Triggering Proactive Business Process Adaptations via Online Reinforcement Learning

Andreas Metzger(✉), Tristan Kley, and Alexander Palm

paluno – The Ruhr Institute for Software Technology,
University of Duisburg-Essen, Essen, Germany
{andreas.metzger,tristan.kley,alexander.palm}@paluno.uni-due.de

Abstract. Proactive process adaptation can prevent and mitigate upcoming problems during process execution by using predictions about how an ongoing case will unfold. There is an important trade-off with respect to these predictions: Earlier predictions leave more time for adaptations than later predictions, but earlier predictions typically exhibit a lower accuracy than later predictions, because not much information about the ongoing case is available. An emerging solution to address this trade-off is to continuously generate predictions and only trigger proactive adaptations when prediction reliability is greater than a predefined threshold. However, a good threshold is not known a priori. One solution is to empirically determine the threshold using a subset of the training data. While an empirical threshold may be optimal for the training data used and the given cost structure, such a threshold may not be optimal over time due to non-stationarity of process environments, data, and cost structures. Here, we use online reinforcement learning as an alternative solution to learn when to trigger proactive process adaptations based on the predictions and their reliability at run time. Experimental results for three public data sets indicate that our approach may on average lead to 12.2% lower process execution costs compared to empirical thresholding.

Keywords: Machine learning · Business process monitoring · Prediction · Adaptation · Accuracy · Earliness

1 Introduction

Predictive business process monitoring predicts how an ongoing case will unfold [2,16,35]. To this end, predictive business process monitoring uses the sequence of events produced by the execution of a case to make predictions about the future state of the case. If the predicted future state of the case indicates a problem, the ongoing case may be proactively adapted; e.g., by re-scheduling process activities or by changing the assignment of resources [19,24,37]. Proactive business process adaptation can prevent the occurrence of problems and it

D. Fahland et al. (Eds.): BPM 2020, LNCS 12168, pp. 273–290, 2020.
https://doi.org/10.1007/978-3-030-58666-9_16

can mitigate the impact of upcoming problems during business process execution [26,28,36]. As an example, a delay in the expected delivery time for a freight transport process may incur contractual penalties. If a delay is predicted, faster, alternative transport activities (such as air delivery instead of road delivery) can be scheduled to prevent the delay and thus the penalty.

Problem Statement. There are two key requirements for predictive business process monitoring when used for proactive process adaptation [20,22,35]. First, predictions should be *accurate*. When adaptation decisions are based on inaccurate predictions, this may imply unnecessary adaptations (e.g., if a delay is falsely predicted) or missed adaptations (e.g., if an actual delay is not predicted). Second, predictions should be produced *early* during process execution. Earlier predictions leave more time for adaptations, which typically have non-negligible latencies. However, there is an important trade-off between generating accurate predictions and generating them early. Earlier predictions typically have a lower accuracy than later predictions, because not much information about the ongoing case is available [20,35].

An emerging solution to address this trade-off is to continuously generate predictions and dynamically determine when a prediction is sufficiently reliable [9,22,34–36]. Only a reliable prediction is used to trigger a proactive process adaptation. Typically, this entails generating, for each prediction, an estimate of the prediction's reliability. A simple example for a reliability estimate is the class probability generated by a random forest classifier. A prediction is considered reliable if the reliability estimate is greater than a given threshold. As an example, all predictions with a class probability higher than 95% may be considered reliable predictions.

By setting different thresholds, one can trade earliness against accuracy [22,36]. If earlier predictions are preferred, a lower threshold may be chosen, at the risk that predictions are not very accurate. On the other hand, if accurate predictions are required, a higher threshold may be chosen, at the risk that adaptations will be triggered too late as to be effective. Experimental results indicate that a higher threshold (and thus more conservative stance in taking adaptation decisions) may help achieve cost savings even if the individual adaptations are expensive [19,22]. Yet, this more conservative stance comes at the expedient of achieving smaller cost saving on average. Based on these observations, a process manager with a lower risk affinity may set a higher threshold than a process manager with a higher risk affinity. However, how to set a concrete threshold that is optimal in the given situation remains open.

Another approach, which eliminates the need to manually set a threshold, is to empirically determine a threshold. This so called *empirical thresholding* is performed via a dedicated training process involving a separate training data set and knowledge about the concrete cost structure of process execution [8,36]. This ensures that the threshold is optimal for the training data used and the given cost structure. However, the threshold may not remain optimal over time due to non-stationarity of process environments, data, and cost structures.

Contributions. Here, we introduce an alternative approach that does not require manually determining a threshold nor does it require a-priori information to empirically determine such threshold. In fact, our approach does not aim to determine an optimal threshold at all. Instead, we combine two emerging machine learning paradigms to decide when to trigger proactive business process adaptations. On the one hand, we use ensembles of deep learning models to generate predictions and their reliability estimates. On the other hand, we use online reinforcement learning (RL) to learn at run time when to trigger an adaptation based on the predictions and their reliability estimates.

We use the predictions and reliability estimates generated by a predictive monitoring system developed in our previous work [20,22]. To provide a more fine-grained signal to online RL, we extended this system from classification to regression in order to generate numeric instead of binary predictions.

We build on our online RL approach presented in earlier work [25], and extend it with respect to three main aspects:

- *Stronger reward function:* The reward function is a key element in RL, because rewards quantify the feedback for executing an action (i.e., an adaptation in our case). RL aims to find an action-selection policy that maximizes long-term rewards. Here, we present a reward function that provides a strong signal to the RL algorithm. For the BPIC2017 data set used in [25], this led to 36.9% lower costs on average than the initial reward function proposed in [25]. In addition, we decouple rewards from the concrete cost structure of process execution, which eliminates the need to re-tune the reward function when the cost structure changes.
- *Additional data sets:* In addition to the BPIC 2017 data set used in [25], we experimentally evaluate our approach on two additional BPM data sets: BPIC 2012 and Traffic. In particular, the Traffic data set visibly exhibits non-stationarity and thus serves to demonstrate how our approach can capture non-stationary process environments.
- *Comparative analysis:* We perform a comparative analysis of our approach with empirical thresholding, as a state-of-the-art approach [8,36]. When compared with empirical thresholding, our approach leads to 12.2% lower process execution costs on average across all three data sets.

In the remainder, Sect. 2 describes our overall approach and its realization. Section 3 reports on the experimental evaluation of our approach. Section 4 discusses validity risks and limitations. Section 5 analyzes related work. Section 6 concludes with an outlook on future work.

2 Overall Approach

Our approach combines two emerging machine learning paradigms as shown in Fig. 1. We leverage *online reinforcement learning (RL)* to learn at run time when to trigger proactive business process adaptations. As input to online RL, we use predictions and their reliability estimates generated via *ensemble deep supervised learning*. We explain these two elements of our approach below.

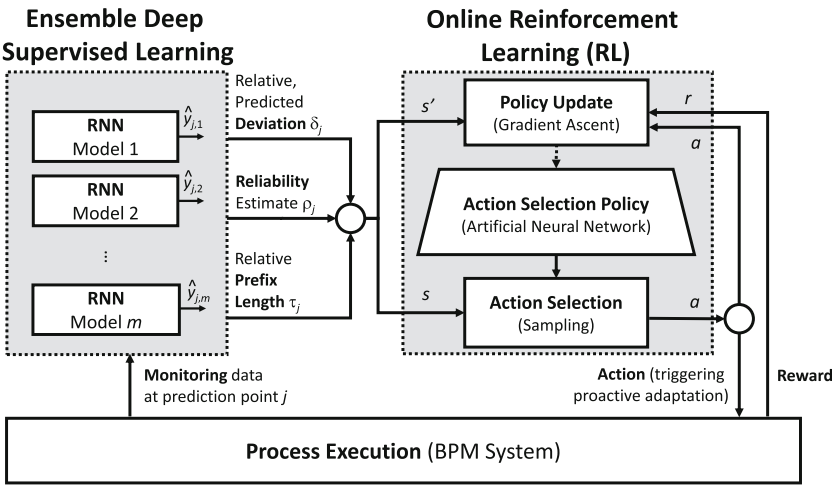


Fig. 1. Combination of machine learning for proactive adaptation

2.1 Ensemble Deep Supervised Learning

As we presented the details of ensemble deep learning for predictive process monitoring in our previous work [20,22], here we only provide a brief summary and the extensions from classification to regression. Regression facilitates computing numeric predictions, which provide a more fine-grained signal to online RL than binary predictions computed via classification.

As shown on the left hand side of Fig. 1, we compute predictions and reliability estimates from ensembles of deep learning models, specifically Recurrent Neural Networks (*RNNs*). RNNs can naturally handle arbitrary length sequences of input data [10] and provide generally high prediction accuracy [7,33]. To implement our approach, we use RNNs with Long Short-Term Memory (LSTM) cells as they better capture long-term dependencies in the data [17,33] and build on the architecture and realization presented by Tax et al. [33].

Ensemble prediction is a meta-prediction technique where the predictions of m prediction models, so called *base learners*, are combined into a single prediction [27,38]. Ensemble prediction is primarily used to increase aggregate prediction accuracy, while it also allows computing reliability estimates [1]. Computing reliability estimates is the main reason why we use ensembles of RNN models in our approach. We use bagging (bootstrap aggregating [6]) as a concrete ensemble technique to build the individual prediction models of our RNN ensemble. Bagging generates m new training data sets from the whole training set by sampling from the whole training data set uniformly and with replacement. For each of the m new training data sets a separate RNN model is trained.

The ensemble of RNN models creates a *prediction* $\hat{y}_j$ at each potential prediction point j. In our case, a prediction is generated after each event of the

running case. These prediction points are thus characterized by their prefix length j, which gives the number of events that were produced up to the prediction point [13,35].

At each prediction point j, each of the m base learners of the ensemble delivers a prediction result $\hat{y}_{i,j}$, with $i = 1, \ldots, m$. Following the typical approach for combining regression models of an ensemble [38], we compute the ensemble prediction as the mean of all predictions:

$$\hat{y}_j = \frac{1}{m} \cdot \sum_{i=1}^{m} \hat{y}_{j,i}.$$

To determine whether this numeric prediction indicates a violation (and thus may require a proactive adaptation), we compute the *relative predicted deviation* δ_j of the prediction from the planned process outcome A:

$$\delta_j = \frac{\hat{y}_j - A}{A}.$$

Without loss of generality, we assume $\delta_j > 0$ means violation. As an example, if the predicted duration of a transport process is longer than its planned duration, this means a violation, as the planned delivery time will not be met.

The *reliability* estimate ρ_j for the ensemble prediction $\hat{y}_j$ is computed as the fraction of the base learners that predicted a violation or a non-violation. We compute the relative deviation $\delta_{i,j}$ for each ensemble prediction and use this to determine the reliability estimate as follows:

$$\rho_j = max_{i=1,\ldots,m}\left(\frac{|i : \delta_{i,j} > 0|}{m}, \frac{|i : \delta_{i,j} \leq 0|}{m}\right).$$

2.2 Online Reinforcement Learning (RL)

RL learns the effectiveness of an agent's actions through the agent's interactions with its environment [31]. As shown on the right hand side of Fig. 1, the agent selects and executes an action a in response to environment state s. As a result, the environment transitions to s' and the agent receives a reward r for executing the action. The goal of RL is to maximize cumulative rewards.

The reward function $r(s)$ specifies the numeric reward the system receives when reaching the new environment state s' after executing action a. Thereby, the reward function quantifies the learning goal to achieve. As an example, a simple reward function provides a positive reward $r = 100$ if action a had the desired effect (i.e., led to the desired new environment state), and a negative reward $r = -100$ if it did not.

The successful application of RL depends on how well the learning problem, and in particular the reward function, is defined [5]. Below, we first explain how we define a suitable learning problem for triggering proactive process adaptations, and then provide details on our concrete RL realization.

Learning Problem. We formalize the learning problem of when to trigger a proactive process adaptation by defining actions a, states s and rewards r.

We define an action a as either triggering ($a = \mathsf{true}$) or not triggering ($a = \mathsf{false}$) a proactive process adaptation.

We build the state s from the output of the predictive monitoring system, which includes the predicted deviation δ_j, the reliability estimate ρ_j as well as information about the current prediction point j given as the relative prefix length τ_j, i.e., each state s is represented by $s = (\delta_j, \rho_j, \tau_j)$. In addition to the relative deviation δ_j and the prediction reliability ρ_j introduced in Sect. 2.1, we also use the relative prediction point τ_j of the current case as input. Using τ_j provides an important signal to the RL algorithm about the earliness of the prediction. This relative prediction point can be computed by dividing the prediction point j by the case length. In situations, where the case length is not known a-priori, the case length may be estimated by the upper bound of process lengths observed in the training data for the RNN ensemble.

Finally, the most important part of formalizing the learning problem is to define suitable rewards r. By giving a reward function, one expresses the learning goal in a declarative fashion. As mentioned above, the aim of the learning process is to maximize cumulative rewards. Therefore, finding a suitable reward function is key to successful learning. One obvious approach is to model the reward function as close as possible to the actual problem domain, which is what we did in our previous work [25]. Here, this would mean to define rewards to be the concrete costs of process execution as encoded in the underlying cost model of the process (see Sect. 3.2). Defining the reward function in such a way has the appeal of a direct mapping between rewards and the behavior of a system in its respective application domain [4]. Yet, it exhibits the following two shortcomings.

First, using the cost model as a reward function means that learning strongly depends on the concrete costs. Whenever the cost model changes, this would require an update of the reward function. We thus decouple the rewards from the actual process costs. In particular, this decoupling should facilitate our RL approach to become more robust against changing cost structures at run time.

Table 1. Reward function r based on prediction contingencies

	Predicted Violation	Predicted Non-violation
Actual Violation	$+1 \cdot (1 - \tau_j)$ (*necessary adaptation*)	-1 (*missed adaptation*)
Actual Non-violation	$-.5 - .5 \cdot (1 - \tau_j)$ (*unnecessary adaptation*)	$+1$ (*no adaptation*)

Second, using the cost model as reward function may not provide a strong enough reward signal to steer the learning process in the right direction and facilitate convergence. In particular, a successful process execution (true negative prediction) would entail zero costs (see Sect. 3.2). However, zero provides only

a weak signal to the learner. We thus formulate strong rewards for each of the prediction contingencies as shown in Table 1.

We provide a strong positive reward signal (+1) if no adaptation was needed and none was triggered (*no adaptation*). Similarly, we provide a strong positive reward signal in the case of a *necessary adaptation*. However, we take into account that not all adaptations triggered may be effective. As explained in Sect. 1, the chances for effective adaptations are the highest at the beginning of the case. We thus discount the reward by a factor of $(1 - \tau_j)$.

We provide a strong negative reward (−1) for a *missed adaptation*. Similarly, we provide a negative reward signal in the case of *unnecessary adaptations* by penalizing the adaptation itself (−.5), as well as the potential compensations that may be required if the adaptation was effective $(-.5 \cdot (1 - \tau_j))$.

We break down the RL problem into suitable episodes, each episode matching the execution of a single case. For each prediction point (process activity), our approach decides whether to adapt or not. Whenever the approach decides to adapt or when the end of the case is reached, we provide a reward r as described above; otherwise, we provide a reward of zero as illustrated in Fig. 2. In order not to discount the reward received at the end of the case, we consequently set the discount factor of the RL algorithm to $\gamma = 1$. The discount factor is a standard hyper-parameter[1] in RL and defines the relevance of future rewards.

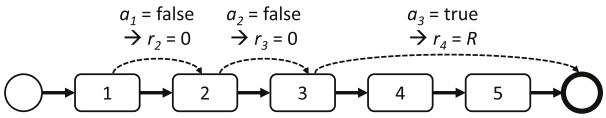


Fig. 2. Episodic learning problem

Realization. We realize our approach using the *policy-based* RL framework for self-adaptive information systems we introduced in [25]. We use policy-based RL, because it offers two main benefits over classical, value-based RL algorithms, such as SARSA and Q-Learning [31]. Value-based RL algorithms often use a lookup table to represent the learned knowledge, which requires manually quantizing environment states to facilitate scalability if the environment has a high number of states or if the environment state is represented by continuous variables. In our case, reliability estimates and predictions are represented by continuous variables. In addition, to facilitate convergence of learning, value-based RL algorithms require manually fine-tuning exploitation versus exploration, i.e., using current knowledge versus gathering new knowledge. When applying RL in an online setting, as we do, value-based RL therefore poses the challenge of when and how to modify the exploration rate to capture non-stationary environments [31].

[1] Hyper-parameters are used to configure the machine learning algorithms and thereby control the learning process.

The fundamental idea behind policy-based RL is to directly use and maximize a parametrized stochastic action selection policy [23,32]. This *action selection policy* (see Fig. 1) maps states to a probability distribution over the action space (i.e., set of possible actions). Actions are selected by sampling from this probability distribution. In our approach, we represent the policy as an artificial neural network (concretely a multi-layer perceptron), because this facilitates generalizing over unseen states.

At run time, *action selection* (see Fig. 1) samples an action a from the probability distribution given by the policy for the current state s. Action selection determines whether there is a need for triggering a proactive adaptation. The generated actions constitute proactive adaptation triggers.

The actual online learning happens via *policy updates* (see Fig. 1). During a policy update, the policy parameters are perturbed based on the rewards received, such that the resulting probability distribution is shifted towards a direction which increases the likelihood of selecting actions leading to a higher cumulative reward. As we represent the policy as an artificial neural network, the policy parameters (the weights of the neural network) are updated via so-called policy gradient methods. These methods update the policy according to the gradient of a given objective function, such as average rewards over the last q actions. Because of this sampling and the probabilistic nature of the policy, policy-based RL does not require manually fine-tuning the balance between exploitation and exploration. Exploration is automatically performed, because sampling over the probability distribution given by the policy leads to some degree of random action selection.

As a concrete policy-based RL algorithm, we use proximal policy optimization (PPO [29]) in the form of OpenAI's baseline implementation[2]. PPO is rather robust for what concerns hyper-parameter settings. Thereby, we avoid the need for extensive hyper-parameter tuning compared to other policy-based RL algorithms. In addition, PPO avoids too large policy updates by using a so called clipping function. A too large policy update may destroy the learned knowledge in a single policy update. To represent the policy, we used multi-layer perceptrons with two hidden layers of 64 neurons each. The input layer consists of three neurons representing the three state variables; the output layer consists of one neuron representing the action variable.

3 Experimental Evaluation

To demonstrate the feasibility and applicability of our approach, we experimentally analyze whether we can learn good proactive adaptation policies at run time. To this end, we analyze how learning evolves over time. We also compare the process execution costs of our approach with the costs when using thresholds determined by empirical thresholding as state-of-the-art baseline.

[2] https://openai.com/blog/openai-baselines-ppo/.

3.1 Experimental Setup

The experimental setup includes the prototypical implementation of our approach, which consists of code to train and evaluate the RNN-LSTM ensembles, as well as code to perform RL at run time. We use three real-world data sets that are among the ones frequently used to evaluate predictive process monitoring approaches [35]. Table 2 provides key characteristics of these data sets. We chose data sets, which had a large enough size – in terms of number of cases and predictions – to observe the effects of learning (also see the discussion in Sect. 4). Similar to [35], we only considered prefix lengths up to a certain length (99% quantile in our case) in order not to bias the results towards very long cases. All artifacts, including code, data sets, prediction results, as well as experimental outcomes, are publicly available to facilitate replicability[3].

Table 2. Data sets used in experiments

Name	Domain	# Cases	# Predictions	Max. Prefix Len.
BPIC2012	Credit application	4,361	57,665	49
BPIC2017	Credit application	10,500	232,397	73
Traffic	Traffic fines	50,117	140,536	6

3.2 Cost Model

To quantify and compare process execution costs, we use a cost model based on [22,36]. Table 3 shows the cost model in the structure of a contingency table.

Table 3. Cost model

	Prediction $\hat{y}_j$ = positive		$\hat{y}_j$ = negative
	Effective adaptation	Non-effective adaptation	
Actual y_j = positive	Adaptation costs	Adaptation costs + Penalty	Penalty
y_j = negative	Adaptation costs + Compensation costs	Adaptation costs	0

A proactive adaptation decision entails different costs depending on whether the decision is based on a false positive or a false negative prediction. In addition, the costs depend on whether a proactive adaptation was *effective* or not. We consider an adaptation non-effective if the problem persists after the adaptation.

[3] https://git.uni-due.de/predictive-process-monitoring/bpm-2020-artefacts/.

The cost model considers three main cost factors of proactive process adaptation. *Adaptation costs* incur, because the adaptation of a running case typically requires effort and resources. As an example, adding more staff to speed up the execution of a case incurs additional personnel costs for the execution of this case. *Penalty costs* incur if the process outcome does not match the planned or expected outcome. As an example, a penalty may have to be paid for late deliveries in a transport process. Penalties may be faced in two situations. First, a necessary proactive adaptation may be missed. Second, a proactive adaptation may not be effective and thus the problem remains after adaptation. *Compensation costs* incur if the adaptation turns out to be an adaptation based on false positive predictions, and thus such wrong adaptation may require roll-back or compensation activities. As an example, if a credit card is falsely blocked, this may entail additional costs for issuing a new credit card to the customer to compensate for this false blocking.

To keep the number of experimental variables manageable, we express adaptation costs c_a and compensation costs c_c as a fraction κ of penalty costs c_p, i.e., $c_a = c_c = \kappa \cdot c_p$. By varying κ, we can reflect different situations that may be faced in practice concerning how costly a process adaptation or compensation in relation to a penalty may be.

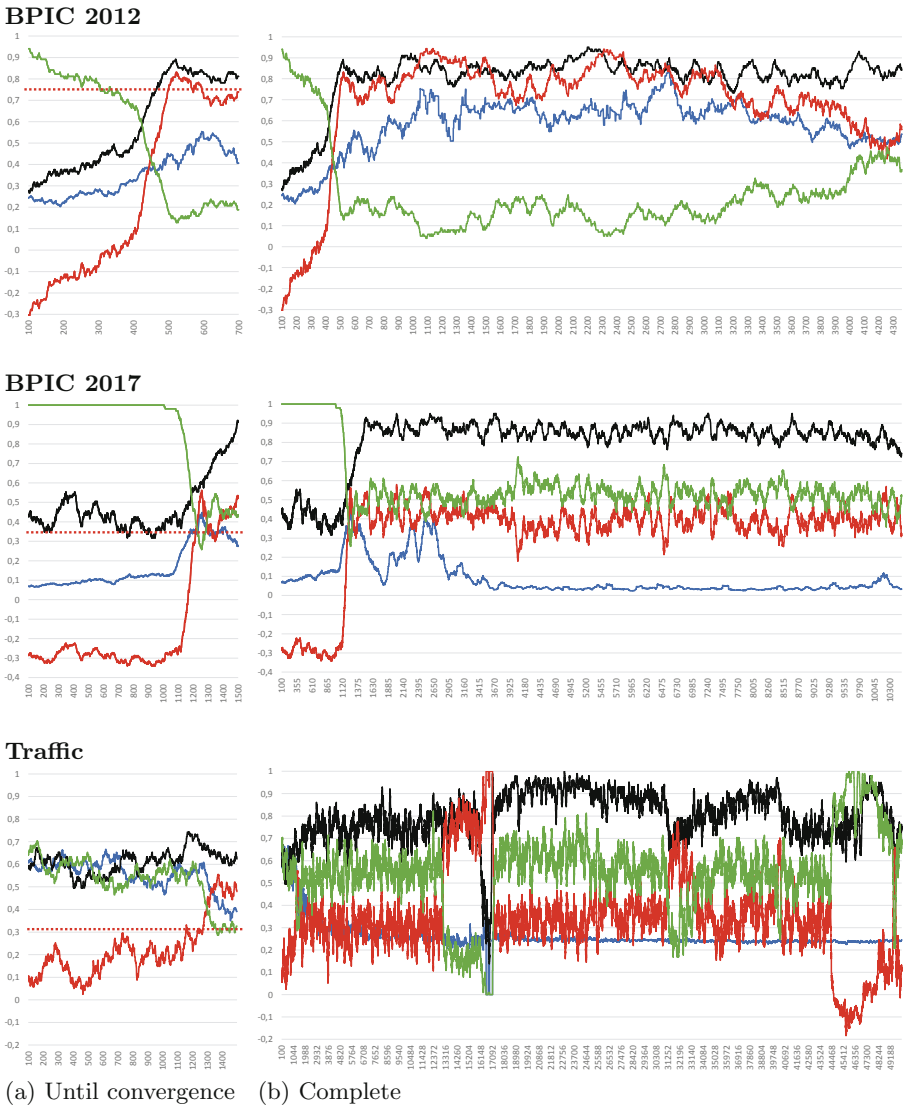
We use α to represent the probability that an adaptation is effective. To model the fact that earlier prediction points provide more options and time for proactive adaptations than later prediction points, earlier proactive adaptations are given a higher α than later adaptations. Specifically, we vary α in our experiments such that it linearly decreases from $\alpha_{max} = 1$ for the first prediction point to α_{min} for the last prediction point. We consider two different settings for α_{min} to cover different possible circumstances: (1) $\alpha_{min} = 0.5$ to model that late adaptations may still be feasible, (2) $\alpha_{min} = 0$ to model that late adaptations are not feasible.

3.3 Experimental Results

We first analyze how the learning process evolves over time in order to demonstrate that RL is able to learn when to trigger proactive process adaptations. Figure 3 show the respective results for the three data sets.

The charts show how the rate of adaptations, earliness, the rate of correct adaptation decisions, and overall rewards evolve (costs are discussed further below). We measure earliness in terms of relative prefix-length when an adaptation was made, i.e., 0 means an adaptation was made at the beginning of the process, while 1 means it was made at the end. Charts start at case # 100, because the points in the charts are averaged over the last 100 cases (for stability reasons).

Part (a) of the charts shows the learning process until convergence can be observed, while part (b) shows the learning process for the whole test data set. We consider convergence to happen as soon as cumulative rewards averaged over the last 100 cases reaches the cumulative rewards averaged across the whole data set, which is indicated as the dashed red line.



(a) Until convergence (b) Complete

Fig. 3. Learning behavior; **green**: rate of adaptations; **blue**: earliness (0 = beginning, 1 = end of process); **black**: rate of correct adaptation decisions; **red**: overall reward/100 (Color figure online)

Across all four data sets, the convergence of the learning process is evident when observing the development of the reward curve. Convergence happens after around 500 cases for BPIC 2012, 1200 for BPIC 2017, and 1300 for Traffic. It can also be seen that the approach indeed is able to learn when to adapt in order to maximize rewards. For all three data sets, the approach starts with

a very high rate of adaptations that are triggered very early in the process. However, this has negative impact on rewards, as the approach has not yet learned that (1) adaptations should only be triggered for positive predictions, (2) not all predictions may be accurate, (3) there is a trade-off between accuracy and earliness. This is also evident in the rate of correct adaptation decisions (which is very low before convergence). After the point of convergence is reached, it can be observed that the approach has learned to be more conservative with triggering adaptations (the rate of adaptations goes down), and also that later predictions may be more accurate (earliness goes up). This results in a higher rate of correct adaptation decisions and thus a higher reward.

Having observed convergence of learning, we now compare the process execution costs of our approach with the process execution costs when using empirical thresholding [36]. Results are given in Tables 4 and 5 for the different settings of $\alpha_{\min}$. We used different concrete process execution costs (expressed as different settings for κ; see Sect. 3.1). For each κ, we computed the optimal threshold based on a 1/3 subset of the test data set and compared the costs of both approaches for the remaining 2/3 of the test data set. For the RL approach, this implies that we compare the results once learning has converged. Note that for Traffic, we consider an excerpt of the data set (including only the first 14,000 cases), because the data set visibly exhibits non-stationarity after that point, which we further analyze below.

Table 4. Average process execution costs of *RL* and empirical *thr*esholding for different relative costs κ and $\alpha_{\min} = 0.5$ (all differences are significant with a p-value < 0.001)

$\alpha_{min} = 0.5$	κ = .1			κ = .2			κ = .3			κ = .4			κ = .5			
	RL	Thr	*Diff*	RL	Thr	*Diff*	RL	Thr	*Diff*	RL	Thr	*Diff*	RL	Thr	*Diff*	*Avg*
Traffic (excerpt)	27,7	25,2	*9,1%*	37,5	41,4	*-10,5%*	47,2	57,6	*-22,0%*	56,9	73,8	*-29,6%*	66,7	89,9	*-34,9%*	***-14,6%***
BPIC 2012	17,8	17,8	*0,0%*	20,3	21,1	*-3,6%*	22,8	23,6	*-3,3%*	25,3	25,9	*-2,1%*	27,8	28,1	*-1,1%*	***-2,0%***
BPIC 2017	8,2	8,8	*-7,8%*	14,8	15,2	*-2,5%*	21,5	21,5	*0,2%*	28,2	27,6	*1,9%*	34,8	33,7	*3,3%*	***-1,7%***
Average			***0,4%***			***-5,5%***			***-8,4%***			***-9,9%***			***-10,9%***	***-6,1%***
Traffic (full)	17,6	17,4	*1,2%*	23,8	28,0	*-17,6%*	30,1	38,6	*-28,5%*	36,3	49,3	*-35,7%*	42,6	59,9	*-40,8%*	***-20,3%***
Average			***-2,2%***			***-7,9%***			***-10,6%***			***-12,0%***			***-12,9%***	***-8,0%***

Table 5. Average process execution costs for $\alpha_{\min} = 0$

$\alpha_{min} = 0.0$	κ = .1			κ = .2			κ = .3			κ = .4			κ = .5			
	RL	Thr	*Diff*	RL	Thr	*Diff*	RL	Thr	*Diff*	RL	Thr	*Diff*	RL	Thr	*Diff*	*Avg*
Traffic (excerpt)	28,4	27,8	*1,9%*	38,1	43,9	*-15,3%*	47,8	60,0	*-25,5%*	57,5	76,1	*-32,3%*	67,2	92,2	*-37,1%*	***-19,0%***
BPIC 2012	22,3	22,6	*-1,3%*	24,6	25,3	*-2,7%*	27,0	28,0	*-3,7%*	29,4	30,6	*-4,2%*	31,7	32,7	*-3,1%*	***-2,7%***
BPIC 2017	8,3	11,0	*-32,5%*	15,0	17,7	*-18,1%*	21,6	23,4	*-8,4%*	28,2	29,1	*-3,1%*	34,9	33,1	*5,1%*	***-13,5%***
Average			***-1,8%***			***-6,0%***			***-8,1%***			***-9,2%***			***-9,5%***	***-6,4%***
Traffic (full)	17,7	19,8	*-11,7%*	23,9	30,3	*-26,7%*	30,2	40,9	*-35,4%*	36,4	51,5	*-41,2%*	42,7	62,0	*-45,3%*	***-20,3%***
Average			***-15,2%***			***-15,8%***			***-15,9%***			***-16,2%***			***-14,4%***	***-12,2%***

Results indicate that the proactive process adaptations triggered by our approach result on average in 6.1% ($\alpha_{\min} = 0.5$) resp. 6.4% ($\alpha_{\min} = 0$) less process execution costs when compared to empirical thresholding.

As we mentioned above, one of the principle advantages of the RL approach is to capture non-stationarity in the data. The Traffic data set shows such non-stationarity between around case # 14,000 and # 16,000, and again after around case # 45,000. Deeper analysis shows that this is due to the fact that the average prediction accuracy for cases # 14,000 to # 16,000 is 65% higher than the average accuracy for the whole data set, while for all cases after case # 45,000 the average accuracy is 51% lower than the average accuracy for the whole data set. We thus now use the complete Traffic data set to analyze how our approach captures non-stationarity. Results are shown in the last row of Tables 4 and 5 respectively. In the presence of non-stationary, our RL approach shows high improvements over empirical thresholding, leading to 20.3% lower costs on average for both settings of $\alpha_{\min}$. We consider the capability of capturing such non-stationary situation one of the key benefits of our approach. Overall, this leads to average savings of 8% resp. 12.2% when we compare our approach for all three complete data sets against empirical thresholding.

The above results also show the robustness of RL in the presence of different concrete costs (expressed as different values of κ). In 24 out of the 30 situations, RL delivers the same or less costs than empirical thresholding. Note that while the optimal empirical threshold is computed for each value of κ, the RL approach is independent of κ.

Finally, Table 6 compares the costs of the strong reward function with the costs of the weak reward function we used in our previous work and which we applied to the BPIC 2017 data set [25]. As can be seen, the stronger reward function leads to a clear increase in the performance of the RL approach, leading to 36.9% less costs on average. This is an important improvement contributing to our RL approach being able to outperform empirical thresholding.

Table 6. Average process execution costs of strong and weak reward function

	κ = .1			**κ = .2**			**κ = .3**			**κ = .4**			**κ = .5**			
	r_{strong}	r_{weak}	*Diff*	r_{strong}	r_{weak}	*Diff*	r_{strong}	r_{weak}	*Diff*	r_{strong}	r_{weak}	*Diff*	r_{strong}	r_{weak}	*Diff*	***Avg***
BPIC 2017	8,2	17,4	*-52,9%*	14,8	25,0	*-40,6%*	21,5	32,6	*-34,0%*	28,2	40,2	*-30,0%*	34,8	47,8	*-27,2%*	***-36,9%***

4 Discussion

4.1 Threats to Validity

Internal Validity. We repeated each of the experiments multiple times in order to assess potential random effects due to the stochastic nature of the neural networks. Even though there were differences in the speed of convergence, the principle learning behavior was consistent across these repetitions. We also purposefully used a multi-layer perceptron as a simple neural network to represent

the policy for the RL approach. Using deep learning models, such as RNN-LSTM, to represent the policy may lead to different results.

External Validity. We used three large, real-world data sets from two different application domains, which differ in key characteristics. To consider the effect of cost factors on the overall process execution costs, we varied the relative adaptation and compensation costs (variable κ). However, we used a rather abstract cost model. For instance, we assumed that the concrete costs for each of the cost factors do not depend on the type of adaptation or severity of a problem, i.e., we only used constant cost functions for adaptation costs and penalties. As such, the effects observed may differ when considering different shapes of cost functions faced in practice. Further experiments, e.g., following the setup described in [18], thus would strengthen external validity.

4.2 Limitations

Speed of Convergence. RL may require quite many learning cycles until the learning process converges. In our experiments, learning required feedback from up to 1,300 cases to converge. This was also one of the reasons why our approach was not applicable to the Cargo 2000 data set, which we used previously in [18,22]. This data set only contains 1,313 cases and we could not reasonably observe convergence. Until RL has converged, the system may execute inefficient adaptations, because not enough observations have yet been made. Inefficient adaptations may lead to negative effects, because they are executed in the live system. Solutions to speed up convergence include finding good initial estimates for the learned knowledge and offline learning via simulations of the system.

Non-stationarity. While our RL approach can capture non-stationarity at run time, we still rely on predictions generated by a supervised deep learning model that is trained once on training data. As non-stationarity of the process data may have an impact on the prediction model (e.g., see the differences in accuracy for the Traffic data set discussed above), one enhancement may be to exploit process drift detection [14,15] and re-train the prediction model if needed.

Multiple Adaptions. We currently formulate the RL problem in such a way as to consider only one possible adaptation. In practice, different adaptation alternatives may exist. In principle, the RL problem can be extended to accommodate more than one action. The RL agent can learn when it is better to execute which of these adaptation alternatives; e.g., by considering different costs for different adaptation actions [8].

Parallel Process Instances. Our RL approach does not consider the influence adaptation actions may have on concurrently running cases. One potential enhancement thus may be to build on existing work such as [3], which facilitates predictions across multiple cases.

5 Related Work

We discuss related work along two complementary streams: (1) addressing the trade-off between earliness and accuracy in predictive process monitoring, and (2) using RL in the context of business process management.

Balancing Earliness and Accuracy. In the literature, the trade-off between prediction accuracy and earliness was approached from different angles.

Several authors use prediction earliness as a dependent variable in their experiments. This means they evaluate their predictive process monitoring techniques by considering prediction earliness in addition to prediction accuracy. As an example, Kang et al. [12], Teinemaa et al. [35], and we in our earlier work [21] measured the accuracy of different prediction techniques for the different prediction points along process execution. Results presented in the aforementioned works clearly show the trade-off between prediction earliness and accuracy. However, it was left open how to resolve the trade-off between accuracy and earliness.

In our previous work, we use reliability estimates computed from ensembles of RNN-LSTM models to decide on proactive adaptation [22]. We dynamically determine the earliest prediction with sufficiently high reliability and in turn use this prediction as basis for proactive adaptation. However, our previous approach required the manual definition of a reliability threshold.

Teinemaa et al. introduce the concept of alarm-based prescriptive process monitoring [8,36]. They use class probabilities generated by random forests (ensembles of decision trees) as reliability estimates to determine whether to raise an alarm to trigger a proactive adaptation. The reliability thresholds above which alarms are raised are determined empirically using a dedicated training data set, taking into account a concrete cost model. This ensures that the threshold is optimal for the specific training data used and the given cost model. Yet, the threshold may not be optimal over time due to non-stationarity of process environments and data, and concrete cost structures.

Reinforcement Learning in BPM. In the literature, a few different RL approaches in the context of BPM were proposed. Huang et al. employ RL for the dynamic optimization of resource allocation in operational business processes [11]. However, they do not consider the proactive adaptation of processes with respect to resources at run time. Also, they use Q-Learning as a classical RL algorithm, and thus assume the environment can be represented by a finite, discrete set of states. As mentioned above, our approach does not have this limitation, but it can directly handle large and continuous environments.

Silvander proposes using Q-Learning with function approximation via a deep neural network (DQN) for the optimization of business processes [30]. They suggest defining a so called ϵ decay rate, to reduce the amount of exploration over time. However, they do not consider using RL at run time, and thus do not take into account how to increase the rate of exploration in the presence of non-stationarity. As mentioned above, our approach does not require tuning the exploration rate and thus does not face this problem.

In our previous work, we proposed a generic framework and implementation for using policy-based RL for self-adaptive information systems [25]. Here, we customized this framework specifically for the problem of triggering proactive process adaptations. In particular, we integrated this framework with our work on predictive process monitoring [22]. In addition, we formulated the reward function in a much stronger and more robust way by decoupling it from actual process costs. Finally, we performed a much broader range of experiments, including two additional data sets and the comparison with a state-of-the-art baseline.

6 Conclusions and Perspectives

We combined two emerging machine learning paradigms – ensemble deep supervised learning and policy-based reinforcement learning – to learn when to trigger proactive process adaptations at run time. Experimental results indicate lower process execution costs when compared to the state of the art, in particular in the presence of non-stationarity.

We plan enhancing our approach in order to speed up the convergence of the reinforcement learning process; e.g., by using simulations to perform offline pre-training. In addition, we plan further experiments to analyze potential benefits of using deep learning models to represent the reinforcement learning policy. Finally, we plan collecting empirical insights from applying the approach in actual business operations.

Acknowledgments. We cordially thank the anonymous reviewers for their constructive comments. Our research received funding from the EU's Horizon 2020 R&I programme under grants 871493 (DataPorts) and 780351 (ENACT).

References

1. Bosnic, Z., Kononenko, I.: Comparison of approaches for estimating reliability of individual regression predictions. Data Knowl. Eng. **67**(3), 504–516 (2008)
2. Cabanillas, C., Di Ciccio, C., Mendling, J., Baumgrass, A.: Predictive task monitoring for business processes. In: Sadiq, S., Soffer, P., Völzer, H. (eds.) BPM 2014. LNCS, vol. 8659, pp. 424–432. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10172-9_31
3. Conforti, R., de Leoni, M., Rosa, M.L., van der Aalst, W.M.P., ter Hofstede, A.H.M.: A recommendation system for predicting risks across multiple business process instances. Decis. Support Syst. **69**, 1–19 (2015)
4. D'Angelo, M., et al.: On learning in collective self-adaptive systems: state of practice and a 3D framework. In: Litoiu, M., Clarke, S., Tei, K. (eds.) 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS@ICSE 2019, Montreal, QC, Canada, pp. 13–24. ACM (2019)
5. Dewey, D.: Reinforcement learning and the reward engineering principle. In: 2014 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, 24–26 March 2014. AAAI Press (2014)

6. Dietterich, T.G.: Ensemble methods in machine learning. In: Kittler, J., Roli, F. (eds.) MCS 2000. LNCS, vol. 1857, pp. 1–15. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45014-9_1
7. Evermann, J., Rehse, J., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
8. Fahrenkrog-Petersen, S.A., et al.: Fire now, fire later: alarm-based systems for prescriptive process monitoring. CoRR abs/1905.09568 (2019)
9. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Petrucci, G., Yeshchenko, A.: An eye into the future: leveraging a-priori knowledge in predictive business process monitoring. In: Carmona, J., Engels, G., Kumar, A. (eds.) BPM 2017. LNCS, vol. 10445, pp. 252–268. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65000-5_15
10. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge (2016)
11. Huang, Z., van der Aalst, W.M.P., Lu, X., Duan, H.: Reinforcement learning based resource allocation in business process management. Data Knowl. Eng. **70**(1), 127–145 (2011)
12. Kang, B., Kim, D., Kang, S.: Real-time business process monitoring method for prediction of abnormal termination using KNNI-based LOF prediction. Expert Syst. Appl. **39**(5), 6061–6068 (2012)
13. Leontjeva, A., Conforti, R., Di Francescomarino, C., Dumas, M., Maggi, F.M.: Complex symbolic sequence encodings for predictive monitoring of business processes. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 297–313. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_21
14. Liu, N., Huang, J., Cui, L.: A framework for online process concept drift detection from event streams. In: 2018 International Conference on Services Computing, SCC 2018, San Francisco, CA, USA, pp. 105–112. IEEE (2018)
15. Maaradji, A., Dumas, M., La Rosa, M., Ostovar, A.: Fast and accurate business process drift detection. In: Motahari-Nezhad, H.R., Recker, J., Weidlich, M. (eds.) BPM 2015. LNCS, vol. 9253, pp. 406–422. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23063-4_27
16. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: a survey. IEEE Trans. Serv. Comput. **11**(6), 962–977 (2018)
17. Mehdiyev, N., Evermann, J., Fettke, P.: A multi-stage deep learning approach for business process event prediction. In: Loucopoulos, P., Manolopoulos, Y., Pastor, O., Theodoulidis, B., Zdravkovic, J. (eds.) 19th Conference on Business Informatics, CBI 2017, Thessaloniki, Greece, pp. 119–128. IEEE Computer Society (2017)
18. Metzger, A., Bohn, P.: Risk-based proactive process adaptation. In: Maximilien, M., Vallecillo, A., Wang, J., Oriol, M. (eds.) ICSOC 2017. LNCS, vol. 10601, pp. 351–366. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69035-3_25
19. Metzger, A., Föcker, F.: Predictive business process monitoring considering reliability estimates. In: Dubois, E., Pohl, K. (eds.) CAiSE 2017. LNCS, vol. 10253, pp. 445–460. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59536-8_28
20. Metzger, A., Franke, J., Jansen, T.: Ensemble deep learning for proactive terminal process management at duisport. In: vom Brocke, J., Mendling, J., Rosemann, M. (eds.) Business Process Management Cases, vol. 2. Springer, Heidelberg (2020)
21. Metzger, A., Neubauer, A.: Considering non-sequential control flows for process prediction with recurrent neural networks. In: 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA, Prague, Czech Republic, pp. 268–272. IEEE Computer Society (2018)

22. Metzger, A., Neubauer, A., Bohn, P., Pohl, K.: Proactive process adaptation using deep learning ensembles. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 547–562. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_34
23. Nachum, O., Norouzi, M., Xu, K., Schuurmans, D.: Bridging the gap between value and policy based reinforcement learning. In: Advances in Neural Information Processing Systems 12 (NIPS 2017), pp. 2772–2782 (2017)
24. Nunes, V.T., Santoro, F.M., Werner, C.M.L., Ralha, C.G.: Real-time process adaptation: a context-aware replanning approach. IEEE Trans. Syst. Man Cybern. Syst. **48**(1), 99–118 (2018)
25. Palm, A., Metzger, A., Pohl, K.: Online reinforcement learning for self-adaptive information systems. In: Dustdar, S., Yu, E., Salinesi, C., Rieu, D., Pant, V. (eds.) CAiSE 2020. LNCS, vol. 12127, pp. 169–184. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49435-3_11
26. Park, G., Song, M.: Prediction-based resource allocation using LSTM and minimum cost and maximum flow algorithm. In: International Conference on Process Mining (ICPM 2019), Aachen, Germany, pp. 121–128 (2019)
27. Polikar, R.: Ensemble based systems in decision making. IEEE Circuits Syst. Mag. **6**(3), 21–45 (2006)
28. Poll, R., Polyvyanyy, A., Rosemann, M., Röglinger, M., Rupprecht, L.: Process forecasting: towards proactive business process management. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNCS, vol. 11080, pp. 496–512. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98648-7_29
29. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR abs/1707.06347 (2017)
30. Silvander, J.: Business process optimization with reinforcement learning. In: Shishkov, B. (ed.) BMSD 2019. LNBIP, vol. 356, pp. 203–212. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-24854-3_13
31. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (2018)
32. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems 12 (NIPS 1999), pp. 1057–1063 (2000)
33. 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
34. Teinemaa, I., Dumas, M., Maggi, F.M., Di Francescomarino, C.: Predictive business process monitoring with structured and unstructured data. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 401–417. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_23
35. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: review and benchmark. TKDD **13**(2), 17:1–17:57 (2019)
36. Teinemaa, I., Tax, N., de Leoni, M., Dumas, M., Maggi, F.M.: Alarm-based prescriptive process monitoring. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) BPM 2018. LNBIP, vol. 329, pp. 91–107. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98651-7_6
37. Weber, B., Sadiq, S.W., Reichert, M.: Beyond rigidity - dynamic process lifecycle support. Comput. Sci. - R&D **23**(2), 47–65 (2009)
38. Zhou, Z.H.: Ensemble Methods: Foundations and Algorithms. Chapman and Hall/CRC, Boca Raton (2012)