Incremental Predictive Process Monitoring: The Next Activity Case

Stephen Pauwels(✉) and Toon Calders

University of Antwerp, Antwerp, Belgium
{stephen.pauwels,toon.calders}@uantwerpen.be

Abstract. Next-activity prediction methods for business processes are always introduced in a static setting, implying a single training phase followed by the application of the learned model during the test phase. Real-life processes, however, are often dynamic and prone to changes over time. Therefore, all state-of-the-art methods need regular retraining on new data to be kept up to date. It is, however, not straightforward to determine when to retrain nor what data to use; for instance, should all historic data be included or only new data? Updating models that still perform at an acceptable level wastes a potentially large amount of computational resources while postponing an update too much will deteriorate model performance. In this paper, we present incremental learning strategies for updating these existing models that do not require fully retraining them, hence reducing the number of computational resources needed while still maintaining a more consistent and correct view of the process in its current form. We introduce a basic neural network method consisting of a single dense layer. This architecture makes it easier to perform fast updates to the model and enables us to perform more experiments. We investigate the differences between our proposed incremental approaches. Experiments performed with a prototype on real-life data show that these update strategies are a promising way forward to further increase the power and usability of state-of-the-art methods.

Keywords: Business process · Event prediction · Incremental learning · Neural networks · Dynamic Bayesian Network

1 Introduction

Predictive process monitoring uses historical data to predict several aspects of ongoing business processes, such as remaining time prediction, outcome prediction, and next-activity prediction. Recently proposed next-activity prediction methods always assume a static setting, where we divide the datasets into fixed training and test parts. One important aspect of Business Processes, however, is that they are inherently dynamic and that different time periods in the log can have different characteristics. Although some authors propose to retrain the model regularly to incorporate the changes in the data, this might not be the most efficient way of updating a model in terms of both runtime and accuracy. Furthermore, no method describes which events to use to retrain the model. It can be beneficial not to use all available historical events as they are no longer relevant for future activities, due to drifts in the processes [4]. No state-of-the-art method proposes how to perform these updates or has been evaluated under these dynamic circumstances.

In this paper, we explore different strategies that can be used for incremental learning of next-activity prediction models. Ranging from completely retraining the models for every new batch of data to only using the newly arrived data to update the existing model. These update strategies can shed new light on the performance of the different methods, as not all methods are equally well suited to be adapted for a dynamic environment.

It is important to note that the strategies proposed in this paper apply to a variety of existing methods, especially neural networks. We aim at showing that the update strategies have significant benefits in a dynamic environment without specifying which of the described models is better.

To visualize the performance over time (with more or less data used for training/updating) we propose a graphical representation of the average accuracy on a given time within a given window. This sliding window technique gives an accurate view of how the predictive performance of the models changes over time.

We show that some incremental methods outperform the completely retrained models despite the *catastrophic forgetting* [26] property of neural networks. Which is the phenomenon when a neural network forgets and ignores the original input when retraining the model with new data. This phenomenon is often a potential risk when updating neural networks. However, we can leverage catastrophic forgetting to gradually forget older, less relevant, events in the presence of concept drift.

Because we needed a lightweight neural network for our initial experiments, we created a basic neural network architecture that consists of a single dense layer. During the experiments, we show that this new architecture (which requires only a limited amount of computational resources) performs on par or outperforms the selected more complex state-of-the-art architectures.

The contributions of our paper are the following:

1. We introduce a simple, but accurate, neural network architecture for next-activity prediction.
2. We compare different update strategies in terms of accuracy and runtime.

The next section gives an overview of related work on incremental learning, concept drift and predictive process monitoring and positions our paper within the field of incremental predictive process monitoring. Section 3 explains the different strategies that can be used for updating. In Sect. 4 we introduce our basic neural network architecture. Experiments on all possible update strategies are performed in Sect. 5.

2 Related Work

2.1 Predictive Process Monitoring

Predictive Process Monitoring aims at correctly predicting various aspects of running business processes. Existing methods deal with predicting the remaining time [1], the outcome [30] or the next activity [27,33] of a running case. In this paper, we focus on predicting the next activity in a running case.

In recent years different types of models are proposed to predict the next activity in a business process. With neural networks becoming extremely popular, lots of new methods are proposed every year.

The first type of network uses a Long-Short-Term-Memory (LSTM) architecture. This type of model is capable of learning the behavior of a sequence of events (hence the memory). Recent methods using this technique are proposed by Evermann et al. [15], Tax et al. [28], Lin et al. [18] and Camargo et al. [7].

The LSTM architecture has also been used in combination with Generative Adversarial Networks by Taymouri et al. [29]. In this type of model, the network exists out of two parts: the first part tries to predict the best activity as well as possible, while the other tries to divide the real activities that happened from the predicted ones. Both parts of the model are trying to outperform the other. In this way, the predictive model gets more accurate feedback about its performance.

LSTM models, however, require significant training time. To address this performance issue posed by the LSTM models, Convolutional Neural Networks (CNNs) were proposed. They also can incorporate the sequential nature of a business process but can train more efficiently. Methods using the CNN architecture have been proposed by Di Mauro et al. [8] and Pasquadibisceglie et al. [22,23].

Pauwels et al. [24] propose to build models using Dynamic Bayesian Networks (DBN). This method is based on techniques in which the data is preprocessed so that it incorporates the time aspect of an event log. The DBN model learns different dependencies between attributes (from both the control-flow and data perspective) that are present in the data and depict the conditional probability of a certain activity happening, given a certain history of events. The activity in the current timestep can depend on every attribute in a previous time step in the k-context log.

2.2 Concept Drift Detection

Concept drift in process mining is well described by Bose et al. [4]. Bose et al. show that concept drift can occur in all perspectives (control-flow, data, resource) and that different types of drift exist, each of which may require a different approach to deal with it. In their paper, Bose et al. focus on detecting the drift points. A disadvantage of this technique is, however, that drifts are only detected after they occurred, leading to a delay in the ability to update existing models.

To detect drifts, Bose et al. propose a sliding window approach. This window is divided into two sub-windows which we want to compare to each other to determine if drift has occurred between the two windows. The characterization of changes between windows is done using a statistical test, such as the Kolmogorov-Smirnov test, the Mann-Whitney U test, or the Hotelling T^2 test.

2.3 Incremental Learning Algorithms

Different applications exist that make use of the dynamic nature of business processes. Some of these applications which are adapted for use in an online setting are: business process discovery [6], conformance checking [5] and concept drift detection [21].

Learning in a setting in which data changes its nature and characteristics over time is a known and well-studied problem within machine learning [17,25]. A popular solution is to use a sliding window to move over the data that indicates which data to use for building a model at a particular moment. One of the downsides of such a sliding window is that they are often of a fixed size, which is determined a priori by the user. This can lead to window sizes that are too big, and thus too insensitive for changes, or too small, in which case too many changes are detected. The correct window size depends on the data itself, and can also vary over time. Therefore, Bifet and Gavalda proposed a learning technique that uses an adaptive window size (`ADWIN`) [3]. On the one hand, when the data is stationary the window size grows, and on the other, the window size shrinks when changes are detected. In contrast to other proposed adaptive methods, Bifet and Gavalda show that the performance of their adaptive window is guaranteed by providing bounds on the false positive and false negative rates.

Another application of incremental learning algorithms is when the data arrives in the form of a data stream in which the compute resources are not able to keep all the arrived data in memory. Hoeffding trees, as proposed by Domingos and Hulten [11], are incremental decision trees that are learned from a massive data stream. This method does, however, assume that the distribution generating the arriving samples does not change over time. Hoeffding trees can be learned in a constant time proportional to the number of attributes.

Gama et al. [16] consider concept drift as described by Bose et al. but propose the use of different incremental algorithms to deal with these changes. This incremental learning overcomes, by constantly updating the learned models, the issue of concept drift often being unexpected and unpredictable. Incremental learning is thus able to update the model in a timely manner, well ahead of models using a concept drift detection method. We continue some of the ideas presented in this work and further elevate them for use with neural networks and next-activity prediction. Gama et al. indicate that besides the types proposed by Bose et al., also outliers may occur in the data. These outliers do not follow the general behavior and should be ignored, rather than incorporated in the model.

Also for neural networks, the task of incrementally learning has already been studied [20,26]. These studies show a typical behavior that occurs when updating existing neural networks called *catastrophic forgetting*. Catastrophic forgetting occurs when a neural network loses the information it learned in previous iterations after new data was used to update the model. Often this forgetting poses significant challenges when updating an existing model, as knowledge learned during the first training phases needs to be remembered. When looking at the dynamic nature of Business Processes, however, we can employ this catastrophic forgetting to our benefit as a natural way of updating the model, while gradually forgetting details of the obsolete distribution.

2.4 Incremental Predictive Process Monitoring

Maisenbacher et al. [19] and Di Francescomarino et al. [9] use the above-mentioned incremental learning algorithms to create incremental models that can predict the outcome of a running case. The main focus of their work is using incremental classifiers that can classify ongoing traces based on their predicted outcome. Maisenbacher et al. look at different existing approaches, like the ones described above, and explore the performances of these approaches when applied to business processes. Di Francescomarino et al. focus on a clustered-based and index-based technique to predict the outcome for an ongoing trace. One of the disadvantages of their work is that they do not show if existing methods could be adapted to incorporate the incremental learning aspect. Their work does indicate the potential benefit of incremental learning in predictive process monitoring.

Berti et al. [2] propose a method for remaining time prediction that can deal with concept drift in the data, by only training on the relevant part of the data that correctly behaves according to the current business process. One disadvantage of their approach is that they need existing concept drift detection methods (like the one proposed by Bose et al.). Knowing which intervals behave in a static way they propose to use distance functions between an ongoing trace and the traces present in the current static interval. Using this distance function they calculate the reliability of a trace in the context of predicting the remaining time for an ongoing trace. In contrast to the approach proposed by Maisenbacher et al., Berti et al. require some a priori knowledge about the different drifts present in the data, making it less suitable for online use.

3 Update Strategies

Different strategies exist that deal with the presence of drifts in the data. We divide them into two main categories; the first category trains a new model (*reset*), the second category updates the existing model (*update*). In the remainder of this paper, we use *learning* to either indicate reset or update. Next, we take a look at how we can select the data used for learning the model.

3.1 Data Selection

Relearning a model after every event is both infeasible and unnecessary. Therefore we divide the event log into windows of a certain size. These windows can be divided by specifying the number of events per window, or by specifying a time interval for every window (days, weeks, months, ...). A log consists of an ordered list of windows $w_0, w_1, \ldots, w_n$ where w_t is the window arrived at time t and n the latest time present in the log. We have for w_i and w_j with $i < j$ that all events in window w_i occurred before all event in window w_j.

Using these windows we can define three different strategies to determine which data to use for learning a model. The first strategy uses all historical data to learn the model (w_0, w_1, ..., w_{t-1}). The next strategy only takes a limited, fixed amount of the immediate history (windows) into account ($w_{t-\ell}$, $w_{t-\ell+1}$, ..., w_{t-1} with ℓ the number of windows to consider). The last method retrains its model when a drift occurred and updates the model after every window with all windows starting from the last drift point up to the current window (w_{d_t}, w_{d_t+1}, ..., w_{t-1} with d_t the time of the latest detected drift).

We can combine all strategies with both the reset and update methods. This leads to six different options for incremental predictive models. The options using the reset strategy can be used straightforwardly with existing methods. When using the update options, existing approaches possibly need extra adaptation.

In this paper, we consider both neural net methods and Dynamic Bayesian Networks. Both these types of methods are already learned in an iterative process, and can thus easily be extended to our incremental approach. Every iteration during training results in a (slightly) adapted model. When, according to a selected loss function, this model performs better than the previous one, we keep this model to start the next iteration with. Updating these models thus only implies that we have to perform extra iterations on the existing model using our updated data.

3.2 Update Existing Methods

In this section, we describe in more detail how the incremental aspect can be added to both neural networks and dynamic bayesian networks based on how these models are learned from data.

Drift-Based Predictions. We use the method proposed by [4] to detect drifts present in the data. To update the model, we retrain the model after every batch using all available data starting from the last seen drift until the most recent used batch of events.

Incremental Neural Networks. Training a neural network is an iterative procedure that tries to optimize a certain model score. This score indicates how accurate the current model is for predicting events (validates the model). Neural network learners use a subset of events from the training data for this validation and the remaining events for actually updating the parameters of the model.

Using new data for updating the model causes the model to diverge from what it originally learned, as it is now validating using new data. Therefore, the model can potentially be less optimal for the original data that was used for the initial training. This is the typical catastrophic forgetting phenomenon that occurs when updating a neural network. For a dataset that contains drift, this is, however, a positive feature rather than a weakness.

As all considered neural network methods train their model in the same way, we can use this incremental method on every proposed neural network model such as the Single Dense Layer (SDL), LSTM, and CNN based methods.

Incremental Dynamic Bayesian Networks. Dynamic Bayesian Networks consist of a model structure and model parameters. Both the structure and parameters are learned from data, and can thus both be updated separately.

The structure of a DBN describes the conditional dependencies between attributes in the data. We learn the model using a hill-climbing algorithm, where we perform iterations of adding or removing dependencies as long as these actions improve the model score. To update the structure we run this algorithm on the existing model. While improvements can be made to the structure, the algorithm will continue to perform iterations.

The parameters of the DBN describe the conditional probabilities for all attributes, and can be calculated as follows:

$$P(A|Pa(A)) = \frac{P(A \cap Pa(A))}{P(Pa(A))} \tag{1}$$

Where $Pa(A)$ are the attributes on which A depends.

To easily being able to compute these probabilities, we create an inverted index for every attribute. Instead of keeping track of the different values that occur in a single row, we keep track of the rows in which a certain value occurs. Due to this inverted index, the DBN method is very suitable for an incremental setting.

Example 1. Consider the event log as shown in Table 1a. We can create an inverted index for the attributes Activity, Role, and Department by listing for every value of the attributes in which event they occur using the event ID. The inverted indexes are shown in Tables 1b, c, and d.

For calculating a conditional probability we can use set operations when dealing with categorical values only:

$$P(Act = A|Role = r_1, Dept = d_0) = \frac{P(Act = A, Role = r_1, Dept = d_0)}{P(Role = r_1, Dept = d_0)} \tag{2}$$

$$= \frac{\#(\{0\} \cap \{0,1\} \cap \{0,1,3\})}{\#(\{0,1\} \cap \{0,1,3\})} = \frac{1}{2} \tag{3}$$

To further improve our implementation we can only keep track of the counts we need in Eq. 3. To update the model, we increment the values that correspond with the combinations of the nominator and denominator.

Table 1. Example log with extra attributes Role and Department (a) and the inverted indexes for Activity (b), Role (c) and Department (d).

eventID	Activity	Role	Department
0	A	r_1	d_0
1	B	r_1	d_0
2	C	r_2	d_1
3	B	r_2	d_0

(a)

Value	IDs
A	{0}
B	{1,3}
C	{2}

(b)

Value	IDs
r_1	{0,1}
r_2	{2,3}

(c)

Value	IDs
d_0	{0,1,3}
d_1	{2}

(d)

4 Reference Model: Single Dense Layer (SDL)

Incrementally updating existing models adds a level of complexity and cost. To reduce the overall cost of learning and updating the models we propose a neural network with a single dense layer for predicting the next event. We did not find any paper exploring the use of simple NNs with only dense layers, without the need of using Petri Nets as intermediate structure [31]. As we show in Sect. 5, such a shallow fully-connected network benefits from low runtimes and high accuracy both in an incremental and non-incremental situation.

We use a prefix-based approach, as this kind of approach captures the sequential nature of the event logs, while at the same time giving an easy, flattened data structure. The SDL network consists of a layer of input cells, corresponding to the number of history steps and the number of attributes used. The selection of these attributes depends on the data used and can vary significantly between datasets. We encode the data using an encoding layer, which encodes the data using one-hot encoding. In the case of numeric attributes, we do not have to encode the values and can use them as-is. These cells are then concatenated before they are linked with a dense layer with as many cells as there are activities and a dropout of 0.2, as proposed by other methods. As the output layer, we use a softmax layer, ensuring that the network returns a probability distribution over all possible events. All input cells $i_0, i_1, \ldots, i_n$ represent the activities and extra attributes present in the entire prefix. We thus have $n = |\mathcal{A}| * k$ input cells in the network, with $\mathcal{A}$ the set of all considered attributes (from both activity and data perspective), and k the size of the history taken into account. The cells $e_0, e_1, \ldots, e_n$ create a integer encoding of the attributes.

Example 2. Suppose we have an event log containing both the activity and resource executing the activity. If we want to create a model with prefix size

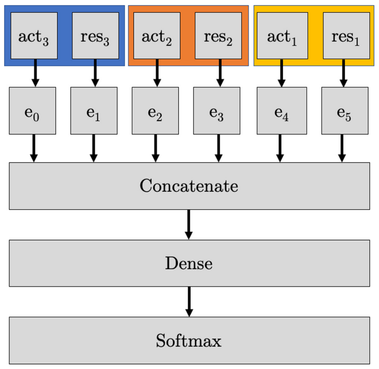

Fig. 1. Example SDL network that takes 3 previous time steps into account from both the activity and resource attribute.

$k = 3$, we have the following input cells: $i_0, i_1, i_2, i_3, i_4, i_5$. Where i_0, i_2, i_4 represent the activities and i_1, i_3, i_5 the resources for the 3 time steps in the prefix. A visual overview of the resulting network can be found in Fig. 1.

5 Experiments

To test the different update strategies we selected 4 methods, each using a different type of model. These methods were selected based on their diversity to cover a wide variety of state-of-the-art methods with shown performance [24]. For this study we use the following methods:

- **Dynamic Bayesian Networks (DBN)**: Pauwels et al. [24]
- **Single Dense Layer NN (SDL)**
- **Long-Short-Term-Memory NN (LSTM)**: Tax et al. [28]
- **Convolutional Neural Network (CNN)**: Di Mauro et al. [10]

In the first place, we are interested in how the accuracy of the different methods changes over time. We define accuracy as the portion of correctly predicted activities. For this purpose, we use a sliding window of fixed size for which we calculate the accuracy obtained within this window. We then use a graphical representation with the event index (in chronological order) on the X-axis and the window accuracy on the Y-axis. This *accuracy-plot* gives us an easy tool to compare different methods and see how the accuracy changes over time. We can use these graphs to see if the tested model does suffer from drifts in the data and if it can recover from changes.

As described in Sect. 3, first a batch size has to be determined that indicates the frequency of performing an update to the model. In our study we tested three different ways of dividing the data into batches; by day, week, or month. Preliminary tests show only minor differences between these batch sizes. Therefore, we decided to use monthly batches for our experiments.

Table 2. Overview of the different datasets. Including the number of days, weeks and month in the train and test parts used.

Dataset				Train			Test		
	#cases	#events	#activities	#days	#weeks	#months	#days	#weeks	#months
Helpdesk	4,580	21,348	14	623	97	23	683	112	27
BPIC11	1,143	150,291	624	622	90	21	550	79	19
BPIC12	13,087	262,200	36	87	14	3	80	12	4
BPIC15_1	1,199	52,217	398	562	112	26	595	122	30
BPIC15_2	832	44,354	410	555	116	28	550	117	28
BPIC15_3	1,409	59,681	383	678	117	29	604	117	28
BPIC15_4	1,053	47,293	356	621	126	30	516	107	26
BPIC15_5	1,156	59,083	389	566	112	27	569	121	29

During our experiments, we are interested in the difference between all update strategies and how a model that uses updates, performs in contrast to a static one. We are thus less interested in determining the single best model to use when incorporating updates in our next-activity predictor.

When evaluating the performance of the update strategies we use an *interleaved-test-then-train* approach. Each batch of data is first used to test the model before we use it for updating the model. All code used for the experiments can be found in our Github Repository[1].

5.1 Dataset Selection

To best test the update capabilities of the models, we need datasets where some drifts occur in the activity perspective. We start with looking at the following datasets which are often used in the literature and have different characteristics:

- **Helpdesk** [32]: a log containing ticket requests of the helpdesk from an Italian software company
- **BPIC11** [14]: a log of a Dutch academic hospital. It shows the different activities and phases the patients go through.
- **BPIC12** [12]: a log containing applications for personal loans. The log contains three intertwined processes.
- **BPIC15** [13]: a log containing building permit applications from five different Dutch municipalities. The log is splitted into five sublogs, one for every municipality (**BPIC15_1** to **BPIC15_5**).

The events in the datasets were first sorted according to timestamp and then split 50/50 in train and test set in chronological order. The details of the different datasets can be found in Table 2. The train set is used to train the initial model and the test set is used to test the method and incrementally update the initial models. To best answer our research questions, we use datasets that do contain

[1] https://github.com/StephenPauwels/edbn.

variation over time in the activity perspective. To look if any drifts occur in the datasets we train a model on the first half of the data. We then test the remaining half using this static model. Using our accuracy-plot, which uses a sliding window to calculate the accuracy, we can then look for changes in accuracy over time.

Figure 2 shows some (small) changes in accuracy over time for the Helpdesk, BPIC11, and BPIC15 datasets when not using any update strategy. The BPIC15 datasets suffer the highest loss of accuracy. This experiment also shows the constant performance of the BPIC12 data. This dataset is useful to see the performance of our incremental learners when no drifts are present. Ideally, the incremental algorithms should perform similarly to the non-incremental ones.

5.2 Baseline Comparison

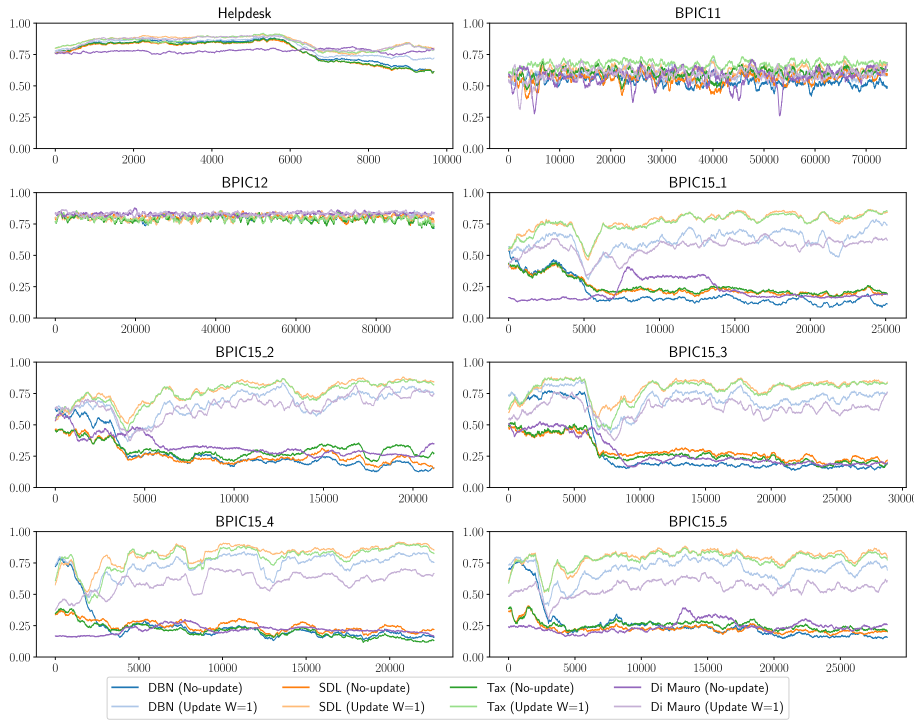

Fig. 2. Accuracy-plot for both a static model and an incremental model using a window size of 1.

A first question that needs to be answered is how much accuracy gain there is when using an incremental algorithm in contrast to the non-incremental ones. Figure 2 shows the average accuracies for the four methods on all datasets. This experiment shows the need for an incremental approach when utilizing prediction models in a real-life setting. The results from the BPIC15 datasets show a large

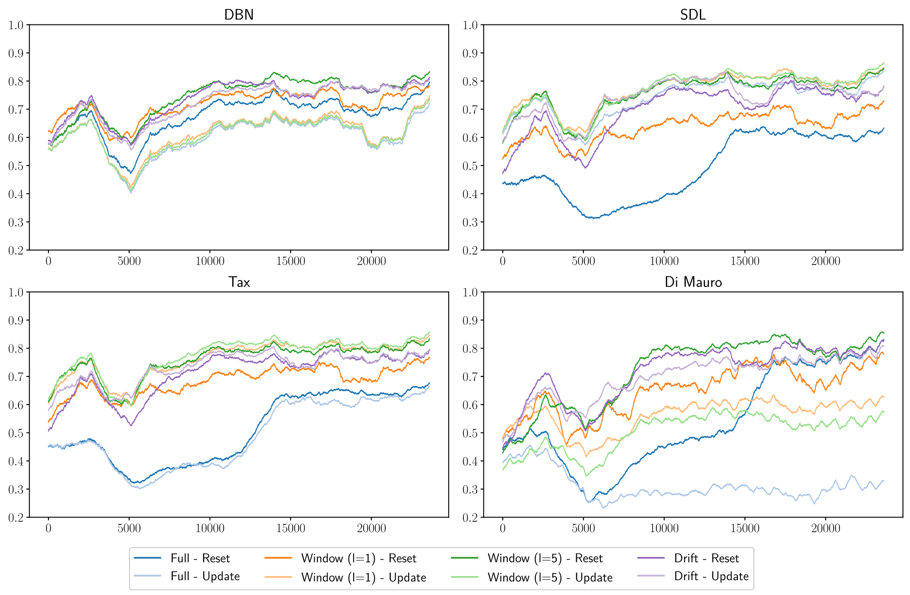

Fig. 3. Accuracy-plot for the different update strategies for the BPIC15_1 dataset.

increase in accuracy. Although the models first seem to suffer from the drift in the data, they all recuperate fast and can keep fairly high accuracies, much higher than the ones seen in the literature for this dataset.

This graph also shows another important aspect of the incremental algorithms; as we do not know in advance if and when a drift occurs, the data may contain no drift at all. Our results on BPIC12 show that the incremental approaches have the same accuracy as the non-incremental ones. This indicates that we can use the incremental versions in all situations, without having to compromise on accuracy.

5.3 Update Strategy

To compare the different incremental approaches we introduced, we selected a single dataset and ran all options for all methods. Figure 3 shows three different observations. We can see that there is little difference between the options for the DBN method. This can be explained by the fact that the DBN model has no way to forget older events.

The SDL and LSTM (Tax et al.) methods show similar behavior. Using all data to update the model, consistently shows the worst performance, as using the full dataset ensures that the model is unable to forget the older events that became irrelevant. We see that using an update strategy using a window scores the best for both methods, thus making use of the catastrophic forgetting to gradually replace the older with newer knowledge. The use of drift detection

Table 3. Runtime (average *(stdev)*) for the update iterations for the BPIC15_1 dataset (in seconds)

Strategy	Batch	DBN	SDL	Tax	Di Mauro
Initial training		48	78	433	188
Reset	Full	106 *(25)*	73 *(18)*	985 *(394)*	450 *(132)*
	Window (size 1)	17 *(4)*	5 *(1)*	37 *(10)*	20 *(9)*
	Window (size 5)	26 *(6)*	14 *(1)*	175 *(39)*	68 *(14)*
	Drift	28 *(15)*	14 *(10)*	187 *(178)*	98 *(87)*
Update	Full	60 *(12)*	56 *(12)*	505 *(101)*	132 *(27)*
	Window (size 1)	1 *(0.5)*	1 *(0.5)*	11 *(4)*	3 *(1)*
	Window (size 5)	7 *(0.8)*	6 *(1)*	59 *(7)*	15 *(2)*
	Drift	3 *(5)*	2 *(1)*	13 *(6)*	6 *(7)*

also gives good results, but this involves running an extra algorithm for every batch to decide whether a drift has occurred.

The architecture using convolutional neural networks (Di Mauro et al.) shows different behavior, in the graph we see that retraining completely achieves the highest accuracy. This can be due to the difference in nature of the convolution layers used in this model.

5.4 Runtime Results

Table 3 shows the average time for each update using the different strategies. Overall we see that SDL is the fastest algorithm to learn, followed by DBN, Di Mauro, and the LSTM architecture of Tax show to be the slowest.

5.5 Overall Results

Table 4 shows an overview of the accuracy obtained by all methods, using the different update strategies on all datasets. This table confirms the behavior that we saw in the previous experiments. We see consistent results for all different methods.

These results also show the performance of our SDL method in comparison to existing methods (both with and without the incremental aspect). We see that our new architecture performs at par with existing methods. On top of that, as the complexity of our model is fairly low, training this model takes considerably less time than training the existing methods.

Table 4. Average accuracy for all methods on all different update settings. Batches grouped using months.

Dataset	Update strategy	Batch	DBN	SDL	LSTM	CNN
Helpdesk	No-update		0.78	0.77	0.77	0.78
	Reset	Full	0.82	0.80	0.78	0.81
		Window (size 1)	0.82	0.83	0.83	0.82
		Window (size 5)	0.84	0.84	0.82	0.83
		Drift	0.83	0.82	0.81	0.82
	Update	Full	0.81	0.80	0.77	0.80
		Window (size 1)	0.81	0.85	0.84	0.84
		Window (size 5)	0.81	0.84	0.83	0.84
		Drift	0.83	0.85	0.84	0.84
BPIC11	No-update		0.54	0.56	0.60	0.57
	Reset	Full	0.58	0.62	0.66	0.63
		Window (size 1)	0.51	0.56	0.58	0.55
		Window (size 5)	0.58	0.63	0.66	0.62
		Drift	0.56	0.60	0.64	0.60
	Update	Full	0.60	0.62	0.60	0.64
		Window (size 1)	0.58	0.62	0.67	0.59
		Window (size 5)	0.58	0.63	0.68	0.60
		Drift	0.57	0.60	0.62	0.58
BPIC12	No-update		0.80	0.81	0.79	0.83
	Reset	Full	0.81	0.81	0.79	0.83
		Window (size 1)	0.79	0.79	0.80	0.82
		Window (size 5)	0.81	0.81	0.80	0.83
		Drift	0.81	0.81	0.79	0.83
	Update	Full	0.81	0.81	0.79	0.83
		Window (size 1)	0.81	0.80	0.80	0.83
		Window (size 5)	0.81	0.81	0.80	0.83
		Drift	0.81	0.80	0.79	0.83
BPIC15_1	No-update		0.20	0.24	0.25	0.21
	Reset	Full	0.68	0.50	0.52	0.55
		Window (size 1)	0.71	0.64	0.68	0.65
		Window (size 5)	0.74	0.75	0.75	0.71
		Drift	0.73	0.69	0.70	0.71
	Update	Full	0.61	0.74	0.50	0.32
		Window (size 1)	0.62	0.76	0.76	0.56
		Window (size 5)	0.61	0.76	0.77	0.50
		Drift	0.72	0.73	0.73	0.69

(*continued*)

Table 4. (*continued*)

Dataset	Update strategy	Batch	DBN	SDL	LSTM	CNN
BPIC15_2	No-update		0.27	0.26	0.30	0.34
	Reset	Full	0.68	0.48	0.51	0.74
		Window (size 1)	0.71	0.60	0.66	0.61
		Window (size 5)	0.73	0.73	0.74	0.74
		Drift	0.73	0.67	0.69	0.71
	Update	Full	0.67	0.73	0.49	0.42
		Window (size 1)	0.67	0.76	0.75	0.65
		Window (size 5)	0.66	0.76	0.76	0.65
		Drift	0.73	0.72	0.71	0.66
BPIC15_3	No-update		0.30	0.30	0.28	0.28
	Reset	Full	0.71	0.50	0.53	0.51
		Window (size 1)	0.74	0.69	0.71	0.71
		Window (size 5)	0.76	0.77	0.78	0.72
		Drift	0.76	0.72	0.72	0.74
	Update	Full	0.70	0.76	0.51	0.40
		Window (size 1)	0.70	0.79	0.78	0.63
		Window (size 5)	0.70	0.79	0.79	0.58
		Drift	0.76	0.77	0.76	0.75
BPIC15_4	No-update		0.25	0.25	0.21	0.21
	Reset	Full	0.74	0.52	0.54	0.60
		Window (size 1)	0.73	0.68	0.71	0.69
		Window (size 5)	0.79	0.79	0.79	0.74
		Drift	0.75	0.58	0.58	0.61
	Update	Full	0.74	0.79	0.52	0.40
		Window (size 1)	0.74	0.81	0.79	0.60
		Window (size 5)	0.73	0.81	0.81	0.54
		Drift	0.75	0.79	0.79	0.65
BPIC15_5	No-update		0.27	0.23	0.26	0.24
	Reset	Full	0.68	0.50	0.52	0.58
		Window (size 1)	0.74	0.68	0.71	0.71
		Window (size 5)	0.75	0.76	0.77	0.71
		Drift	0.76	0.71	0.72	0.74
	Update	Full	0.67	0.75	0.50	0.37
		Window (size 1)	0.69	0.79	0.78	0.55
		Window (size 5)	0.68	0.78	0.78	0.50
		Drift	0.76	0.77	0.76	0.72

6 Conclusion

In this paper, we looked at the situation in which we want to predict the next activity in a business process in a situation where the underlying process can be subject to change. We changed the standard way of evaluating the performance of the methods from using a static training and testing part of the data to a way that we can use all previous events to help predict the next event.

We looked at different options on how to select the data to use for updating the models. We can use the full data, a sliding window technique, or the data after the last detected drift. Updating the models with only a limited number of events reduces the computational resources needed for learning. Furthermore, we looked at the difference in performance when strictly updating an existing model, or by completely retraining models.

We showed that neural network methods can take advantage of the catastrophic forgetting phenomenon that occurs when updating these networks. As the event logs can have dynamic underlying processes, it is logical to give more importance to more recent events. The proposed methods for updating existing models did improve the overall accuracy greatly, without suffering from performance loss when no drift or variation is present in the data.

The DBN method also improved when using an incremental learning approach, but often this improvement was less than the improvement observed for neural network methods. One reason for this is that the current update method for DBNs has no mechanism that forgets older events and/or gives more priority to newer events. In future research, we would like to take a closer look at how to incorporate this in the DBN method to improve the results and make it more flexible when used with changing processes.

Our new architecture showed to perform at par or even outperform some state-of-the-art methods but at a substantially lower computational complexity. In the light of incremental learners, this lower complexity comes at an extra advantage, as updating can be done faster or more often. But, as shown in the experiments, the use of the SDL method should not be limited to incremental settings.

As we only use four different methods in this paper, we cannot make strong conclusions about the best way of solving the incremental next-activity prediction problem. This is often highly dependent on the characteristics of the considered process. We showed that we can leverage existing state-of-the-art methods to cope with variation and drifts in the data by adding a basic incremental framework. Some of the datasets used in our experiments often get ignored in the literature due to their dynamic nature. We showed that, when using a suitable update strategy, most methods are ready to be used in a more challenging environment than the test settings and datasets used most of the time in literature.

References

1. Van der Aalst, W.M., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Inf. Syst. **36**(2), 450–475 (2011)
2. Berti, A.: Improving process mining prediction results in processes that change over time. Data Anal. **2016**, 49 (2016)
3. Bifet, A., Gavalda, R.: SIAM: learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM International Conference on Data Mining (SDM), pp. 443–448. SIAM (2007)
4. Bose, R.J.C., Van Der Aalst, W.M., Žliobaitė, I., Pechenizkiy, M.: Dealing with concept drifts in process mining. IEEE Trans. Neural Netw. Learn. Syst. **25**(1), 154–171 (2013)
5. Burattin, A., Carmona, J.: A framework for online conformance checking. In: Teniente, E., Weidlich, M. (eds.) BPM 2017. LNBIP, vol. 308, pp. 165–177. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-74030-0_12
6. Burattin, A., Cimitile, M., Maggi, F.M., Sperduti, A.: Online discovery of declarative process models from event streams. IEEE Trans. Serv. Comput. **8**(6), 833–846 (2015). https://doi.org/10.1109/TSC.2015.2459703
7. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate LSTM models of business processes. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 286–302. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_19
8. 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
9. Di Francescomarino, C., Ghidini, C., Maggi, F.M., Rizzi, W., Persia, C.D.: Incremental predictive process monitoring: How to deal with the variability of real environments. arXiv preprint arXiv:1804.03967 (2018)
10. Di Mauro, N., Appice, A., Basile, T.M.A.: Activity prediction of business process instances with inception CNN models. In: Alviano, M., Greco, G., Scarcello, F. (eds.) AI*IA 2019. LNCS (LNAI), vol. 11946, pp. 348–361. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-35166-3_25
11. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71–80 (2000)
12. van Dongen, B.: BPI challenge (2012). https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
13. van Dongen, B.: BPI challenge (2015). https://doi.org/10.4121/uuid:31a308ef-c844-48da-948c-305d167a0ec1
14. van Dongen, B.: Real-life event logs - hospital log, March 2011. https://doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
15. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. Decis. Support Syst. **100**, 129–140 (2017)
16. Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. (CSUR) **46**(4), 1–37 (2014)
17. Gepperth, A., Hammer, B.: Incremental learning algorithms and applications. In: European Symposium on Artificial Neural Networks (ESANN) (2016)
18. Lin, L., Wen, L., Wang, J.: MM-PRED: a deep predictive model for multi-attribute event sequence. In: Proceedings of the 2019 SIAM International Conference on Data Mining, pp. 118–126. SIAM (2019)

19. Maisenbacher, M., Weidlich, M.: Handling concept drift in predictive process monitoring. SCC **17**, 1–8 (2017)
20. McCloskey, M., Cohen, N.J.: Catastrophic interference in connectionist networks: the sequential learning problem. In: Psychology of Learning and Motivation, vol. 24, pp. 109–165. Elsevier (1989)
21. Ostovar, A., Maaradji, A., La Rosa, M., ter Hofstede, A.H.M., van Dongen, B.F.V.: Detecting drift from event streams of unpredictable business processes. In: Comyn-Wattiau, I., Tanaka, K., Song, I.-Y., Yamamoto, S., Saeki, M. (eds.) ER 2016. LNCS, vol. 9974, pp. 330–346. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46397-1_26
22. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Using convolutional neural networks for predictive process analytics. In: 2019 International Conference on Process Mining (ICPM), pp. 129–136. IEEE (2019)
23. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Predictive process mining meets computer vision. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 176–192. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58638-6_11
24. Pauwels, S., Calders, T.: Bayesian network based predictions of business processes. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 159–175. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58638-6_10
25. Rokach, L., Maimon, O.: Clustering methods. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook. Springer, Boston (2005). https://doi.org/10.1007/0-387-25465-X_15
26. Serrà Julià, J., Surís, D., Miron, M., Karatzoglou, A.: Overcoming catastrophic forgetting with hard attention to the task. In: Dy, J., Krause, A., (eds.) Proceedings of the 35th International Conference on Machine Learning (ICML 2018), 10–15 July 2018, Stockholmsmässan, Sweden [Massachusetts: PMLR; 2018], pp. 4548–4557. Proceedings of Machine Learning Research (2018)
27. Tax, N., Teinemaa, I., van Zelst, S.J.: An interdisciplinary comparison of sequence modeling methods for next-element prediction. Softw. Syst. Model. **19**(6), 1345–1365 (2020)
28. 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
29. Taymouri, F., Rosa, M.L., Erfani, S., Bozorgi, Z.D., Verenich, I.: Predictive business process monitoring via generative adversarial nets: the case of next event prediction. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 237–256. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58666-9_14
30. Teinemaa, I., Dumas, M., Rosa, M.L., Maggi, F.M.: Outcome-oriented predictive process monitoring: review and benchmark. ACM Trans. Knowl. Discov Data (TKDD) **13**(2), 1–57 (2019)
31. Theis, J., Darabi, H.: Decay replay mining to predict next process events. IEEE Access **7**, 119787–119803 (2019)
32. Verenich, I.: Helpdesk, mendeley data, v1 (2016). https://doi.org/10.17632/39bp3vv62t.1
33. Weinzierl, S., et al.: An empirical comparison of deep-neural-network architectures for next activity prediction using context-enriched process event logs. arXiv preprint arXiv:2005.01194 (2020)