DeepProcess: Supporting Business Process Execution Using a MANN-Based Recommender System

Asjad Khan[1], Hung Le[2](✉), Kien Do[2], Truyen Tran[2], Aditya Ghose[1], Hoa Dam[1], and Renuka Sindhgatta[3]

[1] University of Wollongong, Wollongong, Australia
maak458@uowmail.edu.au, {aditya,hoa}@uow.edu.au
[2] Deakin University, Geelong, Australia
{lethai,dkdo,truyen.tran}@deakin.edu.au
[3] IBM India Research Laboratory, Bangalore, India
renuka.sr@in.ibm.com

Abstract. Process-aware Recommender systems can provide critical decision support functionality to aid business process execution by recommending what actions to take next. Based on recent advances in the field of deep learning, we present a novel memory-augmented neural network (MANN) based approach for constructing a process-aware recommender system. We propose a novel network architecture, namely Write-Protected Dual Controller Memory-Augmented Neural Network(DCw-MANN), for building prescriptive models. To evaluate the feasibility and usefulness of our approach, we consider three real-world datasets and show that our approach leads to better performance on several baselines for the task of suffix recommendation and next task prediction.

1 Introduction

Business process management assists organizations in planning and executing activities that collectively deliver business value, usually in the form of a product or a service. Flexible execution of business process instances entails multiple critical decisions, involving various actors and objects, which can have a major impact process performance and achieving desired process outcomes [32]. These decisions therefore require careful attention, as sub-optimal decisions during process execution, can lead to cost overruns, missed deadlines and the risk of failure [11]. While the problem of predicting the behaviour of a given process instance has been studied extensively, using these predictions to support operational decision-making of the kinds outlined above remains a challenge [6,22]. *Process-Aware Recommender Systems* have been proposed to assist knowledge workers in operational decision-making, for instance, by recommending actions leading to process end, managing resource allocation policies and so on [1,28]. In this work, we present a novel *Process-Aware Recommender System* for supporting organizations and process owners in operational decision-making (related to control-flow).

H. Hacid et al. (Eds.): ICSOC 2021, LNCS 13121, pp. 19–33, 2021.
https://doi.org/10.1007/978-3-030-91431-8_2

Recent advances in neural network architectures and learning algorithms have led to the popularization of *Deep Learning* methods which are particularly good at automated feature discovery and learning robust representations from large quantities of raw data, thus significantly reducing the need to hand-craft features which is typically required when using traditional machine learning techniques [20]. Deep Learning based techniques such as Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) have generated considerable interest recently for tackling various Process Analytics tasks (e.g. predictive monitoring). However, LSTMs and GRU methods lack the capacity to solve complex, structured tasks that, for example, require reasoning and planning [10,13]. To tackle such complex tasks, two promising approaches based on neural networks have been proposed: Memory Networks and Neural Turing Machines, both being instantiations of *Memory-Augmented Neural Networks* (MANN) [12]. In this paper, we investigate the applicability of MANNs for building a *Process-Aware Recommender System* that can provide process execution decision support of the kind discussed above.

Contributions: We propose a novel neural network architecture, namely *Write-Protected Dual Controller Memory-Augmented Neural Network(DCw-MANN)*, for building a Process-Aware Recommender System, where we introduce several modifications to the existing Differential Neural Computer(DNC) architecture: (i) *separating the encoding phase and decoding phase, resulting in dual controllers, one for each phase*; (ii) *implementing a write-protected policy for memory during the decoding phase.* We evaluate the effectiveness of our approach on three world datasets for the task of *generating suffix recommendations* that lead to optimal outcomes.

The paper is organized as follows: In Sect. 2, we provide the necessary background on Process Analytics and Deep Learning techniques upon which our proposed method is built. In Sect. 3, we explain the technical workings of our Process-Aware Recommender System, designed to tackle a number of prescriptive process analytics tasks. Implementation details and experimental results are reported in Sect. 4. Finally, Sect. 5 discusses related work, followed by Sect. 6 which concludes the paper and outlines future work.

2 Preliminaries

We first briefly present the existing work upon which our method is built, including event log presentation, recurrent neural networks, and Long Short-Term Memory (LSTM).

2.1 Process Analytics

Process analytics involves a sophisticated layer of data analytics built over the traditional notion of process mining [33]. Compared to Process mining, Process analytics addresses the more general problem of leveraging data generated by, or associated with, process execution to obtain actionable insights about business processes. Process analytics leverages a range of data, including, but not limited to process logs, event logs [26], provisioning logs, decision logs and process context [29] and answers queries that have a number of real world applications particularly related to prescriptive analytics such as resource optimisation and instance prioritisation. In this paper we focus on event logs and assume that when a business process instance is executed, its execution trace is recorded as an event log. An event log is a sequence of events, naturally ordered by the associating timestamps.

In predictive analytics, we study techniques that allow us to predict how the future of a given process instance will unfold and the likely occurrence of future process events [10]. It can be considered as computing (a) a set of functions and (b) a set of computer programs that carry out computation, over a (partially executed) process instance. An example of case (a) is computing remaining time of a process instance, which is the *sequence-to-vector* setting. An example of case (b) is a continuation of a partially executed process, which is the *sequence-to-sequence* setting.

Prescriptive business process monitoring techniques and Process Aware Recommender systems are for providing decision-support to process users. Applications of such system include, offering recommendations about: *(i)* next activities to execute, *(ii)* resource allocation support, *(iii)* Cost and time optimization and *(iv)* risk-mitigation by raising alarms or recommending actions to prevent undesired outcomes [8,35].

2.2 Sequence Modeling with Deep Learning

Recurrent Neural Nets(RNNs), especially the Long Short-Term Memory (LSTM) have brought about breakthroughs in solving complex sequence modelling tasks in various domains such as video understanding, speech recognition and natural language processing [20,27]. Similarly, it has been shown that LSTM can consistently outperform classical techniques for a number of process analytics tasks such as predicting the next activity, time to the next activity etc. [24,31].

Recurrent neural network (RNN) is a model of dynamic processes, and to some degree, a model of computer programs. At each time step t, a RNN reads an *input vector* $\boldsymbol{x}_t$ into a *hidden state vector* $\boldsymbol{h}_t$ and predicts an *output vector* $\boldsymbol{y}_t$. The state dynamic can be abstracted as a recurrent relation: $\boldsymbol{h}_t = \text{RNN}\left(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t\right)$. The vanilla RNN is parameterized as follows:

$$\begin{aligned}\boldsymbol{h}_t &= \sigma\left(W_h \boldsymbol{h}_{t-1} + V\boldsymbol{x}_t + \boldsymbol{b}_h\right)\\ \boldsymbol{y}_t &= W_y \boldsymbol{h}_t + \boldsymbol{b}_y\end{aligned}$$

where $(W_h, W_y, V, \boldsymbol{b}_h, \boldsymbol{b}_y)$ are learnable parameters, and σ is a point-wise non-linear function.

Although theoretically powerful, vanilla RNNs cannot learn from long-sequences due to a problem known as vanishing or exploding gradients. A powerful solution is Long Short-Term Memory (LSTM) [16]. LSTM introduces one more vector called "memory" $\boldsymbol{c}_t$, which, together with the state $\boldsymbol{h}_t$, specify the dynamic as: $(\boldsymbol{h}_t, \boldsymbol{c}_t) = \mathrm{LSTM}\left(\boldsymbol{h}_{t-1}, \boldsymbol{c}_{t-1}, \boldsymbol{x}_t\right)$. In most implementations, this is decomposed further as:

$$\begin{aligned}\boldsymbol{c}_t &= \boldsymbol{f}_t * \boldsymbol{c}_{t-1} + i_t * \tilde{\boldsymbol{c}}_t\\ \boldsymbol{h}_t &= \boldsymbol{o}_t * \tanh\left(\boldsymbol{c}_t\right)\end{aligned}$$

where $\tilde{\boldsymbol{c}}_t$ is a candidate memory computed from the input, $\boldsymbol{f}_t, \boldsymbol{i}_t, \boldsymbol{o}_t \in (\mathbf{0}, \mathbf{1})$ are gates, and $*$ denotes point-wise multiplication. $\boldsymbol{f}_t$ determines how much the previous memory is maintained; $\boldsymbol{i}_t$ controls how much new information is stored into memory, and $\boldsymbol{o}_t$ controls how much memory is read out. The candidate memory and the gates are typically parameterized as:

$$\begin{aligned}\tilde{\boldsymbol{c}}_t &= \tanh\left(W_c \boldsymbol{h}_{t-1} + V_c \boldsymbol{x}_t + \boldsymbol{b}_c\right)\\ \begin{bmatrix}\boldsymbol{f}_t\\ \boldsymbol{i}_t\\ \boldsymbol{o}_t\end{bmatrix} &= \mathrm{sigm}\left(\begin{bmatrix}W_f\\ W_i\\ W_o\end{bmatrix} \boldsymbol{h}_{t-1} + \begin{bmatrix}V_f\\ V_i\\ V_o\end{bmatrix} \boldsymbol{x}_t + \begin{bmatrix}\boldsymbol{b}_f\\ \boldsymbol{b}_i\\ \boldsymbol{b}_o\end{bmatrix}\right)\end{aligned}$$

where $(W_{c,f,i,o}, V_{c,f,i,o}, \boldsymbol{b}_{c,f,i,o})$ are learnable parameters.

3 Approach

While LSTMs can theoretically deal with long event sequences, the long-term dependencies between distant events in a process get diffused into the memory vector. LSTM partly solves the gradient issue associated with the vanilla RNN but it may not be very effective on complex process executions that contain multiple computational steps and long-range dependencies. Keeping this in mind, we explore the application of an expressive sequential process model, that would allow storing and retrieval of intermediate process states in a long-term memory. This is akin to the capability of a trainable Turing machine. Closest to a Turing machine is an instantiation of MANN, known as Differential Neural Computer (DCN) [13]. MANNs can be considered as a recurrent net augmented with an external memory module [13,30]. Because of this memory module MANNs have certain advantages over traditional LSTMs when tackling highly complex sequence modeling problems such as question answering [30] and algorithmic tasks [13]. The memory $\boldsymbol{c}_t$ compresses the entire history into a single vector, and thus the process structure is somewhat lost. For example, if two distant events are highly dependent, there are no easy ways to enforce this relationship through the forgetting gates. Another critical issue is that if a process involves multiple intermediate results for latter use, there are no mechanism to store these results into the flat memory vector $\boldsymbol{c}_t$. These drawbacks demand an external memory to store temporary computational results, akin to the role of RAM in modern computers. The key idea behind these architectures is that all memory operations, including addressing, reading and writing are differentiable. This enables end-to-end gradient-based training. MANNs have found many applications, e.g., question answering [18,30] and simple algorithmic tasks [12]. Overall, Encoder-decoder architectures like *memory-augmented neural nets* are geared to solve sequence to-sequence problems and are naturally a good fit for tackling the problem of *optimal path recommendation.*

We adapted the most advanced variant of MANNs to date, the Differential Neural Computer (DNC) [13]. In most popular implementations, DNC can be considered as a LSTM augmented with an external memory module M. The LSTM plays the role of a controller, which is akin to a CPU, where the memory $\boldsymbol{c}_t$ is akin to registers in the CPU. At each time step, the controller (i) reads an input, (ii) updates its own internal memory and states, (iii) writes the new information data into the external memory, and (iv) finally reads the updated memory to produce an output. In a typically implementation, the external memory is a matrix of N slots, each slot is a vector. To interface with the external memory, the controller computes keys $\boldsymbol{k}_t$ for locating slots for reading and writing. The memory slot is found using cosine similarity between the key and the slot content. This mechanism of locating memory slot is known as *content-based addressing.* In addition, DNC also supports *dynamic memory allocation* and *temporal memory linkage* mechanisms for computing one final write-weight and several read-weights. The read-weights are then used to produce a read content from the memory. Multiple reads are then combined with the controller state to produce an output vector $\boldsymbol{o}_t$. For readability, we omit the mathematical details here. Readers are referred to the original paper [13].

We now describe how the DNC can be adapted for prescriptive process analytic tasks, starting from event coding into the model and decoding from it, to specific modifications of the DNC to make it suitable for solving a variety of prescriptive tasks in business processes.

3.1 Events/Resources Coding and Decoding

Discrete events/resources in event log can be coded into MANN in several ways. If the number of unique events/resources is large, embedding into a low-dimensional space is typically employed, that is $a \rightarrow \boldsymbol{x}_a$. Otherwise, a simple one-hot coding will suffice, that is, $a \rightarrow [0, 0, ...1, ...0]$. Continuous resources such as time can be normalized as input variables. Alternatively, these continuous variables can be discretized into symbols that represent intervals. This could enable true *end-to-end* learning. However, we can also employ a certain degree of feature engineering to enhance the input signals as in [31], which has been shown to be highly effective.

For discrete symbol prediction at time t, we can use a softmax:

$$P_t\left(a \mid \text{history}\right) = \frac{\exp\left(\boldsymbol{w}_a \cdot \boldsymbol{o}_t\right)}{\sum_{a'} \exp\left(\boldsymbol{w}_{a'} \cdot \boldsymbol{o}_t\right)} \qquad (1)$$

where $\boldsymbol{o}_t$ is the output vector generated by the controller, and $\boldsymbol{w}_a$ is a trainable parameter vector. The discrete output is simply: $a^* = \arg\max_a P_t\left(a \mid \text{history}\right)$. Continuous prediction is through a function $y_t = f\left(\boldsymbol{o}_t\right)$, which can be itself a feedforward neural net.

Application of these decoding settings in Eq. (1) allows us to solve a variety of predictive and prescriptive tasks like next-activity recommendation, suffix recommendation and so on. Likewise time-to-event estimation is simply continuous prediction.

3.2 Sequence Prediction with Dual-Controllers and Write-Protected Policy

We assume that at the decision point, we are given a partially executed process instance, and we want to prescribe actions for the continuation of a process instance based and optimize for KPIs based on remaining time, or the set of resources needed for completing the instance. Under the MANN formulation, many of those prescriptive tasks can be cast into *sequence prediction*, that is, we generate a sequence of discrete symbols. For example, process continuation is a natural case, where each symbol is an event.

In case of resources prediction, even though there may or may not natural ordering among resources, we can still produce a sequence. Due to the availability of the external memory which stores all the previous knowledge, the strict ordering in the output sequence is not of a major issue, because at any point in the prediction time, the controller can just make use of the external memory (which can be order-free since it is read-only), and relies less on its own internal memory (which is order-dependent). Note that this property is not possible in LSTM, which is sequential by design.

In the DNC setting, this task can be decomposed into dual phases: the *encoding phase*, in which the prefix is read into the memory, and the *decoding phase*, in which the suffix is sequentially generated. Second, in standard DNC operations, the memory is constantly modified at each time step. In the dual-phase setting, there is no need to update the memory since there are no real inputs. Thus we suggest a simple modification, that is, the memory is read-only during the decoding phase. And finally, since the two phases serve different purposes, it might be useful to separate the *encoding controller* from the *decoding controller*. That is, the encoding controller is specialized in keeping the best description of the process thus far, and the decoding controller is optimized to producing the best suffix, given the information pre-computed by the encoding controller. We call this DNC variant *DCw-MANN*, which stands for Write–Protected Dual Controller Memory–Augmented Neural Network. The proposed system learns a highly compact low-dimensional process representation and captures all variations implicit in the given process execution log to enable *near real-time decision support* for tackling multiple prescriptive monitoring tasks.

Model Operations Over Time: The operations of the modified DNC is illustrated in Fig. 1. There are two controllers, the encoder LSTM_{enc} for the encoding phase and the decoder LSTM_{dec} for the decoding phase. Both share the same external memory M. Each controller maintains their own internal memory $\boldsymbol{c}$ and state $\boldsymbol{h}$. In the encoding phase, the prefix is fed into the encoder one event at a time. The external memory is updated a long the way. In decoding phase, the state of the encoder and the memory are passed into the decoder. The long-range dependencies between the input prefix and the output suffix are maintained through the memory look-up operations.

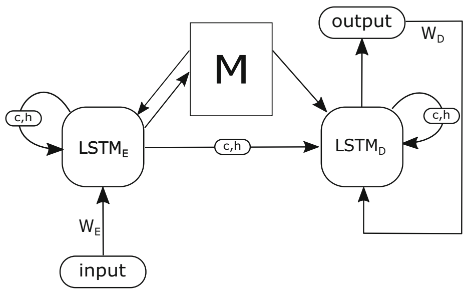

Fig. 1. Write-protected dual controller memory augmented neural network

During the sequence decoding phase, the next symbol at time t is predicted using the information from the memory and previously generated symbols:

$$P_t\left(a \mid a_{t_{pre}+1}, ..., a_{t-1}, \text{history}\right)$$

as in Eq. (1), where the output $\boldsymbol{o}_t$ is generated by the decoder LSTM_{dec}.

3.3 Generating Suffix Recommendations for Decision-Support

Next, our we goal is to learn the task of generating optimal suffix recommendations from partially executed process instances. Process-Aware Recommender Systems, support process users in operational decision-making by continuously monitoring process executions and providing automated recommendations which maximize the likelihood of achieving desired process outcomes. Machine learning based approaches are commonly used to construct data-driven recommender systems where the system attempts to predict the user's interests and recommends items based on those interests. Many of the standard industry recommender systems build a machine learning model by leveraging the user's past behaviour (which is routinely logged) as well as similar actions taken by other users. This model is then used to predict items (or ratings for items) that the users may have an interest in. In Process Analytics, the operative notion of *recommendation*, can be realised by using a machine learning based system, capable of learning from successful (or well-performing) process instances. Weber et al. [28] have explained outcome oriented recommendations based on predictions, as follows: "Recommendations can be considered as predictions about a case, conditioned on the next step that has not been performed yet. In order to recommend to a user what should be the next step in a process, the recommendation service needs to know what the user's target (goal) is, e.g. should the user perform its tasks as soon as possible, or should s/he optimize its outcome in terms of business value".

We have designed our Process-Aware Recommender System to consider the process outcomes by *i)* implementing task conditioning at an architectural level (i.e. using task specific encoders/decoders) *ii)* leveraging past execution data labelled with outcomes (based on performance indicators or non-functional attributes defined for the task). Such labelled data, contains rich knowledge capturing cumulative best practices from the perspective of multiple process users. Our system underpins operational decision support in a manner where good performing instances are leveraged to train a model that can correlate actions with the likelihood of their effectiveness. Here, labels help in differentiating between process instances that performed well based on a pre-defined performance criterion (e.g. through-put time) versus those that performed poorly.

The training examples allow our model to learn the relevant representations from raw data. We trained our proposed machinery in a manner similar to the task of training unsupervised language models, where sequence prediction models are trained with a simple objective: predict the next word, given all of the previous words within some text [25]. Following this approach allows us to learn the prescriptive tasks without the need for explicit supervision. Furthermore, this approach allows us to build a general-purpose model that assumes no domain-specific knowledge of the process, other than the symbolic representation of events (or resources). We finally note that, once the model is trained, it doesn't simply match or repeat the same sequence(recommendations) from training logs rather testing it on an unseen test set(as done in our experiments) shows that our model has learned the task of *recommending an optimal suffix* given a partially executed instance. Good performance on test set also shows that capability of our model to generalise such that it can perform well on a wide variety of future unseen process instances(that were missing from the training data).

4 Evaluation

In the following sections, we explain the experimental setup, we then describe the datasets and pre-processing strategy used for evaluating our proposed approach (Sect. 4.2 and 4.3). We motivate the choice of metrics and describe the baseline methods. We finally present an explanation of model implementation (Sect. 4.4) along with experimental results.

4.1 Datasets

We consider three datasets to evaluate our suffix recommendation engine, whose description is as follows:

- **Moodle Dataset:** This dataset has been created from Moodle's(e-learning platform) issue tracking system. The issue tracking system collects bug reports and allows developers to track the bug resolution process as an issue goes through various development stages. The log contains 10,219 complete processes in total with the number of events in each process ranging from 4 to 23. The preprocessing procedure results in about 32K training prefix/suffix sequences and 8K prefix/suffix sequences. The number of event codes in Moodle dataset is 23.

- **Financial Log:** This log is based on BPI2012 challenge dataset but was pre-processed(see description below) based on a time-based performance metric. After pre-processing we are only left with good performing instances which can be fed to the dataset. The Raw dataset containes about 13,087 cases. The training and testing numbers are approximately 4.2K and 1K, respectively. This dataset has 32 unique type of event codes.
- **IT Incident Management Dataset:** This is an anonymized data set extracted from incident management system supporting an enterprise resource planning (ERP) application. It contains 16,000 tickets(process instances) of IT incident management processes. The log contains the life cycle of a ticket. The ticket is opened by a customer. It is acknowledged typically by a team lead, then it gets assigned to a person working on it and after some analysis and other changes, it gets closed. The group that solved the ticket might not correctly resolve the issue. The log contains the name of the last group that solved the ticket. After splitting, the Incident Mgmt. dataset has about 26K training and 6.5K prefix/suffix sequences. This dataset has 32 unique type of event codes.

4.2 Pre-processing

We take each of these datasets and we split the logs into desirable and undesirable instances (by using the performance of each instance against the stated KPIs) and following the language modeling approach, only train our models using desirable instances. In the Moodle dataset we filter the dataset, by applying a couple of pre-conditions such that each instance should have at least four distinct states[1] and no more than 25 state changes. An undesirable instance examples are chosen with the assumption that bad process instances would shift states back and forth a lot (e.g., issue being reopened multiple times is an Undesirable instance). Hence if more than 25 state changes occur for a given issueID then it would be labelled as an Undesirable instance. Similarly for BPI2012 financial log data we filter cases based on running time. Cases that started in 2012 were filtered out(about 49% because they are not likely to finish. Next, we perform performance filtering using total time duration for each case. Cases with a maximum duration of 1 day 19 h are considered desirable instances while rest of them are labelled as Undesirable performing instances. Each process is a sequence of events and each event is represented by a discrete symbol, which is coded using the one-hot coding scheme introduced in Sect. 3.1. We randomly divide all processes into 80% for training and 20% for testing. Then, we continue splitting each process in the training and test sets into prefix sequence and suffix sequence such that the minimum prefix length is 4.

4.3 Experimental Setup and Modeling

For all experiments, deep learning models are implemented in Tensorflow 1.3.0. Optimizer is Adam [17] with learning rate of 0.001 and other default parameters.

[1] https://docs.moodle.org/dev/Process.

Table 2 describes the hyper-parameter settings, as selected through trial and error. To the best of our knowledge, there is no existing ML based technique for suffix recommendation that considers the problem of outcome based optimal path generation. Therefore, for comparison, we implement custom process-agnostic baselines. For the datasets (Moodle, Financial Log and Incident Mgmt.), the baselines are k-NN, and GRU. The k-NN presents a simple but powerful baseline for the case of vector inputs. Thus it is of interest to see if it works well for sequence inputs as in the case of process analytics. The LSTM, on the hand, has been the state-of-the-art for this domain, as shown in recent work [9,31]. The GRU is a recent alternative to LSTM, which has been shown to be equally effective in NLP tasks [4]. The k-NN works by retrieving k most similar prefixes in the training data. Then suffix and other desirable outcomes are computed from the same outcomes of those retrieved cases. The recommendation is either the average of the retrieved outcomes (if continuous), or the most common outcome (if discrete).

Model Evaluation: Numerical evaluations with comparisons to baselines play a central role when judging research for most recommender systems, therefore we rely on baseline comparisons/benchmarks to evaluate the quality of recommendations produced by our machinery. Our recommender system provides operational decision support for process users and on a higher level, performs utility optimization. Gunawardana et al. [15] provide a survey of evaluation metrics for recommender systems. They observe that the task of optimizing utilities is by far the least explored recommendation task. Hence research/industry standard evaluation metrics do not exist for such task. Prescriptive machine learning models for such tasks are predominantly benchmarked by matching samples against a reference solution (e.g. previously well-performing instances representing ground truth). In our case, we have picked Levenshtein distance metric because it aptly summarizes accuracy/precision in terms of the closeness of recommended sequences to the reference(desired) sequences in high-dimensional vector space. It represents a degree of conformity of evaluated predictions to the true value and is sensitive to differences in error rates, making it effective for judging the effectiveness of our process-aware recommendation machinery. However, these distances have a quadratic time complexity of the sequence length, which can be expensive for long sequences. Hence we build a Trie over the training prefixes for fast retrieval. In our experiments, we choose k to be 1 and 5. We append to the end of each complete process a special token <END> signaling its termination. We train the GRU in the same manner as training a language model [23], which is identical to next activity prediction. After training, a test prefix will be fed to the GRU as prior context and the model will continue recommending the next event step-by-step until the <END> symbol is outputted. In our experiment, we use a hidden vector of size 100 for both GRU and MANN methods.

4.4 Results and Discussion

For evaluation, we use the edit distance (Levenshtein distance) as it is a good indication of sequence similarity where deletion, insertion or substitution are available as in the case of business processes. To account for variable sequence lengths, we normalize this distance over the length of the longer sequence (between 2 sequences). Then, the final metric is calculated as the normalized edit similarity that equals 1− normalized edit distance. Consequently, the predicted sequence is good if its normalized edit similarity to the target sequence is high.

We observe in Table 1 that our MANN based mode outperforms the state of the art LSTM model across all three datasets. The k-NN works surprisingly well. However, it faces some difficulties in this problem of sequence-to-sequence prediction. First, the prefixes can be slightly different but the suffices can differ drastically, e.g., due to a single decisive event. Second, the k-NN does not capture the continuation of a process, and thus suffices from similar instances do not guarantee to be the right continuation. And third, for $k > 1$, there is no easy way to combine multiple suffix sequences, which shows in the worse result than the case $k = 1$.

Table 1. Suffix recommendation task: the average normalized edit similarity between the target suffixes and the suffixes recommended by different models (higher is better).

Method	Moodle	fin_log	Inc_Mgmt
5-NN	0.817	0.588	0.418
1-NN	0.840	0.631	0.432
GRU	0.875	0.559	0.454
LSTM	0.887	0.683	0.497
MANN	**0.888**	**0.691**	**0.502**

Table 2. MANN hyper-parameters. (*) no duplicate.

Hyper-parameters	Moodle	Financial log	Incident Mgmt.
# memory slots	64	64	64
Memory slot size	100	64	100
Controller hidden dim	100	100	100

Since the authors in [31] shared, only public the code for the two-layer LSTMs (one is shared-weight), we can only calculate the parameter size for this configuration, which is about 208K trainable parameters. It should be noted that the best model configurations consisting of 3 or 4 layers may have even more than that number of parameters. Our MANN, by contrast, is much simpler with two one-layer controllers and an external memory hence has fewer parameters (less than 125K). This suggests the ability of the external memory to compress and capture essential information in order to perform better.

Taken together, the results achieved using MANNs demonstrate that our proposed machinery is well-suited for solving both predictive and prescriptive monitoring tasks, with far fewer parameters. We also note that MANNs are relatively new, and we expect that even better performance could be achieved with greater effort in devising encodings for process analytics problems. As well, we have been able to position a range of process analytics problems to leverage future developments/improvements in MANNs. Our approach based on employing labelled datasets should hopefully lead the community to ask a broader range of prescriptive process analytics questions that could be solved using similar machinery as discussed in this paper.

5 Related Work

Predictive business process monitoring is a family of techniques concerned with predicting the future state, outcomes and behaviour of ongoing cases of a business process [32]. Relevant to our work, the task of *next activity prediction* and *Process Path Prediction* has been tackled by approaches like state-transition models, hidden Markov models(HMM) and Probabilistic Finite Automatons(PFA) models [2,3,19]. Tax et al. [31] point out that such approaches are *'tailor-made for specific prediction tasks and not readily generalizable'*. Recently, Deep Learning methods such as LSTMs have shown an advantage over such classical methods for making accurate predictions and solving various predictive monitoring tasks [31]. Several survey papers have reviewed the literature on predictive process monitoring. e.g. Marquez-Chamorro et al. [22] and Di Francescomarino et al. [7] classify the literature based on input data, classification algorithm and prediction target. Similarly, Teinemaa et al. [32] and Verenich et al. [34] also survey the literature by covering various datasets, propose task definitions and provide benchmark comparison of recently proposed algorithms. The output of Predictive business process monitoring techniques, is just *predictions*. Predictions can be used as *early warnings* for taking risk informed decisions but do not explicitly support answering of question like *What action should we take next to achieve a particular goal?* and *Why should we do it?* [21]. Compared to descriptive and predictive business analytics, prescriptive process analytics remains less mature [8]. Marquez et al. [22] point out that *'little attention has been given to providing recommendations'*. Instead of providing specific action recommendations, literature on business process monitoring focuses on forecasting future process events(and outcomes) while leaving the action implementation part to the subjective judgment of process users and business decision makers [6]. Overall, prescriptive business process monitoring techniques [5,14,28] have largely focused on recommending preventive actions in order to support *risk-informed decision making*.

Eili et al. [8] provide a systematic review of Recommender Systems in Process Mining and classify recommendation approaches as *'pattern optimization'*, *'risk minimization'*, or *'metric-based'*. They highlight the fact that compared to descriptive and predictive business analytics, prescriptive process analytics remains less mature [8]. Existing prescriptive business process monitoring techniques [5,14,28] are used to recommend *preventive actions* in order to support *risk-informed decision making*. To the best of our knowledge, the problem of recommending best path (representing sequence of activities leading to the process end), based on pre-defined KPIs hasn't been addressed. Closely related to our work, Weinzier et al. [35] consider problem of recommending *next best actions* that lead to optimal outcomes. Their work however differs from ours, as their technique relies on explicitly adding control-flow knowledge to their proposed technique via formal process model and uses process simulations to verify and filter the predictions of the trained predictive model. Similarly, Groger et al. [14] introduce the concept of recommendation-based business process optimization for data-driven process optimization. Their data-mining driven solution supports adaptive processes and recommends actions for next process step to take for a given process instance in order to avoid performance deviation. Lastly, Schobel et al. [28] propose a technique for early identification of diverging processes that can support operational decision-making processes by for example taking remedial actions as business processes unfold. Overall, prescriptive business process monitoring techniques [5,14,28] are used to recommend preventive actions in order to support *risk-informed decision making*. However, compared to above mentioned work which focuses on *early warning recommendations* (e.g. predicted metric deviation), our work focuses on *best action recommendations* that maximize the likelihood of achieving desirable outcomes.

6 Conclusion

In this paper, we explored the application of recent advances in deep learning for building a *Process-Aware Recommender System*. We investigated a specific type of neural network known as the *memory–augmented neural network (MANN)* for its applications in prescriptive process monitoring tasks. We adapted a recently developed MANN architecture, namely the Differential Neural Computer [13] and proposed several modifications to the default architecture. We performed evaluations using three labelled datasets to show that our proposed approach performs well on the task of suffix recommendation while taking cognisance of the relevant KPIs. Our future work will involve investigating the behaviour of MANNs on highly complex processes that involve multiple intermediate steps and results, and devising ways to visualise how distant events are remembered and linked together.

References

1. Beheshti, A., Yakhchi, S., Mousaeirad, S., Ghafari, S.M., Goluguri, S.R., Edrisi, M.A.: Towards cognitive recommender systems. Algorithms **13**(8), 176 (2020)
2. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. MIS Q. **40**(4), 1009–1034 (2016)
3. Ceci, M., Lanotte, P.F., Fumarola, F., Cavallo, D.P., Malerba, D.: Completion time and next activity prediction of processes using sequential pattern mining. In: Džeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) DS 2014. LNCS (LNAI), vol. 8777, pp. 49–61. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11812-3_5

4. Cho, K.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, pp. 1724–1734. Association for Computational Linguistics, October 2014
5. Conforti, R., de Leoni, M., La Rosa, M., van der Aalst, W.M.P.: Supporting risk-informed decisions during business process execution. In: Salinesi, C., Norrie, M.C., Pastor, Ó. (eds.) CAiSE 2013. LNCS, vol. 7908, pp. 116–132. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38709-8_8
6. Dees, M., de Leoni, M., van der Aalst, W.M.P., Reijers, H.A.: What if process predictions are not followed by good recommendations? (technical report). arXiv preprint arXiv:1905.10173 (2019)
7. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Milani, F.: Predictive process monitoring methods: which one suits me best? In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 462–479. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_27
8. Eili, M.Y., Rezaeenour, J., Sani, M.F.: A systematic literature review on process-aware recommender systems. arXiv preprint arXiv:2103.16654 (2021)
9. Evermann, J., Rehse, J.-R., Fettke, P.: A deep learning approach for predicting process behaviour at runtime. In: Dumas, M., Fantinato, M. (eds.) BPM 2016. LNBIP, vol. 281, pp. 327–338. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58457-7_24
10. Evermann, J., Rehse, J.-R., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
11. Ghattas, J., Soffer, P., Peleg, M.: Improving business process decision making based on past experience. Decis. Support Syst. **59**, 93–107 (2014)
12. Graves, A., Wayne, G., Danihelka, I.: Neural turing machines. arXiv preprint arXiv:1410.5401 (2014)
13. Graves, A., et al.: Hybrid computing using a neural network with dynamic external memory. Nature **538**(7626), 471–476 (2016)
14. Gröger, C., Schwarz, H., Mitschang, B.: Prescriptive analytics for recommendation-based business process optimization. In: Abramowicz, W., Kokkinaki, A. (eds.) BIS 2014. LNBIP, vol. 176, pp. 25–37. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-06695-0_3
15. Gunawardana, A., Shani, G.: A survey of accuracy evaluation metrics of recommendation tasks. J. Mach. Learn. Res. **10**(12) (2009)
16. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
17. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
18. Kumar, A., et al.: Ask me anything: dynamic memory networks for natural language processing. In: International Conference on Machine Learning, pp. 1378–1387. PMLR (2016)
19. Lakshmanan, G.T., Shamsi, D., Doganata, Y.N., Unuvar, M., Khalaf, R.: A Markov prediction model for data-driven semi-structured business processes. Knowl. Inf. Syst. **42**(1), 97–126 (2013). https://doi.org/10.1007/s10115-013-0697-8
20. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015)
21. Lepenioti, K., Bousdekis, A., Apostolou, D., Mentzas, G.: Prescriptive analytics: literature review and research challenges. Int. J. Inf. Manage. **50**, 57–70 (2020)
22. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: a survey. IEEE Trans. Serv. Comput. **11**(6), 962–977 (2017)

23. Mikolov, T., et al.: Statistical language models based on neural networks. Presentation at Google, Mountain View, 2 April 2012
24. Navarin, N., Vincenzi, B., Polato, M., Sperduti, A.: LSTM networks for data-aware remaining time prediction of business process instances. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–7. IEEE (2017)
25. Radford, A., Jeffrey, W., Child, R., Luan, D., Amodei, D., Sutskever, I., et al.: Language models are unsupervised multitask learners. OpenAI Blog **1**(8), 9 (2019)
26. Santiputri, M., Ghose, A., Dam, H.K., Automating the acquisition of process semantics: Mining task post-conditions. Data Knowl. Eng. **109**, 112–125 (2017)
27. Schmidhuber, J.: Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
28. Schonenberg, H., Weber, B., van Dongen, B., van der Aalst, W.: Supporting flexible processes through recommendations based on history. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 51–66. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85758-7_7
29. Sindhgatta, R., Ghose, A., Dam, H.K.: Context-aware analysis of past process executions to aid resource allocation decisions. In: Nurcan, S., Soffer, P., Bajec, M., Eder, J. (eds.) CAiSE 2016. LNCS, vol. 9694, pp. 575–589. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39696-5_35
30. Sukhbaatar, S., Szlam, A., Weston, J., Fergus, R.: End-to-end memory networks. arXiv preprint arXiv:1503.08895 (2015)
31. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_30
32. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: Review and benchmark. ACM Trans. Knowl. Discovery Data (TKDD) **13**(2), 1–57 (2019)
33. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28108-2_19
34. Verenich, I., Dumas, M., Rosa, M.L., Maggi, F.M., Teinemaa, I.:. Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. ACM Trans. Intell. Syst. Technol. (TIST) **10**(4), 1–34 (2019)
35. Weinzierl, S., Dunzer, S., Zilker, S., Matzner, M.: Prescriptive business process monitoring for recommending next best actions. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 193–209. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58638-6_12