Predicting Process Behaviour using Deep Learning

Joerg Evermann[a,1,*], Jana-Rebecca Rehse[b,c], Peter Fettke[b,c]

[a] *Memorial University of Newfoundland, St. John's, NL, Canada*
[b] *German Research Center for Artificial Intelligence, Saarbrücken, Germany*
[c] *Saarland University, Saarbrücken, Germany*

Abstract

Predicting business process behaviour, such as the final state of a running process, the remaining time to completion or the next activity of a running process, is an important aspect of business process management. Motivated by research in natural language processing, this paper describes an application of deep learning with recurrent neural networks to the problem of predicting the next event in a business process. This is both a novel method in process prediction, which has largely relied on explicit process models, and also a novel application of deep learning methods. The approach is evaluated on two real datasets and our results surpass the state-of-the-art in prediction precision. The paper offers recommendations for researchers and practitioners and points out areas for future applications of deep learning in business process management.

Keywords: Process management, Runtime support, Process prediction, Deep learning, Neural networks

1. Introduction

Being able to predict the future behaviour of a business process is an important business capability (Houy et al., 2010). As an application of predictive analytics

*Corresponding author
Email address: jevermann@mun.ca (Joerg Evermann)

in business process management, process prediction exploits data on past process instances to make predictions about current ones (Breuker et al., 2016). Example use cases are customer service agents responding to inquiries about the remaining time until a case is resolved, production managers predicting the completion time of a production process for better planning and higher utilization, or case managers identifying likely compliance violations to mitigate business risk.

We present a novel approach to predicting the next process event using deep learning. While the term "deep learning" has only recently become a popular research topic, it is essentially an application of neural networks and thus looks back on a long history of research (Schmidhuber, 2015). Recent innovations both in algorithms, allowing novel architectures of neural networks, and computing hardware, especially GPU processing, have led to a resurgence in interest for neural networks and popularized the term "deep learning" (LeCun et al., 2015). Our approach is motivated by applications of neural networks to Natural Language Processing (NLP), more specifically the prediction of the next word in a sentence (Sutskever et al., 2011; Graves, 2013; Zaremba et al., 2014). By interpreting process event logs as text, process traces as sentences, and process events as words, these techniques can be applied to predict future process events. The contribution of our research is threefold:

1. We improve on the state-of-the-art in process event prediction. Our results show that the method presented here improves considerably on the precision of next-event prediction for running processes.
2. We demonstrate that an explicit process model is not necessary for prediction and that deep learning algorithms, where the process structure is only implicitly reflected in the neural network parameters, can perform as well as explicit process models.

3. We contribute to process management in general by showcasing the useful application of an artificial intelligence approach, illustrating that business process management can benefit from the application of smart approaches.

Our research is located at the intersection of business process management, machine learning and process mining. We bring together historic process data with an intelligent learning technology and high-performance computing to leverage real-time case management, opening new perspectives into process execution, monitoring, and analysis. We not only provide a new approach to predicting the next process event, but also give a proof-of-concept regarding its feasibility, efficiency, and effectiveness, thus making a valuable contribution to the field of "Smart BPM".

2. Related Work

Process prediction covers an array of different techniques, objectives and data sources. It extends process mining from a post-hoc analysis method to operational decision support (van der Aalst et al., 2010). Most existing process prediction research focuses on prediction of process outcomes, primarily the remaining time to completion, rather than prediction of the next event in a process, as we do here. Only five approaches are concerned with predicting the next event (Le et al., 2012; Lakshmanan et al., 2015; Unuvar et al., 2016; Ceci et al., 2014; Breuker et al., 2016), all of which use an explicit model representation such as a state-transition, HMM (hidden Markov models), or PFA (probabilistic finite automatons) model.

The MSA approach by Le et al. (2012) considers each trace prefix as a state in a state-transition matrix. From the observed prefixes and their next events, a state transition matrix is built. When a running case has reached a state not contained in the state-transition matrix, its similarity to observed traces is computed using string edit distance. The prediction is made from the most similar observed case.

Evaluating the approach on two datasets from a telecommunications company, Le et al. (2012) report accuracies in predicting the next event of up to 25% and 70% for their two datasets.

The approach described by Lakshmanan et al. (2015) and Unuvar et al. (2016) consists of five steps. A process model is mined from existing logs. For each XOR split in the model, a decision tree is mined from case data. These trees are then used to compute the state transition probabilities for a HMM specific to the running case that is to be predicted, from the case data available at that point. This HMM is then used to predict the probabilities of the following event. The approach is evaluated on simulated data. After training the decision trees for the HMM on one half of the traces (training set), each trace in the test set is cut into a prefix and postfix at a random point. For each prefix, the log-likelihood of observing the corresponding postfix is reported.

The approach described by Ceci et al. (2014) uses sequence mining to identify frequent trace prefixes. For each prefix, a regression model is trained to predict remaining time to completion and a decision tree is trained to predict the next event. The algorithm identifies the appropriate prefix of the running case to choose the regression and decision tree model for predicting remaining completion time and the next event. The experimental evaluation uses two datasets, yielding prediction precision values for the next event of approx 65% on one dataset and approx 50% on the other, depending on the type of decision tree used and the frequency threshold for prefixes.

RegPFA (Breuker et al., 2016) a probabilistic finite automaton (PFA) instead of a HMM, allowing the future hidden state to be a function of both the previous hidden state and the previous observed event (which itself is a probabilistic consequence of the previous hidden state). RegPFA uses an EM algorithm to estimate the model

parameters of the PFA. The evaluation uses data from the 2012 and 2013 BPI Challenges (van Dongen, 2012, 2013a,b).

Our approach has the same objective as these five related works, but differs from them in terms of method and process representation. Deep Learning in the form of a recurrent neural network (RNN) is used to predict the next events, using event sequences and associated resource information. Processes are only implicitly represented within the RNN, rather than in an explicit state-transition, PFA or HMM. Because of the non-linear transformations used in neural networks, they are particularly suited to model non-linear relationships. In contrast, many of the existing methods are based on linear assumptions (e.g. regression trees). Thus, neural networks require less restrictive assumptions. Moreover, because models are built implicitly rather than explicitly, their performance does not depend on the any ex-ante assumptions about the model. Overall, our method constitutes an innovative new approach to process prediction.

3. Deep Learning for Process Prediction

3.1. Introduction

A neural network is a special form of an acyclic computational graph (Schmidhuber, 2015). It consists of a layer of input cells, one or more layers of "hidden" cells, and a layer of output cells. Cells in each layer are connected by weighted connections to cells in the previous and following layers, allowing for different architectures. Each cell's output is a function of the weighted sum of its inputs. A simple network architecture is a fully connected network of cells using sigmoid activation functions that form the hidden layer:

$$a_j^l = \sigma\left(\sum_i w_i^{l,j} a_i^{l-1} + b_j^l\right) \quad \text{where} \quad \sigma(x) = \tfrac{1}{1+exp(x)}$$

Here, a_j^l is the output ("activation") of cell j in layer l, $w_j^{l,i}$ is the weight of the connection from cell i on layer $l-1$ to cell j on layer l, a_i^{l-1} is the output of cell i on layer $l-1$ and b_j^l is the "bias" of cell j on layer l.

A neural network is a supervised learning technique where the output of the neural net is compared to a target by means of a loss function. The type of output layer cell and the loss function are often chosen jointly for their computational properties with respect to backpropagation. A typical combination is a softmax layer with a cross-entropy loss function H:

$$y_i = softmax(a)_i = \frac{exp(a_i)}{\sum_j exp(a_j)} \qquad H_{y\prime}(y) = -\sum_i y_i \prime log(y_i)$$

Here, $y\prime$ are the target values and y are the network outputs, computed in turn from the output activations a_i of the next to last network layer.

Gradients of network parameters $(w_j^{l,i}, b_i^l)$ with respect to the loss function are computed using backpropagation and parameters are then adjusted using variants of gradient descent algorithms to minimize the loss function.

3.2. Recurrent Neural Networks (RNN)

In a recurrent neural network, each cell's output is not only connected to the following layer but each cell also feeds back information into itself, allowing it to maintain "state" over time. To make this tractable within an acyclic computational graph and backpropagation, the recurrent network cells are "unrolled", that is, copies of it are produced for time $t, t-1, t-2, \ldots$. The state output of the RNN cell of time $t-1$ is state input to the cell for time t. In general, t need not represent time, but can index any sequence. Depending on how long one wishes to maintain state for, fewer or more cells are unrolled. Fig. 1 shows an RNN architecture with

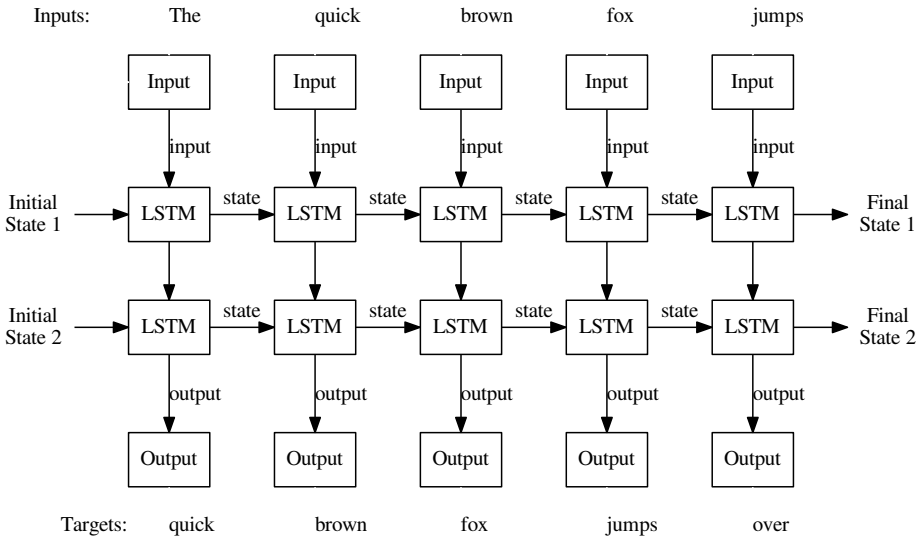

Figure 1: RNN architecture with single hidden layer of LSTM cells, unrolled five steps

an input layer, an output layer and two hidden layers that are unrolled five steps (Graves, 2013). This allows the network to maintain state or context information. For example, when given the input "fox" in Fig. 1 and predicting the correct target (the next word) "jumps", the RNN can make use of the previous words "The quick brown" and thus improve prediction performance. Each layer (each box in Fig. 1) in turn consists of multiple input, output, or hidden cells that are not individually shown in Fig. 1.

3.3. Long Short Term Memory (LSTM)

Recurrent networks with simple sigmoid cells have unsatisfactory performance for long time or sequence distances, leading to the development of long short term memory (LSTM) cells (Hochreiter and Schmidhuber, 1997), defined by the vector equations in Fig 2. An LSTM cell accepts C_{t-1} as state information from the prior unrolled cell on the same level, x_t as input from cells on the previous layer and h_{t-1} as input from the prior cell on the same layer. In turn, it passes C_t as new state

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \quad (1)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \quad (2)$$
$$\bar{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \quad (3)$$
$$C_t = f_t \times C_{t-1} + i_t \times \bar{C}_t \quad (4)$$
$$o_t = \sigma\left(W_o \cdot [h_{t-1}, x_t] + b_o\right) \quad (5)$$
$$h_t = o_t \times \tanh(C_t) \quad (6)$$

Figure 2: Definition of LSTM cell

information to the subsequent unrolled cell and provides h_t as output to the next layer and also to the subsequent unrolled cell on the same level.

All variables C, x, h, f, i, o in Fig. 2 are vectors in $\mathbb{R}^m$; the W and b (weights and biases) are "trainable" parameters of suitable dimensions. The functions tanh() and $\sigma()$ are applied elementwise, $[\ldots]$ represents vector concatenation. Eq 1–6 describe m individual LSTM cells, each operating on a single input from $\mathbb{R}$. The number of individual cells on each layer, i.e. the dimensionality m can be freely chosen.

The intuition behind these definitions is as follows. Eq. 1 represents the "forget gate" that determines, based on the inputs x_t and h_{t-1}, which part of the state to forget. Note that the prior state C_{t-1} is multiplied by f_t in Eq. 4 to derive the new state. With $\sigma()$ in Eq. 1 yielding values between 0 and 1, some information in C_{t-1} will, to some degree, be "forgotten". Eq. 2 and 3 determine how the inputs x_t and h_{t-1} contribute to the updated cell state. Eq. 2 represents the "input gate" and determines which values of the state to update; some of the i_t will be close to zero, others close to one. Eq. 3 computes new candidate values $\bar{C}$, which are multiplied with i_t in Eq. 4 when computing the new state C_t. Eq. 4 is the actual state update equation. The left term represents the "forgetting" of some prior state information while the right term represents the addition of new information to the cell state. Eq. 5 represents the "output gate". It determines which values of the cell state are provided as output to the following layer and subsequent unrolled cell. It is multiplied in Eq. 6 with the tanh of the cell state to produce the final output h_t.

To further improve the performance of LSTM cells, Gers and Schmidhuber (2000)

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i\right) \\
o_t &= \sigma\left(W_o \cdot [C_t, h_{t-1}, x_t] + b_o\right)
\end{aligned}
$$

Figure 3: Definition of peepholes for LSTM cells

introduce the idea of "peepholes", which allow the forget and input gates to look ("peep") at the prior state, and which allows the output gate to also look ("peep") at the current state C_{t-1}, with the equations in Fig. 3 replacing those in Fig. 2. While there exist other variants to this basic LSTM cell, (Greff et al., 2015) conclude after an experimental evaluation that "the most commonly used LSTM architecture, performs reasonably well on various datasets and using any of eight possible modifications does not significantly improve the LSTM performance".

3.4. Process Prediction

This work is motivated by the application of deep learning to natural language processing (NLP). In recent years, NLP research has moved from explicit representations of language models to statistical methods, specifically to recurrent neural networks (for example, Sutskever et al., 2011; Graves, 2013, 2012; Zaremba et al., 2014). A typical NLP application trains the RNN on sequences of input words to predict the next word, e.g. to provide word suggestions for user input. As shown in Fig 1, the target words are simply the input words shifted by one position, so that for each input word the following word is the target to be predicted.

Words are mapped into an n-dimensional "embedding" space $\mathbb{R}^{v \times m}$ using an "embedding matrix", which is essentially a look-up matrix of dimensions $v \times m$ where v is the size of the "vocabulary", i.e. the number of unique event types, and m is the chosen dimensionality of the embedding space and hence the size of each LSTM hidden layer (cf. Section 3.3). The input layer in Fig. 1 is an embedding

lookup function yields a numeric vector from $\mathbb{R}^m$ for each word in the vocabulary, which forms the input x_t in Eq. 1, 2 and 5 for the first LSTM layer. The embedding matrix is also a trainable parameter that is learned during training. The output layer produces a probability distribution over the words in the vocabulary. Each output "box" in Fig. 1 represents a softmax layer with v individual cells. The word with the highest probability is selected as the predicted word and compared to the target. Precision is assessed as the proportion of correct predictions.

In this work, we apply a recurrent neural network to the problem of predicting the next event in a process from a sequence of observed events. In implementing process prediction using RNN, our main idea is *to view an event log as a text, traces in the log as sentences, and events in a trace as analogous to words in a sentence.* As natural language is constrained by grammatical and morphological rules, such as noun and verb agreement for plurals in English, process event sequences are determined or constrained by an internal process logic, for example by decision rules based on case data. Just as linguistic rules and constraints are not explicitly captured in NLP deep-learning approaches (Sutskever et al., 2011; Graves, 2012, 2013; Zaremba et al., 2014) but are learned by the neural network, the process constraints and rules also need not be explicitly represented but can be learned.

However, while there are many similarities between natural language and process traces, there are important differences. First, the size of the vocabulary in process prediction (the number of event types) is much smaller than the size of a natural language vocabulary. Second, the length of a trace can far exceed the typical sentence length in natural language. Together, these two differences result in fewer possible prediction targets (vocabulary size or number of unique process event types) and more information to predict from (sentence or trace prefix length), suggesting that this approach may be able to achieve better results than word prediction in NLP.

Third, in contrast to natural language, process event sequences may represent activities that overlap temporally. But even in that case individual lifecycle events, for example "A-START" and "B-START", are strictly temporally ordered. However, the possibly arbitrary temporal sequence of events of parallel activities across different traces can make the prediction task more difficult, especially when there are no underlying regularities or dependencies imposed by other process characteristics such as resources or case attributes.

To empirically examine the similarity of process event logs and natural language text, we focus on two characteristics. First, the mutual information I of two words in natural language declines with their distance d according to a power law ($I \propto d^{-k}$), rather than an exponential manner ($I \propto \exp -d/k$) but Markov models cannot model this dependency, while RNN can (Lin and Tegmark, 2016). We compare the mutual information in our datasets against that of two natural language text corpora, the British National Corpus[1] and an extract from Project Gutenberg [2] (Fig. 4). As our event logs exhibit exponential decline, an RNN is not strictly necessary as they can be modelled by both Markov-based models and RNNs. This invites a comparison between the RNN and Markov-based approach, which we provide by comparing our method to the PFA method by Breuker et al. (2016). Second, we examine whether Zipf's law (Zipf, 1949) holds for the event logs. Zipf's law states that the frequency of words in a natural language corpus is inversely proportional to the frequency rank of that word. Figure 5) shows that our event logs also adhere to Zipf's law and are, in this characteristic, similar to natural language.

In summary, given that there are both similarities and differences, a proof-of-concept implementation to demonstrate feasibility and an experimental evaluation

[1] `http://www.natcorp.ox.ac.uk/`
[2] `https://www.gutenberg.org/`

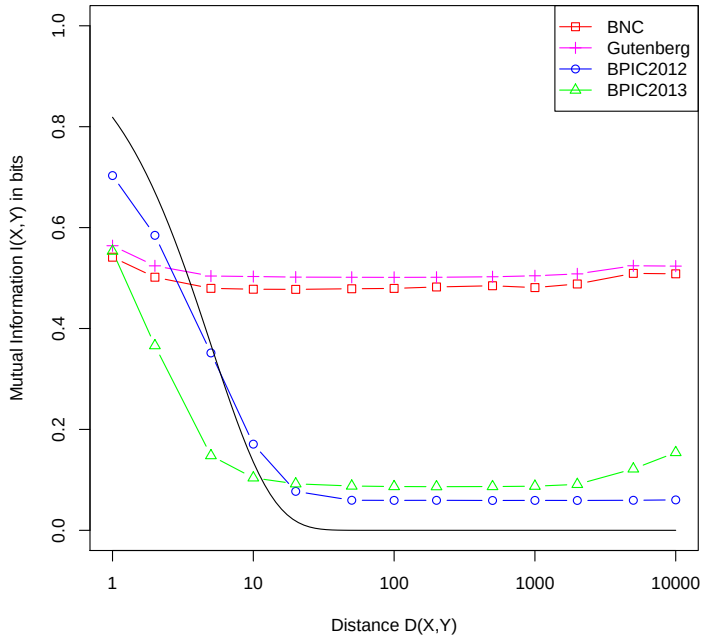

Figure 4: Mutual information of process event logs and English language text by separation distance (solid line is $I = \exp(-d/5)$)

to demonstrate effectiveness and performance are clearly required.

4. Implementation

A number of software frameworks for deep-learning have become available recently (Bahrampour et al., 2015). We implement our approach using Tensorflow as it provides a suitable level of abstraction, provides RNN specific functionality, and can be used on high-performance parallel, cluster, and GPU computing platforms. Code, data and complete results are available[3].

Our network features an architecture as in Fig. 1 with two hidden RNN layers using LSTM cells. This architecture still affords many choices, in particular the dimensionality of the embedding space and the number of unrolled steps.

[3] `http://joerg.evermann.ca/software.html`

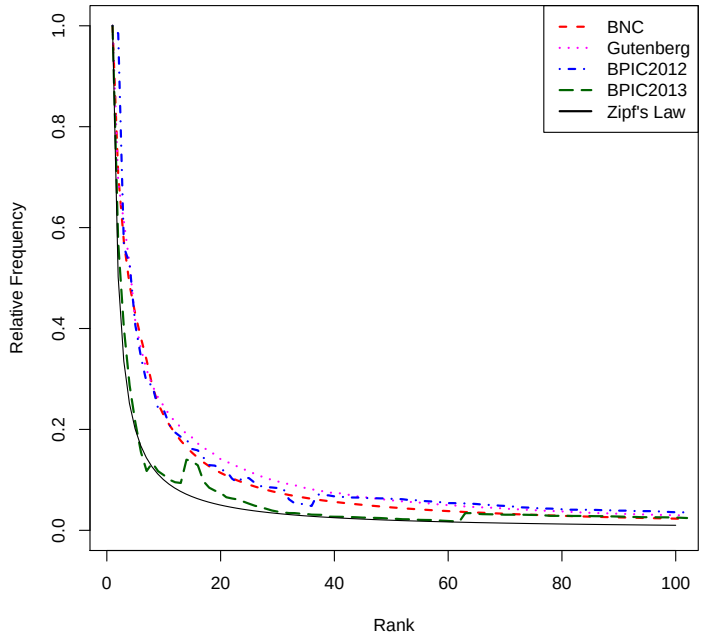

Figure 5: Relative frequency of terms by frequency rank (solid line is Zipf's law)

The dimensionality of the embedding space must be guided by the size of the vocabulary, i.e. the number of unique event types. A larger number of dimensions allows better separation of words in that space, which likely leads to better predictive performance, but at the cost of computational effort. Our baseline in the experimental evaluation (Section 5) is 125 and we explore the effect that varying this parameter has on predictive performance. The number of "unrolled" steps indicates the length of the sequence of words across which state information can be maintained. A larger number of unrolled steps allows the network to take earlier process events into account when predicting the following process event. This, too, is at the expense of computational effort. Our baseline for the experimental evaluation is 20 and we explore the effect of varying this parameter on predictive performance.

In addition to these two parameters, a number of general neural network options are parameterized, dealing with technical details such as gradient computation, ini-

Parameter	*Value*
Num epochs	100
Epochs w/ full base learning rate	50
Batch size	20
Max gradient norm	5.00
Dropout probability	0.20
Initialization scale	0.10
Base learning rate	1.00
Learning rate decay	0.75
LSTM forget bias	0.10

Table 1: RNN Parameters

tialization, and learning rate. Table 1 shows these options and their values in our implementation. Greff et al. (2015) show that these parameters are significantly less important in determining RNN performance than network size and are largely independent in their effect on RNN performance.

Trainable parameters are initialized using a uniform random distribution over $[-0.1, 0.1]$. Training proceeds in batches of size 20. For each batch, the backpropagation algorithm computes the mean gradients for all parameters. Training of the net proceeds in "epochs". Each epoch trains the net on the entire event log. Subsequent epochs maintain the weights W and biases b learned from the previous epoch but reinitialize the states for each layer and then train the net again on the entire event log. The net was trained for 100 epochs. The learning rate is reduced from 1 by a factor of 0.75 each epoch after the 50th. Dropout is a technique to prevent overfitting and improve learning by randomly temporarily removing a cell from the network (Srivastava et al., 2014). Dropout probability for each cell is 0.2.

We ran our experiments on GPU nodes with NVidia K1100M GPUs operating at 705MHz. Training performance is approximately 5500 words per second in the baseline experimental condition. Additionally, we used CPU nodes with Intel Xeon E5-2650 CPUs operating at 2.60GHz, for a training performance of approximately 7000 words per second in our baseline condition.

5. Experimental Evaluation Approach

5.1. Data

To provide a compelling evaluation of our approach, it should be compared to the state-of-the-art in next event prediction. Of the related work discussed in Section 2, only Breuker et al. (2016) make an implementation publicly available and use publicly available data, demonstrating "open research" (van der Aalst et al., 2016). We contacted all authors of the remaining papers twice, but did not receive software or data to use for comparative evaluation.

We use the same datasets as Breuker et al. (2016). The BPI Challenge 2012 dataset (van Dongen, 2012) is a real dataset from a loan application process in a Dutch financial institute with 13087 traces. It can be separated into three sub-processes concerning the application itself (A), the offer (O) and the work item (W) belonging to the application. The BPI Challenge 2013 datasets (van Dongen, 2013a,b) are real datasets from IT incident and problem management processes at Volvo Belgium with 7553 and 2300 traces, respectively.

In addition to separating the BPI 2012 data set by sub-process as done by Breuker et al. (2016), we also use the combined dataset. While Breuker et al. (2016) use only activity completion events for the BPI 2012 dataset, we also test our approach on all events (including the lifecycle transitions "start", "schedule" and "completion"). However, only the "W" subset has events other than completion events. Furthermore, we include an experimental condition where we not only extract the event name, but combine this with the name of the resource associated with the event, when available. We simply concatenate the two character strings to form the composite word. This creates a larger vocabulary which increases the prediction difficulty, but at the same time provides more information to the training algorithm. It also allows prediction of not only the next event but also the resource

Dataset	Number of unique event types ("vocabulary size")		Number of events
		w/ res. info	
BPI2013.Incidents	14	3133	65533
BPI2013.Problems	8	64	9011
BPI2012 (completion events)	24	877	164506
BPI2012 (all events)	37	1349	262200
BPI2012.W (completion events)	7	264	72413
BPI2012.W (all events)	7	736	169507
BPI2012.A	11	302	60849
BPI2012.O	8	313	31244

Table 2: Characteristics of datasets used in experimental evaluation. Event numbers for partial BPI2012 logs do not add up to that of corresponding complete log due to end-of-case marker events added to each trace.

associated with the next event. Because the number of distinct resources in the BPI 2013 Challenge dataset is very large, we combine the organizational group of each event with the event name. Table 2 shows characteristics of the datasets.

The published datasets are transformed using XSL transformations to extract traces, events, and resource information in a suitable format.

5.2. Evaluation Method

There are three random influences that require us to perform multiple runs for each experimental condition. First, the trainable parameters are initialized randomly and different initial values can lead to different outcomes (e.g. convergence to local optima). Second, the traces in a log are in a particular order ("log order") but this is an arbitrary order. Because the LSTM cells maintain state, the order in which traces are used to train the network may have an effect on the outcome. Finally, the selection of the training dataset itself has an influence, and the results obtained with the particular sample may not generalize to others. Recall that both BPI 2012 and 2013 datasets are *sampled* from running systems.

To address these stochastic influences and assess the generalizability of the results, we perform 10-fold cross-validation (Hastie et al., 2009). The separate perfor-

mance assessment of the validation sample and comparison to performance of the training sample provides an assessment of the generalizability to similar datasets. Specifically, it assesses whether the neural network *overfits* the training sample, i.e. exploits idiosyncrasies in the training sample, and what type of performance can realistically be expected on a dataset with similar characteristics. In comparing our results to the state-of-the-art, we note that Breuker et al. (2016) did not cross-validate the results of their stochastic EM-based approach. A fair comparison is therefore to our training results.

We report the prediction precision, defined as the proportion of correct predictions of all predictions made. We report mean and standard deviation of the training precision, as well as the mean and standard deviation of the precision achieved on the validation fold, across all 10 folds.

6. Experimental Results

We train the neural network for 100 epochs on each training dataset. Fig. 6 plots training and validation precision for each epoch for a selection of our datasets (averaged across all 10 training folds). The plot shows that 100 training epochs are sufficient for optimal and stable results. The small BPI 2012 A and O datasets converge quickly to a high precision, whereas this occurs more gradually for the BPI 2013 datasets. Moreover, the BPI 2012 A, BPI 2012 O, and BPI 2013 Problem datasets with relatively small vocabularies converge faster than the BPI 2012 W and BPI 2013 Incident datasets that include resource or organizational group information and consequently have a larger vocabulary. There is a significant improvement in precision when the training rate is adjusted in epoch 50, suggesting that a dynamic training rate is required to prevent suboptimal convergence. The second panel in Fig. 6 shows that validation precision generally follows the behaviour of training

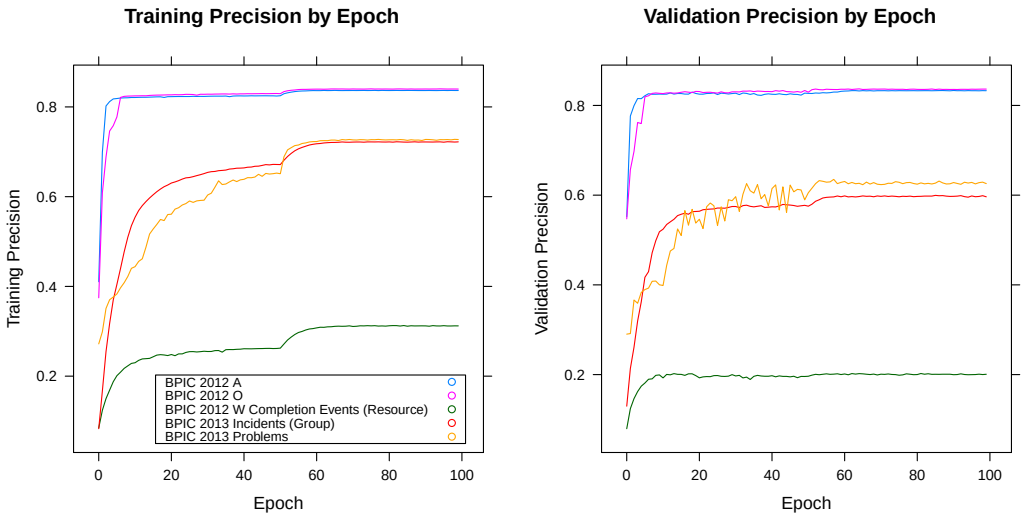

Figure 6: Training precision by training epoch for selected datasets (mean over 10 training folds)

Dataset	Precision in (Breuker et al., 2016)	Training Precision		Validation Precision	
		Mean	SD	Mean	SD
BPI2013.Incidents	.714	.757	.011	.735	.044
BPI2013.Problems	.690	.727	.037	.628	.086
BPI2012 (completion events)		.796	.001	.788	.006
BPI2012 (all events)		.864	.001	.859	.005
BPI2012.W (completion events)	.719	.676	.002	.658	.020
BPI2012.W (all events)		.839	.001	.832	.010
BPI2012.A	.801	.837	.001	.832	.010
BPI2012.O	.811	.841	.001	.836	.010

Table 3: Results and comparison to (Breuker et al., 2016)

precision at a lower level.

6.1. Predicting the Next Event

Table 3 shows our results and a comparison to the best result presented by Breuker et al. (2016) or computed by us using their implementation. The table shows that our approach surpasses the state-of-the-art on many datasets. For the BPI 2013 Incidents data, the BPI 2012 A, and the BPI 2012 O subsets, our training and validation precision values are above the best values reported by Breuker et al.

(2016). For the BPI 2013 Problems dataset, our approach offers better training precision, but the validation precision is lower than the precision reported by Breuker et al. (2016), and on the BPI 2012 W dataset, our approach performs relatively poorly. Computation time for the RegPFA approach ranges from a few hours (BPI 2012 O) to multiple weeks (BPI 2012 W, all events), in constrast to our significantly faster RNN approach (minutes to hours).

Comparing the training and validation precision shows that relatively little overfitting occurs, with the validation precision generally within approximately ± 0.05 of the training precision. The standard deviations for the validation precision are an order of magnitude above those for the training sample because of the much smaller size of the validation sample (one tenth of the training sample size).

Table 3 shows many results with validation precision close to or in excess of 0.8. While we have no comparison to the state-of-the-art on these datasets by Breuker et al. (2016), this level of precision is encouraging for practical applications. It is also generally above the levels reported in related works (see Section 2), although this must be interpreted with caution due to different datasets with possibly very different characteristics.

Prediction is significantly better for the BPI 2012 W dataset with all events compared to the same dataset with completion events only as it exploits the regularity that after every START event for an activity, the corresponding COMPLETE event for that activity follows.

6.2. The Effect of Resource Information

Including the organizational resource or group in the predicting data provides additional information that should lead to improved prediction precision. Including this information in both predictor and predictand allows prediction not only of the next process event but also of the resource associated with that next process event.

Dataset	Training Precision		Validation Precision	
	Mean	SD	Mean	SD
BPI2013.Incidents	.832	.034	.761	.049
BPI2013.Problems	.776	.081	.616	.117
BPI2012 (completion events)	.833	.004	.811	.023
BPI2012 (all events)	.880	.007	.866	.008
BPI2012.W (completion events)	.745	.013	.693	.019
BPI2012.W (all events)	.865	.012	.845	.012
BPI2012.A	.846	.006	.826	.007
BPI2012.O	.870	.015	.833	.015

Table 4: Precision when including resources or organizational groups in only the predictand.

Dataset	Training Precision		Validation Precision	
	Mean	SD	Mean	SD
BPI2013.Incidents	.731	.010	.595	.061
BPI2013.Problems	.764	.034	.517	.043
BPI2012 (completion events)	.622	.005	.594	.010
BPI2012 (all events)	.685	.007	.663	.015
BPI2012.W (completion events)	.313	.012	.208	.023
BPI2012.W (all events)	.609	.018	.559	.018
BPI2012.A (completion events)	.758	.010	.709	.008
BPI2012.O (completion events)	.703	.026	.590	.020

Table 5: Precision when including resources or organizational groups in both the predictor and predictand.

However, this comes at the cost of a larger vocabulary and should lead to lower prediction precision. We explored both options, and the results in Tables 4 and 5 are generally as expected.

When including the information about organizational group or resource only in the predictor (Table 4), the training precision improves considerably for all datasets, by up to 0.07, compared to the baseline in Table 3. However, the validation precision does not follow the same pattern, improving only for some datasets, and even then improving to a lesser extent as the training precision. This suggests that including the organizational information for prediction may be useful but also runs the danger of overfitting the model.

Including the organizational resource or group in both the predictor and predictand affects the prediction precision for the datasets in different ways, as shown in Table 5. Training precision for the BPI 2013 Incident dataset is somewhat lower, whereas training precision increases for the BPI 2013 Problem dataset. However, cross-validated precision drops significantly for both datasets. Training and validation precision for all BPI 2012 datasets also drop significantly. An extreme example is the BPI 2012 W dataset with completion events only, where training precision drops to .313 and validation precision to .208. Here, the resources of events do not follow any pattern or underlying regularity. In contrast, the BPI2012.W with all events shows the regularity that the resource of the completion event is the same as that of the start event, which leads to improved precision performance.

6.3. Predicting Duration of Activities

An RNN can also be used to predict the duration of activities. For this, we quantize the length of a trace in minutes and, for each minute, encode the current activity in the input. Hence, if activity A occurs for 3 minutes, the input consists of the sequence AAA. Only the BPI 2012 W dataset includes both start and completion events to allow determination of the duration of activities[4]. While it is possible to encode idle time between two activities quantized in minutes as well, the BPI 2012 W dataset describes long-running cases, which would have led to very long and monotonous sequences.

The validation precision for this case is 0.942, (SD=0.027), significantly higher than the 0.832 reported in Table 3 for the case with no duration information. This increase in precision is not surprising, as the dataset consists of longer sequences of identical words, thus making prediction of the following word much easier.

[4]This kind of encoding is sensible only when activities do not temporally overlap, which is the case for this dataset.

6.4. The Effect of Embedding Space Dimensionality

To identify the effect of the dimensionality of the embedding on training and validation precision, we repeat our experiments using embedding spaces of 500, 64, 32, 16, and 8 dimensions, keeping the other parameters constant. Table 6 shows the validation precision of selected datasets for different numbers of dimensions. Fig. 7 shows a plot of training and validation precision for three datasets without organizational information, and the same three datasets when combined with organizational information.

As expected, reducing the dimensionality of the embedding has, in general, a detrimental effect on the prediction precision. However, this effect is negligible as long as the dimensionality of the embedding space is greater than the size of the vocabulary. This is the case for all datasets that do not include organizational information. For example, neither the BPI 2013 datasets with 14 and 8 event types, nor the BPI 2012 dataset with 24 event types in Fig. 7 show a marked reduction in training or validation precision when the dimensionality is reduced from 64 to 32 and to 16. However, when the dimensionality is reduced to 8, the reduction in precision is more pronounced for all datasets.

Given the larger vocabulary when including organizational information, the effect of dimensionality is more pronounced at smaller dimensions, as is visible in Fig. 7. The BPI 2013 datasets with 3133 and 64 unique combinations of event types and organizational group, and the BPI 2012 dataset with 877 unique combinations, show a significant reduction in precision when the dimensionality is reduced from 125 to 64. Other datasets with large vocabulary show the same behaviour.

Increasing the dimensionality to 500, done only for selected datasets due to the required computational effort, shows that significant overfitting occurs, especially for datasets with a large vocabulary. As shown in Fig. 7, the training precision for

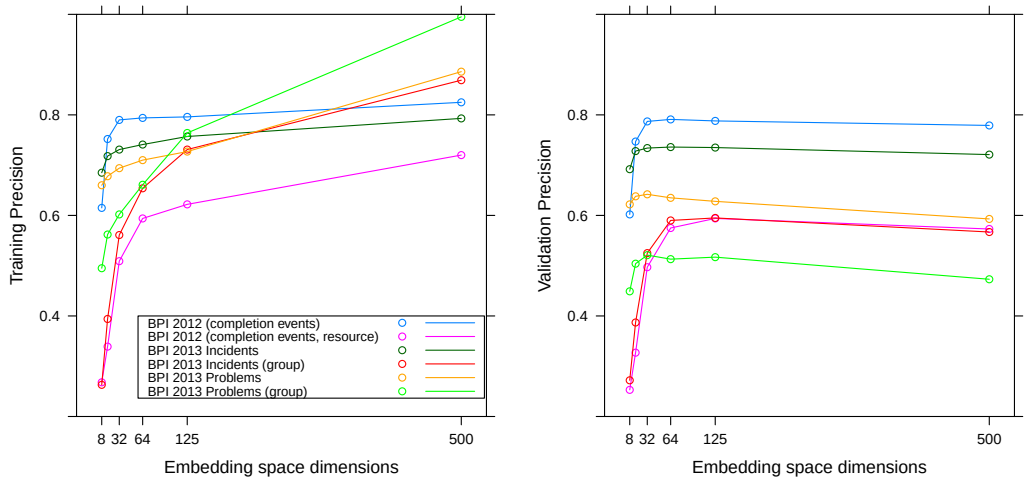

Figure 7: Training and validation precision against embedding space dimensions for selected datasets

the BPI 2012 and BPI 2013 datasets increases significantly whereas the validation precision is not only much lower than the training precision, but decreases with increasing dimensionality. These trends are clear indications of overfitting.

Examining Fig. 7 and the corresponding data in Table 6 shows that our chosen baseline with 125 dimensions is close to the optimum validation precision for all datasets and does not suffer from significant overfitting.

6.5. The Effect of the Number of Unrolled Steps

To identify the effect that the number of unrolled steps has, we repeat our experiments using 10 and 5 unrolled steps, keeping the other parameters constant. Fig. 8 and Table 7 show the training and validation precision of selected datasets against the number of unrolled steps. Neither training nor validation precision is significantly affected by the number of unrolled steps. There appears to be a minor effect for the BPI 2013 Problem dataset (more pronounced when including the organizational group), where the training performance decreases with increasing

Dataset	Dimensionality of Embedding Space					
	500	125	64	32	16	8
BPI2013.Incidents	.721	.735	.736	.734	.728	.692
BPI2013.Problems	.593	.628	.635	.642	.638	.622
BPI2012 (completion events)	.779	.788	.791	.787	.747	.602
BPI2012 (all events)		.859	.859	.854	.772	.578
BPI2012.W (completion events)	.638	.658	.660	.661	.657	.646
BPI2012.W (all events)		.832	.833	.832	.824	.733
BPI2012.A	.833	.832	.833	.834	.832	.799
BPI2012.O	.827	.836	.836	.837	.836	.823
BPI2013.Incidents (with group)	.567	.595	.590	.525	.387	.272
BPI2013.Problems (with group)	.473	.517	.513	.521	.504	.449
BPI2012 (completion events, resource)	.573	.594	.575	.497	.327	.253
BPI2012 (all events, resource)		.663	.618	.456	.328	.188
BPI2012.W (completion events, resource)	.130	.208	.214	.204	.174	.147
BPI2012.W (all events, resource)		.559	.558	.495	.302	.134
BPI2012.A (resource)	.714	.709	.717	.717	.688	.615
BPI2012.O (resource)	.577	.590	.598	.588	.482	.263

Table 6: Validation precision for different dimensions of the embedding space. Due to computational requirements, 500 dimensions applied only to selected datasets.

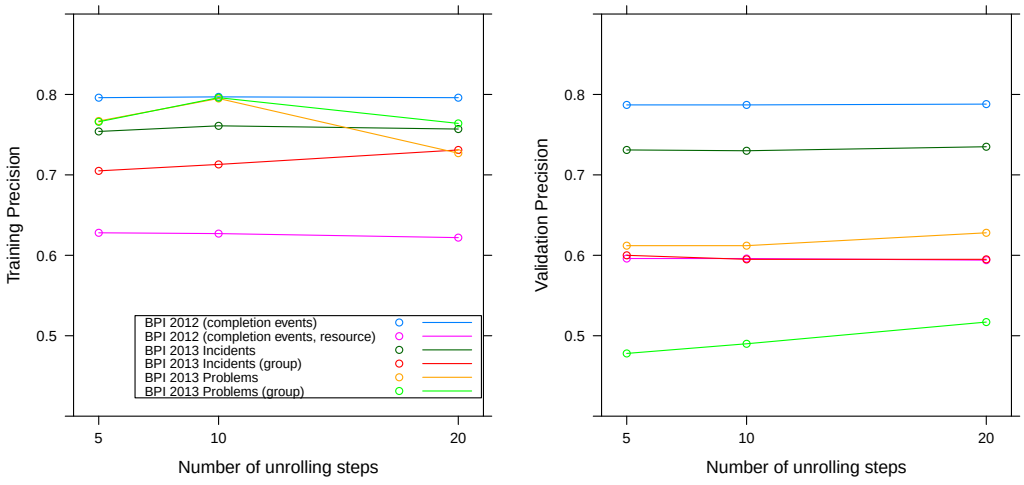

Figure 8: Validation precision for three values of unrolled steps for selected datasets

number of unrolled steps, while the validation performance increases. This indicates that there are few to no long-term dependencies in the different processes that cannot be captured by even five previous steps, which is in line with the mutual information considerations in Section 3.4.

Dataset	Unrolled Steps		
	20	10	5
BPI2013.Incidents	.735	.730	.731
BPI2013.Problems	.628	.612	.612
BPI2012 (completion events)	.788	.787	.787
BPI2012 (all events)	.859	.860	.860
BPI2012.W (completion events)	.658	.657	.657
BPI2012.W (all events)	.832	.831	.831
BPI2012.A	.832	.833	.834
BPI2012.O	.836	.836	.836
BPI2013.Incidents (with group)	.595	.595	.600
BPI2013.Problems (with group)	.517	.490	.478
BPI2012 (completion events, resource)	.594	.596	.596
BPI2012 (all events, resource)	.663	.667	.667
BPI2012.W (completion events, resource)	.208	.201	.201
BPI2012.W (all events, resource)	.559	.558	.558
BPI2012.A (resource)	.709	.710	.712
BPI2012.O (resource)	.590	.590	.590

Table 7: Validation precision for different number of unrolled steps

6.6. Interpreting the RNN

One of the main criticisms of neural networks is the fact that no explicit, human understandable model is created; the learning consists entirely in the optimization of the values of thousands or millions of floating point parameters. We present two ways in which users can gain insight into what the neural net has learned, i.e. the knowledge that is encoded in its structures.

6.6.1. Process Hallucinations

Neural networks, especially recurrent neural networks, can be made to "hallucinate", i.e. to generate output on their own, for example in language modeling (Graves, 2013; Sutskever et al., 2011; Karpathy, 2016) or music creation (van den Oord et al., 2016; Huang and Wu, 2016; Choi et al., 2016). This is done by feeding the net output (prediction) immediately back as new input. Hallucinations can provide insight into what the RNN has learned about the training set. The ability of an RNN to re-produce, on its own, realistic and convincing process traces, demon-

strates that it has correctly learned the important features of the event log used as training sample and thereby validates the trained net and supports the usefulness of the RNN approach for process prediction.

Hallucinations can be initialized in different ways, simply by using the state at the end of the training step ($i = 0$), a random initial state ($i = 1$), or by providing a short "seed" sequence of events ($i = 2$). For the output, one may simply accept the event with the highest probability ($k = 0$), or one samples from events using the output probabilities ($k = 1$). Finally, one can feed the output back to the net for each set of unrolled steps at a time ($m = 0$), or one can do this element-wise at each step ($m = 1$). We explore all combinations of these different methods of producing hallucinations for the full BPI 2012 and BPI 2013 datasets. For each combination of hallucination methods (i.e. each combination of parameters i, k, m) and for each dataset, we train a net with 32 embedding dimensions and 5 unrolled steps for 100 epochs. After training is complete, we produce 20 hallucinations of sequence length 1000 from each trained net. An excerpt of the BPI2012 hallucinations is shown in Fig. 9. The complete set of generated hallucinations is too large to produce here, but is available from the authors.

In addition to visually inspecting the generated hallucinations and judging their realism, we mine a process model from the original event log using the Flexible Heuristics Miner (Weijters and Ribeiro, 2011) with default parameters. We choose the FHM because it performs well on a wide variety of models (Claes and Poels, 2012). We replay both the original log and the generated hallucinations against this model and compute the replay fitness, which indicates whether the model can generate the observed behavior (van der Aalst et al., 2012). A replay fitness of the hallucinations similar to that of the original model is a strong indication that the generated traces are realistic. We also examine whether the frequency distribution

```
ASUBMITTED APARTLYSUBMITTED ADECLINED WAfhandelenleads [EOC] ASUBMITTED
APARTLYSUBMITTED APREACCEPTED WAfhandelenleads WCompleterenaanvraag
WCompleterenaanvraag WCompleterenaanvraag AACCEPTED AFINALIZED OSELECTED
OCREATED OSENT WCompleterenaanvraag WNabellenoffertes WNabellenoffertes
WNabellenoffertes OSENTBACK WNabellenoffertes WValiderenaanvraag
WValiderenaanvraag WValiderenaanvraag WNabellenincompletedossiers OCANCELLED
OSELECTED OCREATED OSENT WNabellenincompletedossiers WNabellenincompletedossiers
WNabellenincompletedossiers WNabellenincompletedossiers
WNabellenincompletedossiers ACANCELLED OCANCELLED WNabellenincompletedossiers
[EOC] ASUBMITTED APARTLYSUBMITTED WAfhandelenleads ADECLINED WAfhandelenleads
[EOC] ASUBMITTED APARTLYSUBMITTED ADECLINED [EOC] ASUBMITTED APARTLYSUBMITTED
APREACCEPTED AACCEPTED OSELECTED AFINALIZED OCREATED OSENT WCompleterenaanvraag
WNabellenoffertes WNabellenoffertes ADECLINED ODECLINED WNabellenoffertes [EOC]
```

Figure 9: Example hallucinations for the BPI2012 dataset ($i = 1, k = 1, m = 0$, "[EOC]" designates end-of-case)

of events in the generated hallucinations matches that in the original dataset using the Kolmogorov-Smirnov test. A non-signifant test (p-value ¿ 0.05) indicates the frequency distributions come from the same underlying population.

Table 8 shows the results for the BPI2012 dataset. Probabilistic output sampling ($k = 1$) produces more realistic results than choosing the most likely output ($k = 0$); the latter produces many long uniform event sequences that are uncharacteristic of the input logs. Element-wise feedback ($m = 0$) performs better than in sets ($m = 1$); the latter results in many empty traces being created. In particular, the combination $k = 1$ and $m = 0$ produces logs that are highly similar to the original log. The type of initialization (i) does not appear to have any significant effect on the generated hallucinations beyond the first few events. Results for the other event logs are similar. In summary, the generated hallucinations show that the neural networks learn the relevant features of the input event logs.

Hallucinations can also be used to predict the remainder of a case. For this, the hallucination is initialized with a trace prefix and then continued until it produces an end-of-case indicator. The hallucination can then be compared with the actual

	$k=0$						
i	full	0	0	1	1	2	2
m	log	0	1	0	1	0	1
Mean trace length	13	56	26	26	13	21	16
Max trace length	96	247	221	249	103	194	106
Replay fitness	.586	.628	.241	.331	.317	.350	.301
KS test p-value	—	.000	.237	.000	.237	.000	.414
	$k=1$						
i	full	0	0	1	1	2	2
m	log	0	1	0	1	0	1
Mean trace length	13	13	11	13	11	12	11
Max trace length	96	63	81	50	55	56	56
Replay fitness	.586	.572	.339	.578	.321	.582	.310
KS test p-value	—	.990	.878	.990	.878	.878	.414

Table 8: Hallucination characteristics for BPI2012 (completion events only) dataset

Dataset	Damerau–Levenshtein Distance	
	Mean	SD
BPI2013.Incidents	.563	.199
BPI2013.Problems	.616	.177
BPI2012 (completion events)	.659	.203
BPI2012.W (completion events)	.703	.205
BPI2012.W (all events)	.697	.211
BPI2012.A	.545	.241
BPI2012.O	.532	.191

Table 9: Mean and standard deviation of Damerau-Levenshtein distance between actual trace remainders and those predicted from a prefix of length 5. Hallucinations produced using probability sampling ($k = 1$) and element-wise feedback ($m = 1$); smaller is better.

trace continuation using a string-edit distance. We trained nets with 32 embedding dimensions and 5 unroll steps for 100 epochs. Using trace prefixes of length 5, we produced hallucinations and compared them to the actual continuation using the normed Damerau–Levenshtein distance, which ranges from 0 to 1 (Table 9).

6.6.2. Hidden State Dynamics

Recent work on understanding RNNs also focuses on visualization. In particular, visualizations of the embedding matrix, the state activation and the state dynamics are useful in understanding how an RNN encodes knowledge (Karpathy et al., 2015; Li et al., 2015; Yosinski et al., 2015; Strobelt et al., 2016).

We export embedding matrices after completion of training with our datasets to create 2D t-SNE plots (Maaten and Hinton, 2008). They are not included as no significant or obvious clustering of events is discernible for any of our datasets.

We use LSTMVis (Strobelt et al., 2016), which is motivated by earlier work by Karpathy et al. (2015), to examine the activation of hidden state cells for different inputs[5]. The visualization assists in identifying cells that are active for some input but not others (the "hypothesis selection process", Strobelt et al., 2016), and then confirming such knowledge encoding hypotheses by comparing activation patterns against similar input (the "matching process", Strobelt et al., 2016)).

Figure 10 shows the tool in use with hidden states on the first level of the trained RNN for the BPI2012 dataset with 16 embedding dimensions. The event sequence "ADECLINED [EOC]" is selected on the timeline (top part of figure). The inputs that match the activation pattern of the selected event sequence are sequences of the same two events, shown at the bottom of the image. This cell appears to represent the ending of cases with declined loan applications. In some cases, another event occurs in between the "ADECLINED" and "[EOC]" and this input is represented by the same cell activation pattern. Other hidden state cells represent the events "WNABELLENOFFERTES", "WCOMPLETERENANVRAAG", and the sequence "AACCEPTED AFINALIZED OSELECTED".

7. Discussion and Conclusion

To our knowledge, this is the first use of RNNs for process prediction and presents a novel approach to predicting the behaviour of running processes. Our results, surpassing or close to the state-of-the-art on two real datasets and with cross-validated

[5]Because the LSTMVis provides an interactive visualization for exploring state dynamics, it is difficult to provide in static form in this paper. The fully interactive visualization is available at http://rnnprocess.evermann.ca

Figure 10: LSTM visualization of states for declined loan application processes

precision in excess of 80% on many problems, demonstrate the feasibility and usefulness of this approach and encourage further work in this direction.

Using deep learning on even relatively small samples requires significant computation resources. With training rates of 5000 to 7000 words per seconds, many of our experimental evaluations take multiple days, especially in light of 10-fold cross-validation and large embedding spaces. However, most of this time is spent on training with backpropagation. Once trained, generating predictions using only forward computation is significantly faster. In practical applications, cross-validation may not be necessary. For example, a company trying to predict the next event in a stable process does not need to consider generalization beyond the training sample. On the other hand, variable processes, i.e. in the presence of concept drift, require repeated checks and validatation of the prediction performance to identify the need for retraining the network. The dimensionality of the embedding space should not

be much smaller than the number of event types, and should not be too large to avoid overfitting. For the processes and event logs considered here, values around 32 to 125 dimensions produce good cross-validated precision.

We have shown that including organizational information into the predictor may help in predicting the next process step. While we did not experiment with predicting the next event and resource from the prior events only, intuition suggests that this type of prediction is more difficult than predicting this information from both prior events and resource, as we have done here. In light of the effect that the inclusion of organizational information has on the vocabulary size, and thus also on the required dimensionality of the embedding space, practical applications must determine whether it is necessary to include this information as it comes at significant computational cost.

As this research is still early work with the deep learning approach for process management, we recognize its limitations of this study and the need for further research. In particular, more exploration of the technical parameters in Table 1 is required. For example, one can introduce additional RNN layers, adjust the learning rate to be more or less "aggressive", adjust the clipping of gradients to allow faster, but possibly sub-optimal, convergence, and adjust the initialization of network parameters.

Another area of inquiry is to add additional information into the predictors and/or the predictands. In our experiments, we have added organizational information by concatenating the process event and the resource, thus extending the vocabulary. One can add case attribute information in a similar way. This requires the identification of a subset of case attributes whose value space is not too large. Similarly, one can include compliance with important LTL goals in the vocabulary. These variations will still yield a sequence of "words" that can be used within the

RNN framework that we have applied here.

Finally, the deep learning approach can also be applied to the prediction of process outcomes. Process outcomes, such as remaining time to completion or violation of an LTL compliance expression, are continuous or categorical but not in the form of sequences. Both are suitable for neural networks but do not require the recurrent neural network architecture used here and more "traditional" architectures need to be explored and evaluated.

Acknowledgements

The authors gratefully acknowledge the support of the Memorial University Center for Health Informatics and Analytics, St. John's, Canada and the Hasso-Plattner-Institute at the University of Potsdam, Germany, in providing access to computing resources.

References

Bahrampour, S., Ramakrishnan, N., Schott, L., Shah, M., 2015. Comparative study of deep learning software frameworks. arXiv preprint arXiv:1511.06435.

Breuker, D., Matzner, M., Delfmann, P., Becker, J., 2016. Comprehensible predictive models for business processes. MIS Quarterly In press.

Ceci, M., Lanotte, P. F., Fumarola, F., Cavallo, D. P., Malerba, D., 2014. Completion time and next activity prediction of processes using sequential pattern mining. In: Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings. pp. 49–61.

Choi, K., Fazekas, G., Sandler, M. B., 2016. Text-based LSTM networks for automatic music composition. CoRR abs/1604.05358.

Claes, J., Poels, G., 2012. Process mining and the prom framework: An exploratory survey. In: Business Process Management Workshops. Vol. 132 of Lecture Notes in Business Information Processing. Springer, pp. 187–198.

Gers, F. A., Schmidhuber, J., 2000. Recurrent nets that time and count. In: IJCNN (3). pp. 189–194.

Graves, A., 2012. Supervised Sequence Labelling with Recurrent Neural Networks. Vol. 385 of Studies in Computational Intelligence. Springer.

Graves, A., 2013. Generating sequences with recurrent neural networks. CoRR abs/1308.0850.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., Schmidhuber, J., 2015. LSTM: A search space odyssey. CoRR abs/1503.04069.

Hastie, T., Tibshirani, R., Friedman, J., 2009. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer Verlag, Berlin, Germany.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Computation 9 (8), 1735–1780.

Houy, C., Fettke, P., Loos, P., van der Aalst, W. M. P., Krogstie, J., 2010. Bpm-in-the-large - towards a higher level of abstraction in business process management. In: EGES/GISP. Vol. 334 of IFIP Advances in Information and Communication Technology. Springer, pp. 233–244.

Huang, A., Wu, R., 2016. Deep learning for music. CoRR abs/1606.04930.

Karpathy, A., 2016. The unreasonable effectiveness of recurrent neural networks. Accessed 6 Dec, 2016.

Karpathy, A., Johnson, J., Li, F., 2015. Visualizing and understanding recurrent networks. CoRR abs/1506.02078.

Lakshmanan, G. T., Shamsi, D., Doganata, Y. N., Unuvar, M., Khalaf, R., 2015. A markov prediction model for data-driven semi-structured business processes. Knowl. Inf. Syst. 42 (1), 97–126.

Le, M., Gabrys, B., Nauck, D., 2012. A hybrid model for business process event prediction. In: SGAI Conf. Springer, pp. 179–192.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. Nature 521, 436–444.

Li, J., Chen, X., Hovy, E. H., Jurafsky, D., 2015. Visualizing and understanding neural models in NLP. CoRR abs/1506.01066.

Lin, H. W., Tegmark, M., Jun. 2016. Critical Behavior from Deep Dynamics: A Hidden Dimension in Natural Language. ArXiv e-prints.

Maaten, L. v. d., Hinton, G., 2008. Visualizing data using t-sne. Journal of Machine Learning Research 9 (Nov), 2579–2605.

Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural Networks 61, 85–117.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15 (1), 1929–1958.

Strobelt, H., Gehrmann, S., Huber, B., Pfister, H., Rush, A. M., 2016. Visual analysis of hidden state dynamics in recurrent neural networks. CoRR abs/1606.07461.

Sutskever, I., Martens, J., Hinton, G. E., 2011. Generating text with recurrent neural networks. In: ICML. Omnipress, pp. 1017–1024.

Unuvar, M., Lakshmanan, G. T., Doganata, Y. N., 2016. Leveraging path information to generate predictions for parallel business processes. Knowl. Inf. Syst. 47 (2), 433–461.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., Kavukcuoglu, K., 2016. Wavenet: A generative model for raw audio. CoRR abs/1609.03499.

van der Aalst, W. M. P., Adriansyah, A., van Dongen, B. F., 2012. Replaying history on process models for conformance checking and performance analysis. Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery 2 (2), 182–192.

van der Aalst, W. M. P., Bichler, M., Heinzl, A., 2016. Open research in business and information systems engineering. Business & Information Systems Engineering 58 (6), 375–379.

van der Aalst, W. M. P., Pesic, M., Song, M., 2010. Beyond Process Mining: From the Past to Present and Future. Springer, Berlin, Heidelberg, pp. 38–52.

van Dongen, B., 2012. Business process intelligence 2013 challenge data set. `http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f`.

van Dongen, B., 2013a. Business process intelligence 2013 challenge data set (closed problems). `http://dx.doi.org/10.4121/uuid:c2c3b154-ab26-4b31-a0e8-8f2350ddac11`.

van Dongen, B., 2013b. Business process intelligence 2013 challenge data set (incident management). `http://dx.doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee`.

Weijters, A. J. M. M., Ribeiro, J. T. S., 2011. Flexible heuristics miner (FHM). In: CIDM. IEEE, pp. 310–317.

Yosinski, J., Clune, J., Nguyen, A. M., Fuchs, T., Lipson, H., 2015. Understanding neural networks through deep visualization. CoRR abs/1506.06579.

Zaremba, W., Sutskever, I., Vinyals, O., 2014. Recurrent neural network regularization. CoRR abs/1409.2329.

Zipf, G. K., 1949. Human behavior and the principle of least effort.