Learning Accurate LSTM Models of Business Processes

Manuel Camargo[1,2][0000−0002−8510−1972], Marlon Dumas[1][0000−0002−9247−7476], and Oscar González-Rojas[2][0000−0002−8296−6620]

[1] University of Tartu, Tartu, Estonia, {manuel.camargo, marlon.dumas}@ut.ee
[2] Universidad de los Andes, Bogotá, Colombia, o-gonza1@uniandes.edu.co

Abstract. Deep learning techniques have recently found applications in the field of predictive business process monitoring. These techniques allow us to predict, among other things, what will be the next events in a case, when will they occur, and which resources will trigger them. They also allow us to generate entire execution traces of a business process, or even entire event logs, which opens up the possibility of using such models for process simulation. This paper addresses the question of how to use deep learning techniques to train accurate models of business process behavior from event logs. The paper proposes an approach to train recurrent neural networks with Long-Short-Term Memory (LSTM) architecture in order to predict sequences of next events, their timestamp, and their associated resource pools. An experimental evaluation on real-life event logs shows that the proposed approach outperforms previously proposed LSTM architectures targeted at this problem.

Keywords: Process mining · Deep learning · Long-short-term memory.

1 Introduction

Models of business process behavior trained with deep learning techniques have recently found several applications in the fields of predictive process monitoring [2,7,13]. Such models allow us to move from predicting boolean, categorical, or numerical performance properties, to predicting what will be the next event in a case, when will it occur, and which resource will trigger it. They also allow us to predict the most likely remaining path of an ongoing case and even to generate entire execution traces of a business process (or entire event logs), which opens up the possibility of using such models for process simulation. Yet another application of such models can be found in the field of anomaly detection [8].

This paper addresses the question of how to use deep learning techniques to train accurate models from business process event logs. This question has been previously addressed in the context of predictive process monitoring by using Recurrent Neural Networks (RNNs) with Long-Short-Term Memory (LSTM) architecture. Specifically, Evermann et al. [2] proposed an approach to generate the most likely remaining sequence of events (suffix) starting from a prefix of an ongoing case. However, this architecture cannot handle numerical variables and hence it cannot generate sequences of timestamped events. This inability to predict timestamps and durations is also shared by the approach of Lin et al. [6]. An alternative approach by Tax et al [13] can predict timestamps but it does not use the embedded dimension of the LSTM network, which forces it to one-hot-encode categorical variables. In particular, it one-hot-encodes the type of each event (i.e. the activity to which the event refers). As a result, its accuracy deteriorates as the number of event types increases. As shown later in this paper, this choice leads to poor accuracy when applied to real-life event logs with a couple of dozen event types.

The paper addresses the limitations of the above approaches by proposing new pre- and post-processing methods and architectures for building and using generative models from event logs using LSTM neural networks. Specifically, the paper proposes an approach to learn models that can generate traces (or suffixes of traces starting from a given prefix) consisting of triplets (event type, role, time-stamp). The proposed approach combines the advantages of Tax et al [13] and Evermann et al. [2] by making use of the embedded dimension while supporting both categorical and numerical attributes in the event log. The paper considers three architectures corresponding to different combinations of shared and specialized layers in the neural network.

The paper reports on two experimental evaluations. The first one compares alternative instantiations of the proposed approach corresponding to different architectures, pre-processing, and post-processing choices. The goal of this evaluation is to derive guidelines as to which design choices are preferable depending on the characteristics of the log. The second evaluation compares the accuracy of the proposed approach relative to the three baselines mentioned above.

The next section provides an overview of RNNs and LSTMs and discusses related work on the use of deep learning techniques in the field of process mining and predictive process monitoring. Section 3 introduces the proposed approach, while Section 4 presents its evaluation. Finally, Section 5 summarizes the contributions and findings and outlines future work.

2 Background and Related Work

2.1 RNN and LSTM networks

Deep Learning is a sub-field of machine learning concerned with the construction and use of networks composed of multiple interconnected layers of neurons (perceptrons), which perform non-linear transformations of data [4]. The main goal of these transformations is train the network to "learn" the behaviors/patterns observed in the data. Theoretically, the more layers of neurons there are in the network, the more it becomes possible to detect higher-level patterns in the data thanks to the composition of complex functions [5].

Recurrent Neural Networks (RNN) contain cyclical connections that have been specially designed for the prediction of sequential data [10]. In this type of data, the state of an observation depends on the state of its predecessor. So the

RNNs use a part of the processed output (h) of the preceding unit of processing (a cell) for the processing of a new input (X). Fig. 1 presents the basic RNN cell structure. Even though, RNNs have a good performance when predicting sequences with short-term temporary dependencies, they fail to account for long-term dependencies. Long Short-Term Memory (LSTM) networks address this problem. In LSTM networks apart from the use of part of the previous output for a new processing, a long-term memory is implemented. In the long term memory, the information flows from cell to cell with minimal variation, keeping certain aspects constant during the processing of all inputs. This constant input allows to remain the coherence of the predictions in long periods of time.

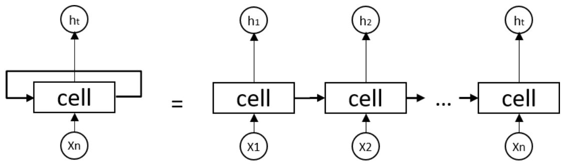


Fig. 1: RNN basic structure

2.2 Related Work

Tax et al. [13] use LSTM networks to predict the type of the next event of an ongoing process case and the time until the next event (its timestamp). In this approach, each event is mapped to a feature vector by encoding the event type using one-hot encoding and supplementing it with features related to the event's occurrence time, such the time of the day, the time since the previous event, and the accumulated duration since the start of the case. The weights in the network are set so as to minimize the cross-entropy between the ground-truth one-hot encoding of the next event and the predicted one-hot encoding as well as the Mean Absolute Error (MAE) between the ground truth time until the next event and the predicted time. The network architecture consists of a shared LSTM layer that feeds two independent LSTM layers specialized in predicting the next event and the other in predicting times. By repeatedly predicting the next event in a case and its timestamp, the authors also use their approach to predict the remaining sequence of events until case completion and the remaining cycle time. The experiments show that the LSTM approach outperforms automata-based approaches for predicting the remaining of sequence of events and the remaining time [1, 9]. In this approach the embedded dimension in LSTMs is not used to capture the event type, but instead the event type is one-hot encoded. This design choice is suitable when the number of event types is low, but detrimental for larger numbers of event types as shown later in this paper.

Evermann et al. [2] also apply LSTM networks to predict the type of the next event of a case. Unlike [13], this approach uses the embedded dimension of LSTMs to reduce the input's size and to include additional attributes such as the resource associated to each event. The network's architecture comprises two LSTM hidden layers. An empirical evaluation shows that this approach sometimes outperforms the approach of [13] at the task of predicting the next event. However, the approach focuses on predicting event types. It cannot handle numerical variables and hence it cannot predict the next event's timestamp. In this paper, we combine the idea of using the embedded dimension from [2] with the idea of interleaving shared and specialized layers from [13] to design prediction architectures that can handle large numbers of event types.

Lin et al. [6] propose an RNN-based approach, namely MM-Pred, for predicting the next event and the suffix of an ongoing case. This approach uses both the control-flow information (event type) and the case data (event attributes). The proposed architecture is composed of encoders, modulators and decoders. Encoders and decoders use LSTM networks to transform the attributes of each event into and from hidden representations. The modulator component infers a variable-length alignment weight vector, in which each weight represents the relevance of the attribute for predicting the future events and attributes. This work suffers from the same limitation as [2]: It does not support the prediction of attributes with numerical domains, including timestamps and durations.

In [7] the authors propose another approach to predict the next event using a multi-stage deep learning approach. In this approach, each event is first mapped to feature vector. Next, transformations are applied to reduce the input's dimensionality, e.g. by extracting n-grams, applying a hash function, and passing the input through two auto-encoder layers. The transformed input is then processed by a feed-forward neural network responsible for the next-event prediction. Again, this approach suffers from the same limitation as [2], namely that it does not handle numerical variables and hence it cannot predict timestamps or durations.

In [8] the authors propose a neural network architecture called BINet for real-time anomaly detection in business process executions. The core of this approach is a GRU neural network trained to predict the next event and its attributes. The approach is designed to assign a likelihood score to each event in a trace, which is then used to detect anomalies. This approach shows that generative models of process behavior can also be used for anomaly detection. In this paper, we do not consider this possible application. Instead, we focus on training models to produce sequences of timestamped events with associated roles.

In [12], the authors compare the performance of several techniques for predicting the next element in a sequence using real-life datasets. Specifically, the authors consider generative Markov models (including all-k markov models, AKOM), RNN models, and automata-based models, and compare them in terms of precision and interpretability. The results that the AKOM model yields the highest accuracy (outperforming an RNN architecture in some cases) while automata-based models have higher interpretability. This latter study addresses the problem of predicting the next event's type, but it does not consider the problem of simultaneously predicting the next event and its timestamp as we do in this paper.

3 Approach

This section describes the method we propose to build predictive models from business process event logs. This method uses LSTM networks to predict sequences of next events, their timestamp, and their associated resource pools. Three LSTM architectures are proposed that seek to improve the learning of the network in relation to the different events logs characteristics. These architectures can accurately reproduce the behavior observed in the log. Fig. 2 summarizes the phases and steps for building predictive models with our method.

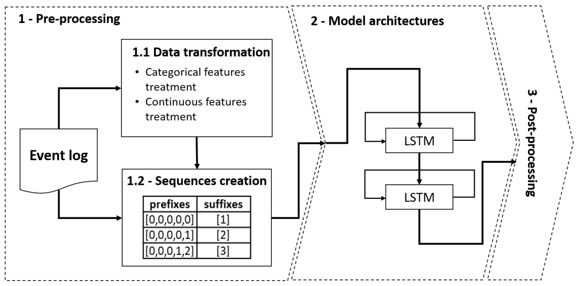


Fig. 2: Phases and steps for building predictive models

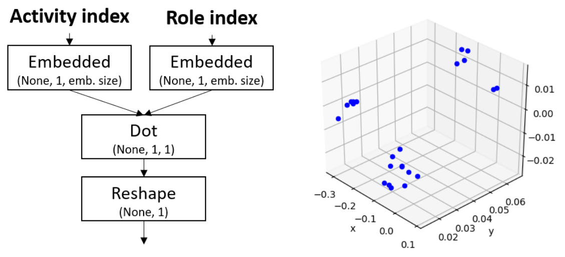
3.1 Pre-processing Phase

Data transformation. According with the attributes nature (i.e. categorical or continuous) specific pre-processing tasks were carried looking for the improvement of the data quality for feeding the LSTM models.

Our main concern in the case of the *categorical attributes* was its transformation into numerical values to be interpreted by the LSTM network without increase the attributes dimensionality. In contrast with approaches that use one-hot encoding (i.e. process flow), which is valid to manage a reduced number of attributes and categories, our model uses activities and resources as categorical attributes. The inclusion of multiple categorical attributes looks for using more information about the process behaviour to improve the prediction accuracy. However, this multiplicity could also increment the number of potential categories exponentially. To deal with this problem, we propose the grouping of resources into roles and the use of embedded dimensions.

On the one hand, the grouping of resources into roles was performed using the algorithm described by Song and Van der Aalst [11]. This algorithm seeks to discover resource pools (called roles in [11]) based on the definition of activity execution profiles for each resource and the creation of a correlation matrix of similarity of those profiles. The use of this algorithm allowed us to reduce the number of categories of this attribute, but keeping enough information to help the LSTM network to make more clear the differences between events.

On the other hand, the use of embedded dimensions helps in the control of the exponential attributes growth while provides more detailed information about the associations between attributes. To exemplify its advantages let's take the event log BPI 2012[3], which has 36 activities and 5 roles. If we use one-hot encoding to represent each unique pair activity-role in the event log, 180 new attributes composed by 179 zeroes are needed. This huge increment in the dimensionality is mostly composed by useless information. In contrast, only 4 dimensions are needed to encode the log if using embedded dimensions to map the categories into a n-dimensional space, in which each coordinate corresponds to a unique category. In this dimensional space the distances between points represents the how close is one activity performed by one role in relation with the same activity performed by other role. This additional information can help the network to understand the associations between events and differentiate them among similar ones. An independent network was trained to coordinate the embedded dimensions. The training network was fed with positive and negative examples of association between attributes, allowing the network to identify and locate near attributes with similar characteristics. The number of embedded dimensions was determined as the fourth root of the number of categories just to avoid a possible collision between them, according to a common recommendation used in the NLP community[4]. The generated values were exported and reused in all the experiments as non-trainable parameters, which allowed not to increase the complexity of the models. The Fig. 3a presents the architecture of the network used for training the embedded layers, and the Fig. 3b shows a representation of the generated 4d space reduced to a 3d space for activities.



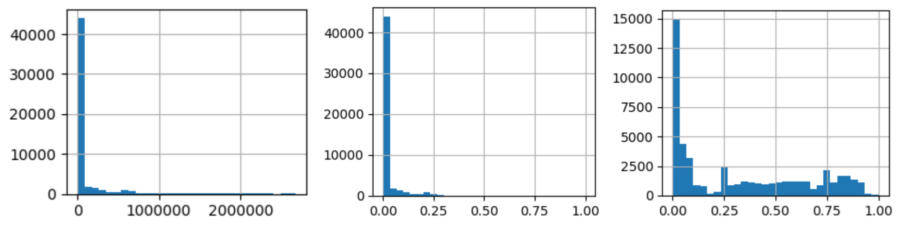
(a) Embedded layers (b) Generated space

Fig. 3: Embedding network architecture and results

In the case of *continuous attributes*, our major concern was the scaling of the values in a $[0, 1]$ range to be interpreted by the predictive models. Our model uses the relative time between activities as categorical input, calculated as the time elapsed between the complete time of one event and the complete time of the previous one. The relative time is easier to interpret by the models and is

[3] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[4] https://www.tensorflow.org/guide/feature_columns

useful to calculate the timestamp of the events in a trace. However, due to the nature of each event log the relative time may have a high variability. This high variability can hide useful information about the process behaviour such as time bottlenecks or anomalous behaviours that can be hide if the attribute scaling is performed without care. If the relative times present low variability, the use of log-normalization could also distort the perception of data. Therefore both techniques were evaluated to determine which best fits to the characteristics of the relative times. Fig. 4 illustrates the results of scaling the relative times in the event log BPI 2012. In particular, the use of log-normalization makes variations in relative times clearly observable.



(a) Original (b) Maximum value (c) Log-normalized

Fig. 4: Scaling of relative times over the maximum value and log-normalization

Sequences creation. We decided to extract n-grams of fixed sizes of each event log trace to create the input sequences and expected events to train the predictive network. N-grams allow to control the temporal dimensionality of the input, and bring clear patterns of sub-sequences describing the execution order of activities, roles or relative times, regardless of the length of the traces. One n-gram is extracted each time-step of the process execution, and is done for each attribute on an independent way, this meant that for our models we count with 3 independent inputs: activities, roles and relative times. Table 1 presents five n-grams extracted from the case id 174770 of the BPI 2012 event log. The numbers in the activities, roles, and times correspond to the indexes and scaled values in the data transformation step.

Time Step	**Activities**	**Roles**	**Relative times**
0	[0 0 0 0 0]	[0 0 0 0 0]	[0. 0. 0. 0. 0.]
1	[0 0 0 0 10]	[0 0 0 0 5]	[0. 0. 0. 0. 0.]
2	[0 0 0 10 7]	[0 0 0 5 5]	[0. 0. 0. 0. 4.73e-05]
3	[0 0 10 7 18]	[0 0 5 5 1]	[0. 0. 0. 4.73e-05 5.51e-01]
4	[0 10 7 18 5]	[0 5 5 1 1]	[0. 0. 4.73e-05 5.51e-01 1]
5	[10 7 18 5 18]	[5 5 1 1 1]	[0. 4.73e-05 5.51e-01 1 7.48e-04]

Table 1: N-grams for case number 174770 of the BPI 2012 event log

3.2 Model Structure Definition Phase

LSTM networks were used as the core of our predictive models since they are a well-known and proven technology to handle sequences, which are the nature of a business process event log. Fig. 5 illustrates the basic architecture of our network consisted of an input layer for each attribute, two stacked LSTM layers and a dense output layer. The first LSTM layer is in charge of provide a sequence output rather than a single value output to fed the second LSTM layer. Additionally, the categorical attributes have an embedded layer for their coding.

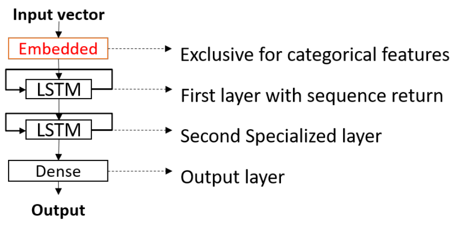
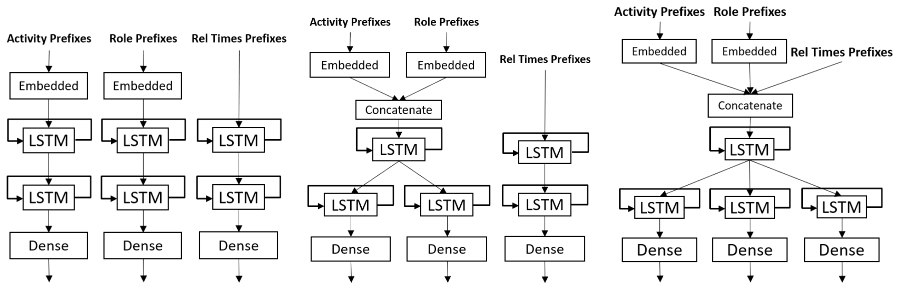


Fig. 5: Baseline architecture

Likewise, three variants of the baseline architecture were tested as is shown in the Fig. 6. The hypothesis behind these approaches is that sharing information between the layers can help to differentiate execution patterns. However, these changes could interfere with the identification of patterns in a log with high variability in relative times or in structure, generating noise in learning.



(a) Specialized (b) Shared categorical (c) Full shared

Fig. 6: Tested architectures

The specialized architecture (see Fig. 6a) does not share any information, in fact can be understood as three independent models. The shared categorical architecture (see Fig. 6b), concatenates the inputs related with activities and roles, and shares the first LSTM layer. Is expected that this architecture avoids the possible noises introduced by sharing information between attributes of different nature (i.e. categorical or continuous). The full shared architecture (see Fig. 6c), concatenates all the inputs and completely shares the first LSTM layer. In the evaluation section, the possibility of an architecture fits better than other in accordance with the nature of each event log is explored.

3.3 Post-processing Phase

Our technique is capable of generate complete traces of business processes starting from a zero prefix size. The way of doing this is by the use of continuous feedback of the model with each new generated event, until the generation of a finalization event (hallucination). This technique has been used by previous approaches, however, we explore the use of arguments of the maxima (arg-max) and random choice as techniques for the category selection of the next predicted event. Arg-max is the technique commonly used to select the next category of a prediction, and consists in selecting the one that has the highest predicted probability. In theory this technique should work well for specific prediction tasks, such as the most likely category of the next event, given an incomplete case. However, if the model is used in a generative way, it could be biased and tends to generate always the same kind of sequences, that is, the most probable ones. To avoid the this, we use the random selection of a new category following the predicted probability distribution. This attribute allows us to generate a greater number of different traces, by not getting stuck in the higher probabilities. This technique also allows us to reveals what the neural network has actually learned from the dynamics observed in the event log. Of course, the introduction of a random element, forces us to perform multiple repetitions of the experiment to find the convergence in the measurements. Both approaches were taken into account in the evaluation of results about the reproduction of the observed current state of business processes.

4 Evaluation

This section describes two experimental evaluations. The first experiment compares different instantiations of the three proposed architectures in terms of pre-processing and post-processing choices. The second experiment compares the proposed approach to the three baselines discussed in Section 2.2 for the tasks of next event, suffix, and remaining time prediction.

4.1 Comparison of LSTM architectures and processing options

Datasets. For this experiment, we use nine real-life event logs from different domains and with diverse characteristics:

- The Helpdesk[5] event log contains records from a ticketing management process of the helpdesk of an Italian software company.

[5] https://doi:10.17632/39bp3vv62t.1

– The two event-logs within BPI 2012[6] are related to a loan application process from a German financial institution. This process is composed by three sub-processes from which we used the W sub-process in order to allow the comparison with the existing approaches [2, 12].
– The event log within BPI 2013[7] is related to a Volvo's IT incident and problem management. We used the complete cases to learn generative models.
– The five event-logs within BPI 2015[8] contain data on building permit applications provided by five Dutch municipalities during a period of four years. The original event log was subdivided in five parts (one per each municipality). All the event logs were specified at a sub-processes level including more than 345 activities. Therefore, it was pre-processed to be managed at a phases level by following with the steps described in [14].

The sequence flow (SF) of each event log was classified as simple, medium, and complex according with its composition in terms of number of traces, events, activities and length of the sequences. In the same way, the time variability (TV) was classified as stable or variable according with the relation between the mean and max duration of each event log (see Table 2).

Event log	**Num. traces**	**Num. events**	**Num. activities**	**Avg. activities per trace**	**Max. activities per trace**	**Mean duration**	**Max. duration**	**SF**	**TV**
Helpdesk	4580	21348	14	4.6	15	40.9 days	59.2 days	simple	stedy
BPI 2012	13087	262200	36	20	175	8.6 days	137.5 days	complex	stedy
BPI 2012 W	9658	170107	7	17.6	156	8.8 days	137.5 days	complex	stedy
BPI 2013	1487	6660	7	4.47	35	179.2 days	6 years, 64 days	simple	irregular
BPI 2015-1	1199	27409	38	22.8	61	95.9 days	4 years, 26 days	medium	irregular
BPI 2015-2	832	25344	44	30.4	78	160.3 days	2 years, 341 days	medium	irregular
BPI 2015-3	1409	31574	40	22.4	69	62.2 days	4 years, 52 days	medium	irregular
BPI 2015-4	1053	27679	43	26.2	83	116.9 days	2 years, 196 days	medium	irregular
BPI 2015-5	1156	36.234	41	31.3	109	98 days	3 years, 248 days	medium	irregular

Table 2: Event logs description

Experimental setup. This experiment compares different instantiations of our approach in terms of their ability to learn execution patterns and to reliably reproduce the behavior registered in the event log. Accordingly, we use the LSTM models to generate full event logs starting from size zero prefixes, and we then compare the generated traces against those in the original log.

We used two metrics to assess the similarity of the generated event logs. The Demerau-Levinstain (DL) algorithm measures the distance between sequences in terms of the number of editions necessary for one string character to be

[6] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[7] https://doi.org/10.4121/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07
[8] https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1

equal to another. This algorithm penalizes each time actions such as insertion, deletion, substitution, and transposition are carried out. Their measurements are commonly scaled by using the maximum size between the two sequences that are compared. Therefore, we use its inverse to measure the similarity between a generated sequence of activities or roles and a sequence observed in the actual event log. Then, a higher value implies a higher similarity among the sequences.

We trivially lift the DL measure (which applies to pairs of strings or traces), to measure the difference between two event logs by pairing each generated trace with the most similar trace (w.r.t. DL distance) of the ground-truth log. Once the pairs (generated trace, ground-truth trace) are formed, we calculate the mean DL between them. The Mean Absolute Error (MAE) metric is used to measure the error in predicting time-stamps. This measure is calculated by taking the absolute value of the distance between an observation and the predicted value, and then calculating the average value of these magnitudes. We use this metric to evaluate the distance between the generated relative time and those observed time, for each pair (generated trace, ground-truth trace).

We used cross validation by splitting the event logs into two folds: 70% for training and 30% for validation. The first fold was used as input to train 2000 models (approximately 220 models per event log). These models were configured with different pre-processing techniques and architectures. The configurations' values were selected randomly from the full search space of 972 combinations.

Then, new event logs of complete events are generated with each trained model (cf. techniques for the selection of the next activity described in the Section 3). Fifteen logs of each configuration were generated and their results averaged. More than 32000 generated event logs were evaluated.

Results and Interpretation. Table 3 summarizes the similarity results of the event logs generated from different model instantiations. The *Pre-processing, Model definition and Post-processing* columns describe the configuration used in each phase for building the evaluated models. The *DL act* and *DL roles* columns measure the similarity in the predicted categorical attributes. The *MAE* column corresponds to the mean absolute error of the cycle time of the predicted traces.

These results indicate that using this approach it is possible to train models that learn and reliably reproduce the observed behavior patterns of the original logs. Additionally, the results suggest that for the LSTM models is more difficult to learn sequences with a greater vocabulary than longer sequences. To learn these patterns, a greater number of examples is required, as can be seen in the results of BPI2012 and BPI2015. Both logs have more than 30 activities, but there is a great difference in the amount of traces (see Table 2). The high degree of similarity of the BPI2012 also suggests that the use of embedded dimensions to handle a high number of event types improves the results, so long as the number of examples is enough to learn the underlying patterns.

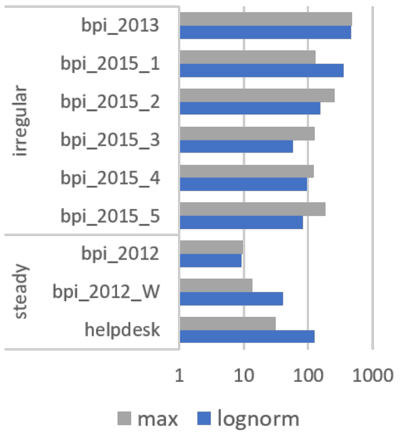
In relation with the architectural components evaluated in this experiment, we analyze them according to the phases to build generative models: preprocessing, model structure and hyper-parameters selection, and prediction.

Event log	Pre-processing		Model definition	Post-processing	DL act.	DL roles	MAE (days)
	Scaling	N-gram size	Architecture	Selection method			
BPI 2012	max	15	specialized	random	**0.8929**	0.7888	9
	max	15	shared cat.	random	0.885	**0.8998**	9
	lognorm	15	concatenated	random	0.8426	0.856	**4**
BPI 2012 W	max	15	specialized	random	**0.8742**	0.8245	11.8
	max	15	concatenated	random	0.7902	**0.8552**	7.3
	max	10	concatenated	random	0.7855	0.8329	**5.9**
BPI 2013	lognorm	10	joint	arg max	0.5442	0.698	**242.6**
	max	15	shared cat.	random	**0.7209**	0.8139	471.5
	lognorm	15	shared cat.	random	0.4416	**0.8475**	472.5
BPI 2015-1	max	10	concatenated	random	**0.4397**	0.8048	76.6
	lognorm	10	specialized	random	0.4228	**0.8498**	79.3
	lognorm	10	concatenated	arg in ax	0.3642	0.5922	**40.1**
BPI 2015-2	lognorm	10	shared cat.	arg in ax	**0.3737**	0.6228	159.4
	max	15	concatenated	random	0.3462	**0.8612**	158.3
	max	10	shared cat.	arg max	0.0431	0.1691	**89**
BPI 2015-3	lognorm	10	concatenated	random	**0.4616**	0.8501	53.2
	lognorm	5	concatenated	random	0.4456	**0.8729**	54.4
	lognorm	15	concatenated	arg max	0.4255	0.7786	**39.6**
BPI 2015-4	lognorm	5	concatenated	arg max	**0.4034**	0.7188	96
	lognorm	5	specialized	random	0.3609	**0.8248**	98.8
	max	5	shared cat.	arg max	0.0581	0.0968	**71.1**
BPI 2015-5	lognorm	15	specialized	random	**0.3633**	0.8653	84.1
	max	5	shared cat.	random	0.3323	**0.9019**	82.5
	lognorm	10	concatenated	arg max	0.3228	0.6547	**49.6**
Helpdesk	max	5	shared cat.	random	**0.9568**	**0.9869**	42.1
	max	5	joint	arg max	0.5773	0.7368	**7.3**

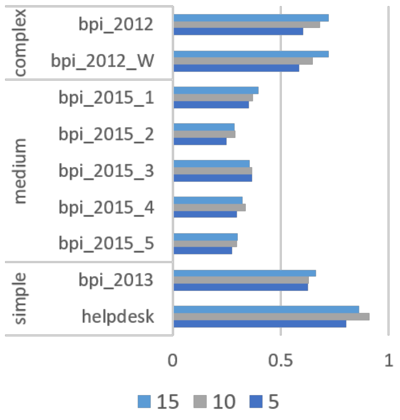
Table 3: Similarity results in event logs for different configurations

Regarding the *pre-processing phase*, Fig. 7a illustrates how logs with little time variability present better results using max value as scaling technique. In contrast, logs that have an irregular structure have lower MAE using log-normalization. Additionally, Fig. 7b presents the results of DL similarity in the use of n-grams of different sizes, in relation to the structure of event logs. We can observe that the use of longer n-grams has better results for logs with longer traces, showing a stable increasing trend. In contrast, it is not clear a trend for the event logs with medium and simple structures. Therefore, the use of long n-grams should be reserved to logs with very long traces.

Regarding the *model structure definition phase*, Fig. 8 illustrates that the concatenated architecture has the lowest overall similarity. In contrast, the model architecture that only shares information between categorical attributes has the median best performance. However, it is not very distant from the specialized architecture, albeit a wider spread. This implies that sharing information between attributes of different nature can generate noise in the patterns that the network is processing, thus, hindering the learning process.



(a) Scaling of relative times results



(b) N-gram size selection

Fig. 7: Preprocessing phase components comparison

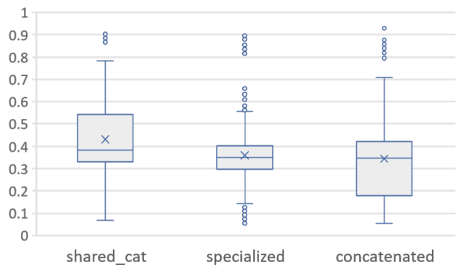
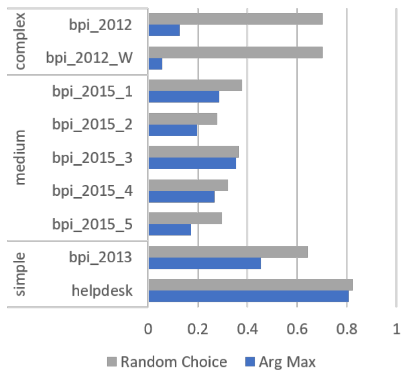


Fig. 8: Shared layers' overall similarity

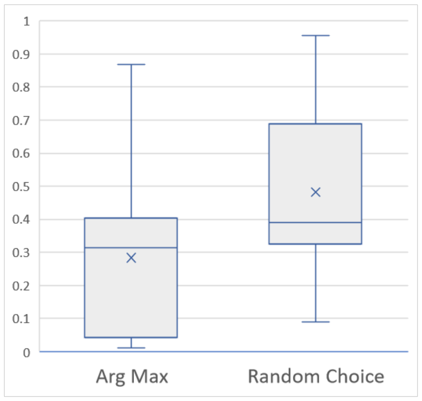
Regarding the *prediction phase*, Fig. 9 shows how random choice outperforms arg-max in all the event logs. This behaviour is even more clear in the event logs with longer and complex traces. The results suggest that random choice is advisable for assess the learning process in spite of the event log structure.

4.2 Comparison Against Baselines

Experimental setup. The aim of this experiment is to assess the relative performance of our approach at the task of predicting the next event, the remaining sequence of events (i.e. suffixes), and the remaining time, for trace prefixes of varying lengths. For *next event prediction*, we feed each model with trace prefixes of increasing length, from 1 up to the length of each trace. For each prefix, we predict the next event and we measure the accuracy (percentage of correct predictions). For suffix and remaining time prediction, we also feed the models with prefixes of increasing lengths. However, this time, we allow the models to hallucinate until the end of the case is reached. The remaining time is then



(a) Similarity per structure type



(b) Overall similarity

Fig. 9: Comparison of next-event selection methods

computed by subtracting the timestamp of the last event in the prefix from the timestamp of the last hallucinated event. As in [13], we use DL as a measure of similarity for suffix prediction and MAE for remaining time prediction. For next event and suffix prediction, we use [13], [2] and [6] as baselines while for remaining time prediction, we only use [13], since [2] and [6] cannot handle this prediction task. We only use the Helpdesk, BPI2012W and BPI2012 event logs, because these are the only logs for which results are reported in [13], [2] and [6]. The results reported for [2] for the Helpdesk and BPI2012 event logs correspond to the re-implementation of this technique reported in [6].

Results and Interpretation. Table 4 summarizes the average accuracy for the next-event prediction task and the average similarity between the predicted suffixes and the actual suffixes. For the task of next-event prediction, our approach performs similar to that of Evermann et al. and Tax et al. while slightly outperforming them for the BPI2012W event log. However, it underperforms the approach by Lin et al. For the task of suffix prediction, our approach outperforms all baselines including that of Lin et al. These results suggest that the measures adopted for the dimensionality control of the categorical attributes, allows our approach to achieve consistently good performance even for long sequences.

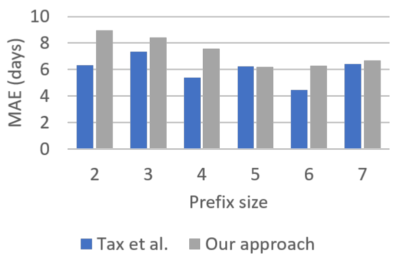
Figure 10 presents the MAE for remaining cycle time prediction. Even though the objective of our technique is not to predict the remaining time, it achieves similar performance at this task relative to Tax et al. – slightly underperforming it in one log, and slightly outperforming it for long suffixes in the other log.

5 Conclusion and Future Work

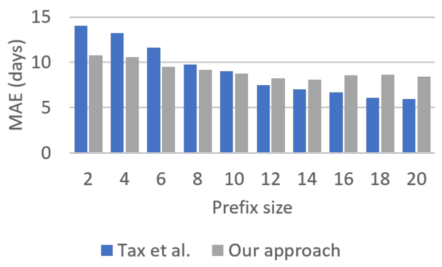
This paper outlined an approach to train LSTM networks to predict the type of the next event in a case, its timestamp, and the role associated to the event. By

	Next event accuracy			**Suffix prediction distance**		
Implementation	**Helpdesk**	**BPI 2012W**	**BPI 2012**	**Helpdesk**	**BPI 2012W**	**BPI 2012**
Our approach	0.789	*0.778*	0.786	*0.917*	*0.525*	*0.632*
Tax et al.	0.712	0.760		0.767	0.353	
Everman et al.	0.798	0.623	0.780	0.742	0.297	0.110
Lin et al.	*0.916*		*0.974*	0.874		0.281

Table 4: Next event and suffix prediction results



(a) Helpdesk



(b) BPI 2012W

Fig. 10: Results of remaining cycle-time MAE in days

iteratively predicting the next event, the approach can also predict the remaining sequence of events of a case (the suffix) and it can also generate entire traces from scratch. The approach consists of a pre-processing phase (scaling and n-gram encoding), an LSTM training phase, and a post-processing phase (selection of the predicted next event among the likely ones). The paper compared several options for each of these phases with respect to the task of generating full traces that closely match the traces in the original log. The evaluation shows that the use of longer n-grams gives higher accuracy, log-normalization is a suitable scaling method for logs with high variability, and randomly selecting the next event using the probabilities produced by the LSTM leads to a wider variety of traces and higher accuracy, relative to always choosing the most likely next event. The paper also showed that the proposed approach outperforms existing LSTM-based approaches for predicting the remaining sequence of events and their timestamps starting from a given prefix of a trace.

We foresee that the proposed approach could be used as a tool for business process simulation. Indeed, in its essence, a process simulator is a generative model that produces sets of traces consisting of event types, resources, and timestamps, from which it calculates performance measures such as waiting times, cycle times, and resource utilization. While process simulators rely on interpretable process models (e.g. BPMN models), any model that can generate traces of events, where each event consists of an event type (activity label), a timestamp, and a resource, can in principle be used to simulate a process. A key challenge to use LSTM networks for process simulation is how to capture "what-if" scenarios (e.g. the effect of removing a task or removing a resource). To this end, we plan to apply techniques to guide the generation of event sequences from LSTM models using constraints along the lines of [3].

Reproducibility The source code, event logs and example models can be downloaded from https://github.com/AdaptiveBProcess/GenerativeLSTM.git

Acknowledgments This research is funded by the Estonian Research Council (IUT20-55) and the European Research Council (Project PIX).

References

1. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Quarterly **40**(4), 1009–1034 (2016)
2. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decision Support Systems **100**, 129–140 (2017)
3. Francescomarino, C.D., Ghidini, C., Maggi, F.M., Petrucci, G., Yeshchenko, A.: An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring. In: 15th International Conference on Business Process Management (BPM). LNCS, vol. 10445, pp. 252–268. Springer (2017)
4. Hao, X., Zhang, G., Ma, S.: Deep Learning. International Journal of Semantic Computing **10**(03), 417–439 (sep 2016). https://doi.org/10.1142/S1793351X16500045
5. Lecun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
6. Lin, L., Wen, L., Wang, J.: MM-Pred: A deep predictive model for multi-attribute event sequence. In: Proceedings of the 2019 SIAM International Conference on Data Mining. pp. 118–126. Society for Industrial and Applied Mathematics (2019). https://doi.org/10.1137/1.9781611975673.14
7. Mehdiyev, N., Evermann, J., Fettke, P.: A Multi-stage Deep Learning Approach for Business Process Event Prediction. In: 19th Conference on Business Informatics (CBI). vol. 01, pp. 119–128. IEEE (2017). https://doi.org/10.1109/CBI.2017.46
8. Nolle, T., Seeliger, A., Mühlhäuser, M.: BINet: Multivariate Business Process Anomaly Detection Using Deep Learning. In: International Conference on Business Process Management (BPM), LNCS, vol. 11080, pp. 271–287. Springer (2018)
9. Polato, M., Sperduti, A., Burattin, A., Leoni, M.D.: Time and activity sequence prediction of business process instances. Computing **100**(9), 1005–1031 (2018)
10. Schmidhuber, J.: Deep Learning in neural networks: An overview. Neural Networks **61**, 85–117 (2015). https://doi.org/10.1016/j.neunet.2014.09.003
11. Song, M., van der Aalst, W.M.: Towards comprehensive support for organizational mining. Decision Support Systems **46**(1), 300 – 317 (2008). https://doi.org/10.1016/j.dss.2008.07.002
12. Tax, N., Teinemaa, I., van Zelst, S.J.: An interdisciplinary comparison of sequence modeling methods for next-element prediction. CoRR **abs/1811.00062** (2018)
13. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Intl. Conference on Advanced Information Systems Engineering (CAiSE), LNCS, vol. 10253, pp. 477–492. Springer (2017)
14. Teinemaa, I., Leontjeva, A., Masing, K.O.: BPIC 2015: Diagnostics of building permit application process in dutch municipalities. BPI Challenge Report **72** (2015)