l_i is calculated from the timestamp t_i and t_{i-1} where $\pi_{time}(e_{i-1}) = t_{i-1}$

The resulting event vector for event e_i is $\langle enc(a_i), enc(c_i), n_i, l_i \rangle$

After normalizing the appropriate numerical values and encoding the categorical values, the event vector from Definition 4.1 can be built. As an example, take an event with an activity, both categorical and numerical object attributes, and a corresponding elapsed time. That example event is then represented as a vector:

$$\text{Event vector: } \langle \underbrace{0,1,0,0}_{\text{activity}}, \underbrace{0,0,0,0,1,0,0,0,0,0}_{\text{categorical attributes}}, \underbrace{0.301,1,0.5,0.5818}_{\text{numerical attributes}}, \underbrace{0.04818532}_{\text{elapsed}} \rangle$$

In previous research [15] only the activity and elapsed time is encoded in the event vector. Object attributes that might have been present in the underlying process and stored in information systems, typically are not preserved in traditional event logs, therefore, they can not be used in the prediction. By leveraging the data contained in OCEL, this thesis augments the event vector with object attributes, which if the features are useful in the prediction, can increase the accuracy.

A sequence of events necessitates an end of sequence (EOS) bit, to signal that the end of the case has been reached. Therefore, the final event is simply zeros and sets the EOS bit. An example case can be:

$$\text{Case: } \langle \langle e_1, 0 \rangle, \langle e_2, 0 \rangle, \langle e_3, 0 \rangle, \langle 0, 1 \rangle \rangle$$

Prefix/Suffix Split

Definition 4.2 (**Prefix and Suffix**)**.** Given a non-empty sequence of n events $\langle e_1, e_2, e_3, \ldots e_n \rangle$ ordered based on the timestamp and related to an object $o \in O$ where $for\ 1 \leq i \leq n$: $o \in \pi_{omap}(e_i)$, a prefix is $\langle e_1, \ldots, e_k \rangle$ for some $k \leq n$. Then a suffix is the remaining events $\langle e_k, \ldots, e_n \rangle$.

When making a prediction, the input is an ongoing process called the *prefix* and the goal is to predict the remainder of the process called the *suffix* as defined in Definition 4.2. When preparing the OCEL data, completed processes need to be split up into prefix/suffix pairs to train the model. For a given process with five events e_i for $1 \leq i \leq 4$, all possible splits are created in Table 4.3:

Table 4.3: Making example case into prefix and suffix splits.

prefix	suffix
e_1	e_2, e_3, e_4
e_1, e_2	e_3, e_4
e_1, e_2, e_3	e_4

This maximizes the amount of training data available and enables the model to predict suffixes from both short and long input prefixes.

4.2 Generator

As seen in Section 4.1, the model needs to be able to take variable length sequences (prefixes) as an input and produce variable length sequences (suffixes) as an output. As

discussed in Chapter 2, this is achieved with an encoder/decoder architecture used in Sequence to Sequence models [32], which are the foundation of the generator. Intuitively the encoder returns a vector based on the input sequence and the decoder takes that as input and returns the output sequence.

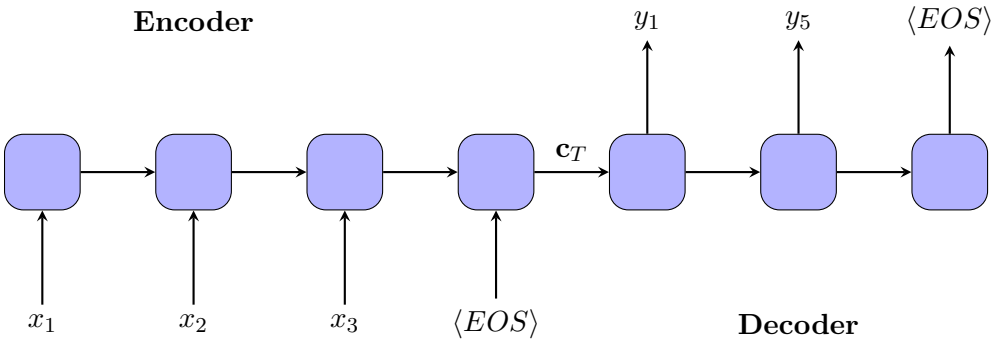


Figure 4.2: Encoder/Decoder seq2seq model mapping $\langle x_1, x_2, x_3 \rangle$ to $\langle y_1, y_2 \rangle$.

Encoder

As illustrated in Figure 4.2 the input sequence is a vector containing event vectors and the $\langle EOS \rangle$ to show the end of the sequence. This is the prefix; the sequence for the ongoing process execution. The encoder is an LSTM network which processes the prefix input sequence $x_1, \ldots, x_t$ of length T. Recall from Definition 2.6 the cell state vector c_t:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

This is a summary of the input sequence for a timestep. The LSTM cell is updated T times, meaning the six equations in Definition 2.6 are calculated, and the final cell state vector $\mathbf{c}_T$ for the entire prefix is given. As shown in Figure 4.2 this is directly passed into the decoder where it is used as the initial cell state.

Decoder

The decoder is another LSTM network that starts generating the output sequence $y_1, \ldots y_{T'}$. Crucially, the output sequence length T' can be different from the input sequence length T. In each of the T' decoder updates, the output y_{t-1} from the previous update is used as the input for the current update.

Conditional Probability

The goal of the generator therefore, is to estimate the conditional probability of the output sequence (suffix) $y_1, \ldots, y_{T'}$ given an input sequence (prefix) $x_1, \ldots, y_T$:

This probability given the encoder output $\mathbf{c}_T$ is formally [32] given in Equation 4.2:

$$p(y_1, \ldots, y_{T'} \mid x_1, \ldots, x_T) = \prod_{t=1}^{T'} p(y_t \mid \mathbf{c}_T, y_1, \ldots, y_{t-1}) \tag{4.2}$$

4.3 Gumble-Softmax

GAN architectures require differentiable data in order to update the generator. However, the vectors that are created from the event logs have some categorical data in them as well, which when sampled from, are not differentiable. As demonstrated in [48], a sample from a categorical distribution can be replaced with a differentiable sample from a Gumbel-Softmax distribution.

Let z be categorical data with the class probabilities $\pi_1, \pi_2, \ldots, \pi_k$. These are one hot encoded as described before. With g_i being the samples from Gumbel(0,1) the Gumbel trick as defined by [49] is given in Equation 4.3:

$$z = \text{one hot}\left(\underset{i}{argmax}[g_i + log(\pi_i)]\right) \tag{4.3}$$

However, the function returns non differentiable values so Softmax is used to obtain a differentiable approximation to arg max for y_i where τ refers to the temperature, as seen in Equation 4.4:

$$y_i = \frac{exp((log(\pi_i) + g_i)/\tau)}{\sum_{i=1}^{k} exp((log(\pi_j) + g_j)/\tau)} \tag{4.4}$$

4.4 GAN Architecture

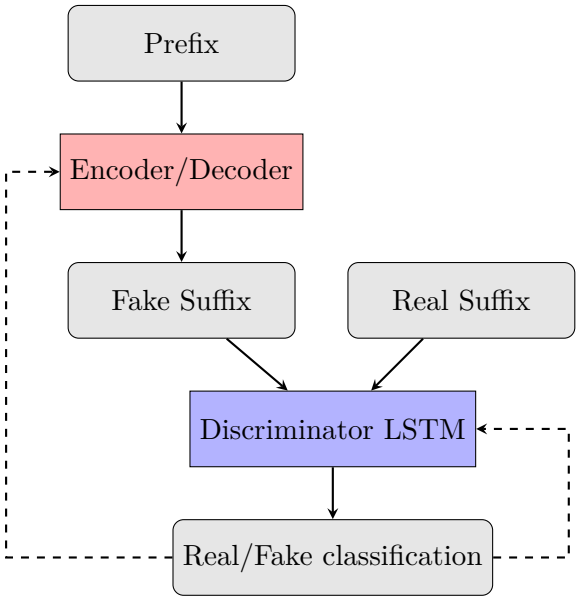


Figure 4.3: GAN architecture for OCEL prediction.

The generator of the GAN consists of an encoder and decoder and receives a prefix as input. Conventional GANs use random noise as the input, but since the goal is to make predictions based on an ongoing process, a prefix is used instead. The generator returns a fake suffix, which is compared to the real suffix by the discriminator, as seen in Figure 4.3. The discriminator consists of an LSTM and a fully connected layer and returns a probability representing if the fake suffix resembles a real suffix.

For a prefix pre, suffix suf, generator G, and discriminator D, the objective functions based on [50] [15] are defined in Definition 4.3:

Definition 4.3 (**Generator and Discriminator objective functions**).

$$\mathcal{L}(D;G) = -log(D(suf)) - log(1 - D(G(pre)))$$

$$\mathcal{L}(G;D) = -log\left(\frac{D(G(pre))}{1 - D(G(pre))}\right)$$

First, the discriminator D is updated by minimizing the loss $\mathcal{L}(D;G)$ while the generator stays fixed. When the discriminator miss-classifies a suffix, the discriminator loss penalizes the discriminator. This occurs through backprapogation, where the weights of the network are adjusted through gradient descent.

Next the generator G is updated by minimizing the loss $\mathcal{L}(G;D)$ while the discriminator stays fixed. The generator is penalized when it produces suffixes that the discriminator is able correctly classify. Backpropagation occurs through both the discriminator and generator so the gradients can be calculated, but only the generator's weights are adjusted in this step.

Chapter 5

Implementation

In this section the implementation of Chapter 4 is detailed for the data preprocessing steps and model initialization and training. Furthermore, hyperparameter tuning of the model is discussed, the web interface is described and the deployment is explained.

5.1 Extracting RCLL dataset

The RCLL is a simulation of factory logistics discussed in more detail in Chapter 6. The events that occur are saved in a database with some unecessary for this thesis information about the route, distance, and more. The relevant event information of indentifier, activity, and timestamp is extracted. The object types of robot, order, and components are extracted as well and the event table of the OCEL standard can be created. The attributes for the objects are extracted to create the object table. The final dataset is stored according to the OCEL standard in the JSON file format.

5.2 Data preprocessing

The data preprocessing module handles the steps discussed in Chapter 4 as well as some common additional functions. The module was written in Python 3, using the pandas data analysis library [1] and PyTorch machine learning library [2].

Importing OCEL
The pm4py-mdl python package [3] is utilized to import the OCEL data. Reading both XML and JSON files as per the OCEL standard is supported and the import methods return two pandas dataframes. One dataframe represents the events and is similar to Table 2.2 and the other represents the objects and is similar in form to Table 2.3.

Elapsed Time Calculation
The elapsed time is calculated for individual cases in the events dataframe. The elapsed time for the first event in a case, therefore, is zero and for every subsequent event, it is the time that passed after the event right before.

[1] https://pandas.pydata.org/
[2] https://pytorch.org/
[3] https://github.com/Javert899/pm4py-mdl

One-hot Encoding
The categorical values including activity names and categorical object attributes are one-hot encoded and the mapping is saved for later reversal. This mapping is also important when predicting for a new prefix, there the categorical values are encoded with this mapping so the model can interpret the data correctly.

Normalizing
Numerical values used in the prediction are normalized and the terms of the normalization saved. When using the model in prediction these terms are needed to reverse the normalization and interpret the results in absolute timestamp terms.

Partitioning
When a mini-batch has randomly chosen inputs, it can have a large amount of variability in the input lengths. While the model can be updated more frequently using Mini-Batch Gradient Descent as shown in Algorithm 1, the downside is that different length cases cause slowdowns in the computations, as observed in [32]. Therefore, the partitions are groups of cases with a similar length, which can be batched together.

Variable Length Splits
Within the created partitions, traces are split into every possible combination of lengths for the prefix and suffix. The prefixes and corresponding suffixes are saved as PyTorch tensors. The tensors are saved as TensorDatasets in DataLoader objects. This enables easy access during training.

5.3 Model

The generator consists of two parts; the encoder and the decoder. The encoder is an LSTM network which maps the inputs to the hidden vector. The decoder is also an LSTM network connected to a fully connected layer and then passed through ReLU and Sigmoid activation functions. A concept called teacher forcing [51] is used in the generator every so often based on randomness. When it is triggered, instead of using the output from the decoder from the previous time step as the input for the current one, the real suffix is used as the input. This can lead to faster training and better results. The discriminator is an LSTM network connected to a fully connected layer.

5.3.1 Training

Training occurs based on the Algorithm 2.

The gradients need to be set to zero in lines 3 and 7 because PyTorch accumulates the gradients on the backward passes. Exploding gradients can occur when large updates to weights cause numerical under or overflow, which happens in LSTM networks since they need to unroll many timesteps as discussed in Chapter 2. The solution to that is gradient norm clipping seen in lines 6 and 9.

5.3.2 Validation and Testing

In the data preprocessing step, the main OCEL data is split into training, validation, and testing sets. Training occurs with the training data, which represents 70% of the total data. As seen in Algorithm 2, during training the model is evaluated on validation data

Algorithm 2: Training GAN

```
for Number of epochs do
    for Number of mini-batches do
        for prefix, suffix in training data do
            forward pass of generator and discriminator;
            zero discriminator gradients;
            update the discriminator to minimize loss;
            clip discriminator gradient norm;
            zero generator gradients;
            update the generator to minimize loss;
            clip generator gradient norm;
        end
        if epoch multiple of 5 then
            run model on validation data;
            save model if beats current best;
        end
    end
end
```

periodically. Validation data represents 10% of the total data. Last, there is the testing data with 20% of the total data. The model is benchmarked using either validation data or testing data in the evaluation function. The model is set to evaluation mode to prevent weights from being updated and a forward pass of the model (generator) is performed. The fake suffix returned from the generator is compared to the real suffix in two metrics, which are discussed in Chapter 6.

5.4 Hyperparameter Tuning

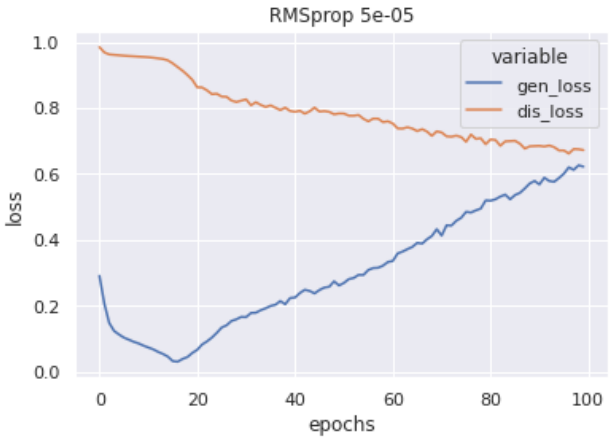


Figure 5.1: Convergence of generator and discriminator loss on synthetic dataset.

Convergence is difficult for GANs due to the nature of the adversarial game; gains by the generator result in losses by the discriminator and vice versa. Ideally, the generator

improves to the point where the discriminator has a 50% accuracy. Then, the loss calculated for each should be about the same, as seen in the convergence graph in Figure 5.1. This can be problematic since the discriminator continues giving feedback to the generator and the resulting fake suffixes can decrease in quality. As a result, the convergence might be shortlived. Training can fail because of model collapse, where the generator can only output a small number of different suffixes. Convergence failure is also possible, where the generator loss keeps increasing while the discriminator loss is near zero, meaning the generator produces very poor quality suffixes that the discriminator can easily identify.

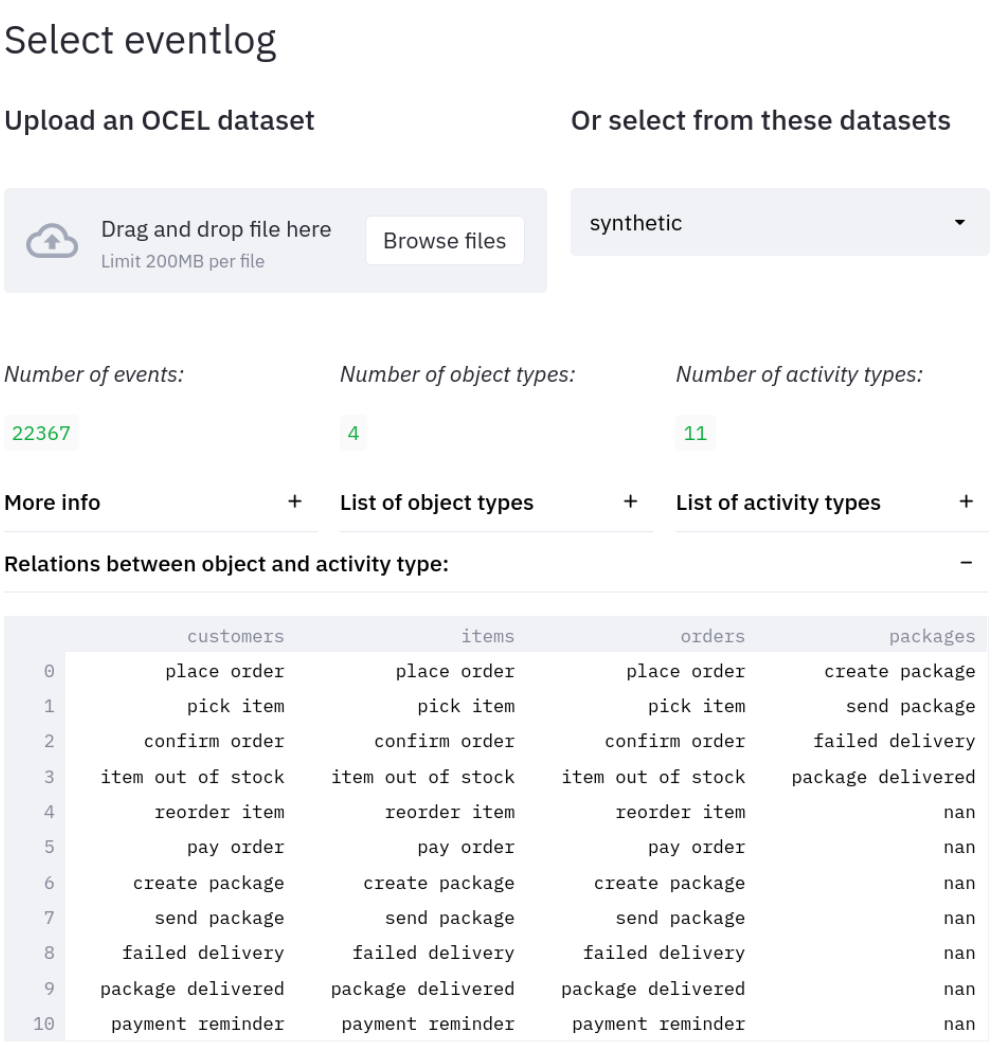
The model has multiple hyperparameters: learning rate, type of optimizer, and the number of layers. The optuna python library[4] was used to aid in empirically finding the best hyperparameters. The hyperparameters are suggested by optuna based on a given range and then multiple trials are conducted to find the best settings. Optuna searches the range of values efficiently using the tree-structured Parzen Estimator algorithm and can prune trials that are performing poorly to avoid wasting resources. The algorithm repeatably decreases the search space, first searching in the range suggested. Based on the evaluation of the model with those hyperparameters, the range is narrowed, leading to an optimal search space. The best hyperparameters found through tuning were: the RMSprop Algorithm for the optimizer, a learning rate of $5.5e^{-5}$, and five layers for the LSTM networks. The values were similar to the options used in past research [15].

5.5 Interface

The web interface is implemented in Python using the streamlit library [5]. As seen in Figure 5.2, the first main section of the interface is *Select eventlog*. There either an event log in the OCEL format can be uploaded, or one of the preloaded datasets picked from the dropdown menu. Next, there is an overview of common statistics regarding the activities and object types present in the data. The choice of an object type to use in the prediction is vital, therefore, a table visualizing the relations between activities and object types is presented. The model can then be trained based on the selected object type and next the prediction starts.

As seen in Figure 5.3, the section for entering a prefix (ongoing case) starts with selecting the object type. Based on that information, the dropdown menu *activity* under "Enter event information" is populated by all the activities that the object type has a relation to. After selecting the activity, the timestamp information and identifier for the object can be entered. By choosing the "Save Event" button the event can be saved and is presented in the table under "Process execution". From there, more events can be created and saved to create a prefix for which the prediction should be performed on. Under the section "Enter object information" the fields are automatically retrieved from the OCEL log, presenting only the object attribute classes that are related to the object type. There, the desired attribute values can be entered. That concludes the creation of a prefix with both event and object attributes. Subsequently, the interface presents the predicted suffix for the prefix that was entered, using the trained model.

[4] https://optuna.org
[5] https://www.streamlit.io/



	customers	items	orders	packages
0	place order	place order	place order	create package
1	pick item	pick item	pick item	send package
2	confirm order	confirm order	confirm order	failed delivery
3	item out of stock	item out of stock	item out of stock	package delivered
4	reorder item	reorder item	reorder item	nan
5	pay order	pay order	pay order	nan
6	create package	create package	create package	nan
7	send package	send package	send package	nan
8	failed delivery	failed delivery	failed delivery	nan
9	package delivered	package delivered	package delivered	nan
10	payment reminder	payment reminder	payment reminder	nan

Figure 5.2: Interface section for selecting the event log and viewing information.

5.6 Deployment

To deploy the model and the interface easily, the container-orchestration system Kubernetes [6] is used. The interface setup and required dependencies are defined in a Dockerfile and a Docker image [7] is built. This represents the information required to configure a container in Kubernetes using the Deployment object. The Kubernetes cluster can then automatically create replica pods based on current load requirements and the interface is exposed by the Service object.

[6] https://kubernetes.io/
[7] https://www.docker.com/

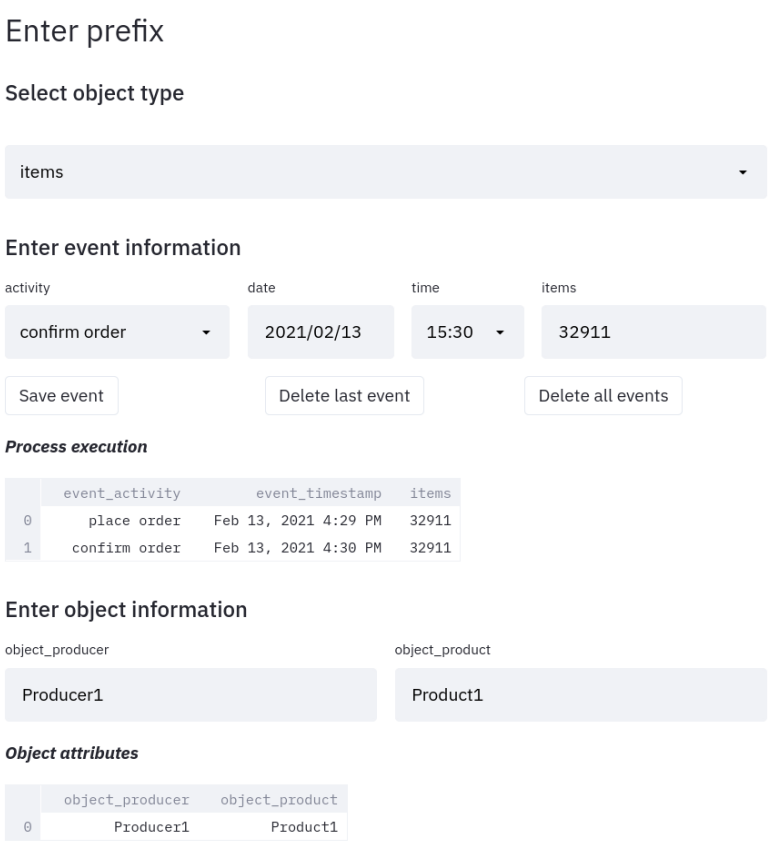


Figure 5.3: Interface section for inputting prefix.

Chapter 6

Evaluation

In this chapter the the experimental setup is detailed. Then the datasets used in evaluating the model are presented and the reslts are discussed.

6.1 Experimental Setup

The model was implemented as described in Chapter 5. The training and testing was performed on an Intel Xeon CPU, 16 GB RAM, and a Nvidia Tesla T4 GPU with 16 GB of GPU memory.

Metrics

The evaluation metrics are kept the same as previous research for uniformity. To compare the predicted suffixes to the real suffixes, the approach for calculating similarity S from [15] is used as described in Equation 6.1:

$$S(s_f, s_r) = 1 - \left(\frac{DL(s_f, s_r)}{Max(|s_f|, |s_r|)} \right) \tag{6.1}$$

Where s_f is the predicted suffix and s_r is the real suffix. Since the suffix is a sequence, Damerau–Levenshtein (DL) distance can be used. It is defined as the minimum number of operations needed to transform one sequence into the other sequence. The allowed operations are insertion, deletion, substitution, or transposition.

As the measure of accuracy of timestamps, Mean Absolute Error is used (MAE) as seen in Equation 6.2:

$$MAE = \frac{\sum_{i=1}^{n} |t_r - t_f|}{n} \tag{6.2}$$

Where t_r is the real timestamp and t_f the predicted timestamp.

6.2 Synthetic Dataset

To validate the approach a synthetic dataset was used. The synthetic dataset represents an order process with different object types. Orders with multiple items are placed by customers. Items are packed into packages and sent out. An excerpt of the data can be seen in Figure A.1 in the Appendix.

6.2.1 Description

The activities modeled in the dataset are:

{place order, pick item, confirm order, item out of stock, reorder item, pay order, create package, send package, failed delivery, package delivered, payment reminder}

The object types in the dataset are:

{customers, items, orders, packages}

The attributes for each object type are shown in Table 6.1:

Table 6.1: Object attributes in synthetic object centric event log.

object type	object attribute
customers	age
items	color
orders	price
packages	weight

An important step in selecting an appropriate object type is understanding the relations between activities and object types as seen in Table 6.2.

Table 6.2: Relations between activities and object types in the synthetic dataset.

customers	*items*	*orders*	*packages*
place order	*place order*	*place order*	$\emptyset$
pick item	*pick item*	*pick item*	$\emptyset$
confirm order	*confirm order*	*confirm order*	$\emptyset$
item out of stock	*item out of stock*	*item out of stock*	$\emptyset$
reorder item	*reorder item*	*reorder item*	$\emptyset$
pay order	*pay order*	*pay order*	$\emptyset$
create package	*create package*	*create package*	*create package*
send package	*send package*	*send package*	*send package*
failed delivery	*failed delivery*	*failed delivery*	*failed delivery*
package delivered	*package delivered*	*package delivered*	*package delivered*
payment reminder	*payment reminder*	*payment reminder*	$\emptyset$

Additionally, to better understand the cases that are captured by the different object types, some statistics are given in Table 6.3. These represent data after outliers outside of two standard deviations in terms of case length had been removed, which is also what is

used when training. For example, in the case of the object type *customers*, this does not remove any data, but in the case of *orders*, the number of cases decreases by less than 1% and the max length for a case is decreased from 41 to 28.

Table 6.3: Statistics about synthetic object centric event log.

		Length of Case			Case time (days)		
	Number of cases	Max	Min	Mean	Max	Min	Mean
customers	17	1470	1171	1315.7	461.03	383.7	422.5
items	7860	10	7	7.82	103.87	0.51	14.48
orders	1923	28	7	15.57	114.69	1.67	18.9
packages	1238	4	3	3.14	7.71	0.03	1.69

6.2.2 Results

Table 6.4: Sequence similarity and MAE results for synthetic object centric event log.

	Sequence similariy S			MAE (normalized)		
	Thesis	Taymouri [15]	Tax[13]	Thesis	Taymouri[15]	Tax[13]
customers	0.0986	0.0921	0.0371	0.4559	0.4983	0.5839
items	0.5102	0.5113	0.4781	0.0384	0.0461	0.0438
orders	0.3502	0.3288	0.3013	0.0336	0.0310	0.0487
packages	0.8654	0.8241	0.8121	0.0673	0.0632	0.0729

The predominant trend that can be observed in Table 6.4 is that the approach detailed in this thesis generally performs better than previous approaches across different object types. The largest improvement in sequence similarity can be observed for the object type *orders*. This should mean that the object attribute *price* for object type *orders* is a good feature for the prediction of suffixes. To prove that assumption the model was trained without the object attributes, meaning the model is almost identical in strucutre to the approach by Taymouri [15]. As expected, the results were equal. For the object types items and packages less significant improvements are observed. Therefore, it is possible the respective object attributes are less useful features for the model. The absolute results of the object type *packages* were the best, most likely due to the low average case length and little variance in possible event flow.

The model is not able to predict suffixes well for the object type *customer*. Taking into account the statistics from Table 6.3, this makes sense. There are only a few cases, making learning features difficult. Furthermore, the average case length for cases of object type *orders* is significantly longer than the other object types' average case length. Very long cases clearly are difficult for the model to predict. In the case of the synthetic event log, the object type *customers* is not very useful in predicting an order process, since customers make multiple orders in the dataset. This can hold interesting data about the ordering habits of a customer, or can be used to predict when a customer might pay an order. However, the structure of the model in this thesis as empirically demonstrated, is better suited to shorter cases, therefore the metrics of sequence similarity and MAE are very low.

6.3 RCLL Dataset

The RoboCup Logisitcs League (RCLL) is a simulation of factory logistics. There are two teams of three robots and the goal is to fulfill orders from a central system. The products that can be ordered consist of the following components: a base, optional rings, and a cap. There can be between zero and three rings, but there is always one base and one cap according to the order. On a whole, the steps are to collect the base, mount rings per the order, mount the cap, and deliver the order. To achieve this, the robots work together navigating a map to access all the resources and workstations. An excerpt of the data can be seen in Figure A.2 in the Appendix.

6.3.1 Description

The activities modeled in the dataset are:

{clear-mps, deliver, discard-unknown, fill-cap, fill-rs, get-base-to-fill-rs, mount-first-ring, mount-next-ring, process-mps, produce-c0, produce-cx}

The object types in the RCLL object centric event log:

{robot, order, products}

The attributes for each object type are shown in Table 6.5:

Table 6.5: Object attributes in RCLL object centric event log.

object type	object attribute
robot	none
order	delivery begin, delivery end
components	station, cost

The object attributes of delivery begin/end refer to the delivery window that is associated with the order. In the RCLL simulation early delivery is considered fatal but late delivery may be acceptable. The object attribute station and cost only applies if the component is a ring. The station is the name of a ring station where the particular component needs to be mounted at and the cost refers to how expensive it is to mount. All object types are related to all activities in the RCLL data and some statistics are given in Table 6.6.

Table 6.6: Statistics about RCLL object centric event log.

		Length of Case			Case time (Minutes)		
	Number of cases	Max	Min	Mean	Max	Min	Mean
robot	3	94	88	91.67	16.65	4.711	11.838
orders	26	16	5	9.81	13.74	0.77	3.02
components	80	18	5	11.1	13.73	0.77	3.02

Table 6.7: Sequence similarity and MAE results for RCLL object centric event log.

	Sequence similariy S			MAE (normalized)		
	Thesis	Taymouri [15]	Tax [13]	Thesis	Taymouri [15]	Tax [13]
robot	0.1408	0.1377	0.1128	0.4977	0.4992	0.5182
order	0.5242	0.4822	0.4627	0.0325	0.0312	0.0387
components	0.4629	0.4315	0.4179	0.0278	0.0281	0.0391

6.3.2 Results

The trends observed in the synthetic dataset continue to be present with the RCLL data, as seen in Table 6.7. Overall, in both Sequence similarity and MAE, the model proposed in this thesis meets or exceeds the previous research. For the object type *order* the related object attributes enable the model to achieve a significant improvement in sequence similarity as opposed to the previous research. The improvement is less noticeable in the timstamp prediction, which can lead to future research. The object type *robot* proved to be difficult to predict, most likely on account of the very long average case lengths. This continues the trend seen in the synthetic data that the model struggles with longer cases. Furthermore, since there were no object attributes present for the object type *robot*, the results are more or less the same as the approach by Taymouri [15].

Chapter 7

Conclusion

In this section, the thesis is summarized and an outlook for applications and expansions are given.

7.1 Summary

In this thesis, using object-centric event logs in state-of-the-art predictive models was discussed. As demonstrated, the relations that frequently exist between object types and activities in real processes can not be accurately modeled by traditional event logs. By not forcing the selection of a single case id, the OCEL standard enables the preservation of such relations, as well as storing object attributes. In addition, the object attributes contained in OCEL data can hold important features for the prediction of next events and timestamps (suffix) based on an ongoing case (prefix). While this had been used in past research in the form of case attributes in traditional event logs, the most recent methods were not utilizing this information.

The model proposed by this thesis augments the recent advancements in predictive methods with the rich data contained in OCEL. The model makes use of a Generative Adversarial Network (GAN) in a Sequence to sequence model using encoder and decoder Long Short-Term Memory layer (LSTM). An individual event is encoded with activity and timestamp information, as well as the relevant attributes. Numerical and categorical data is encoded and normalized where applicable. The model was first trained and evaluated on a synthetic event log to validate the approach. The synthetic event log represents an order process with multiple object types, including *customers, items, orders*, and *packages*. The metrics for testing the model are a measure of sequence similarity to evaluate the predicted suffix and mean absolute error for the predicted timestamps. The results showed that overall, the approach presented in this thesis outperformed the previous approaches in both metrics. For the object type orders, a noticeable increase in sequence similarity was achieved. For the object type customers, the results were poor, which can be explained by the long average case lengths and the low number of cases available. These trends are also observed on a real event log of RCLL data, which is a simulation of factory logistics. Again, object types which have short average case lengths were able to perform better than previous research.

7.2 Outlook

The main goal of the thesis was to use object-centric event logs with predictive methods. By using object attributes as a feature, the model was able to outperform past approaches. This can be applied to any OCEL data with object attributes to improve the prediction methods. However, there are some drawbacks, the main one being that only activities related to the chosen object type can be predicted. This can lead to an incomplete or disjointed view of the underlying process, where intermediate activities that are not related to an object type can not be predicted by this approach. Therefore, the main area of future research is developing a method to predict a case using multiple object types, not just one as in this thesis. This would allow for all activities to be predicted and can lead to more useful real world applications. Correlating the disjoint events was the topic of research in [52]. By defining correlation profiles, different views on a process can be gained and such research could be explored in the future. However, some more incremental research based on the current work is possible as well. The encodings chosen for numerical and categorical values were as simple as possible, to demonstrate the approach working well independent of a special data preprocessing step. This can be improved by taking into account business hours as a part of the timestamp prediction if such data is consistent with those rules. This can lead to significant increases in the accuracy since the model presented in this thesis could predict an event to occur ten minutes after closing when it should instead occur ten minutes after opening the next business day. Furthermore, there is some recent research to learn the time constraints of events using LSTM layers in Time2Vec [53]. Instead of determining the business hours beforehand, Time2Vec can learn the periodic and non-periodic patterns in the time data. Furthermore, research to determine the most useful types of object attributes can be conducted. In this thesis, it was not exhaustively explored why certain attributes could lead to better results and a more rigorous understanding of the factors that make certain attributes perform well could lead to insights into how to improve predictions.

Appendix A

Appendix

event id	activity	timestamp	items	orders	customers	packages
1.0	place order	2019-05-20 09:07:47	{880003, 880002, 880001, 880004}	{990001}	{Marco Pegoraro}	∅
2.0	place order	2019-05-20 10:35:21	{880005, 880006, 880008, 880007}	{990002}	{Gyunam Park}	∅
3.0	pick item	2019-05-20 10:38:17	{880006}	{990002}	{Gyunam Park}	∅
4.0	confirm order	2019-05-20 11:13:54	{880003, 880002, 880001, 880004}	{990001}	{Marco Pegoraro}	∅
5.0	pick item	2019-05-20 11:20:13	{880002}	{990001}	{Marco Pegoraro}	∅
6.0	place order	2019-05-20 12:30:30	{880012, 880011, 880009, 880010}	{990003}	{Majid Rafiei}	∅
7.0	confirm order	2019-05-20 12:34:16	{880012, 880011, 880009, 880010}	{990003}	{Majid Rafiei}	∅

Figure A.1: First few events in event table from synthetic OCEL

event id	activity	timestamp	components	order	robot
2	FILL-CAP	2020-06-11 15:21:48.150	{CAP_GREY_1, BASE_SILVER_1}	{O1}	{0}
6	PRODUCE-C0	2020-06-11 15:22:06.251	{CAP_GREY_1, BASE_SILVER_1}	{O1}	{2}
15	DELIVER	2020-06-11 15:22:41.322	{CAP_GREY_1, BASE_SILVER_1}	{O1}	{2}
70	FILL-CAP	2020-06-11 15:25:31.418	{RING_YELLOW_3, CAP_BLACK_3, BASE_SILVER_3}	{O12}	{1}
78	FILL-RS-1	2020-06-11 15:25:57.641	{RING_YELLOW_3, CAP_BLACK_3, BASE_SILVER_3}	{O12}	{1}
98	FILL-CAP	2020-06-11 15:27:01.212	{BASE_BLACK_2, CAP_GREY_2}	{O11}	{0}
102	PROCESS-MPS	2020-06-11 15:27:12.408	{BASE_BLACK_2, CAP_GREY_2}	{O11}	{0}
107	CLEAR-MPS	2020-06-11 15:27:26.223	{BASE_BLACK_2, CAP_GREY_2}	{O11}	{1}
114	DELIVER	2020-06-11 15:27:48.902	{BASE_BLACK_2, CAP_GREY_2}	{O11}	{1}

Figure A.2: First few events in event table from RCLL OCEL

Bibliography

[1] Wil van der Aalst. *Process Mining: Data Science in Action*. Springer-Verlag, 2 edition, . ISBN 978-3-662-49850-7. doi: 10.1007/978-3-662-49851-4. URL `https://www.springer.com/gp/book/9783662498507`.

[2] Alasdair Gilchrist. *Industry 4.0*. Apress. ISBN 978-1-4842-2046-7. doi: 10.1007/978-1-4842-2047-4.

[3] Somayya Madakam and Siddharth Tripathi. Internet of things (IoT): A literature review. 03(5):164. doi: 10.4236/jcc.2015.35021. Number: 05 Publisher: Scientific Research Publishing.

[4] Mirko Polato, Alessandro Sperduti, Andrea Burattin, and Massimiliano de Leoni. Time and activity sequence prediction of business process instances. URL `http://arxiv.org/abs/1602.07566`.

[5] Samuel Mann, Jan Pennekamp, Tobias Brockhoff, Anahita Farhang, Mahsa Pourbafrani, Lukas Oster, Merih Seran Uysal, Rahul Sharma, Uwe Reisgen, Klaus Wehrle, et al. Connected, digitalized welding production—secure, ubiquitous utilization of data across process layers. In *Advanced Joining Processes*, pages 101–118. Springer, 2020.

[6] Merih Seran Uysal, Sebastiaan J van Zelst, Tobias Brockhoff, Anahita Farhang Ghahfarokhi, Mahsa Pourbafrani, Ruben Schumacher, Sebastian Junglas, Günther Schuh, and WM van der Aalst. Process mining for production processes in the automotive industry. In *Industry Forum at BPM*, volume 20, 2020.

[7] Mohammad Reza Harati Nik, Wil MP van der Aalst, and Mohammadreza Fani Sani. Bipm: Combining bi and process mining. In *DATA*, pages 123–128, 2019.

[8] Joerg Evermann, Jana-Rebecca Rehse, and Peter Fettke. Predicting process behaviour using deep learning. 100:129–140. ISSN 01679236. doi: 10.1016/j.dss.2017.04.003. URL `http://arxiv.org/abs/1612.04600`.

[9] Francesco Folino, Massimo Guarascio, and Luigi Pontieri. Discovering context-aware models for predicting business process performances. In *On the Move to Meaningful Internet Systems: OTM 2012*, Lecture Notes in Computer Science, pages 287–304. Springer. ISBN 978-3-642-33606-5. doi: 10.1007/978-3-642-33606-5_18.

[10] Dominic Breuker, Martin Matzner, Patrick Delfmann, and Jörg Becker. Comprehensible predictive models for business processes. 40(4):1009–1034. ISSN 0276-7783. doi: 10.25300/MISQ/2016/40.4.10.

[11] IEEE standard for eXtensible event stream (XES) for achieving interoperability in event logs and event streams. pages 1–50. doi: 10.1109/IEEESTD.2016.7740858. Conference Name: IEEE Std 1849-2016.

[12] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, and Donato Malerba. Using convolutional neural networks for predictive process analytics. In *2019 International Conference on Process Mining (ICPM)*, pages 129–136. doi: 10.1109/ICPM.2019.00028.

[13] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. Predictive business process monitoring with LSTM neural networks. 10253:477–492. doi: 10.1007/978-3-319-59536-8_30. URL `http://arxiv.org/abs/1612.02130`.

[14] Farbod Taymouri, Marcello La Rosa, Sarah Erfani, Zahra Dasht Bozorgi, and Ilya Verenich. Predictive business process monitoring via generative adversarial nets: The case of next event prediction. URL `http://arxiv.org/abs/2003.11268`.

[15] Farbod Taymouri and Marcello La Rosa. Encoder-decoder generative adversarial nets for suffix generation and remaining time prediction of business process models. URL `http://arxiv.org/abs/2007.16030`.

[16] Wil van der Aalst. Object-centric process mining: Dealing with divergence and convergence in event data. In *Software Engineering and Formal Methods*, Lecture Notes in Computer Science, pages 3–25. Springer International Publishing, . ISBN 978-3-030-30446-1. doi: 10.1007/978-3-030-30446-1_1.

[17] Anahita Farhang Ghahfarokhi, Gyunam Park, Alessandro Berti, and Wil MP van der Aalst. Ocel: A standard for object-centric event logs. In *European Conference on Advances in Databases and Information Systems*, pages 169–175. Springer, 2021.

[18] Anahita Farhang Ghahfarokhi, Alessandro Berti, and Wil MP van der Aalst. Process comparison using object-centric process cubes. *arXiv preprint arXiv:2103.07184*, 2021.

[19] Anahita Farhang Ghahfarokhi and Wil MP van der Aalst. A python tool for object-centric process mining comparison. *arXiv e-prints*, pages arXiv–2202, 2022.

[20] Alessandro Berti, Anahita Farhang Ghahfarokhi, Gyunam Park, and Wil MP van der Aalst. A scalable database for the storage of object-centric event logs. *arXiv preprint arXiv:2202.05639*, 2022.

[21] Guangming Li, Eduardo González López de Murillas, Renata Medeiros de Carvalho, and Wil M. P. van der Aalst. Extracting object-centric event logs to support process mining on databases. In *Information Systems in the Big Data Era*, Lecture Notes in Business Information Processing, pages 182–199. Springer International Publishing, . ISBN 978-3-319-92901-9. doi: 10.1007/978-3-319-92901-9_16.

[22] Wil van der Aalst, M. H. Schonenberg, and M. Song. Time prediction based on process mining. 36(2):450–475, . ISSN 0306-4379. doi: 10.1016/j.is.2010.09.001. URL `http://www.sciencedirect.com/science/article/pii/S0306437910000864`.

[23] Anahita Farhang Ghahfarokhi, Gyunam Park, Alessandro Berti, and Wil van der Aalst. OCEL standard. URL `http://ocel-standard.org/1.0/specification.pdf`.

[24] B. F. van Dongen, R. A. Crooy, and Wil van der Aalst. Cycle time prediction: When will this case finally be finished? In *On the Move to Meaningful Internet Systems: OTM 2008*, Lecture Notes in Computer Science, pages 319–336. Springer. ISBN 978-3-540-88871-0. doi: 10.1007/978-3-540-88871-0_22.

[25] Arik Senderovich, Chiara Di Francescomarino, Chiara Ghidini, Kerwin Jorbina, and Fabrizio Maria Maggi. Intra and inter-case features in predictive process monitoring: A tale of two dimensions. In *Business Process Management*, Lecture Notes in Computer Science, pages 306–323. Springer International Publishing. ISBN 978-3-319-65000-5. doi: 10.1007/978-3-319-65000-5_18.

[26] Juergen Schmidhuber. Deep learning in neural networks. 61:85–117. ISSN 08936080. doi: 10.1016/j.neunet.2014.09.003. URL `http://arxiv.org/abs/1404.7828`.

[27] Michael Nielsen. *Neural networks and deep learning*. Determination Press.

[28] Shi Yan. Understanding LSTM and its diagrams. URL `https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714`.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. 9(8):1735–1780. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. Publisher: MIT Press.

[30] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, and David Warde-Farley. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc.

[31] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. URL `http://arxiv.org/abs/1511.06434`.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. URL `http://arxiv.org/abs/1409.3215`.

[33] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In *Advances in Database Technology — EDBT'98*, Lecture Notes in Computer Science, pages 467–483. Springer. ISBN 978-3-540-69709-1. doi: 10.1007/BFb0101003.

[34] Wil van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. 16(9):1128–1142, . doi: 10.1109/TKDE.2004.47.

[35] A. Weijters and Wil van der Aalst. Rediscovering workflow models from event-based data using little thumb. 10:151–162. doi: 10.3233/ICA-2003-10205.

[36] Mai Le, Bogdan Gabrys, and Detlef Nauck. A hybrid model for business process event prediction. In *Research and Development in Intelligent Systems XXIX*, pages 179–192. Springer. ISBN 978-1-4471-4739-8. doi: 10.1007/978-1-4471-4739-8_13.

[37] Geetika T. Lakshmanan, Davood Shamsi, Yurdaer N. Doganata, Merve Unuvar, and Rania Khalaf. A markov prediction model for data-driven semi-structured business processes. 42(1):97–126. ISSN 0219-3116. doi: 10.1007/s10115-013-0697-8.

[38] Andreas Rogge-Solti and Mathias Weske. Prediction of remaining service execution time using stochastic petri nets with arbitrary firing delays. In *Service-Oriented*

Computing, Lecture Notes in Computer Science, pages 389–403. Springer. ISBN 978-3-642-45005-1. doi: 10.1007/978-3-642-45005-1_27.

[39] Ilya Verenich, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, and Irene Teinemaa. Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. 10(4):34:1–34:34. ISSN 2157-6904. doi: 10.1145/3331449.

[40] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. Learning accurate LSTM models of business processes. In *Business Process Management*, Lecture Notes in Computer Science, pages 286–302. Springer International Publishing. ISBN 978-3-030-26619-6. doi: 10.1007/978-3-030-26619-6_19.

[41] Alessandro Berti and Wil van der Aalst. Extracting multiple viewpoint models from relational databases. URL `http://arxiv.org/abs/2001.02562`.

[42] A. P. Simović, S. Babarogić, and O. Pantelić. A domain-specific language for supporting event log extraction from ERP systems. In *2018 7th International Conference on Computers Communications and Control (ICCCC)*, pages 12–16. doi: 10.1109/ICCCC.2018.8390430.

[43] E. González López de Murillas, H. A. Reijers, and W. M. P. van der Aalst. Case notion discovery and recommendation: automated event log building on databases. 62(7):2539–2575. ISSN 0219-3116. doi: 10.1007/s10115-019-01430-6.

[44] Wil M. P. van der Aalst and Alessandro Berti. Discovering object-centric petri nets. URL `http://arxiv.org/abs/2010.02047`.

[45] David Cohn and Richard Hull. Business artifacts: A data-centric approach to modeling business operations and processes. 32:3–9.

[46] N. C. Narendra, Y. Badr, P. Thiran, and Z. Maamar. Towards a unified approach for business process modeling using context-based artifacts and web services. In *2009 IEEE International Conference on Services Computing*, pages 332–339. doi: 10.1109/SCC.2009.14.

[47] Guangming Li, Renata Medeiros de Carvalho, and Wil M. P. van der Aalst. Object-centric behavioral constraint models: a hybrid model for behavioral and data perspectives. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pages 48–56. Association for Computing Machinery, . ISBN 978-1-4503-5933-7. doi: 10.1145/3297280.3297287.

[48] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. URL `http://arxiv.org/abs/1611.01144`.

[49] Chris J. Maddison, Daniel Tarlow, and Tom Minka. A* sampling. URL `http://arxiv.org/abs/1411.0030`.

[50] Casper Kaae Sønderby, Jose Caballero, Lucas Theis, Wenzhe Shi, and Ferenc Huszár. Amortised MAP inference for image super-resolution. URL `http://arxiv.org/abs/1610.04490`.

[51] Ronald J. Williams and David Zipser. A learning algorithm for continually running

fully recurrent neural networks. 1(2):270–280. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270.

[52] Guangming Li, Renata M. de Carvalho, and Wil Aalst. Configurable event correlation for process discovery from object-centric event data. pages 203–210, . doi: 10.1109/ICWS.2018.00033.

[53] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. URL `http://arxiv.org/abs/1907.05321`.