Enhancing Deep Sequence Generation with Logical Temporal Knowledge

Elena Umili[1,*], Gabriel Paludo Licks[1] and Fabio Patrizi[1]

[1]*Sapienza University of Rome*

Abstract

Despite significant advancements in deep learning for sequence forecasting, neural models are typically trained only on data, and the incorporation of high-level prior logical knowledge in their training is still an hard challenge. This limitation hinders the exploitation of background knowledge, such as common sense or domain-specific information, in predictive tasks performed by neural networks. In this work, we propose a principled approach to integrate prior knowledge in Linear Temporal Logic over finite traces (LTL_f) into deep autoregressive models for multistep symbolic sequence generation (i.e., suffix prediction) at training time. Our method involves representing logical knowledge through continuous probabilistic relaxations and employing a differentiable schedule for sampling the next symbol from the network. We test our approach on synthetic datasets based on background knowledge in Declare, inspired by Business Process Management (BPM) applications. The results demonstrate that our method consistently improves the performance of the neural predictor, achieving lower Damerau-Levenshtein (DL) distances from target sequences and higher satisfaction rates of the logical knowledge compared to models trained solely on data.

Keywords

Suffix prediction, Neurosymbolic AI, Deep learning with logical knowledge, Linear Temporal Logic

1. Introduction

This paper addresses the problem of suffix prediction by exploiting both data and prior logic temporal knowledge. The task of suffix prediction is particularly important in Business Process Management (BPM) for forecasting the future of a process trace, enabling resource allocation, and the anticipation of future steps in a process based on historical data.

Recently, there has been significant interest in employing deep learning techniques for suffix prediction in BPM [1], involving the use of Recurrent Neural Networks (RNNs) [2], Transformers [3], and Deep Reinforcement Learning algorithms [4]. Despite the significant advancements in machine learning and deep learning, some studies show how deep models can surprisingly fail to satisfy even the most basic logical constraints [5] [6] derived from commonsense or additional domain-knowledge. This occurs because these models are trained solely on data, with the integration of logical knowledge into the training process remaining an open challenge. Due to this, while in many application fields, such as BPM, both data and logical knowledge about the process are often available, the latter remain typically unused. This work explores a

PMAI@ECAI24: International ECAI Workshop on Process Management in the AI era, October 19, 2024, Santiago De Compostela, Spain

*Corresponding author.

umili@diag.uniroma1.it (E. Umili); licks@diag.uniroma1.it (G. P. Licks); patrizi@diag.uniroma1.it (F. Patrizi)

novel approach, aimed at bridging this gap, by taking advantage of prior knowledge expressed in Linear Temporal Logic over finite traces (LTL_f), when training a deep generative model for suffix prediction.

Very few methods incorporate logical knowledge into deep sequence generation [6, 7]. Specifically, STLnet [6] employs specifications in Signal Temporal Logic on continuous temporal sequences for applications other than BPM, however, the attempt to apply it to our domains produced poor results. On the other hand, Di Francescomarino et al. [7] apply their method to suffix prediction in BPM but use background knowledge only at *test time*, employing a variant of beam search that also takes into account the temporal formula modeling some process properties. Thus, the proposed method does not fully integrate background knowledge into the training process but, rather, a smart way to query the pre-trained network, considering the LTL_f specification. Interestingly, though, this approach can be seamlessly combined with ours, to further maximize constraint satisfaction.

Our contribution is a method to integrate the knowledge of certain process properties, expressed in LTL over finite traces (LTL_f), into the training process of a sequence neural predictor. This enables us to leverage both these sources of information –data and background LTL_f knowledge– at training time. Specifically, we achieve this by defining a differentiable counterpart of the LTL_f knowledge and leveraging a differentiable sampling process known as Gumbel-Softmax trick [8]. By utilizing these two elements, we define a logical loss function that can be used in conjunction with any other loss employed by the predictor. This ensures that the network learns to generate traces similar to those seen in the training dataset and to satisfy the given temporal specifications *at the same time*. Through preliminary experiments on synthetic data, we show that incorporating our loss function at training-time results, for RNN models, in predicted suffixes that feature both a lower Damerau-Levenshtein (DL) distance [9] from the target suffixes and a higher rate of satisfaction of the LTL_f constraints.

We observe that, although grounded here on a typical BPM task, our approach can be applied to any scenario involving multi-step symbolic sequence generation through deep learning models, thus making the main results of this work potentially of general interest to the wider audience focusing on Machine Learning and, in particular, neurosymbolic AI.

2. Related Work

Recently, there has been significant interest in employing deep Neural Networks (NN) in predictive monitoring of business processes, for tasks such as next activity prediction, suffix prediction, and attribute prediction [1]. Despite significant advances in the field, nearly all works rely on training these models solely on data without utilizing any formal prior knowledge about the process. They mainly focus on two aspects: (i) enhancing the neural model, ranging from RNNs [10, 11, 7, 12], Convolutional NN (CNN) [13], Generative Adversarial Networks (GANs) [14, 12], Autoencoders [12], and Transformers [12]; and (ii) wisely choosing the sampling technique to query the network at test time to generate the suffix, mostly using greedy search [10], random search [11], or beam search [7], and more recently, policies trained with Reinforcement Learning (RL) [15, 4]. Among all these works, only one exploits prior process knowledge [7], expressed as a set of LTL formulas, but it uses this knowledge only at test time, modifying the beam search

sampling algorithm to select potentially compliant traces with the background knowledge.

In this work, we take a radically different approach by introducing a principled way to integrate background knowledge in LTL_f with a deep NN model for suffix prediction at *training time*. This is based on defining a logical loss that can be combined with the loss of any autoregressive neural model and any sampling technique at test time, that draws inspiration from the literature in Neurosymbolic (NeSy) AI [16]. In this field, many prior works focus on exploiting temporal logical knowledge in deep learning tasks, but none have been used for multi-step symbolic sequence generation. T-leaf [17] creates a semantic embedding space to represent both formulas and traces and uses it in tasks such as sequential action recognition and imitation learning, which do not involve multi-step prediction. [18] modifies Logic Tensor Networks (LTN) [19] to integrate LTL_f background knowledge in image sequence classification tasks rather than generative tasks. STLnet [6] adopts a student-teacher training scheme where the student network proposes a suffix based on the data, that is corrected by the teacher network to satisfy the formula. This work uses Signal Temporal Logic (STL) formulas and is applied to continuous trajectories rather than discrete traces. Our attempts to apply it to discrete data and LTL_f formulas translated into STL yielded poor results, as the resulting STL formulas were extremely challenging for the framework to handle.

Our work is the first to integrate temporal knowledge in the generation of multi-step symbolic sequences. It is based on encoding LTL_f formulas using a matrix representation that we previously used for very different tasks –such as learning RL policies for non-Markovian tasks [20] and inducing automata from a set of labeled traces [21]– that we adapt here for use in the generative task of suffix prediction.

3. Background and Notation

Linear Temporal Logic and Deterministic Finite Automata Linear Temporal Logic (LTL) [22] is a language which extends traditional propositional logic with modal operators. With the latter we can specify rules that must hold *through time*. In this work we use LTL interpreted over finite traces (LTL_f) [23], which model finite, but length-unbounded process executions, and is thus adequate for finite-horizon problems. Given a finite set P of atomic propositions, the set of LTL_f formulas ϕ is inductively defined as follows:

$$\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi \wedge \phi \mid X\phi \mid \phi U \phi, \tag{1}$$

where $p \in P$. We use $\top$ and $\bot$ to denote true and false respectively. X (Strong Next) and U (Until) are temporal operators. Other temporal operators are: N (Weak Next) and R (Release) respectively, defined as $N\phi \equiv \neg X \neg \phi$ and $\phi_1 R \phi_2 \equiv \neg(\neg\phi_1 U \neg\phi_2)$; G (globally) $G\phi \equiv \bot R \phi$ and F (eventually) $F\phi \equiv \top U \phi$. A trace $x = [x^{(1)}, x^{(2)}, \ldots, x^{(l)}]$ is a sequence of propositional assignments to the propositions in P, where $x^{(t)} \subseteq P$ is the set of all and only propositions that are true at instant t. Additionally, $|x| = l$ represents the length of x. Since every trace is finite, $|x| < \infty$ and $x \in (2^P)^*$. If the propositional symbols in P are all *mutually exclusive*, e.g. the domain produces one and only one symbol true at each step, we have that $x^{(t)} \in P$. As customary in BPM, we make this assumption, known as the *Declare assumption* [24]. By $x \vDash \phi$ we denote that the trace x satisfies the LTL_f formula ϕ. We refer the reader to [23] for a formal description of the LTL_f

semantics. Any LTL_f formula ϕ can be translated into a Deterministic Finite Automaton (DFA) [23] $A_\phi = (\Sigma, Q, q_0, \delta, F)$, with Σ the automaton alphabet, Q the finite set of states, $q_0 \in Q$ the initial state, $\delta : Q \times \Sigma \to Q$ the transition function and $F \subseteq Q$ the set of final states, s.t. for a trace $x \in 2^P$, we have that $x \in L(A_\phi)$ iff $x \vDash \phi$, where $L(A_\phi)$ denotes the language (of words) accepted by A_ϕ. Depending on whether the Declare assumption holds or not, Σ can be P or 2^P, respectively. In our case, the former holds.

Deep Autoregressive Models and Suffix Prediction Deep autoregressive models are a class of deep learning models that automatically predict the next component in a sequence by using the previous elements in the sequence as inputs. These models can be applied to both continuous and categorical (symbolic) data, finding applications in various generative AI tasks such as Natural Language Processing (NLP) and Large Language Models (LLM) [25, 26], image synthesis [27, 28], and time-series prediction [29]. They encompass deep architectures such as RNNs and Transformers and, in general, any neural model capable of estimating the probability

$$p(x^{(t)} \mid x^{(1)}, x^{(2)}, \dots, x^{(t-1)}) = p(x^{(t)} \mid x^{<t}) \tag{2}$$

where $x = [x^{(1)}, x^{(2)}, \dots, x^{(l)}]$ is a sequence of data. Its probability can be calculated as

$$p(x) = \prod_{i=1}^{l} p(x^{(i)} \mid x^{(<i)}) \tag{3}$$

In suffix prediction in BPM, given a subsequence (or prefix) of *past events* $p_t = [e^{(1)}, e^{(2)}, \dots, e^{(t)}]$ that the process has produced up to the current time step t, with $e^{(i)}$ in a finite set of activities $\mathscr{A}$, we aim to complete the trace by generating the sequence of future events, called also *suffix* $s_t = [\tilde{e}^{(t+1)}, \tilde{e}^{(t+2)}, \dots]$. Note that we denote *ground-truth* events observed in the domain as e and *predicted* events as $\tilde{e}$, that may in general deviate from actual future events. This task can also be accomplished using autoregressive models, by choosing at each step the most probable next event according to the neural network, concatenating it with the prefix, and continuing to predict the next event in this manner until a special *end-of-trace* symbol is predicted or the trace has reached a maximum number of steps T. We denote the trace obtained in this way as $\tilde{\sigma} = p_t + s_t$, with:

$$\tilde{e}^{(k)} = \underset{a \in \mathscr{A}}{\text{argmax}}\, p(\tilde{e}^{(k)} = a \mid e^{(1)}, \dots, e^{(t)}, \tilde{e}^{(t+1)}, \dots, \tilde{e}^{(k-1)}) \qquad t < k \leq T \tag{4}$$

This method of generating the suffix is a greedy search strategy and generally may not produce the *most probable suffix*, e.g., the trace that maximizes the probability in Equation 3. Other non-optimal sampling strategies commonly used for this task include Beam Search, Random Sampling, and Temperature Sampling [4].

4. Method

Our method assumes an autoregressive neural model to estimate the probability of the next event given a trace of previous events (Equation 2). For explanation purposes, we consider a

simple RNN-based next-activity predictor, where:

$$\begin{aligned} h^{(t)} &= f(e^{(t)}, h^{(t-1)}; \theta_f) \\ y^{(t)} &= g(h^{(t)}; \theta_g) \end{aligned} \tag{5}$$

where $e^{(t)} \in [0,1]^{|\mathscr{A}|}$ is the event of the trace at time t encoded as one-hot vector. $h^{(t)} \in \mathbb{R}^{d_h}$ is the *hidden state* of the RNN, a continuous vector representation of past events in the trace having size d_h. $f : [0,1]^{|\mathscr{A}|} \times \mathbb{R}^{d_h} \to \mathbb{R}^{d_h}$ is a parametric recursive function calculating the current hidden state $h^{(t)}$ given the previous one $h^{(t-1)}$ and the current event $e^{(t)}$ as input. We denote its learnable parameters as θ_f. Finally $g : \mathbb{R}^{d_h} \to [0,1]^{|\mathscr{A}|}$ is the learnable function, with parameters θ_g, mapping the hidden state to the probability vector of the next event $y^{(t)}$. Function f and g can be implemented by an RNN and a multi-layer perceptron (MLP), respectively, as shown in Figure 1. The model parameters are usually trained with a supervised loss $L_{\mathscr{D}}$ evaluated on a dataset of N ground truth traces $\mathscr{D} = \{\sigma_1, \sigma_2, \dots, \sigma_N\}$ recorded observing the process. The loss formulation for a trace $\sigma_k \in \mathscr{D}$ of length l is as follows:

$$L_{\mathscr{D}}(\sigma_k, \theta_f, \theta_g) = \frac{1}{l} \sum_{t=1}^{l} \text{crossentropy}(g(h(\sigma_k^{\leq t})), \sigma_k^{t+1}) \tag{6}$$

This loss teaches the network to produce the next symbol in the trace to mimic as closely as possible the data contained in the dataset. In our method, this loss is combined with a *logic* loss L_ϕ, which imposes the satisfaction of prior knowledge in the following way

$$L = \alpha L_{\mathscr{D}} + (1 - \alpha) L_\phi \tag{7}$$

where α is a constant chosen between 0 and 1 balancing the influence of each loss on the training process.

4.1. Logic Loss Calculation

We assume known certain process properties that need to be enforced at training time. Such properties, which constitute the *background* (or *prior*) knowledge we have about the process, are expressed as an LTL_f formula ϕ and constitute the basis for defining the logic loss L_ϕ.

Essentially, L_ϕ estimates the probability that the suffix generated by the RNN satisfies ϕ and is enforced to be as close as possible to 1. Observe that, while the supervised loss in Equation 6 takes into account only the *next-step* prediction, the satisfaction of ϕ must be checked over the *entire* predicted trace, which is produced by querying the network multiple times. This is because compliance of the trace to ϕ can be evaluated only one the complete trace. Indeed, the fact that the predicted partial trace $\tilde{\sigma}$ satisfies (or violates) ϕ before termination does not preclude the entire predicted trace from violating (or satisfying) ϕ.

Additionally, notice that the next-activity predictor is trained with *perfectly one-hot* (or symbolic) data and produces a (continuous) probability vector as output, which may be very different from a one-hot vector. Therefore, we cannot directly feed the network with the probabilities predicted in previous steps, as this may generate unexpected outputs. Instead, we need to *sample* from these probabilities, in order to produce one-hot vectors again. At

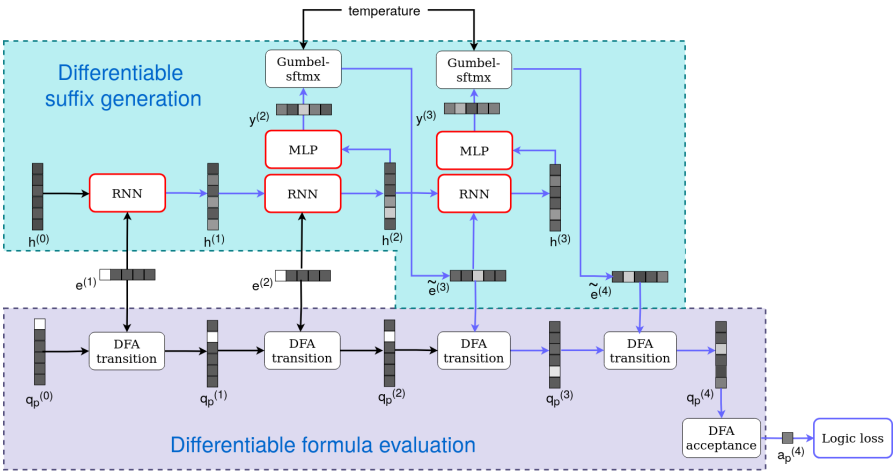


Figure 1: Logic loss computation by using a differentiable procedure for both suffix generation and formula evaluation. The violet arrows represents connections that are interested by backpropagation of the loss, and the elements highlighted with a red border are the modules whose parameters are affected by the logic loss.

the same time, this sampling process needs to guarantee differentiability, so as to ensure backpropagation's applicability.

The calculation of the logic loss is exemplified in the following steps:

1. given a prefix p_t, generate a suffix s_t that is at the same time: *(i)* very probable according to the network; *(ii)* differentiable; and *(iii)* at least *nearly symbolic*, i.e., as close as possible to a one-hot vector;
2. calculate the probability that the predicted trace $\tilde{\sigma} = p_t + s_t$ satisfies ϕ through a differentiable procedure;
3. enforce maximization of the obtained probability.

Given the differentiability of both the knowledge evaluator and the sampling process, the loss can be backpropagated through the suffix and influence the parameters θ_f and θ_g of the next activity predictor.

Differentiable Suffix Generation To sample the next activity $\tilde{e}^{(t+1)}$ from the probability distribution returned by the neural network $y^{(t)}$, while ensuring differentiability of the output sample, we use the Gumbel-Softmax reparameterization trick [8, 30]:

$$\tilde{e}^{(t+1)} = \text{softmax}\left(\frac{\log(y^{(t)}) + G}{\tau}\right) \tag{8}$$

where G is a random sample drawn from the Gumbel distribution, and τ is a temperature parameter used to interpolate between discrete one-hot-encoded categorical distributions and continuous categorical probabilities. When $\tau \to 0$, the output samples tend to be one-hot, while they tend to the continuous probabilities in $y^{(t)}$ when $\tau = 1$.

Differentiable LTL$_f$ Evaluation In order to evaluate the LTL$_f$ specification in a differentiable way, we represent ϕ through a matrix representation that is generally used to encode Probabilistic Finite Automata (PFA) [21, 31, 20]. We first translate ϕ into the equivalent DFA A_ϕ [32] and simplify the automaton with the Declare assumption. We then represent the automaton $A_\phi = (P, Q, q_0, \delta, F)$ as an *input vector* $v_0 \in [0,1]^{1\times|Q|}$, a *transition matrix* $M_\delta \in [0,1]^{|P|\times|Q|\times|Q|}$, and a *finality vector* $v_F \in [0,1]^{|Q|\times 1}$. The input vector v_0 takes the value 1 at index q if $q = q_0$, and 0 otherwise. The matrix M_δ takes the value 1 at index (p, q, q') if $\delta(q, p) = q'$, and 0 otherwise. We denote $M_\delta[p] \in [0,1]^{|Q|\times|Q|}$ as the 2D transition matrix for symbol p. The output vector v_F has a 1 at position q if $q \in F$, and 0 otherwise. Given a string $x = [x^{(1)}x^{(2)} \dots x^{(l)}]$, we denote $q_p = [q_p^{(0)}, q_p^{(1)}, \dots, q_p^{(l)}]$ as the sequence of probabilities to visit a certain state, and $a_p = [a_p^{(0)}, a_p^{(1)}, \dots, a_p^{(l)}]$ as the probabilities that the sequence is accepted. Here, $q_p^{(t)} \in [0,1]^{1\times|Q|}$ is a row vector containing at position q the probability of being in state q at time t, while $a_p^{(t)} \in [0,1]$ is a scalar representing the probability that the sequence is accepted at time t.

$$\begin{aligned} q_p^{(0)} &= v_0 \\ q_p^{(t)} &= q_p^{(t-1)} \times M_\delta[x^{(t)}] \quad \forall t > 0 \\ a_p^{(t)} &= q_p^{(t)} \times v_F \end{aligned} \tag{9}$$

Here, $\times$ denotes the inner product. Therefore, the probability of a string being accepted is the probability of being in a final state in the last computed state $a_p^{(l)}$. Note that $x^{(t)} \in P$ is an *integer*, while the predicted symbol sampled by the network $\tilde{e}^{(t)}$ (Equation 8) is a *probability vector*, which *tends* to approximate a one-hot vector but is not guaranteed to be composed only of zeros and ones and may contain some noise. For this reason, we adapt Equation 9 to our case by adding the expectation computation over the next automaton state.

$$\begin{aligned} q_p^{(t)} &= \sum_{p=1}^{|P|} (q_p^{(t-1)} \times M_\delta[p])\tilde{e}^{(t)}[p] \quad \forall t > 0 \\ a_p^{(t)} &= q_p^{(t)} \times v_F \end{aligned} \tag{10}$$

Enforcing Compliance with the Formula In summary, given a prefix $p_t = [e^{(1)}, e^{(2)}, \dots, e^{(t)}]$, we calculate the suffix $s_t = [\tilde{e}^{(t+1)}, \dots, \tilde{e}^{(T)}]$ up to a time-step $T > t$ by iterating through the following steps at each time step: (i) producing the RNN probability for the next symbol (Eq. 5), (ii) differentiably sampling the next activity from this probability (Eq. 8), and (iii) concatenating the next activity to the current trace. The trace generated in this way, $\tilde{\sigma} = p + s$, is processed by the differentiable automaton (Eq. 10), which produces an acceptance probability at the last step, $a_p^{(T)}$. We enforce this probability to stay close to 1 by minimizing the loss

$$L_\phi = -\log(a_p^{(T)}). \tag{11}$$

The entire process is illustrated in Figure 1. The figure shows how the loss is backpropagated through both the suffix prediction and knowledge evaluation processes, affecting the parameters of the learnable functions f and g of the next activity predictor.

5. Experiments

In this section, we describe the setup for our method's empirical evaluation and discuss the results achieved. The experiments are reproducible using our implementation at https://github.com/whitemech/suffix-prediction-pmai2024.

5.1. Experimental Setup

We aim to evaluate our method on a diverse set of formulas with varying levels of difficulty. To achieve this, we construct random formulas by combining Declare constraints [33] as follows: *(i)* we set the maximum number of conjuncts C and the maximum number of disjuncts D; *(ii)* we generate a random formula ϕ as the disjunction of up to D disjuncts, each of them is the conjunction of up to C randomly selected Declare constraints, possibly negated, with each constraint having random symbols from $\mathscr{A}$; *(iii)* we evaluate ϕ on a set of random strings and select the formula for the experiment only if the satisfaction rate is between 10% and 90%, in order to eliminate formulas that are too obvious or too difficult.

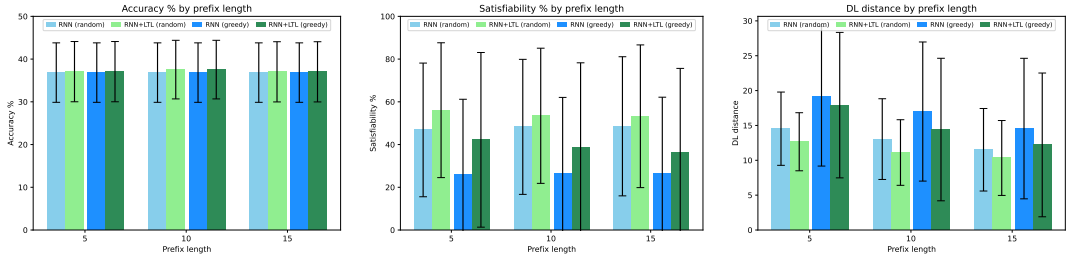
For each random formula, we construct a dataset $\mathscr{D}$ composed of N traces of length l that are compliant with the formula. We then split the dataset into 90% for training and 10% for testing. We generate 25 dataset-formula pairs by setting $D = C = 5$, $\mathscr{A} = \{a, b, c\}$, $N = 1000$, and $l = 20$. These pairs are used to evaluate our framework Additionally, each experiment's training and testing are performed on different prefix lengths $|p| \in 5, 10, 15$ to evaluate the model's ability to predict shorter and longer suffixes. During testing, the suffix prediction stops when the model outputs the *end-of-trace* symbol or when the predicted suffix reaches a maximum of $T = 2 \times l = 40$ timesteps. Each experiment is repeated 5 times with different seeds to ensure robustness and reliability of the results.

5.2. Empirical Results

Our empirical evaluation compares the performance of four models:

- **RNN (random)**: An RNN trained on the training data, queried with random sampling. The next activity is selected by randomly sampling according to the probability distribution predicted by the RNN output layer.
- **RNN+LTL (random)**: An RNN trained with the combined loss in Eq 7, where L_ϕ is calculated using the random formula associated with the dataset, and queried with random sampling.
- **RNN (greedy)**: An RNN trained only with data, with suffixes generated using greedy sampling (Eq. 4), always taking the most probable next event predicted by the output layer. This model is equivalent to the one proposed in [2].
- **RNN+LTL (greedy)**: Similar to the previous model but with the addition of LTL background knowledge during training.

We instantiate all models as LSTMs with two layers and 100 neurons in each layer (as in [10]), a batch size of 64, and an Adam optimizer with a learning rate of 5×10^{-4}. The models



(a) Next activity pred. acc. % (b) Satisfaction rate % (c) DL distance

Figure 2: Empirical results.

that incorporate LTL background knowledge use α set to 0.6 (Eq. 7), τ equal to 0.5 (Eq. 8), and T equal to $2 \times l = 40$ timesteps (Eq. 11).

The plots in Figure 2 show the average and standard deviation of the results obtained from running experiments on all 25 datasets. The evaluated metrics are the model's test accuracy on the next activity prediction (Figure 2a), the satisfaction rate of test suffixes (Figure 2b), and the Damerau-Levenshtein (DL) distance (Figure 2c) of the predicted traces against the ground truth traces. The empirical results show that the integration of background knowledge does not affect the accuracy of next activity prediction, indicating that the predictor retains its capability to mimic training data when combined with logical knowledge. However, RNN+LTL models consistently achieve the best performance in suffix prediction. In terms of the satisfiability percentage metric, utilizing LTL background knowledge consistently results in an improved satisfaction rate over using only data, regardless of the sampling technique employed. Varying prefix lengths display no significant impact on the satisfaction rate, implying that the models are equally capable of sampling traces that satisfy the formulas for any input prefix length. The DL distance comparison further supports the superiority of models that utilize LTL background knowledge, showing a lower distance between the sampled traces and the ground truth traces for both sampling strategies used. The prefix length, however, does impact the DL distance, with shorter prefix inputs leading to a higher distance between the predicted and ground truth traces. In general, random sampling shows better performance compared to greedy sampling in terms of both metrics. Additionally, the DL distance results of the random sampling methods exhibit a more stable standard deviation than those of the greedy sampling methods. Overall, our empirical evaluation supports the hypothesis that integrating background knowledge into the models consistently results in improved performance.

6. Conclusions and Future Work

In conclusion, we present a novel NeSy framework for incorporating background knowledge in LTL_f into the training process of a deep autoregressive model for multistep symbolic sequence generation (i.e., suffix prediction). Our method involves: (i) translating the LTL_f formula into a DFA, (ii) representing the DFA probabilistically through a neural framework, and (iii) softly enforcing DFA acceptance on differentiable suffixes obtained via Gumbel-Softmax sampling.

This process allows us to define a logical loss that smoothly guides the predicted suffixes to satisfy the given knowledge, which can be integrated with other loss functions of the predictor. Experimental results on synthetic datasets demonstrate that applying our loss function to RNN models consistently produces suffixes with both lower DL distances and higher rates of LTL_f knowledge satisfaction, regardless of the sampling technique used for querying the network. In future work, we plan to apply our framework to realistic BPM datasets and more advanced neural predictors, such as GANs and Transformers.

Acknowledgments

This work has been partially supported by PNRR MUR project PE0000013-FAIR , the Sapienza Project MARLeN, the EU H2020 project AIPlan4EU (No. 101016442), the ERC-ADG WhiteMech (No. 834228), and the PRIN project RIPER (No. 20203FFYLK).

References

[1] E. Rama-Maneiro, J. C. Vidal, M. Lama, Deep learning for predictive business process monitoring: Review and benchmark, IEEE Transactions on Services Computing 16 (2020) 739–756.

[2] N. Tax, I. Verenich, M. L. Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings, 2017, pp. 477–492. URL: https://doi.org/10.1007/978-3-319-59536-8_30. doi:10.1007/978-3-319-59536-8_30.

[3] G. Rivera Lazo, R. Ñanculef, Multi-attribute transformers for sequence prediction in business process management, in: Discovery Science: 25th International Conference, DS 2022, Montpellier, France, October 10–12, 2022, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2022, p. 184–194. doi:10.1007/978-3-031-18840-4_14.

[4] E. Rama-Maneiro, F. Patrizi, J. C. Vidal, M. Lama, Towards learning the optimal sampling strategy for suffix prediction in predictive monitoring, in: Proc. of CAISE 2024, 2024, p. To Appear.

[5] E. Giunchiglia, M. C. Stoian, S. Khan, F. Cuzzolin, T. Lukasiewicz, ROAD-R: the autonomous driving dataset with logical requirements, Mach. Learn. 112 (2023) 3261–3291. doi:10.1007/S10994-023-06322-Z.

[6] M. Ma, J. Gao, L. Feng, J. Stankovic, Stlnet: Signal temporal logic enforced multivariate recurrent neural networks, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, volume 33, Curran Associates, Inc., 2020, pp. 14604–14614.

[7] C. Di Francescomarino, F. M. Maggi, G. Petrucci, A. Yeshchenko, An eye into the future: Leveraging a-priori knowledge in predictive business process monitoring, in: Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings, 2017, pp. 252–268. doi:10.1007/978-3-319-65000-5_15.

[8] E. Jang, S. Gu, B. Poole, Categorical reparameterization with gumbel-softmax, in: 5th

International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[9] F. J. Damerau, A technique for computer detection and correction of spelling errors, Commun. ACM 7 (1964) 171–176. doi:10.1145/363958.363994.

[10] N. Tax, I. Verenich, M. L. Rosa, M. Dumas, Predictive business process monitoring with LSTM neural networks, in: Advanced Information Systems Engineering - 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings, 2017, pp. 477–492. doi:10.1007/978-3-319-59536-8_30.

[11] J. Evermann, J. Rehse, P. Fettke, Predicting process behaviour using deep learning, Decis. Support Syst. 100 (2017) 129–140. URL: https://doi.org/10.1016/j.dss.2017.04.003. doi:10.1016/J.DSS.2017.04.003.

[12] I. Ketykó, F. Mannhardt, M. Hassani, B. F. van Dongen, What averages do not tell - predicting real life processes with sequential deep learning, CoRR abs/2110.10225 (2021). URL: https://arxiv.org/abs/2110.10225. arXiv:2110.10225.

[13] N. D. Mauro, A. Appice, T. M. A. Basile, Activity prediction of business process instances with inception CNN models, in: AI*IA 2019 - Advances in Artificial Intelligence - XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19-22, 2019, Proceedings, 2019, pp. 348–361. URL: https://doi.org/10.1007/978-3-030-35166-3_25. doi:10.1007/978-3-030-35166-3_25.

[14] F. Taymouri, M. L. Rosa, S. M. Erfani, A deep adversarial model for suffix and remaining time prediction of event sequences, in: Proceedings of the 2021 SIAM International Conference on Data Mining, SDM 2021, Virtual Event, April 29 - May 1, 2021, 2021, pp. 522–530. URL: https://doi.org/10.1137/1.9781611976700.59. doi:10.1137/1.9781611976700.59.

[15] A. Chiorrini, C. Diamantini, A. Mircoli, D. Potena, A preliminary study on the application of reinforcement learning for predictive process monitoring, in: Process Mining Workshops - ICPM 2020 International Workshops, Padua, Italy, October 5-8, 2020, Revised Selected Papers, 2020, pp. 124–135. doi:10.1007/978-3-030-72693-5_10.

[16] T. R. Besold, A. S. d'Avila Garcez, S. Bader, H. Bowman, P. M. Domingos, P. Hitzler, K. Kühnberger, L. C. Lamb, P. M. V. Lima, L. de Penning, G. Pinkas, H. Poon, G. Zaverucha, Neural-symbolic learning and reasoning: A survey and interpretation, in: Neuro-Symbolic Artificial Intelligence: The State of the Art, 2021, pp. 1–51. doi:10.3233/FAIA210348.

[17] Y. Xie, F. Zhou, H. Soh, Embedding symbolic temporal knowledge into deep sequential models, CoRR abs/2101.11981 (2021). URL: https://arxiv.org/abs/2101.11981. arXiv:2101.11981.

[18] E. Umili, R. Capobianco, G. D. Giacomo, Grounding ltlf specifications in image sequences, in: Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning, KR 2023, Rhodes, Greece, September 2-8, 2023, 2023, pp. 668–678. URL: https://doi.org/10.24963/kr.2023/65. doi:10.24963/KR.2023/65.

[19] S. Badreddine, A. d'Avila Garcez, L. Serafini, M. Spranger, Logic tensor networks, Artificial Intelligence 303 (2022) 103649. URL: https://www.sciencedirect.com/science/article/pii/S0004370221002009. doi:https://doi.org/10.1016/j.artint.2021.103649.

[20] E. Umili, F. Argenziano, A. Barbin, R. Capobianco, Visual reward machines, in: Proceedings of the 17th International Workshop on Neural-Symbolic Learning and Reasoning, La Certosa di Pontignano, Siena, Italy, July 3-5, 2023, 2023, pp. 255–267. URL: https://ceur-ws.

org/Vol-3432/paper23.pdf.
[21] E. Umili, R. Capobianco, DeepDFA: a transparent neural network design for dfa induction, 2023. doi:10.13140/RG.2.2.25449.98401.
[22] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977, IEEE Computer Society, 1977, pp. 46–57. URL: https://doi.org/10.1109/SFCS.1977.32. doi:10.1109/SFCS.1977.32.
[23] G. De Giacomo, M. Y. Vardi, Linear temporal logic and linear dynamic logic on finite traces, in: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, IJCAI '13, AAAI Press, 2013, p. 854–860.
[24] G. De Giacomo, R. De Masellis, M. Montali, Reasoning on ltl on finite traces: Insensitivity to infiniteness, Proceedings of the AAAI Conference on Artificial Intelligence 28 (2014). URL: https://ojs.aaai.org/index.php/AAAI/article/view/8872. doi:10.1609/aaai.v28i1.8872.
[25] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, G. Lample, Llama: Open and efficient foundation language models, CoRR abs/2302.13971 (2023). URL: https://doi.org/10.48550/arXiv.2302.13971. doi:10.48550/ARXIV.2302.13971. arXiv:2302.13971.
[26] OpenAI, Gpt-4 technical report, 2024. arXiv:2303.08774.
[27] A. van den Oord, N. Kalchbrenner, K. Kavukcuoglu, Pixel recurrent neural networks, in: M. F. Balcan, K. Q. Weinberger (Eds.), Proceedings of The 33rd International Conference on Machine Learning, volume 48 of *Proceedings of Machine Learning Research*, PMLR, New York, New York, USA, 2016, pp. 1747–1756.
[28] T. Salimans, A. Karpathy, X. Chen, D. P. Kingma, Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications, in: ICLR, 2017.
[29] A. Casolaro, V. Capone, G. Iannuzzo, F. Camastra, Deep learning for time series forecasting: Advances and open problems, Information 14 (2023). URL: https://www.mdpi.com/2078-2489/14/11/598. doi:10.3390/info14110598.
[30] C. J. Maddison, A. Mnih, Y. W. Teh, The concrete distribution: A continuous relaxation of discrete random variables, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017. URL: https://openreview.net/forum?id=S1jE5L5gl.
[31] E. Umili, Neurosymbolic integration of linear temporal logic in non symbolic domains, in: Multi-Agent Systems - 20th European Conference, EUMAS 2023, Naples, Italy, September 14-15, 2023, Proceedings, 2023, pp. 521–527. URL: https://doi.org/10.1007/978-3-031-43264-4_41. doi:10.1007/978-3-031-43264-4_41.
[32] F. Fuggitti, Ltlf2dfa, 2019. doi:10.5281/zenodo.3888410.
[33] G. De Giacomo, M. Dumas, F. M. Maggi, M. Montali, Declarative process modeling in bpmn, in: J. Zdravkovic, M. Kirikova, P. Johannesson (Eds.), Advanced Information Systems Engineering, Springer International Publishing, Cham, 2015, pp. 84–100.