Predictive Business Process Monitoring with LSTM Neural Networks

Niek Tax[1], Ilya Verenich[2,3], Marcello La Rosa[2], and Marlon Dumas[3]

[1] Eindhoven University of Technology, The Netherlands
`n.tax@tue.nl`
[2] Queensland University of Technology, Australia
`{ilya.verenich,m.larosa}@qut.edu.au`
[3] University of Tartu, Estonia
`marlon.dumas@ut.ee`

Abstract. Predictive business process monitoring methods exploit logs of completed cases of a process in order to make predictions about running cases thereof. Existing methods in this space are tailor-made for specific prediction tasks. Moreover, their relative accuracy is highly sensitive to the dataset at hand, thus requiring users to engage in trial-and-error and tuning when applying them in a specific setting. This paper investigates Long Short-Term Memory (LSTM) neural networks as an approach to build consistently accurate models for a wide range of predictive process monitoring tasks. First, we show that LSTMs outperform existing techniques to predict the next event of a running case and its timestamp. Next, we show how to use models for predicting the next task in order to predict the full continuation of a running case. Finally, we apply the same approach to predict the remaining time, and show that this approach outperforms existing tailor-made methods.

1 Introduction

Predictive business process monitoring techniques are concerned with predicting the evolution of running cases of a business process based on models extracted from historical event logs. A range of such techniques have been proposed for a variety of prediction tasks: predicting the next activity [2], predicting the future path (continuation) of a running case [25], predicting the remaining cycle time [27], predicting deadline violations [22] and predicting the fulfillment of a property upon completion [20]. The predictions generated by these techniques have a range of applications. For example, predicting the next activity (and its timestamp) or predicting the sequence of future activities in a case provide valuable input for planning and resource allocation. Meanwhile, predictions of the remaining execution time can be used to prioritize process instances in order to fulfill service-level objectives (e.g. to minimize deadline violations).

Existing predictive process monitoring approaches are tailor-made for specific prediction tasks and not readily generalizable. Moreover, their relative accuracy varies significantly depending on the input dataset and the point in time when the prediction is made. A technique may outperform another one for one log and a given prediction point (e.g. making prediction at the mid-point of each trace), but under-perform it for another log at the same prediction point, or for the same log at an earlier prediction point [12,22]. In some cases, multiple techniques need to be combined [22] or considerable tuning is required (e.g. using hyperparameter optimization) [11] in order to achieve more consistent accuracy.

Recurrent neural networks with Long Short-Term Memory (LSTM) architectures [14] have been shown to deliver consistently high accuracy in several sequence modeling application domains, e.g. natural language processing [23] and speech recognition [13]. Recently, Evermann et al. [9] applied LSTMs to predictive process monitoring, specifically to predict the next activity in a case.

Inspired by these results, this paper investigates the following questions: (i) can LSTMs be applied to a broad range of predictive process monitoring problems, and how? and (ii) do LSTMs achieve consistently high accuracy across a range of prediction tasks, event logs and prediction points? To address these questions, the paper puts forward LSTM architectures for predicting: (i) the next activity in a running case and its timestamp; (ii) the continuation of a case up to completion; and (iii) the remaining cycle time. The outlined LSTM architectures are empirically compared against tailor-made approaches with respect to their accuracy at different prediction points, using four real-life event logs.

The paper is structured as follows. Section 2 discusses related work. Section 3 introduces foundational concepts and notation. Section 4 describes a technique to predict the next activity in a case and its timestamp, and compares it against tailor-made baselines. Section 5 extends the previous technique to predict the continuation of a running case. Section 6 shows how this latter method can be used to predict the remaining time of a case, and compares it against tailor-made approaches. Section 7 concludes the paper and outlines future work directions.

2 Related Work

This section discusses existing approaches to predictive process monitoring for three prediction tasks: time-related predictions, predictions of the outcome of a case and predictions of the continuation of a case and/or characteristics thereof.

2.1 Prediction of time-related properties

A range of research proposals have addressed the problem of predicting delays and deadline violations in business processes. Pika et al. [24] propose a technique for predicting deadline violations. Metzger et al. [21,22] present techniques for predicting "late show" events (i.e. delays between the expected and the actual time of arrival) in a freight transportation process. Senderovich et al. [28] apply queue mining techniques to predict delays in case executions.

Another body of work focuses on predicting the remaining cycle time of running cases. Van Dongen et al. predict the remaining time by construction regression models based on case variables [8]. Van der Aalst et al. [1] propose a remaining time prediction method by constructing a transition system from the event log using set, bag, or sequence abstractions. Rogge-Solti & Weske [27] use stochastic Petri nets to predict the remaining time of a process, taking into account elapsed time since the last observed event. Folino et al. [10] develop an ad-hoc clustering approach to predict remaining time and overtime faults. In this paper, we show that the problem of predicting the remaining cycle time can be approached as a special case of predicting the continuation of a process.

2.2 Prediction of case outcome

The goal of approaches in this category is to predict cases that will end up in an undesirable state. Maggi et al. [20], propose a framework to predict the outcome of a case (normal vs. deviant) based on the sequence of activities executed in a given case and the values of data attributes of the last executed activity in a case. This latter framework constructs a classifier on-the-fly (e.g. a decision tree or random forest) based on historical cases that are similar to the (incomplete) trace of a running case. Other approaches construct a collection of classifiers offline. For example, [19] construct one classifier for every possible prediction point (e.g. predicting the outcome after the first event, the second one and so on). Meanwhile, [12] apply clustering techniques to group together similar prefixes of historical traces and then construct one classifier per cluster.

The above approaches require one to extract a feature vector from a prefix of an ongoing trace. De Leoni et al. [18] propose a framework that classifies possible approaches to extract such feature vectors.

In this paper, we do not address the problem of case outcome prediction, although the proposed architectures could be extended in this direction.

2.3 Prediction of future event(s)

The next-activity prediction problem is that of determining what is the next activity in a running case. Breuker et al. [3] use probabilistic finite automaton to tackle this problem, while Evermann et al. [9] use LSTMs. We take this latter approach as a baseline and show an LSTM architecture that solve the next-activity prediction problem with higher accuracy than [9] and [3], and that can be generalized to other prediction problems.

Pravilovic et al. [26] propose an approach that predicts both the next activity and its attributes (e.g. the involved resource). In this paper we use LSTMs to tackle a similar problem: predicting the next activity and its timestamp.

Lakshmanan et al. [16] use Markov chains to estimate the probability of future execution of a given task in a running case. Meanwhile, Van der Spoel et al [29] address the more ambitious problem of predicting the entire continuation of a case using a shortest path algorithm over a causality graph. Polato et al. [25] refine this approach by mining an annotated transition system from an event log and annotating its edges with transition probabilities. In this paper, we take this latter approach as a baseline and show how LSTMs can improve over it while providing higher generalizability.

3 Background

In this section we introduce concepts used in later sections of this paper.

3.1 Event logs, traces and sequences

For a given set A, A^* denotes the set of all sequences over A and $\sigma = \langle a_1, a_2, \dots, a_n \rangle$ a sequence of length n; $\langle\rangle$ is the empty sequence and $\sigma_1 \cdot \sigma_2$ is the concatenation of sequences σ_1 and σ_2. $hd^k(\sigma) = \langle a_1, a_2, \dots, a_{min(k,n)} \rangle$ is the prefix of length k of sequence σ and $tl^k(\sigma) = \langle a_{max(n-k+1,1)}, \dots, a_n \rangle$ is its suffix. For example, for sequence $\sigma_1 = \langle a, b, c, d, e \rangle$, $hd^2(\sigma_1) = \langle a, b \rangle$ and $tl^2(\sigma_1) = \langle c, d, e \rangle$.

Let $\mathcal{E}$ be the event universe, i.e., the set of all possible event identifiers, and $\mathcal{T}$ the time domain. We assume that events are characterized by various properties, e.g., an event has a timestamp, corresponds to an activity, is performed by a particular resource, etc. We do not impose a specific set of properties, however, given the focus of this paper we assume that two of these properties are the timestamp and the activity of an event, i.e., there is a function $\pi_\mathcal{T} \in \mathcal{E} \to \mathcal{T}$ that assigns timestamps to events, and a function $\pi_\mathcal{A} \in \mathcal{E} \to \mathcal{A}$ that assigns to each event an activity from a finite set of process activities $\mathcal{A}$.

An *event log* is a set of events, each linked to one trace and globally unique, i.e., the same event cannot occur twice in a log. A trace in a log represents the execution of one case.

Definition 1 (Trace, Event Log). *A* trace *is a finite non-empty sequence of events $\sigma \in \mathcal{E}^*$ such that each event appears only once and time is non-decreasing, i.e., for $1 \leq i < j \leq |\sigma| : \sigma(i) \neq \sigma(j)$ and $\pi_\mathcal{T}(\sigma(i)) \leq \pi_\mathcal{T}(\sigma(j))$. $\mathcal{C}$ is the set of all possible traces. An* event log *is a set of traces $L \subseteq \mathcal{C}$ such that each event appears at most once in the entire log.*

Given a trace and a property, we often need to compute a sequence consisting of the value of this property for each event in the trace. To this end, we lift the function f_p that maps an event to the value of its property p, in such a way that we can apply it to sequences of events (traces).

Definition 2 (Applying Functions to Sequences). *A function $f \in X \to Y$ can be lifted to sequences over X using the following recursive definition: (1) $f(\langle\rangle) = \langle\rangle$; (2) for any $\sigma \in X^*$ and $x \in X$: $f(\sigma \cdot \langle x \rangle) = f(\sigma) \cdot \langle f(x) \rangle$.*

Finally, $\pi_\mathcal{A}(\sigma)$ transforms a trace σ to a sequence of its activities. For example, for trace $\sigma = \langle e_1, e_2 \rangle$, with $\pi_\mathcal{A}(e_1) = a$ and $\pi_\mathcal{A}(e_2) = b$, $\pi_\mathcal{A}(\sigma) = \langle a, b \rangle$.

3.2 Neural Networks & Recurrent Neural Networks

A neural network consists of one layer of *inputs units*, one layer of outputs units, and multiple layers in-between which are referred to as *hidden units.* The outputs of the input units form the inputs of the units of the first *hidden layer* (i.e., the

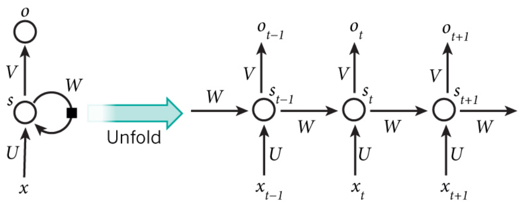

Fig. 1. A simple recurrent neural network (taken from [17]).

first layer of hidden units), and the outputs of the units of each hidden layer form the input for each subsequent hidden layer. The outputs of the last hidden layer form the input for the output layer. The output of each unit is a function over the weighted sum of its inputs. The weights of this weighted sum performed in each unit are learned through gradient-based optimization from training data that consists of example inputs and desired outputs for those example inputs. Recurrent Neural Networks (RNNs) are a special type of neural networks where the connections between neurons form a directed cycle.

RNNs can be unfolded, as shown in Figure 1. Each step in the unfolding is referred to as a time step, where x_t is the input at time step t. RNNs can take an arbitrary length sequence as input, by providing the RNN a feature representation of one element of the sequence at each time step. s_t is the hidden state at time step t and contains information extracted from all time steps up to t. The hidden state s is updated with information of the new input x_t after each time step: $s_t = f(Ux_t + Ws_{t-1})$, where U and W are vectors of weights over the new inputs and the hidden state respectively. Function f, known as the activation function, is usually either the hyperbolic tangent or the logistic function, often referred to as the sigmoid function: $sigmoid(x) = \frac{1}{1+exp(-x)}$. In neural network literature the sigmoid function is often represented with the letter σ, but we will fully write *sigmoid* to avoid confusion with traces. o_t is the output at step t.

3.3 Long Short-Term Memory for Sequence Modeling

A Long Short-Term Memory model (LSTM) [14] is a special Recurrent Neural Network architecture that has powerful modeling capabilities for long-term dependencies. The main distinction between a regular RNN and a LSTM is that the latter has a more complex memory cell C_t replacing s_t. Where the value of state s_t in a RNN is the result of a function over the weighted average over s_{t-1} and x_t, the LSTM state C_t is accessed, written, and cleared through controlling gates, respectively o_t, i_t, and f_t. Information on a new input will be accumulated to the memory cell if i_t is activated. Additionally, the past memory cell status C_{t-1} can be "forgotten" if f_t is activated. The information of C_t will be propagated to the output h_t based on the activation of output gate o_t. Combined, the LSTM model can be described with the following formulas:

$$\begin{aligned} f_t &= sigmoid(W_f \cdot [h_{t-1}, x_t] + b_f) & C_t &= f_t * C_{t-1} + i_i * \tilde{C}_t \\ i_t &= sigmoid(W_i \cdot [h_{t-1}, x_t] + b_i) & o_t &= sigmoid(W_o[h_{t-1}, x_t] + b_o) \\ \tilde{C}_t &= tanh(W_c \cdot [h_{t-1}, x_t] + b_C) & h_t &= o_t * tanh(C_t) \end{aligned}$$

In these formulas all W variables are weights and b variables are biases and both are learned during the training phase.

4 Next Activity and Timestamp Prediction

In this section we present and evaluate multiple architectures for next event and timestamp prediction using LSTMs.

4.1 Approach

We start by predicting the next activity in a case and its timestamp, by learning an activity prediction function f_a^1 and a time prediction function f_t^1. We aim at functions f_a^1 and f_t^1 such that $f_a^1(hd^k(\sigma)) = hd^1(tl^k(\pi_{\mathcal{A}}(\sigma)))$ and $f_t^1(hd^k(\sigma)) = hd^1(tl^k(\pi_{\mathcal{T}}(\sigma)))$ for any prefix length k. We transform each event $e \in hd^k(\sigma)$ into a feature vector and use these vectors as LSTM inputs $x_1, \ldots, x_k$. We build the feature vector as follows. We start with $|A|$ features that represent the type of activity of event e in a so called *one-hot encoding*. We take an arbitrary but consistent ordering over the set of activities A, and use $index \in A \rightarrow \{1, \ldots, |A|\}$ to indicate the position of an activity in it. The one-hot encoding assigns the value 1 to feature number $index(\pi_{\mathcal{A}}(e))$ and a value of 0 to the other features. We add three time-based features to the one-hot encoding feature vector. The first time-based feature of event $e = \sigma(i)$ is the time between the previous event in the trace and the current event, i.e., $fv_{t1}(e) = \begin{cases} 0 & \text{if } i = 1, \\ \pi_{\mathcal{T}}(e) - \pi_{\mathcal{T}}(\sigma(i-1)) & \text{otherwise.} \end{cases}$. This feature allows the LSTM to learn dependencies between the time differences at different points (indexes) in the process. Many activities can only be performed during office hours, therefore we add a time feature fv_{t2} that contains the time within the day (since midnight) and fv_{t3} that contains the time within the week (since midnight on Sunday). fv_{t2} and fv_{t3} are added to learn the LSTM such that if the last event observed occurred at the end of the working day or at the end of the working week, the time until the next event is expected to be longer.

At learning time, we set the target output o_a^k of time step k to the one-hot encoding of the activity of the event one time step later. However, it can be the case that the case ends at time k, in which case there is no new event to predict. Therefore we add an extra element to the output one-hot-encoding vector, which has value 1 when the case ends after k. We set a second target output o_t^k equal to the fv_{t1} feature of the next time step, i.e. the target is the time difference between the next and the current event. However, knowing the timestamp of the current event, we can calculate the timestamp of the following event. We optimize the weights of the neural network with the Adam learning algorithm

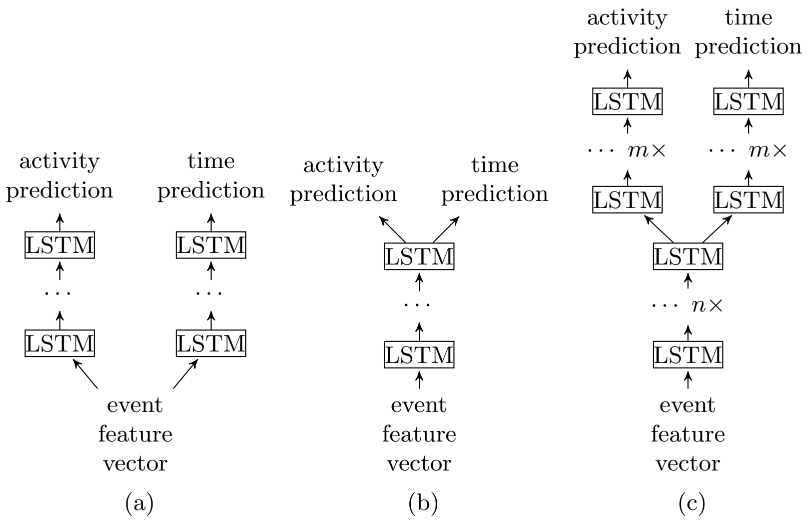

Fig. 2. Neural Network architectures with single-task layers *(a)*, with shared multi-tasks layer *(b)*, and with $n + m$ layers of which n are shared *(c)*.

[15] such that the cross entropy between the ground truth one-hot encoding of the next event and the predicted one-hot encoding of the next event as well as the mean absolute error (MAE) between the ground truth time until the next event and the predicted time until the next event are minimized.

Modeling the next activity prediction function f_a^1 and time prediction function f_t^1 with LSTMs can be done using several architectures. Firstly, we can train two separate models, one for f_a^1 and one for f_t^1, both using the same input features at each time step, as represented in Figure 2 (a). Secondly, f_a^1 and f_t^1 can be learned jointly in a single LSTM model that generates two outputs, in a multi-task learning setting [4] (Figure 2 (b)). The usage of LSTMs in a multi-task learning setting has shown to improve performance on all individual tasks when jointly learning multiple natural language processing tasks, including part-of-speech tagging, named entity recognition, and sentence classification [6]. A hybrid option between the architecture of Figures 2 (a) and (b) is an architecture of a number of shared LSTM layers for both tasks, followed by a number of layers that specialize in either prediction of the next activity or prediction of the time until the next event, as shown in Figure 2 (c).

It should be noted that activity prediction function f_a^1 outputs the probability distribution of various possible continuations of the partial trace. For evaluation purposes, we will only use the most likely continuation.

We implemented the technique as a set of Python scripts using the recurrent neural network library `Keras` [5]. The experiments were performed on a single NVidia Tesla k80 GPU, on which the experiments took between 15 and 90 seconds per training iteration depending on the neural network architecture. The execution time to make a prediction is in the order of milliseconds.

4.2 Experimental setup

In this section we describe and motivate the metrics, datasets, and baseline methods used for evaluation of the predictions of the next activities and of the timestamps of the next events. To the best of our knowledge, there is no existing technique to predict both the next activity and its timestamp. Therefore, we utilize one baseline method for activity prediction and a different one for timestamp prediction.

Well-known error metrics for regression tasks are Mean Absolute Error (MAE) and Root Mean Square Error (RMSE). Time differences between events tend to be highly varying, with values at different orders of magnitude. We evaluate the predictions using MAE, as RMSE would be very sensitive to errors on outlier data points, where the time between two events in the log is very large.

The remaining cycle time prediction method proposed by van der Aalst et al. [1] can be naturally adjusted to predict the time until the next event. To do so we build a transition system from the event log using either set, bag, or sequence abstraction, as in [1], but instead we annotate the transition system states with the average time until the next event. We will use this approach as baseline to predict the timestamp of next event.

We evaluate the performance of predicting the next activity and its timestamp on two datasets. We use the chronologically ordered first 2/3 of the traces as training data, and evaluate the activity and time predictions on the remaining 1/3 of the traces. We evaluate the next activity and the timestamp prediction on all prefixes $hd^k(\sigma)$ of all trace σ in the set of test traces for $2 \leq k < |\sigma|$. We do not make any predictions for the trace prefix of size one, since for those prefixes there is insufficient data available to base the prediction upon.

Helpdesk dataset This log contains events from a ticketing management process of the help desk of an Italian software company[1]. The process consists of 9 activities, and all cases start with the insertion of a new ticket into the ticketing management system. Each case ends when the issue is resolved and the ticket is closed. This log contains around 3,804 cases and 13,710 events.

BPI'12 subprocess W dataset This event log originates from the Business Process Intelligence Challenge (BPI'12)[2] and contains data from the application procedure for financial products at a large financial institution. This process consists of three subprocesses: one that tracks the state of the application, one that tracks the states of work items associated with the application, and a third one that tracks the state of the offer. In the context of predicting the coming events and their timestamps we are not interested in events that are performed automatically. Thus, we narrow down our evaluation to the work items subprocess, which contains events that are manually executed. Further, we filter the log to retain only events of type *complete*. Two existing techniques [3,9] for the next activity prediction, described in Section 2, have been evaluated on this event log with identical preprocessing, enabling comparison.

[1] doi:10.17632/39bp3vv62t.1

[2] doi:10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

			Helpdesk					BPI'12 W				
Layers	Shared	N/l	MAE in days				Acc.	MAE in days				Acc.
			Prefix 2	4	6	All		Prefix 2	10	20	All	
					LSTM							
4	4	100	3.64	2.79	2.22	3.82	0.7076	1.75	1.49	1.02	1.61	0.7466
4	3	100	3.63	2.78	2.21	3.83	0.7075	1.74	1.47	1.01	1.59	0.7479
4	2	100	3.59	2.82	2.27	3.81	0.7114	1.72	**1.45**	1.00	1.57	0.7497
4	1	100	3.58	2.77	2.24	3.77	0.7074	1.70	1.46	1.01	1.59	0.7522
4	0	100	3.78	2.98	2.41	3.95	0.7072	1.74	1.47	1.05	1.61	0.7515
3	3	100	3.58	2.69	2.22	3.77	0.7116	**1.69**	1.47	1.02	1.58	0.7507
3	2	100	3.59	2.69	2.21	3.80	0.7118	**1.69**	1.47	1.01	1.57	0.7512
3	1	100	3.55	2.78	2.38	3.76	**0.7123**	1.72	1.47	1.04	1.59	0.7525
3	0	100	3.62	2.71	2.23	3.82	0.6924	1.81	1.51	1.07	1.66	0.7506
2	2	100	3.61	2.64	**2.11**	3.81	0.7117	1.72	1.46	1.02	1.58	0.7556
2	1	100	3.57	**2.61**	**2.11**	3.77	0.7119	**1.69**	**1.45**	1.01	**1.56**	**0.7600**
2	0	100	3.66	2.89	2.13	3.86	0.6985	1.74	1.46	0.99	1.60	0.7537
1	1	100	**3.54**	2.71	3.16	**3.75**	0.7072	1.71	1.47	**0.98**	1.57	0.7486
1	0	100	3.55	2.91	2.45	3.87	0.7110	1.72	1.46	1.05	1.59	0.7431
3	1	75	3.73	2.81	2.23	3.89	0.7118	1.73	1.49	1.07	1.62	0.7503
3	1	150	3.78	2.92	2.43	3.97	0.6918	1.81	1.52	1.14	1.71	0.7491
2	1	75	3.73	2.79	2.32	3.90	0.7045	1.72	1.47	1.03	1.59	0.7544
2	1	150	3.62	2.73	2.23	3.83	0.6982	1.74	1.49	1.08	1.65	0.7511
1	1	75	3.74	2.87	2.35	3.87	0.6925	1.75	1.50	1.07	1.64	0.7452
1	1	150	3.73	2.79	2.32	3.92	0.7103	1.72	1.48	1.02	1.60	0.7489
					RNN							
3	1	100	4.21	3.25	3.13	4.04	0.6581					
2	1	100	4.12	3.23	3.05	3.98	0.6624					
1	1	100	4.14	3.28	3.12	4.02	0.6597					
					Time prediction baselines							
Set abstraction [1]			6.15	4.25	4.07	5.83	-	2.71	1.64	1.02	1.97	-
Bag abstraction [1]			6.17	4.11	3.26	5.74	-	2.89	1.71	1.07	1.92	-
Sequence abstraction [1]			6.17	3.53	2.98	5.67	-	2.89	1.69	1.07	1.91	-
					Activity prediction baselines							
Evermann et al. [9]			-	-	-	-	-	-	-	-	-	0.623
Breuker et al. [3]			-	-	-	-	-	-	-	-	-	0.719

Table 1. Experimental results for the Helpdesk and BPI'12 W logs.

4.3 Results

Table 1 shows the performance of various LSTM architectures on the helpdesk and the BPI'12 W subprocess logs in terms of MAE on predicted time, and accuracy of predicting the next event. The specific prefix sizes are chosen such that they represent *short*, *medium*, and *long* traces for each log. Thus, as the BPI'12 W log contains longer traces, the prefix sizes evaluated are higher for this log. In the table, *all* reports the average performance on all prefixes, not just the three prefix sizes reported in the three preceding columns. The number of shared layers represents the number of layers that contribute to both time and activity prediction. Rows where the numbers of shared layers are 0 correspond to the architecture of Figure 2 (a), where the prediction of time and activities is performed with separate models. When the number of shared layers is equal to the number of layers, the neural network contains no specialized layers, corresponding to the architecture of Figure 2 (b). Table 1 also shows the results of predicting the time until the end of the next event using the adjusted method from van der Aalst et al. [1] for comparison. All LSTM architectures outperform the baseline approach on all prefixes as well as averaged over all prefixes on both datasets. Further, it can be observed that the performance gain between the best LSTM model and the best baseline model is much larger for the short prefix than for the long prefix. The best performance obtained on next activity prediction over all prefixes was a classification accuracy of 71% on the helpdesk log. On the BPI'12 W log the best accuracy is 76%, which is higher than the 71.9% accuracy on this log reported by Breuker et al. [3] and the 62.3% accuracy reported by Evermann et al. [9]. In fact, the results obtained with LSTM are consistently higher than both approaches. Even though Evermann et al. [9] also rely on LSTM in their approach, there are several differences which are likely to cause the performance gap. First of all, [9] uses a technique called *embedding* [23] to create feature descriptions of events instead of the features described above. Embeddings automatically transform each activity into a "useful" large dimensional continuous feature vector. This approach has shown to work really well in the field of natural language processing, where the number of distinct words that can be predicted is very large, but for process mining event logs, where the number of distinct activities in an event log is often in the order of hundreds or much less, no useful feature vector can be learned automatically. Second, [9] uses a two-layer architecture with 500 neurons per layer, and does not explore other variants. We found performance to decrease when increasing the number of neurons from 100 to 150, which makes it likely that the performance of a 500 neuron model will decrease due to overfitting. A third and last explanation for the performance difference is the use of multi-task learning, which as we showed, slightly improves prediction performance on the next activity.

Even though the performance differences between our three LSTM architectures are small for both logs, we observe that most best performances (indicated in bold) of the LSTM model in terms of time prediction and next activity prediction are either obtained with the completely shared architecture of Figure 2 (b) or with the hybrid architecture of Figure 2 (c). We experimented with decreasing the number of neurons per layer to 75 and increasing it to 150 for architectures with one shared layer, but found that this results in decreasing performance in both tasks. It is likely that 75 neurons resulted in underfitting models, while 150 neurons resulted in overfitting models. We also experimented with traditional RNNs on one layer architectures, and found that they perform significantly worse than LSTMs on both time and activity prediction.

5 Suffix Prediction

Using functions f_a^1 and f_t^1 repeatedly allows us to make longer-term predictions that predict further ahead than a single time step. We use $f_a^\perp$ and $f_t^\perp$ to refer to activity and time until next event prediction functions that predict the whole continuation of a running case, and aim at those functions to be such that $f_a^\perp(hd^k(\sigma)) = tl^k(\pi_\mathcal{A}(\sigma))$ and $f_t^\perp(hd^k(\sigma)) = tl^k(\pi_\mathcal{T}(\sigma))$

5.1 Approach

The suffix can be predicted by iteratively predicting the next activity and the time until the next event, until the next activity prediction function f_a^1 predicts the end of case, which we represent with $\perp$. More formally, we calculate the complete suffix of activities as follows:

$$f_a^{\perp}(\sigma) = \begin{cases} \sigma & \text{if } f_a^1(\sigma) = \perp \\ f_a^{\perp}(\sigma \cdot e), \text{with } e \in \mathcal{E}, \pi_{\mathcal{A}}(e) = f_a^1(\sigma) \wedge & \\ \quad \pi_{\mathcal{T}}(e) = (f_t^1(\sigma) + \pi_{\mathcal{T}}(\sigma(|\sigma|))) & \text{otherwise} \end{cases}$$

and we calculate the suffix of times until the next events as follows:

$$f_t^{\perp}(\sigma) = \begin{cases} \sigma, & \text{if } f_t^1(\sigma) = \perp \\ f_t^{\perp}(\sigma \cdot e), \text{with } e \in \mathcal{E}, \pi_{\mathcal{A}}(e) = f_a^1(\sigma) \wedge & \\ \quad \pi_{\mathcal{T}}(e) = (f_t^1(\sigma) + \pi_{\mathcal{T}}(\sigma(|\sigma|))) & \text{otherwise} \end{cases}$$

5.2 Experimental Setup

For a given trace prefix $hd^k(\sigma)$ we evaluate the performance of $f_a^{\perp}$ by calculating the distance between the predicted continuation $f_a^{\perp}(hd^k(\sigma))$ and the actual continuation $\pi_{\mathcal{A}}(tl^k(\sigma))$. Many sequence distance metrics exist, with Levenshtein distance being one of the most well-known ones. Levenshtein distance is defined as the minimum number of insertion, deletion, and substitution operations needed to transform one sequence into the other.

Levenshtein distance is not suitable when the business process includes parallel branches. Indeed, when $\langle a, b \rangle$ are the next predicted events, and $\langle b, a \rangle$ are the actual next events, we consider this to be only a minor error, since it is often not relevant in which order two parallel activities are executed. However, Levenshtein distance would assign a cost of 2 to this prediction, as transforming the predicted sequence into the ground truth sequence would require one deletion and one insertion operation. An evaluation measure that better reflects the prediction quality of is the Damerau-Levenstein distance [7], which adds a swapping operation to the set of operations used by Levenshtein distance. Damerau-Levenshtein distance would assign a cost of 1 to transform $\langle a, b \rangle$ into $\langle b, a \rangle$. To obtain comparable numbers for different numbers of predicted events we normalize the Damerau-Levenshtein distance by the maximum of the length of the ground truth suffix and the length of the predicted suffix and subtract the normalized Damerau-Levenshtein distance from 1 to obtain Damerau-Levenshtein Similarity (DLS).

To the best of our knowledge, the most recent method to predict an arbitrary number of events ahead is the one by Polato et al. [25]. The authors first extract a transition system from the log and then learn a machine learning model for each transition system state to predict the next activity. They evaluate on predictions of a fixed number of events ahead, while we are interested in the continuation of the case until its end. We redid the experiments with their ProM plugin to obtain the performance on the predicted full case continuation.

For the LSTM experiments, we use a two-layer architecture with one shared layer and 100 neurons per layer, which showed good performance in terms of next activity prediction and predicting the time until the next event in the previous experiment. In addition to the two logs introduced in the previous section, we evaluate prediction of the suffix on an additional dataset, described below, which becomes feasible now that we have fixed the LSTM architecture.

Environmental permit dataset This is a log of an environmental permitting process at a Dutch municipality.[1] Each case refers to one permit application. The log contains 937 cases and 38,944 events of 381 event types. Almost every case follows a unique path, making the suffix prediction more challenging.

5.3 Results

Table 2 summarizes the results of suffix prediction for each log. As can be seen, the LSTM outperforms the baseline [25] on all logs. Even though it improves over the baseline, the performance on the BPI'12 W log is low given that the log only contains 6 activities. After inspection we found that this log contains many sequences of two or more events in a row of the same activity, where occurrences of 8 or more identical events in a row are not uncommon. We found that LSTMs have problems dealing with this log characteristic, causing it to predict overly long sequences of the same activity, resulting in predicted suffixes that are much longer than the ground truth suffixes. Hence, we also evaluated suffix prediction on a modified version of the BPI'12 W log where we removed repeated occurrences of the same event, keeping only the first occurrence. However, we can only notice a mild improvement over the unmodified log.

Method	Helpdesk	BPI'12 W	BPI'12 W (no duplicates)	Environmental permit
Polato [25]	0.2516	0.0458	0.0336	0.0260
LSTM	**0.7669**	**0.3533**	**0.3937**	**0.1522**

Table 2. Suffix prediction results in terms of Damerau-Levenshtein Similarity.

6 Remaining Cycle Time Prediction

Time prediction function $f_t^{\perp}$ predicts the timestamps of all events in a running case that are still to come. Since the last predicted timestamp in a prediction generated by $f_t^{\perp}$ is the timestamp of the end of the case, it is easy to see that $f_t^{\perp}$ can be used for predicting the remaining cycle time of the running case. For a given unfinished case σ, $\hat{\sigma_t} = f_t^{\perp}(\sigma)$ contains the predicted timestamps of the next events, and $\hat{\sigma_t}(|\hat{\sigma_t}|)$ contains the predicted end time of σ, therefore the estimated remaining cycle time can be obtained through $\hat{\sigma_t}(|\hat{\sigma_t}|) - \pi(\sigma(|\sigma|))$.

6.1 Experimental Setup

We use the same architecture as for the suffix prediction experiments. We predict and evaluate the remaining time after each passed event, starting from prefix size 2. We use the remaining cycle time prediction methods of van der Aalst et al. [1] and van Dongen et al. [8] as baseline methods.

[1] doi:10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270

6.2 Results

Figure 3 shows the mean absolute error for each prefix size, for the four logs (Helpdesk, BPI'12 W, BPI'12 W with no duplicates and Environmental Permit). It can be seen that LSTM consistently outperforms the baselines for the Helpdesk log. An exception is the BPI'12 W log, where LSTM performs worse than the baselines on short prefixes. This is caused by the problem that LSTMs have in predicting the next event when the log has many repeated events, as described in Section 5. This problem causes the LSTM to predict suffixes that are too long compared to the ground truth, and, thereby, also overestimating the remaining cycle time. We see that the LSTM does outperform the baseline on the modified version of the BPI'12 W log where we only kept the first occurrence of each repeated event in a sequence. Note that we do not remove the last event of the case, even if it is a repeated event, as that would change the ground truth remaining cycle time for the prefix.

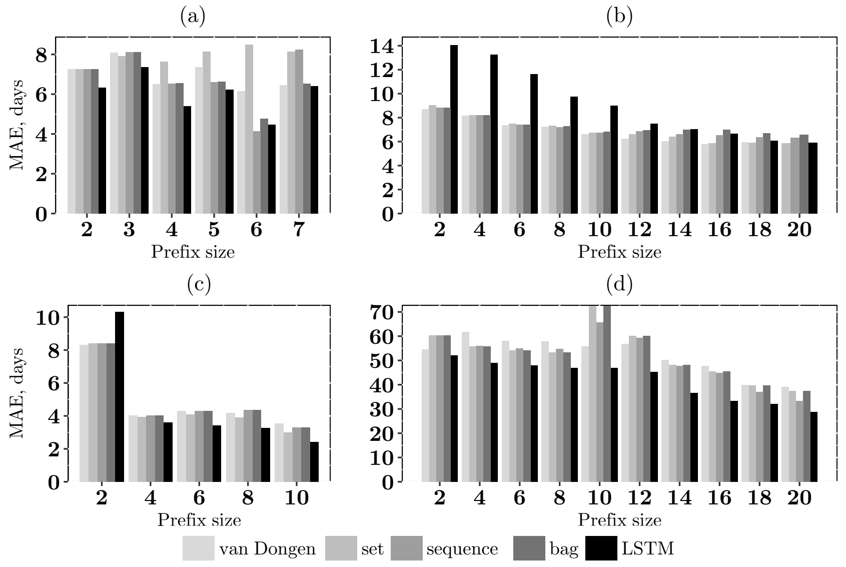

Fig. 3. MAE values using prefixes of different lengths for *helpdesk* (a), *BPI'12 W* (b), *BPI'12 W (no duplicates)* (c) and *environmental permit* (d) datasets.

7 Conclusion & Future Work

The foremost contribution of this paper is a technique to predict the next activity of a running case and its timestamp using LSTM neural networks. We showed that this technique outperforms existing baselines on real-life data sets. Additionally, we found that predicting the next activity and its timestamp via a single model (multi-task learning) yields a higher accuracy than predicting them using separate models. We then showed that this basic technique can be generalized to address two other predictive process monitoring problems: predicting the entire continuation of a running case and predicting the remaining cycle time. We empirically showed that the generalized LSTM-based technique outperforms tailor-made approaches to these problems. We also identified a limitation of LSTM models when dealing with traces with multiple occurrences of the same activity, in which case the model predicts overly long sequences of the same event. Addressing this latter limitation is a direction for future work.

The technique can be readily extended by adding event attributes to the feature vectors. In future work, we plan on extending the feature vector with additional attributes (e.g. resources). Furthermore, we plan to extend the multi-task learning approach to predict other attributes of the next activity besides its timestamp. Finally, we plan to extend the proposed technique to other prediction tasks, such as prediction of case outcomes and aggregate performance indicators.

Reproducibility. The source code and supplementary material required to reproduce the experiments reported in this paper can be found at `http://verenich.github.io/ProcessSequencePrediction`.

Acknowledgments. This research is funded by the Australian Research Council (grant DP150103356), the Estonian Research Council (grant IUT20-55) and the RISE_BPM project (H2020 Marie Curie Program, grant 645751).

References

1. van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Information Systems 36(2), 450–475 (2011)
2. Becker, J., Breuker, D., Delfmann, P., Matzner, M.: Designing and implementing a framework for event-based predictive modelling of business processes. In: Proceedigns of the 6th International Workshop on Enterprise Modelling and Information Systems Architectures. pp. 71–84. Springer (2014)
3. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Quarterly (In press.) (2016)
4. Caruana, R.: Multitask learning. Machine Learning 28(1), 41–75 (1997)
5. Chollet, F.: Keras. `https://github.com/fchollet/keras` (2015)
6. Collobert, R., Weston, J.: A unified architecture for natural language processing: Deep neural networks with multitask learning. In: ICML. pp. 160–167. ACM (2008)
7. Damerau, F.J.: A technique for computer detection and correction of spelling errors. Communications of the ACM 7(3), 171–176 (1964)
8. van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle time prediction: when will this case finally be finished? In: CoopIS. pp. 319–336. Springer (2008)
9. Evermann, J., Rehse, J.R., Fettke, P.: A deep learning approach for predicting process behaviour at runtime. In: Proceedings of the 1st International Workshop on Runtime Analysis of Process-Aware Information Systems. Springer (2016)
10. Folino, F., Guarascio, M., Pontieri, L.: Discovering context-aware models for predicting business process performances. In: CoopIS. pp. 287–304 (2012)
11. Francescomarino, C.D., Dumas, M., Federici, M., Ghidini, C., Maggi, F.M., Rizzi, W.: Predictive business process monitoring framework with hyperparameter optimization. In: CAiSE. pp. 361–376. Springer (2016)

12. Francescomarino, C.D., Dumas, M., Maggi, F.M., Teinemaa, I.: Clustering-based predictive process monitoring. CoRR abs/1506.01428 (2015), `http://arxiv.org/abs/1506.01428`, to appear in Transactions on Services Computing
13. Graves, A., Mohamed, A.R., Hinton, G.: Speech recognition with deep recurrent neural networks. In: IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 6645–6649. IEEE (2013)
14. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Computation 9(8), 1735–1780 (1997)
15. Kingma, D., Ba, J.: Adam: A method for stochastic optimization. In: Proceedings of the 3rd International Conference for Learning Representations (2015)
16. Lakshmanan, G.T., Shamsi, D., Doganata, Y.N., Unuvar, M., Khalaf, R.: A markov prediction model for data-driven semi-structured business processes. Knowledge and Information Systems 42(1), 97–126 (2015)
17. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436–444 (2015)
18. de Leoni, M., van der Aalst, W.M.P., Dees, M.: A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. Information Systems 56, 235–257 (2016)
19. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: BPM, pp. 297–313. Springer (2015)
20. Maggi, F.M., Di Francescomarino, C., Dumas, M., Ghidini, C.: Predictive monitoring of business processes. In: CAiSE. pp. 457–472. Springer (2014)
21. Metzger, A., Franklin, R., Engel, Y.: Predictive monitoring of heterogeneous service-oriented business networks: The transport and logistics case. In: 2012 Annual SRII Global Conference. pp. 313–322. IEEE (2012)
22. Metzger, A., Leitner, P., Ivanovic, D., Schmieders, E., Franklin, R., Carro, M., Dustdar, S., Pohl, K.: Comparing and combining predictive business process monitoring techniques. IEEE Trans. Systems, Man, and Cybernetics: Systems 45(2), 276–290 (2015)
23. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems. pp. 3111–3119 (2013)
24. Pika, A., van der Aalst, W.M.P., Fidge, C.J., ter Hofstede, A.H.M., Wynn, M.T.: Predicting deadline transgressions using event logs. In: BPM. pp. 211–216. Springer (2012)
25. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Time and activity sequence prediction of business process instances. arXiv preprint arXiv:1602.07566 (2016)
26. Pravilovic, S., Appice, A., Malerba, D.: Process mining to forecast the future of running cases. In: International Workshop on New Frontiers in Mining Complex Patterns. pp. 67–81. Springer (2013)
27. Rogge-Solti, A., Weske, M.: Prediction of remaining service execution time using stochastic Petri nets with arbitrary firing delays. In: ICSOC. pp. 389–403. Springer (2013)
28. Senderovich, A., Weidlich, M., Gal, A., Mandelbaum, A.: Queue mining - predicting delays in service processes. In: CAiSE. pp. 42–57 (2014)
29. van der Spoel, S., van Keulen, M., Amrit, C.: Process prediction in noisy data sets: a case study in a dutch hospital. In: International Symposium on Data-Driven Process Discovery and Analysis. pp. 60–83. Springer (2012)