Learning Uncertainty with Artificial Neural Networks for Improved Remaining Time Prediction of Business Processes

Hans Weytjens(✉) and Jochen De Weerdt

Research Centre for Information Systems Engineering (LIRIS),
KU Leuven, Leuven, Belgium
{hans.weytjens,jochen.deweerdt}@kuleuven.be

Abstract. Artificial neural networks will always make a prediction, even when completely uncertain and regardless of the consequences. This obliviousness of uncertainty is a major obstacle towards their adoption in practice. Techniques exist, however, to estimate the two major types of uncertainty: model uncertainty and observation noise in the data. Bayesian neural networks are theoretically well-founded models that can learn the model uncertainty of their predictions. Minor modifications to these models and their loss functions allow learning the observation noise for individual samples as well. This paper is the first to apply these techniques to predictive process monitoring. We found that they contribute towards more accurate predictions and work quickly. However, their main benefit resides with the uncertainty estimates themselves that allow the separation of higher-quality from lower-quality predictions and the building of confidence intervals. This leads to many interesting applications, enables an earlier adoption of prediction systems with smaller datasets and fosters a better cooperation with humans.

Keywords: Process mining · Remaining time prediction · Bayesian neural networks · Concrete dropout · Uncertainty · Heteroscedasticity · Convolutional neural networks · Long short-term memory models

1 Introduction

Modern information systems and data availability led to the acceleration of process mining research and deployment of its algorithms in industry in recent years. Process mining analyzes event data generated by such information systems with the goal of process discovery, process conformance checking and process enhancement. Predictive process monitoring is an important sub-field of process mining and concerns predicting next events, process outcomes and remaining execution times. Recent advances in machine learning propelled predictive process monitoring to the next level and many researchers intensified the use of artificial neural networks (NNs) for their predictions.

However, the adoption of these powerful and versatile NNs has not followed suit in practice. Practitioners are reluctant to use NNs that cannot explain their

A. Polyvyanyy et al. (Eds.): BPM 2021, LNCS 12875, pp. 141–157, 2021.
https://doi.org/10.1007/978-3-030-85469-0_11

predictions. A related, consequential problem is that NNs are unaware of the uncertainty of their predictions. They will always make a prediction, even when confronted with inputs they were never trained on. This can lead to potentially expensive or even catastrophic mistakes. Uncertainty awareness would therefore be a tremendous asset.

The uncertainty of predictions is the subject of this paper. Our core contribution is the introduction of NN-based uncertainty estimation techniques including heteroscedasticity learning and loss attenuation, concrete dropout and Bayesian neural networks (BNNs) to predictive process monitoring. We test their impact on overall prediction quality, uncertainty estimation quality, and computational time in a carefully designed experimental assessment using three public real-life datasets. Furthermore, we shed light on the practical applications. We consider the problem of remaining execution time prediction of ongoing processes which is highly relevant in practice, as it allows management to stop or alter running processes or initiate other actions. For example, an organization can inform its customers about the expected feedback/fulfillment time for their requests/orders and divert cases with long expected remaining times to a special track to speed them up.

We define our learning problem and position this paper relative to other work in Sect. 2. Section 3 explains two types of uncertainty before introducing techniques adapting plain-vanilla NNs to learn them. We then derive the precise questions we seek to address with our experiments. Section 4 describes the setup of these experiments, whose results are presented in Sect. 5. We subsequently present applications enabled by the uncertainty estimates in Sect. 6 before summarizing our findings and formulating paths for future research in the final Sect. 7.

2 Remaining Time Prediction: Definition and Related Work

In predictive process monitoring, datasets are event logs describing processes, often called *cases*. These cases consist of *events*. A number of attributes, also called *features* or variables, describe these cases and events. In remaining time prediction problems, every event is associated with a *target* feature describing the remaining time until completion of the case. A *prefix* is an ongoing, incomplete case, with the *prefix length* its number of completed events. Our learning problem is to train a learner using a training dataset containing events, described by their features and organized in prefixes that are labeled with targets, with the goal of predicting the targets of unseen prefixes.

In 2008, the first published research on process remaining time prediction [1] used non-parametric regressions, followed a few years later by [2] proposing to build an annotated transition system. Later, increasingly sophisticated approaches [3] deployed classic machine learning techniques such as support vector regression and naive Bayes and included the events' attributes other than activity name and time into their calculations. Recently, long short-term memory models (LSTMs) entered the scene [4,5]. Such deep learning techniques permit the substitution of automatic feature engineering for the error-prone, domain-knowledge-based manual feature engineering of the classic machine learning techniques. The authors of [6] provide an overview of papers until 2017. Our paper further extends this line of research by complementing the point estimates of these NN with predictions of the respective uncertainty. As such, we realize our goal of not only improving the overall quality of these point estimates, but also of unlocking many applications based on the knowledge of the predictions' uncertainty.

3 Estimating Uncertainty

In the context of predicting with models, we can distinguish two kinds of uncertainty [7]. The first, the *epistemic* (a.k.a. reducible) uncertainty expresses the model's uncertainty and finds its origin in the paucity of training data. Adding more samples to the training dataset will reduce the epistemic uncertainty. The first two graphs in Fig. 1 visualize two examples. The second type of uncertainty, the *aleatoric uncertainty* is a measure for the observation noise of the underlying distribution that generated the samples. It is often expressed as σ and will not decrease by observing more data. Many models in practice assume the aleatoric noise to be constant or homoscedastic (as in the third graph in Fig. 1). In reality, heteroscedasticity (fourth graph in Fig. 1) is probably much more common: the aleatoric noise varies across the domain.

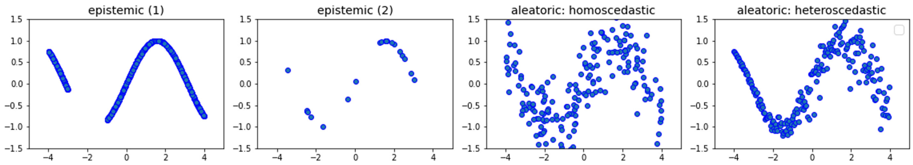

Fig. 1. Examples of uncertainty types

3.1 Estimating Epistemic Uncertainty with Bayesian Neural Networks

In regular, *deterministic* neural networks, the maximum likelihood estimate (MLE) of a model $\mathcal{H}$'s weights $\boldsymbol{\omega}$ maximizes the probability $p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{\omega}, \mathcal{H})$ of the observed outcomes $\boldsymbol{Y}$ given corresponding inputs $\boldsymbol{X}$. Prediction leads to a point estimate $y^* = \mathcal{H}(\boldsymbol{x}^*, \boldsymbol{\omega})$. Whilst good function approximators, (unregularized) deterministic NNs are prone to overfitting, especially when dealing with small training sets, and therefore struggle dealing with points $\boldsymbol{x}^*$ far away from the training data $\boldsymbol{X}$. Deterministic models have no knowledge of their point predictions' uncertainty.

The Bayesian approach is *stochastic* by nature: we look for the maximum a posteriori (MAP) distribution of the weights $\boldsymbol{\omega}$ given the training set $[\boldsymbol{X}, \boldsymbol{Y}]$, that can be expressed using the Bayesian rule:

$$p(\boldsymbol{\omega}|\boldsymbol{X}, \boldsymbol{Y}, \mathcal{H}) = \frac{p(\boldsymbol{Y}|\boldsymbol{X}, \boldsymbol{w}, \mathcal{H}).p(\boldsymbol{\omega}|\mathcal{H})}{p(\boldsymbol{Y}, \boldsymbol{X})} \text{ or posterior} = \frac{\text{likelihood x prior}}{\text{evidence}}$$

Note that the likelihood equals the MLE problem above. To predict the outcome for a given $\boldsymbol{x}^*$, we marginalize the likelihood over $\boldsymbol{\omega}$, a process called *inference* ($\mathcal{H}$ dropped to simplify notation):

$$p(y^*|\boldsymbol{x^*}, \boldsymbol{X}, \boldsymbol{Y}) = \int p(y^*|\boldsymbol{x}^*, \boldsymbol{\omega}).p(\boldsymbol{\omega}|\boldsymbol{X}, \boldsymbol{Y}).d\boldsymbol{\omega} \tag{1}$$

This is no longer a point estimate, but rather a distribution from which moments (mean, variance, etc.) can be derived. These statistics provide both a point estimate (mean) and a measure of the uncertainty of that estimate (variance), opening a range of possibilities that will be the subject of this paper. Under certain assumptions, there is an analytical solution to compute the posterior $p(\boldsymbol{\omega}|\boldsymbol{X}, \boldsymbol{Y})$ [8] but it is prohibitively computationally-expensive, as would be Markov Chain Monte Carlo sampling. Consequently, we resort to seeking a closed, approximate function $q_\theta(\boldsymbol{\omega})$ over the same domain $\boldsymbol{\omega}$ and parameterized by θ. This can be achieved by minimizing the Kullback-Leibler (KL) divergence between the two distributions:

$$\min KL\left(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega}|\boldsymbol{X}, \boldsymbol{Y})\right) = \int q_\theta(\boldsymbol{\omega}).\log\frac{q_\theta(\boldsymbol{\omega})}{p(\boldsymbol{\omega}|\boldsymbol{X}, \boldsymbol{Y})}$$

After some mathematical manipulations, the minimization problem above is equivalent to maximizing the *evidence lower bound* (ELBO):

$$\text{ELBO} = \mathbf{E}_{q_\theta(\omega)}\log p(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{\omega}) - KL\left(q_\theta(\boldsymbol{\omega})||p(\boldsymbol{\omega})\right) = \boxed{1} - \boxed{2} \tag{2}$$

Maximizing $\boxed{1}$ is the standard MLE approach with $\boxed{2}$ acting as a regularizer keeping the approximative posterior $q_\theta(\boldsymbol{\omega})$ as closely as possible to the prior $p(\boldsymbol{\omega})$. Unlike $\boxed{2}$, the (derivative of) $\boxed{1}$ cannot be computed in closed form. Since the density function $q_\theta(\boldsymbol{\omega})$ in $\frac{\partial}{\partial\theta}\int q_\theta(\boldsymbol{\omega})\log p(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{\omega}).d\boldsymbol{\omega}$ itself depends on θ, regular Monte Carlo (MC) integration is not feasible either. [9] proposes to use the so called *reparameterization trick* [10] to solve $\frac{\partial}{\partial\theta}\int q_\theta(\boldsymbol{\omega})\log p(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{\omega}).d\boldsymbol{\omega}$. It involves expressing $\boldsymbol{\omega}$ as a deterministic function $g(\epsilon, \theta)$ in which ϵ is a unconditional parameter, allowing to sample ϵ from $\mathcal{N}(0, I)$ rather than sampling $\boldsymbol{\omega}$ from $q_\theta(\boldsymbol{\omega})$. The above approach is called *stochastic variational inference.* Often, a Gaussian distribution is placed over every weight ω in the network with $\omega = g(\epsilon, \theta) = \mu + \sigma.\epsilon$. This method has two serious drawbacks: it doubles the number of parameters to be estimated (μ and σ instead of a single ω for every node) and requires relatively complex coding.

Dropout [11] is a popular regularization technique to prevent NNs from overfitting. It resembles training a large number of networks in parallel by dropping out, or randomly ignoring the outputs of nodes (including the network's inputs) during training by multiplying each one by a parameter ϵ sampled from a Bernoulli distribution with probability p. By simply transforming this stochasticity from the feature space in the NNs' dropout scenario to the weight space in BNNs, maximizing ELBO equals minimizing the NNs' dropout loss function with an additional L2 regularizer [12]. We are, thus, able to use standard NNs with easy-to-implement dropout regularization as BNNs, overcoming the drawbacks aforementioned. *Concrete dropout* [13] eliminates the need for tuning the dropout parameters p_i (for each layer i) by automatically optimizing p_i, replacing the discrete Bernoulli distribution with a continuous relaxation (concrete distribution relaxation [14]). In the traditional approach, dropout layers are placed between the convolutional layers in CNNs and only after the inputs and the last LSTM layer in LSTMs. This traditional approach leads to unsatisfying results. In our BNNs (we used both LSTMs and CNNs, see Subsect. 3.3), we therefore applied dropout to the inner-product layers (kernels) [15] in CNNs and to all eight weight matrices within the LSTM cells [16] which reduces overfitting problems more successfully.

After training the model as described above, we proceed to inference or prediction by using MC sampling again, performing T stochastic forward passes of our trained model. The predictive mean of Eq. 1 is estimated by the predictive mean of the MC samples:

$$\mathbf{E}_{p(y^*|\boldsymbol{x}^*,\boldsymbol{X},\boldsymbol{Y})}[y^*] \approx \frac{1}{T}\sum_t \mathcal{H}(\boldsymbol{x}^*, \hat{\boldsymbol{\omega}}) \tag{3}$$

with $\hat{\boldsymbol{\omega}}$ indirectly sampled from $q_\theta(\boldsymbol{\omega})$ by sampling ϵ from $\mathcal{N}(0, I)$. The variance is given by:

$$\mathrm{Var}_{p(y^*|x^*,\boldsymbol{X},\boldsymbol{Y})}[y^*] \approx \sigma^2 + \frac{1}{T}\sum_t \mathcal{H}(\boldsymbol{x}^*, \hat{\boldsymbol{\omega}})^2 - \left(\frac{1}{T}\sum_t \mathcal{H}(\boldsymbol{x}^*, \hat{\boldsymbol{\omega}})\right)^2 = \sigma^2 + \boxed{3} \tag{4}$$

$\boxed{3}$ is the sample variance of the T stochastic forward passes and can be interpreted as the model's or epistemic uncertainty. Adding more samples to the training dataset will reduce it. Hence, BNNs enable the ability to gauge the model's uncertainty for every prediction made.

3.2 Estimating Heteroscedastic Aleatoric Uncertainty

The σ in the above Eq. 4 is the aleatoric uncertainty. As most models assume σ to be constant, or homoscedastic, over the entire domain, they do not include it in their loss functions (the last term in Eq. 5 is simply dropped). However, learning an individual σ_n for each sample n would be valuable to better assess the variance of our predictions in Eq. 4. This is achieved by doubling the last dense layer in the model (unsupervised learning) [7]. By re-completing the loss function (ignoring the regulation term) to include the learned σ_n:

$$L = \min \frac{1}{N}\sum \frac{1}{2\sigma_n^2}(\boldsymbol{y}_n - \mathcal{H}(\boldsymbol{x}_n))^2 + \frac{1}{2}\log\sigma_n^2 \tag{5}$$

it becomes less sensitive to noisy data, as it will predict high uncertainty for poor predictions and vice versa. This process is called *loss attenuation* and should lead to better overall predictions. The second term in Eq. 5 ensures that the model does not simply predict high uncertainty for every sample.

3.3 LSTM Vs. CNN

The techniques described above all depend on the underlying NNs. LSTMs [17] have been the intuitive instrument of choice in predictive process monitoring problems. An LSTM processes every sequence of events it is presented one time step at a time. At any given time step, it will pass a vector containing information about the current and previous time steps to the next time step, until reaching the last one whose output is propagated to the next layer. In contrast, convolutional neural networks (CNN) [18] work with fixed-sized, spatially-organized data. A series of alternating convolution layers applying weight-sharing filters and dimension-reducing pooling layers enables the models to automatically recognize patterns and extract features from the input data. These features are then passed to a series of dense layers for the final regression. Interpreting time as a spatial dimension, one-dimensional CNNs can be successfully applied to sequence processing as well, as a growing body of research (e.g. [19]) points out. This thesis is supported by [20] for the related case of process outcome prediction. We, therefore, ran our experiments using both CNNs and LSTMs to gain further insight into the applicability of both models.

3.4 Objectives

Equipped with this understanding, we can now translate our research goal of investigating uncertainty for remaining time prediction into more detailed objectives. First, we assess the effect on the overall quality of point estimates of the following techniques (Subsect. 5.1):

1. **Heteroscedasticity**: Estimating the observation loss for individual samples (σ_n) permits loss attenuation. Can it improve point estimates?
2. **Dropout**: BNNs resemble NNs with dropout regularization. What are the merits of isolated dropout in a non-Bayesian context?
3. **Concrete dropout**: allows in-model estimating the dropout parameters p_i. How does it affect results?
4. **BNN**: Using the heteroscedastic NNs with concrete dropout, we apply MC sampling (T stochastic forward passes) and average to calculate point estimates (Eq. 3). Do we get better predictions?
5. **CNN/LSTM/base case**: We compare CNNs to LSTMs, as well as to a baseline to get an intuition for the absolute performances.

From the theory, we expect each of the first four techniques to contribute to better point estimates. CNNs should produce results at least at par with LSTMs.

Second, we investigate whether the uncertainty estimates' succeed in separating good from bad predictions and in building reliable confidence intervals based on these uncertainty estimates (Subsect. 5.2) as is theoretically expected. Third (Subsect. 5.3), we wish to gain insights in the computation time for training and inference respectively. Finally, our fourth objective (Sect. 6) is to explore and assess applications stemming from the knowledge of predictions' uncertainties.

4 Experimental Setup

4.1 Datasets

We used three publicly available datasets from the BPI Challenges[1]. BPIC_2017[2] is a rich and large dataset containing logs of a loan application process at a Dutch bank. BPIC_2019[3], while comparable in size, has much shorter cases and concerns a purchase order handling process. BPIC_2020[4] is a collection of five smaller datasets related to travel administration at a university. The five subsets are records of processes covering international declaration documents (Intl. Declarations), expense claims (Travel Costs), travel permits (Permits), pre-paid travel costs and requests for payment (Payments) and domestic declaration documents (Domestic Declarations). Our target for all these datasets was defined as the fractional number of days until case completion.

4.2 Preprocessing

To maintain a realistic setting, we refrained from filtering. Other than adding a few synthetic features based on the event time stamps (e.g. event number, elapsed time since previous event, day of the week, ...), we did not apply any domain knowledge whatsoever to our approach. The chronologically 15% last starting cases (10% for BPIC_2020) were withheld as a test dataset. Since the duration of a case is only known at its end (when the process is finished), we deleted all cases from the remaining training set that ended after the start of the first test dataset case[5]. This left us with approximately two thirds of the original cases for BPIC_2017 and BPIC_2019. Given the shorter recording time frame for BPIC_2020, this approach drastically reduced the number of samples for training, especially where cases take longer (Intl. Declarations is only left with 57 events from five cases in the training set). With longer cases (with more deviations) and more levels for the categorical variables, BPIC_2017 differs significantly from BPIC_2019. To add further variety, we worked with more features in BPIC_2017 (10) than in BPIC_2019 (5). To observe how results depend on the training set size, we performed our experiments on different shares of the available training

[1] https://data.4tu.nl (4TU Centre for Research Data).

[2] https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884.

[3] https://data.4tu.nl/articles/dataset/BPI_Challenge_2019/12715853.

[4] https://data.4tu.nl/collections/BPI_Challenge_2020/5065541.

[5] A theoretical possibility of data leakage remains. In reality, some case variables such as "Amount" are possibly unknown at the beginning of the case, even though every event log has a value for them.

samples for both large datasets (keeping the same test sets), ranging from 0.1% to 100%. Table 1 shows the respective datasets' key statistics and illustrates their differences.

Table 1. Statistics of the used datasets.

Dataset	Avg. case length	Share of events used	Training events	Validation events	Test events	Range features	Categorical features	Levels
BPIC_2017	38.5	.001	629	220	181, 189	5	5	113
		.002	1,286	363				
		...	...	...				
		.5	327,959	79,190				
		1	655,271	159,306				
BPIC_2019	5.2	.001	625	192	162, 753	3	2	18
		.002	1,263	341				
		...	...	...				
		.5	328,994	85,622				
		1	657,187	171,724				
Intl. declarations	29.6	1	57	20	4, 416	3	3	18
Travel costs	7.7	1	1,706	412	1, 652	5	9	74
Permits	10.0	1	8,030	2,132	6, 537	5	9	94
Payments	5.3	1	21,049	5,743	3, 746	4	8	107
Domestic declarations	8.1	1	23,434	6,216	3, 533	4	6	66

Range features were standardized. The number of levels of categorical variables was not clipped (non-frequent labels may be a reason for uncertain estimates). The labels were mapped to integers that were then passed to an embedding layer in the neural networks. All possible prefixes were derived from the cases and then standardized to a pre-determined *sequence length* by padding the shorter and truncating the longer ones. All experiments were coded in Python/Pytorch and ran on a desktop with a 3.50 Ghz CPU, 64 Gb of RAM and GeForce 1080 GPU. Our code is published on GitHub[6] for reproducibility. The metric used was the mean absolute error (MAE).

4.3 Estimating the Epistemic, Aleatoric and Total Uncertainty

In the case of BNNs, we performed $T = 50$ stochastic forward passes (MC sampling) for every prefix in the test set, each time with a different mask over the weights, by sampling a different ϵ for every ω at every run (as per Eq. 3). The final predictions are the averages over these 50 samples, discussed in Sect. 5. Using their variance, we calculated the model's uncertainty, i.e. the epistemic uncertainty, for every prediction in the test set using $\boxed{3}$ in Eq. 4. Moreover, we computed the per-point aleatoric uncertainty in an additional final dense layer in the models and included it in the loss function as in Eq. 5. We added together both types of uncertainty to calculate the total uncertainties used in Subsect. 5.2 and Sect. 6. All predictions in the following are averages of 20 runs of the respective models.

[6] https://github.com/hansweytjens/uncertainty-remaining_time.

4.4 Base Case

Despite the widespread use of public datasets in predictive process monitoring, assessing the quality of different methods remains hard as the filtering of the datasets, other preprocessing steps, model architectures, etc. are far from uniform across papers. Furthermore, the metrics used allow for comparisons of the methods within a paper but fail to convey an intuition about their absolute merits. To remedy the latter, we included the transition system-based method [2] as a baseline in our experiments.

5 Results

5.1 Overall Performance

We investigated whether the techniques in Subsect. 3.4 contribute to achieving more accurate point estimates. The results are summarized in Fig. 2 in which every row pertains to a dataset (BPIC_2017, BPIC_2019, BPIC_2020 respectively). Every column compares two or more techniques and will be discussed in the five following subsections. The horizontal axis in the graphs for BPIC_2017 and BPIC_2019 represents the share of the available training set that was used for training, ranked from small to large. In the last row, however, it is the five sub-datasets that are ranked from small to large. The vertical axis represents the models' MAE, with the scale being shared throughout the respective rows. Note that we normalized the MAE in the last row, with the respective base cases equal to one.

Loss Attenuation Inconclusive (Fig. 2: Column 1). We found no evidence in our experiments for the theoretically-derived hypothesis that learning the heteroscedastic uncertainty and using it by introducing loss attenuation (Eq. 5) in the loss functions leads to more accurate predictions. The black lines in Fig. 2 represent the plain-vanilla NNs, whereas the cyan lines stand for models including the technique. Results on BPIC_2017 and BPIC_2020 significantly worsened. Only in the case of BPIC_2019 did the technique lower MAE. Two effects could explain that. First, the added complexity may require larger datasets. Second, for datasets with rather homoscedastic aleatoric noise, or datasets with a rather randomly distributed heteroscedastic aleatoric noise, one cannot expect superior results from introducing loss attenuation. We did not further investigate this matter. Nevertheless, learning heteroscedastic uncertainty is indispensable for judging the quality of predictions. We will treat this in Subsect. 5.2.

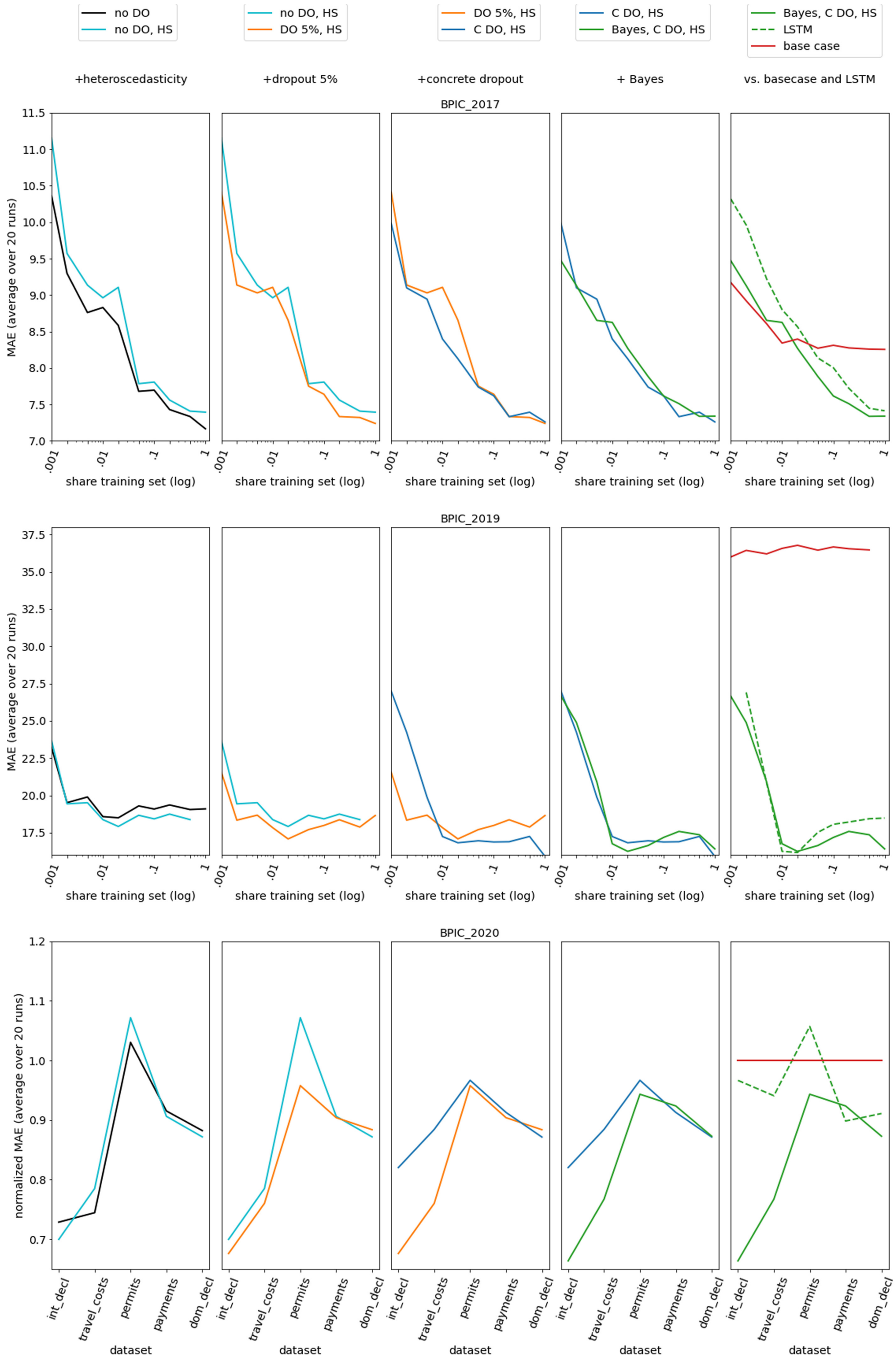

Fig. 2. Overall results on complete test sets. no DO: no dropout = plain-vanilla NN, HS: heteroscedastic, DO 5%: 5% dropout probability, C DO: concrete dropout, Bayes: BNN. Rows show three datasets, stepwise different techniques in columns. BPIC_2020 results normalized with base case = 1.

Dropout Effectively Combats Overfitting (Fig. 2: Column 2). The heteroscedastic models are again represented by the cyan lines in column 2. They already included an early-stopping mechanism. But, since the validation sets were in certain cases very small, and some concept drift may exist in the datasets, some overfitting still happened. In line with expectations, the dropout mechanism (orange lines) successfully further reduced overfitting on practically all datasets and training set sizes.

Concrete Dropout Works for Medium to Large Datasets (Fig. 2: Column 3). When comparing classic dropout with a fixed dropout parameter (orange lines) to concrete dropout (blue lines), our experiments suggest that, for some very small to small datasets (BPIC_2019 $<$ 1%, BPIC_2020), concrete dropout negatively affects the overall quality of the predictions. For all other datasets, concrete dropout appeared to work or even improve results as expected. The use of concrete dropout also eliminates the need for the expensive optimization of the dropout parameter(s) $p_{(i)}$ that requires part of the training set to be set aside as a validation set.

Bayesian Learning Improves Results for Very Small Datasets (Fig. 2: Column 4). Until now, we used deterministic NNs to arrive at such models using concrete dropout (blue lines). In column 4, we introduce stochastic NNs in the form of BNNs (green lines), that predict distributions of which the arithmetic averages yield point estimates. BPIC_2017 and especially BPIC_2020 support the claim that BNNs produce superior results for smaller datasets. For larger datasets, the effect is negligible, possibly slightly negative. As explained in Sect. 3, the variance of the produced distributions can be interpreted as a measure for the models' (epistemic) uncertainty, a property we use below. As mentioned in Sect. 3, BNNs by default add L2 regularization to the dropout models. Since the combination of these regularization techniques (in our case even with early-stopping on top) makes these models so robust to overfitting, it is recommended to build models with large capacity to avoid underspecification and train them sufficiently long.

CNNs Outperform LSTMs, BNNs Outperform the Base Cases (Fig. 2: Column 5). The models in columns 1–4 were all CNNs. When comparing the last one (BNN, full green line) with an otherwise identical LSTM model (dotted green line), it becomes apparent that the CNNs nearly always outperformed the LSTMs. Of course, the chosen architectures (number of layers, nodes, etc.) influenced these outcomes, but the results support similar findings in [19,20]. Unless otherwise mentioned, we will use these heteroscedastic Bayesian CNNs with concrete dropout in the remainder of this paper and simply refer to them as BNN. With the exception of shares of less than 2% of the BPIC_2017 dataset and of the BPIC_2019 Permits dataset, the BNNs outperformed the base cases.

5.2 Uncertainty Estimates

We analyze the quality of the total uncertainty estimates, focusing on their correlation with the quality of the predictions and on the reliability of confidence intervals based on them.

Certainty of Predictions Correlates Strongly with Accuracy. We ranked the predictions in the test set and then retained different shares of the predictions while rejecting the others for different uncertainty thresholds (100%, 75%, 50%, 25%, 10%, 5% best). Figure 3 shows how well this worked for all datasets and dataset sizes: higher uncertainty led to worse predictions, without fail. Unfortunately, the quality of the uncertainty estimates suffered together with the quality of the predictions when datasets became too small, thus also reducing the possibility to separate good from bad predictions as can be witnessed at the left end of the graphs in Fig. 3.

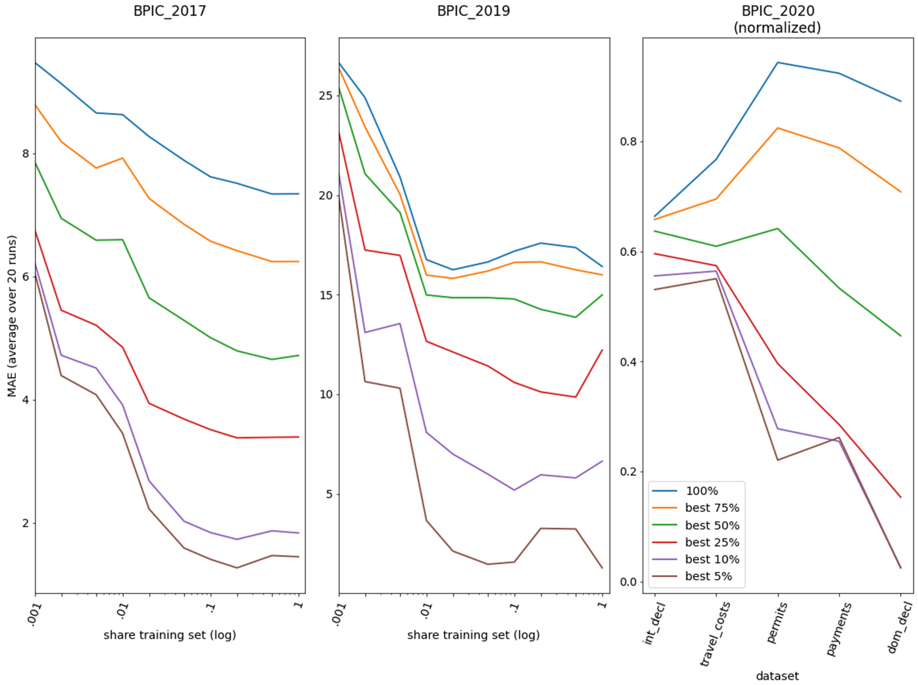

Fig. 3. We ranked the samples in the test sets based on the sum of the predicted epistemic and aleatoric uncertainties. In all three datasets, we observe lower MAE (better predictions) for lower levels of uncertainty. We used BNNs with concrete dropout and heteroscedasticity.

Predictions with Confidence Intervals. To build a confidence interval around a point estimate, the product of a so called *critical value* (z^* in statistics) and the uncertainty is added/subtracted to/from that point estimate to determine the upper/lower bound of the confidence interval. For each desired confidence level (50%, 75%, 90%, 95%, 99%) we computed the required critical value based on the last 5,000 samples in the training set. Since the BPIC_2017 dataset exhibits drift (changes over time), it did not suffice to determine these critical values only once: they had to be calculated online, as can be seen in the left part of Fig. 4. In the right part of Fig. 4, the real shares of true values in the respective confidence intervals are shown. They oscillate around their ideal values (horizontal lines), proving their reliability.

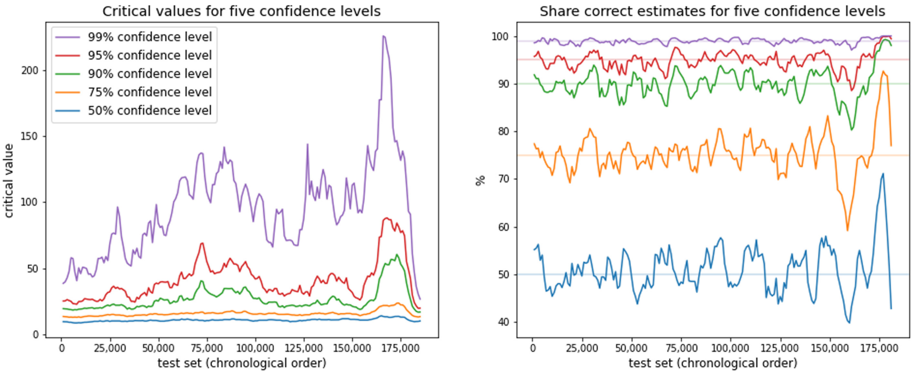

Fig. 4. Left: critical values for confidence levels of 50%, 75%, 90%, 95% and 99% computed on 5,000 preceding samples every 1,000th sample in the test set. Right: Corresponding share of true values in following 5,000 samples within the confidence interval. Dataset is BPIC_2017 (complete).

5.3 Computation Time

BNNs Train and Predict Relatively Fast. To gain an insight in the computation time of BNNs, we disabled the early stopping mechanism and trained the models for 20 epochs on the complete BPIC_2017 training set. Training the BNNs took around 335 s, approximately 38% more than the corresponding plain-vanilla deterministic models' 242 s. As inference requires MC sampling (we performed 50 MC forward passes), BNN predictions took longer (32 vs 0.65 s for all 181,189 test set points). Whilst in most settings the inference time is low enough to ignore, this may not be the case in certain online environments requiring near-instantaneous decisions.

Compared to plain-vanilla, deterministic models, the BNNs' hyperparameter space is definitely of a lower dimensionality. There is no need to determine values for the dropout parameter(s) p_i (assuming concrete dropout), model size (we can safely use large-capacity BNNs), number of epochs trained, etc. This may turn their small speed disadvantage into a considerable advantage.

CNNs Outspeed LSTMs. As already observed in previous work [20], CNNs train nearly an order of magnitude faster than LSTMs requiring non-parallelizable sequential calculations. The custom coding to implement dropout within the LSTM cells prevented us from using the very efficient standard PyTorch neural network libraries we used for the CNNs. As a result, our LSTM models slowed down even further and kept us from publishing a fair speed comparison in our specific setting.

6 Applications of Uncertainty

The knowledge of a prediction's quality opens the door to useful practical applications:

Higher Accuracy and Acceptance of Prediction Systems. The previous section demonstrated how the techniques we introduced will generally lead to more accurate overall predictions. However, a yet much higher accuracy can be reached by concentrating on the most certain predictions. An organization requiring a given accuracy threshold can now deploy a prediction system that does not reach that threshold overall but that is aware which of its predictions are expected to surpass it. Predictions that do not reach the (un)certainty threshold can be ignored or passed to humans or another system. In summary, not only can models produce better predictions, but they will also flag potentially incorrect, absurd or even dangerous predictions.

Improved Human-Machine Symbiosis. The ability to isolate inaccurate predictions permits two-track systems. Cases with good predictions remain on the automated track. Cases with predictions below an uncertainty threshold are passed to the human track. These latter cases will generally be the hardest to solve, more irregular, more interesting ones which could lead to more satisfying work for the involved humans and a better leverage of their cognitive faculties.

Working with Smaller Datasets: Earlier Adoption of Prediction Systems. As Figs. 2 and 3 show, the lack of data often leads to underperforming predictions systems. Organizations will not deploy them or delay their adoption until they feel their dataset is large enough. This may lead to a competitive disadvantage in this digital era requiring rapid innovations, speedy implementation and constant learning where waiting for perfection is no longer an option. The ability to identify predictions that meet a pre-set uncertainty threshold allows for a much faster adoption of prediction systems. Originally, only a relatively small share of the best predictions is actually used. But as the dataset grows, that share continually increases. During this phase-in period, the organization will gain invaluable information to further improve its systems and data collection otherwise lost when remaining on the sidelines.

Uncertainty-Based Analysis. The estimates of the predictions' uncertainty enables further analysis. For example, as in Fig. 5, we can plot the test set uncertainty in function of the prefix length and the real number of remaining days (unknown to the model). Given their high aleatoric uncertainty, the model is rightfully very uncertain about the prefixes of length one (first column). The model clearly gains in confidence when prefixes get longer, at least for the most common remaining time lengths (lower than 4 days, lowest four rows). When prefixes start getting longer than six events, the model becomes increasingly wary of its predictions again. Indeed, parts of the domain with fewer samples (e.g. prefix length > six events, real remaining time > 50 days) should have a higher epistemic, and hence total uncertainty. Outliers, such as the confident predictions of prefixes with length five or those in the second row (10–19 days) of prefix length one, deserve closer attention and may lead to interesting insights. Of course, the uncertainty can be plotted against any other feature as well. A detailed analysis falls outside this paper's scope.

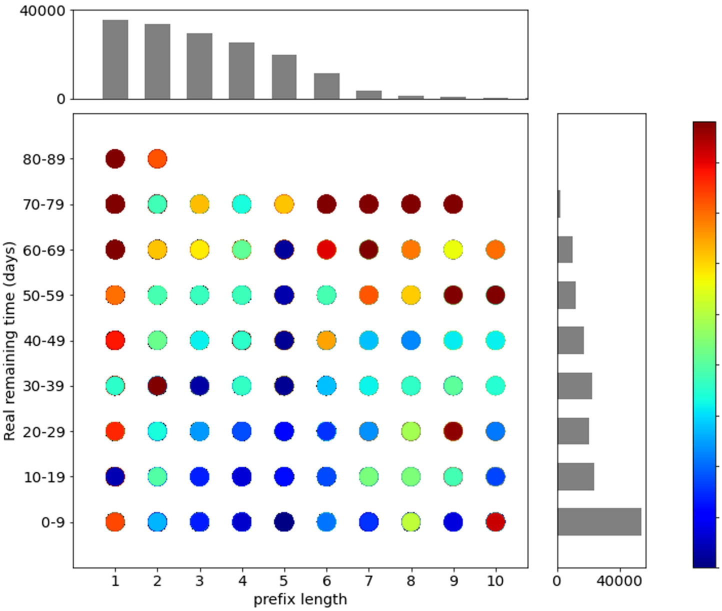

Fig. 5. BPIC_2019, 20% of training set: Uncertainty (blue = low, red = high) in function of prefix length and real number of remaining days. Grey bars indicate frequency of occurence. Prefix length cut of at 10, corresponding to >99% of samples in test set. (Color figure online)

7 Conclusion and Future Work

The stochastic Bayesian approach leads neural networks to predict distributions rather than point estimates. These distributions can be used both to derive more precise point estimates (mean) and to estimate the model's epistemic uncertainty (variance). It can be proven that BNNs are nearly identical to deterministic NNs with dropout, which makes them easy to implement. Concrete dropout renders optimizing the dropout parameters p_i obsolete. A dataset's heteroscedastic aleatoric noise can be learned in-model by means of a simple modification to the model and its loss function (loss attenuation). Whilst inconclusive on the benefits of loss attenuation, this paper shows how dropout, concrete dropout and BNNs generally contribute to more accurate remaining time predictions. CNNs prove to work better and faster than LSTMs. Not all of these techniques work well on all datasets: small datasets pose problems for concrete dropout while they benefit from the Bayesian models that themselves add no value with larger datasets. The presented techniques require little extra coding, learn nearly as fast and are less data-hungry than corresponding regular neural networks. Rather than improving overall accuracy, however, the main benefits of learning uncertainty reside with the new options this knowledge enables. Users can set thresholds to retain those predictions that meet any required accuracy, build confidence intervals around predictions, divide cases between computers and humans in a clever way, adopt prediction models earlier before huge datasets are collected, gain additional insights e.g. in the search for anomalies, etc. We hope that the techniques we proposed help remove some of the barriers that slow down or prevent the adoption of neural networks and could help to extract more value from information systems.

This new field of research can be extended in a variety of ways. First, the validity of our results should be tested on a diverse range of datasets to reach more general conclusions. Also other predictive process monitoring regression and classification problems are logical extensions. Dropout is not the only option to implement variational inference, other methods could be tested as well and may have other characteristics. We also believe that the knowledge of uncertainties can lead to more applications than the ones here presented. As we only concentrated on the total uncertainty, evaluating the respective merits of epistemic and aleatoric uncertainty constitutes another path for future research.

References

1. van Dongen, B.F., Crooy, R.A., van der Aalst, W.M.P.: Cycle time prediction: when will this case finally be finished? In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5331, pp. 319–336. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88871-0_22
2. Van der Aalst, W.M.P., Schonenberg, M.H., Song, M.: Time prediction based on process mining. Inf. Syst. **36**, 450–475 (2011)
3. Polato, M., Sperduti, A., Burattin, A., de Leoni, M.: Data-aware remaining time prediction of business process Instances. Presented at the (2014)

4. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. Lecture Notes Computer Science, vol. 10253, pp. 477–492 (2017)
5. Navarin, N., Vincenzi, B., Polato, M., Sperduti, A.: LSTM networks for data-aware remaining time prediction of business process instances. arXiv:1711.03822v1 (2017)
6. Verenich, I., Dumas, M., La Rosa, M., Maggi, F.M., Teinemaa, I.: Survey and cross-benchmark comparison of remaining time prediction methods in business process monitoring. ACM Trans. Intell. Syst. Technol. (TIST) **10**(4), 1–34 (2019)
7. Kendall, A., Gal, Y.: What uncertainties do we need in Bayesian deep learning for computer vision? Presented at the (2017)
8. MacKay, D.: Bayesian methods for neural networks: theory and applications. In: Neural Networks Summer School. University of Cambridge (1995)
9. Gal, Y.: Uncertainty in deep learning: PhD thesis. University of Cambridge (2016). http://mlg.eng.cam.ac.uk/yarin/thesis/thesis.pdf
10. Kingma, D.P., Welling, M.: Auto-encoding variational Bayes. arXiv:1312.6114 (2014)
11. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014)
12. Gal, Y., Ghahramani, Z.: Dropout as a Bayesian approximation: representing model uncertainty in deep learning. In: Proceedings of the 33rd International Conference on Machine Learning, vol. 48 (2016)
13. Gal, Y., Hron, J., Kendall, A.: Concrete dropout. Presented at the (2017)
14. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: a continuous relaxation of discrete random variables arXiv:1611.00712v3 (2017)
15. Gal, Y., Ghahramani, Z.: Bayesian convolutional neural networks with Bernoulli approximate variational inference. arXiv:1506.02158v1 (2015)
16. Gal, Y., Ghahramani, Z.: A theoretically grounded application of dropout in recurrent neural networks. Presented at the (2016)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
18. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Presented at the (1998)
19. Bai, S.J., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271v2 (2018)
20. Weytjens, H., De Weerdt, J.: Process outcome prediction: CNN vs. LSTM (with attention). Presented at the (2020)