Frequentist Uncertainty in Recurrent Neural Networks via *Blockwise* Influence Functions

Ahmed M. Alaa [1] **Mihaela van der Schaar** [1 2]

Abstract

Recurrent neural networks (RNNs) are instrumental in modelling sequential and time-series data. Yet, when using RNNs to inform decision-making, predictions by themselves are not sufficient — we also need estimates of predictive *uncertainty*. Existing approaches for uncertainty quantification in RNNs are based predominantly on Bayesian methods; these are computationally prohibitive, and require major alterations to the RNN architecture and training. Capitalizing on ideas from classical jackknife resampling, we develop a *frequentist* alternative that: (a) does not interfere with model training or compromise its accuracy, (b) applies to *any* RNN architecture, and (c) provides theoretical coverage guarantees on the estimated uncertainty intervals. Our method derives predictive uncertainty from the variability of the (jackknife) sampling distribution of the RNN outputs, which is estimated by repeatedly deleting "blocks" of (temporally-correlated) training data, and collecting the predictions of the RNN re-trained on the remaining data. To avoid exhaustive re-training, we utilize *influence functions* to estimate the effect of removing training data blocks on the learned RNN parameters. Using data from a critical care setting, we demonstrate the utility of uncertainty quantification in sequential decision-making.

1. Introduction

Recurrent neural networks (RNNs) are sequence-based models of key importance in a wide range of application domains (Sundermeyer et al., 2012; Kalchbrenner & Blunsom, 2013). Because of their ability to handle temporal sequences, RNNs have been used in applications that involve sequential decision making over time — examples of such applications include: stock market predictions (Bao et al., 2017; Selvin et al., 2017), service demand forecasting (Wen et al., 2017; Zhu & Laptev, 2017), medical prognoses (Shickel et al., 2017; Lim et al., 2018; Alaa & van der Schaar, 2019) and reinforcement learning (Bakker, 2002; Wang et al., 2018).

[1]University of California, Los Angeles [2]Cambridge University. Correspondence to: Ahmed M. Alaa <ahmedmalaa@ucla.edu>.

Proceedings of the 37th International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

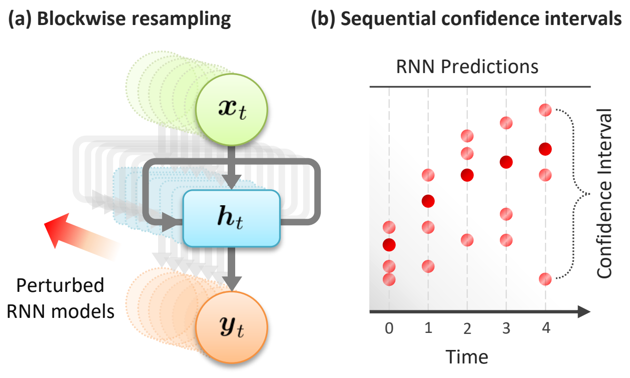

Figure 1. **Pictorial depiction of our approach.** (a) Blockwise resampling is applied to the RNN in a *post-hoc* fashion to create N perturbed versions of the model $\{\text{RNN}_1, \ldots, \text{RNN}_N\}$ (illustrated through the shaded diagrams) by repeatedly deleting N "blocks" of training data, and using *influence functions* to estimate the parameters of the perturbed models. Here, h_t is the RNN's hidden state, x_t is its input and y_t is its output, (b) Confidence intervals are constructed on the basis of the variability of the N perturbed RNNs' outputs (highlighted in light red color). Here, we set $N = 3$.

In high-stakes applications where RNNs would inform critical decisions, reliable estimates of *predictive uncertainty* are indispensable for accurate risk assessment (Gollier, 2018). For instance, optimal decisions on *when* to administer an aggressive treatment to a hospitalized patient (based on their temporal lab measurements) requires estimating the uncertainty in their predicted outcomes in order to properly weigh risks against benefits (Dusenberry et al., 2019). Whereas various methods for uncertainty estimation in feed-forward networks have been recently proposed (Lakshminarayanan et al., 2017; Malinin & Gales, 2018; Maddox et al., 2019), equivalent methods for RNN models are still lacking.

In this paper, we develop a principled *frequentist* approach for estimating uncertainty in the predictions of RNN models. (A high-level depiction of our approach is given in Figure 1.) Our method builds on classical *jackknife* resampling (Miller,

1974) to create multiple "perturbed" versions of an RNN by resampling its trained parameters (Figure 1.a). Through the RNN's forward pass, we estimate the sampling distribution of the model's predictions by collecting the outputs of its perturbed copies — confidence intervals on the model's predictions are then constructed based on the jackknife residuals and the variability of the resampled outputs (Figure1.b). The entire procedure is applied in a *post-hoc* fashion, hence it neither interferes with model training nor compromises its accuracy. Moreover, since the jackknife is model-agnostic, our approach covers a wide range of RNN architectures.

Because RNNs process (temporally-correlated) sequences, our jackknife resampling scheme operates on "blocks" of dependent data points instead of individual observations. That is, in order to resample the parameters $\hat{\boldsymbol{\theta}}_{\text{RNN}}$ of an RNN trained on a data set $\mathcal{D}$, we repeatedly delete temporally contiguous blocks of data from $\mathcal{D}$, and estimate the parameters of the perturbed version of the model — trained on the remaining data — through the following relationship:

$$\mathcal{I}(\text{Data Block}) \approx \hat{\boldsymbol{\theta}}_{\text{RNN}}(\mathcal{D} \setminus \text{Data Block}) - \hat{\boldsymbol{\theta}}_{\text{RNN}}(\mathcal{D}),$$

where $\mathcal{I}(.)$ is a *blockwise influence function* that quantifies the effect of removing a data block in $\mathcal{D}$ on the learned RNN parameters — this is a generalization of conventional influence functions defined with respect to individual data points (Hampel et al., 2011; Koh & Liang, 2017). In Section 4, we develop a stochastic estimation algorithm, based on blockwise batch sampling, to compute $\mathcal{I}(.)$ in linear time for RNN models trained via backpropagation through time (BPTT) or any of its variants. Conducting our blockwise resampling scheme requires only access to the model's loss gradients, and is independent of the model's specific architecture.

Using the "leave-one-block-out" training residuals and the variability of the RNN predictions — estimated via influence functions — we construct a sequence of confidence intervals that "sandwiches" the sequential predictions made by the RNN model. In Section 3, we extend the classical notion of *coverage probability* of confidence intervals, i.e., the likelihood that a confidence interval contains the true label (Marra & Wood, 2012), to the sequential prediction setup, and in Section 4, we show that under certain conditions, confidence intervals estimated through blockwise resampling can achieve any pre-specified coverage probability.

Finally, in Section 5, we demonstrate the utility of our proposed method in the context of sequential decision-making in critical care medicine using real-world data, in addition to synthetic simulations. More specifically, we show that uncertainty estimates generated by our method (on top of an RNN model) can inform clinicians on *when* antibiotics can be *safely* prescribed to hospitalized critically-ill patients so that improvements in their health outcome are maximized while taking into account uncertainties about side effects.

2. Related Work

Driven by the desire to suppress the "overconfidence" exhibited by poorly calibrated — but highly discriminative — deep learning models, various methods for uncertainty estimation in standard feed-forward neural networks have been recently proposed (Hernández-Lobato & Adams, 2015; Gal & Ghahramani, 2016a; Lakshminarayanan et al., 2017; Malinin & Gales, 2018). However, the literature on uncertainty quantification in RNNs is rather scarce. In what follows, we provide a brief glimpse at three strands of relevant literature; a more elaborate discussion on these methods, the history of jackknife resampling, and the connections between our work and modern applications of influence functions (Koh & Liang, 2017; Koh et al., 2019) is provided in Appendix A.

Bayesian RNNs. The most prevalent approaches for RNN uncertainty estimation hinge on Bayesian modeling (Fortunato et al., 2017; Chien & Ku, 2015; Zhu & Laptev, 2017; Mirikitani & Nikolaev, 2009). By specifying a prior over the RNN parameters, these approaches estimate a model's uncertainty via its posterior credible intervals. However, exact inference in Bayesian deep learning is generally intractable or computationally infeasible. Hence, practical Bayesian RNNs rely on dropout-based approximate inference (Gal & Ghahramani, 2016a;b), which is often hard to calibrate, and is generally ill-posed as its posterior distributions do not concentrate asymptotically (Osband, 2016; Hron et al., 2017). Moreover, Bayesian methods require significant modifications to model architecture and training which limits their versatility. Our paper addresses these issues by developing a frequentist post-hoc alternative to Bayesian methods.

Quantile RNNs. Similar to quantile regression (Koenker & Hallock, 2001), quantile RNNs (Q-RNNs) learn prediction intervals by explicitly modeling the cumulative distribution function of the prediction targets (Taylor, 2000; Wen et al., 2017; Gasthaus et al., 2019). The key difference between these approaches and ours is that Q-RNNs learn uncertainty jointly with the main prediction task, which as we show in Section 5, can compromise their predictive accuracy. Moreover, Q-RNNs are bespoke models that must be retailored for each new prediction task, whereas our approach can be applied in a post-hoc fashion to any RNN architecture. Empirical comparisons between our method and the 2 approaches discussed above are conducted in Section 5.

Probabilistic RNNs. This class of models capture the structural uncertainty within the stochastic dynamics of sequential data. Existing variants of probabilistic RNNs are mainly based on deep state-space models (Krishnan et al., 2017; Rangapuram et al., 2018; Alaa & van der Schaar, 2019). Probabilistic models are bespoke to the application at hand, and may not be appropriate for some application domains. Moreover, most probabilistic models entail restrictive assumptions on sequence dynamics (such as Markovianity).

3. Frequentist Uncertainty in RNNs

This Section lays out the class of sequential prediction tasks and RNN architectures under study (Section 3.1), provides a formal description of sequential data (Section 3.2), and extends classical notions of frequentist coverage to the sequential setup under consideration (Section 3.3).

3.1. Sequential Prediction with RNNs

We consider a broad class of sequential prediction tasks, to which we provide an encapsulating formulation as follows. Given a (temporal) input sequence $\boldsymbol{x} = [x_1, \ldots, x_T]$ of length T, our objective is to predict a corresponding output sequence $\boldsymbol{y} = [y_1, \ldots, y_{T+T'}]$, where $T' \geq 0$ is the prediction horizon, i.e., the number of future time steps predicted at each time step t. To this end, an RNN is trained to learn the conditional expectation of future values of $\boldsymbol{y}$, i.e.,

$$\mathrm{RNN}(x_1, \ldots, x_t; \boldsymbol{\theta}) \approx \mathbb{E}[\, \boldsymbol{y}_{t:t+T'} \,|\, \boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t}\,], \tag{1}$$

where $\boldsymbol{x}_{t:t'} = [x_t, \ldots, x_{t'}]$, $\boldsymbol{\theta}$ is the RNN model parameters, and the expectation $\mathbb{E}[\,.\,]$ is taken with respect to the randomness of the stochastic process $\mathcal{S} = (\boldsymbol{x}, \boldsymbol{y})$, which we thoroughly describe in Section 3.2. Throughout the paper, we abstract away the RNN model through a sequence of functions $\{f_t\}_{t=1}^{T}$, $f_t : \mathcal{X}^t \to \mathcal{Y}^{T'}$, where $\mathcal{X}$ and $\mathcal{Y}$ are the input and output spaces, respectively. The RNN prediction for the t'-th horizon at time t is denoted by $f_{t,t'}(\boldsymbol{x}_{1:t}; \boldsymbol{\theta})$.

Model training. A training data set $\mathcal{D}_n = \{\mathcal{S}^{(i)}, 1 \leq i \leq n\}$ comprises a set of n sequences[1], where $\mathcal{S}^{(i)} = (\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ is the i-th sequence. The model is trained by minimizing the loss function $L(\mathcal{D}_n; \boldsymbol{\theta}, \boldsymbol{w}) = \sum_{i=1}^{n} L(\mathcal{S}^{(i)}; \boldsymbol{\theta}, \boldsymbol{w})$, where

$$L(\mathcal{S}^{(i)}; \boldsymbol{\theta}, \boldsymbol{w}) \triangleq \sum_{t,t'} w_{t,t'}^{(i)} \cdot \ell(y_{t+t'}^{(i)}, f_{t,t'}(\boldsymbol{x}_{1:t}^{(i)}; \boldsymbol{\theta})). \tag{2}$$

That is, the model's loss at the t'-th horizon in the t-th time step for the i-th sequence is weighted by $w_{t,t'}^{(i)} \in \mathbb{R}_+$. We represent the weights $\{w_{t,t'}^{(i)} : \forall i, t, t'\}$ via the (flattened) vector $\boldsymbol{w} \in \mathbb{R}^{nTT'}$. The choice of the loss $\ell(.)$ depends on whether $\boldsymbol{y}$ is a regression or a classification target.

Which RNN architectures does our formulation cover? For different settings of the weight vector $\boldsymbol{w}$ and the prediction horizon T', the general formulation in (2) encapsulates many prediction setups and RNN architectures common in various applications. A non-exhaustive list of examples for such architectures are provided in Table 1 — these include the *many-to-many* architecture used in sequence labeling tasks (Luong et al., 2016), the *many-to-one* architecture used in sequence classification tasks (Graves et al., 2006), and the *sequence-to-sequence* architecture, usually implemented via encoder-decoder schemes (Sutskever et al., 2014).

[1]Our formulation and proposed methods can be straightforwardly extended to accommodate variable-length sequences.

Table 1. Generality of the formulation in (2).

RNN model	**Weight vector $\boldsymbol{w}$**
Sequence labeling	$T' = 0,\ w_{t,0}^{(i)} = 1,\ \forall\, i, t.$
Sequence classification	$T' = 0,\ w_{t,0}^{(i)} = \mathbf{1}_{\{t=T\}},\ \forall\, i.$
Sequence-to-sequence	$T' > 0,\ w_{t,t'}^{(i)} = \mathbf{1}_{\{t=T\}},\ \forall\, i, t'.$

We do not pose any assumptions on the type of representations learned by the RNN — our proposed methods apply to LSTM, GRU and attention-based architectures. While our work is primarily motivated by applications in which the sequence $\mathcal{S}$ is *temporal*, all methods presented in Section 4 are applicable to other sequential setups, such as natural language processing, where sequences are not necessarily defined over time (Auger & Roy, 2008).

3.2. Sequential Data as Stochastic Processes

Given a probability space $(\Omega, \mathcal{H}, \mathbb{P})$, we view the sequence $\mathcal{S} = (\boldsymbol{x}, \boldsymbol{y})$ as a *stochastic process* comprising a collection of random variables on Ω indexed by time steps $\{1, \ldots, T\}$. A filtration $\mathcal{H}_t, 1 \leq t \leq T$, is a (weakly) increasing collection of σ-algebras on Ω, and is always bounded above by $\mathcal{H}$, i.e., $\mathcal{H}_t \subseteq \mathcal{H}$. The filtration $\mathcal{H}_t$ comprises the information that the RNN model uses to make a prediction at the t-th time step, i.e., the history of the process up to time t.

We characterize the stochastic process $\mathcal{S}$ via its β-*mixing* rate. Formally, the β-mixing coefficient for $\mathcal{S}$ generated under the distribution $\mathbb{P}$ is defined as (Mcdonald et al., 2011)

$$\beta_T(\tau) \triangleq \sup_t \| \, \mathbb{P}_{1:t} \otimes \mathbb{P}_{t+\tau:T} - \mathbb{P}_{t,\tau} \, \|_{TV}, \tag{3}$$

where $\|.\|_{TV}$ is the total variation norm, $\mathbb{P}_{t:t'}$ is the joint distribution of $(\boldsymbol{x}_{t:t'}, \boldsymbol{y}_{t:t'})$, and $\mathbb{P}_{t,t'}$ is the joint distribution of $(\boldsymbol{x}_{1:t}, \boldsymbol{y}_{1:t})$ and $(\boldsymbol{x}_{t+t':T}, \boldsymbol{y}_{t+t':T})$. The β-coefficient $\beta_T(\tau)$ measures the extent to which the trajectory of $\mathcal{S}$ after τ time steps in the future depends on its history — smaller values for $\beta_T(\tau)$ mean that $\mathcal{S}$ forgets its past. When $\beta_T(\tau) = 0$ for all τ, then the data at all time steps are independent, whereas a non-zero $\beta_T(\tau) = 0$ for all τ corresponds to sequences where samples across all time steps exhibit correlation.

3.3. Sequential Confidence Intervals

Our key objective is to quantify the uncertainty in the RNN's sequential predictions $\{f_t(\boldsymbol{x}_{1:t}; \boldsymbol{\theta})\}_{t=1}^{T}$. Following a formal frequentist approach to the problem, Section 4 develops a procedure which maps the filtrations $\{\mathcal{H}_t\}_t$ to a sequence of predictive confidence intervals[2] $\{\mathcal{C}_t(\mathcal{H}_t; \alpha)\}_t$ that contain the true output $\boldsymbol{y}$ with a *coverage probability* of $(1 - \alpha)$.

[2]For simplicity of exposition, we consider one-dimensional outputs; for which uncertainty is quantified via prediction intervals. If $\boldsymbol{y}$ is multidimensional, then $\mathcal{C}_t$ is a *confidence region*.

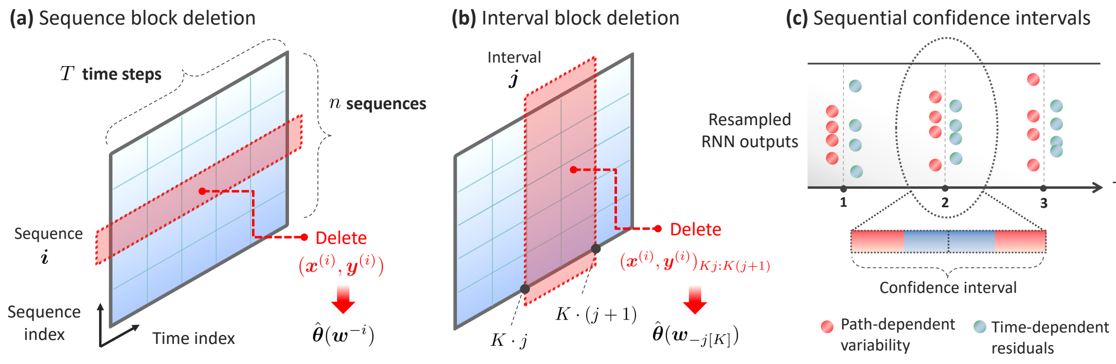
(a) Sequence block deletion
T time steps
n sequences
Sequence i
Sequence index
Time index
Delete
$(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$
$\hat{\boldsymbol{\theta}}(\boldsymbol{w}^{-i})$
(b) Interval block deletion
Interval j
Delete
$(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})_{Kj:K(j+1)}$
$K \cdot j$
$K \cdot (j+1)$
$\hat{\boldsymbol{\theta}}(\boldsymbol{w}_{-j[K]})$
(c) Sequential confidence intervals
Resampled RNN outputs
Time
1
2
3
Confidence interval
Path-dependent variability
Time-dependent residuals

Figure 2. **Uncertainty estimation in RNNs via BJ resampling.** (a) *Sequence block deletion.* In this panel, we represent the data set $\mathcal{D}_n$ as an $n \times T$ matrix. Sequence deletion is conducted by iteratively deleting rows from this matrix. (b) *Interval block deletion.* The columns of the matrix $\mathcal{D}_n$ are grouped into $\lfloor T/K \rfloor$ partitions, each spanning K columns — interval deletion proceeds by deleting one partition at a time. The data set $\mathcal{D}_n$ is resampled $n \cdot \lfloor T/K \rfloor$ times by repeatedly applying sequence and interval block deletion in a nested fashion as shown in Algorithm 1. (c) *Confidence intervals construction.* By re-training the RNN model once on each resampled data set, we collect multiple perturbed versions of the trained parameters $\hat{\boldsymbol{\theta}}$. For a new test sequence, confidence intervals are determined by the quantiles of the leave-one-block-out residuals (red), which only depend on the training data and are fixed for each time step, in addition to the predictions made by the perturbed models for the test sequence (blue), which account for path-dependent variability of the RNN outputs.

Time-dependent coverage. Conventional notions of coverage pertain to "static" regression setups (Lawless & Fredette, 2005). In our sequential setup, confidence estimates are issued for each time step in a test sequence — so how can we extend the notion of coverage to accommodate temporality? To capture the temporal aspect, we coin the notion of "time-dependent coverage" as follows. An α-level time-dependent coverage is satisfied if for every t, confidence intervals cover true outcomes with probability $1-\alpha$, i.e.,

$$\mathbb{P}(\varphi(y_{t+t'}) \in \mathcal{C}_{t,t'}(\mathcal{H}_t;\alpha)) \geq 1-\alpha,\ \forall\, t, t'. \qquad (4)$$

Here, $\varphi(.)$ is a function that maps y_t to the true class probabilities $\{1-p, p\}$ if y_t is a binary classification target, and is an identity function if y_t is a regression target. (4) is an appropriate design objective in applications where sensitive decisions are made repeatedly at each time step — we provide an example for such application in Section 5.

Distinguishing two types of uncertainty. We note that, for classification outputs, i.e., $y_t \in \{0,1\}$, we differentiate between the notions of calibration (or risk) and model uncertainty (Fedorov, 2013; Osband, 2016). Calibration refers to the accuracy of the model's point estimates of the class probabilities, whereas model uncertainty refers to the model's confidence in those point estimates. Calibration is also often referred to as *aleatoric uncertainty* (Gal, 2016) — it reflects the inherent noise uncertainty in the data itself, rather than the model's epistemic uncertainty. Our sought-after confidence intervals $\{\mathcal{C}_t\}_t$ implicitly quantify both types of uncertainty, but do not explicitly disentangle them, i.e., poor calibration would implicitly manifest through wider confidence intervals. Explicit assessment of model calibration can be carried out by computing Brier scores (Avati et al., 2018).

4. The Blockwise Infinitesimal Jackknife

How can we construct confidence intervals $\{\mathcal{C}_t\}_t$ that satisfy the coverage constraints in (4)? In this Section, we do so through a formal resampling procedure that estimates the variability in the *sampling distribution* of an RNN's output by resampling its trained parameters $\hat{\boldsymbol{\theta}}(\boldsymbol{w})$. (Figure 2 provides a high-level illustration of our procedure.)

4.1. Jackknifing via Block Deletion

Given a data set $\mathcal{D}_n$ and a weight vector $\boldsymbol{w}$, the trained RNN parameters, obtained by minimizing the loss in (2), are

$$\hat{\boldsymbol{\theta}}(\boldsymbol{w}) = \arg\min_{\boldsymbol{\theta}}\ L(\mathcal{D}_n;\boldsymbol{\theta},\boldsymbol{w}). \qquad (5)$$

We use *jackknife* resampling to estimate the sampling distribution of $\hat{\boldsymbol{\theta}}(\boldsymbol{w})$. The jackknife entails repeatedly re-training the RNN on leave-one-out (LOO) subsets of the training data $\mathcal{D}_n$, with one *individual* data point deleted from $\mathcal{D}_n$ at a time (Miller, 1974; Efron, 1992). Unlike the standard regression setup, wherein all training data points are independent (Tukey, 1958), here we deal with temporally-correlated data sequences. Hence, to preserve the temporal dependence in the data, we delete *blocks* of temporally contiguous data points instead of individual observations — we denote this "leave-one-block-out" scheme as the *blockwise* jackknife.

Two types of data blocks can be used in our blockwise jackknife (BJ) scheme. The first is the *sequence block*, depicted in Figure 2.a, which comprises the entirety of a single sequence $\mathcal{S} = (\boldsymbol{x}, \boldsymbol{y})$ in the training data $\mathcal{D}_n$. The second is the *interval block*, depicted in Figure 2.b, which comprises the set of all sub-sequences of length $K \leq T$ occupying the same time indexes across all sequences in $\mathcal{D}_n$. If we think of $\mathcal{D}_n$ as an $n \times T$ matrix, then sequence and interval blocks represent horizontal and vertical partitions of $\mathcal{D}_n$, respectively. BJ operates by repeatedly deleting interval and/or sequence blocks, and training the RNN on the remaining data. Block deletion is achieved by setting specific elements of $\boldsymbol{w}$ to zero, leading to an altered weight vector $\boldsymbol{w}'$ as follows:

(a) Sequence block deletion. Deletion of the i-th sequence $\mathcal{S}^{(i)}$ in $\mathcal{D}_n$ entails setting the weights $w^{(i)}_{t,t'} = 0,\, \forall t, t'$. We denote the resulting weight vector as $\boldsymbol{w}^{-i}$. Re-training using $\boldsymbol{w}^{-i}$ is equivalent to training on the data set $\mathcal{D}_n / \{\mathcal{S}^{(i)}\}$.

(b) Interval block deletion. For interval blocks of length K, we divide the sequence length T into $\lfloor T/K \rfloor$ equal-sized blocks. Deletion of the j-th interval block entails setting the weights $w^{(i)}_{j',t'} = 0, \forall i, t', K \cdot j \leq j' \leq K \cdot (j+1)$. We denote the corresponding weight vector as $\boldsymbol{w}_{-j[K]}$. When deleting an interval block, we stitch together the remaining segments of each sequence, preserving temporal order.

Sequence blocks are more relevant in applications where we would have many (short) training sequences (e.g., medical time series data), whereas interval blocks are relevant to applications with few (long) sequences (e.g., financial data). As shown in Algorithm 1, BJ resampling generally proceeds by deleting sequence and interval blocks in a nested fashion (we denote the weight vector with the i-th sequence and the j-th interval deleted as $\boldsymbol{w}^{-i}_{-j[K]}$.) Using the RNN parameters re-trained on $\boldsymbol{w}^{-i}_{-j[K]}$, we collect the leave-one-block-out (LOBO) error residuals of the re-trained RNN models, i.e.,

$$r^{(i,j)}_{t,t'} = \left|\, y^{(i)}_{t+t'} - f_{t,t'}(\boldsymbol{x}^{(i)}_{1:t}; \hat{\boldsymbol{\theta}}(\boldsymbol{w}^{-i}_{-j[K]}))\,\right|, \tag{6}$$

for all t' and $K \cdot j \leq t \leq K \cdot (j+1)$. That is, the LOBO residuals correspond to the residual errors of the RNN re-trained with the i-th sequence and the j-th interval deleted, and evaluated on the data block defined by the intersection between the two deleted blocks, i.e., the intersection between the red blocks in Figures 2.a and 2.b. In most practical applications, we would need to delete only either sequence or interval blocks, depending on the nature of the application at hand.

4.2. Constructing Confidence Intervals

The sequence of confidence intervals $\{\mathcal{C}_{t,t'}\}_t$ is a *bivariate* stochastic process on $(\Omega, \mathcal{H}, \mathbb{P})$ that "sandwiches" the true output over time. Given the resampled RNN parameters $\{\hat{\boldsymbol{\theta}}(\boldsymbol{w}^{-i}_{-j[K]})\}_{i,j}$ obtained through the BJ procedure in Section 4.1, we describe how to construct the confidence intervals $\{\mathcal{C}_{t,t'}\}_t$ in this Section. Before proceeding, we first define some notation. For any set $\mathcal{V} = \{v_i\}_{i=1}^n$, define the $(1-\alpha)$ and α empirical quantiles $\widehat{Q}^+_\alpha\{\mathcal{V}\}$ and $\widehat{Q}^-_\alpha\{\mathcal{V}\}$ as

$$\widehat{Q}^+_\alpha\{\mathcal{V}\} \triangleq \text{the } \lceil(1-\alpha)(n+1)\rceil\text{-th smallest value in } \mathcal{V},$$
$$\widehat{Q}^-_\alpha\{\mathcal{V}\} \triangleq \text{the } \lfloor\alpha(n+1)\rfloor\text{-th smallest value in } \mathcal{V}.$$

Algorithm 1 BJ-based uncertainty estimation in RNNs

```
1: Input: Trained RNN parameters $\hat{\boldsymbol{\theta}}(\boldsymbol{w})$, data set $\mathcal{D}_n$,
2:        interval block length $K$, coverage level $\alpha$,
3:        test input sequence $\boldsymbol{x}^*$.
4: Output: Confidence intervals $\{\mathcal{C}_t(\mathcal{H}^*_t;\alpha)\}_t$.
5: // Resampling sequence and interval blocks
6: for $i = 1$ to $n$ do
7:   for $j = 1$ to $\lfloor \frac{T}{K} \rfloor$ do
8:     $\boldsymbol{w}' \leftarrow \boldsymbol{w}^{-i}_{-j[K]}$.
9:     $\mathcal{I}_{\boldsymbol{w}}(\boldsymbol{w}') \leftarrow \textsc{BlockwiseInfluence}(\boldsymbol{w}, \boldsymbol{w}', \hat{\boldsymbol{\theta}})$.
10:    $\hat{\boldsymbol{\theta}}(\boldsymbol{w}') \leftarrow \hat{\boldsymbol{\theta}}(\boldsymbol{w}) + \mathcal{I}_{\boldsymbol{w}}(\boldsymbol{w}')$.
11:    for $t = K \cdot j$ to $K \cdot (j+1)$ do
12:      $r^{(i,j)}_{t,t'} \leftarrow \left|\, y^{(i)}_{t+t'} - f_{t,t'}(\boldsymbol{x}^{(i)}_{1:t}; \hat{\boldsymbol{\theta}}(\boldsymbol{w}'))\,\right|, \forall t'$.
13:    end for
14:  end for
15: end for
16: // Constructing confidence intervals
17: Construct the prediction variability set $\mathcal{V}_{t,t'}(\boldsymbol{x}^*, \gamma)$.
18: $f^U_{t,t'} \leftarrow \widehat{Q}^+_\alpha\{\mathcal{V}_{t,t'}(\boldsymbol{x}^*; +1)\}, \forall t, t'$.
19: $f^L_{t,t'} \leftarrow \widehat{Q}^-_\alpha\{\mathcal{V}_{t,t'}(\boldsymbol{x}^*; -1)\}, \forall t, t'$.
20: Return $\mathcal{C}_{t,t'} = [f^L_{t,t'}, f^U_{t,t'}], \forall t, t'$.
```

For a test sequence $\boldsymbol{x}^* \in \mathcal{X}^T$, define $\mathcal{V}_{t,t'}(\boldsymbol{x}^*, \gamma) = \{v^{(i)}_{t,t'}\}_i$ as a *prediction variability* set, with $\gamma \in \{-1, +1\}$, where the elements of $\mathcal{V}_{t,t'}$ are constructed as (Barber et al., 2019b)

$$v^{(i)}_{t,t'}(\boldsymbol{x}^*_{1:t}; \gamma) = \underbrace{f_{t,t'}(\boldsymbol{x}^*_{1:t}; \hat{\boldsymbol{\theta}}(\boldsymbol{w}^{-i}_{-j[K]}))}_{\textbf{Path-dependent}} + \underbrace{\gamma \cdot r^{(i,j)}_{t,t'}}_{\textbf{Time-dependent}},$$

with j satisfying $K \cdot j \leq t \leq K \cdot (j+1)$ for a given t. Each element in $\mathcal{V}_{t,t'}$ comprises two component. The first component represents the *path-dependent* variability of the resampled RNN outputs on the new test sequence $\boldsymbol{x}^*$ (pink dots in Figure 2.c) — this reflects the uncertainty of the model about its prediction for this specific path of the stochastic processes in $(\Omega, \mathcal{H}, \mathbb{P})$, i.e., the variability will be larger if $\boldsymbol{x}^*$ is a "rarely seen" sequence in the training data $\mathcal{D}_n$. The second component is the set of LOBO residuals of the RNN model on the training data, which depend only on the time step t, and are fixed for any new test sequence (blue dots in Figure 2.c). The residual terms reflect how well the RNN model generalizes to a new sequence at each time step t. The lower limit of the confidence interval $\mathcal{C}_{t,t'} = [f^L_{t,t'}, f^U_{t,t'}]$ is constructed as $f^L_{t,t'} = \widehat{Q}^-_\alpha\{\mathcal{V}_{t,t'}(\boldsymbol{x}^*; -1)\}$, whereas the upper limit on confidence is $f^U_{t,t'} = \widehat{Q}^+_\alpha\{\mathcal{V}_{t,t'}(\boldsymbol{x}^*; +1)\}$. As we will show in Section 4.4, constructing confidence intervals using the quantiles of the prediction variability set $\mathcal{V}_{t,t'}$ ensures that the coverage condition in (4) is satisfied.

4.3. Blockwise Influence Functions

Implementing exact BJ resampling requires re-training the RNN model $n \cdot \lfloor T/K \rfloor$ times, which is computationally infeasible for most relevant applications that entail large data sets and long sequences. To avoid re-training the RNN, we use influence functions to estimate $\hat{\boldsymbol{\theta}}(\boldsymbol{w}^{-i}_{-j[K]})$ without explicit model fitting — this corresponds to an "infinitesimal" version of our jackknife scheme (Efron, 1992).

The *blockwise* influence function, denoted as $\mathcal{I}_{\boldsymbol{w}}(\boldsymbol{w}')$, quantifies the change in RNN parameters when training using the weight vector $\boldsymbol{w}'$ instead of $\boldsymbol{w}$ through a first-order Taylor approximation given by (Hampel et al., 2011):

$$\hat{\boldsymbol{\theta}}(\boldsymbol{w}') \approx \hat{\boldsymbol{\theta}}(\boldsymbol{w}) + \mathcal{I}_{\boldsymbol{w}}(\boldsymbol{w}'), \tag{7}$$

and can be computed as follows (Koh et al., 2019):

$$\mathcal{I}_{\boldsymbol{w}}(\boldsymbol{w}') = H^{-1}_{\boldsymbol{w}}\, g_{\boldsymbol{w}}(\boldsymbol{w}'), \tag{8}$$

where $H_{\boldsymbol{w}} = \nabla^2_{\theta} L(\mathcal{D}_n; \hat{\boldsymbol{\theta}}(\boldsymbol{w}), \boldsymbol{w})$ is the Hessian of the loss function, and $g_{\boldsymbol{w}}(\boldsymbol{w}') = \nabla_{\theta} L(\mathcal{D}_n; \hat{\boldsymbol{\theta}}(\boldsymbol{w}), \boldsymbol{w}')$. In what follows, we develop an algorithm for efficiently computing (8) (the function BLOCKWISEINFLUENCE in Algorithm 1).

Computing Blockwise Influence. The bottleneck in computing the influence function in (8) is the inversion of the Hessian matrix $H_{\boldsymbol{w}}$; for P RNN parameters in $\hat{\boldsymbol{\theta}}$, the complexity of computing $H^{-1}_{\boldsymbol{w}}$ is $\mathcal{O}(P^3)$. Moreover, because the RNN parameters are shared across time steps, the complexity of gradient computation using backpropagation through time (BPTT) is $\mathcal{O}(PT)$. This computational burden can be significant when performing interval block deletion on long sequences, since deleting the j-th interval block entails altering the RNN hidden states in time steps $t \geq K \cdot (j+1)$ and re-computing all gradients for those time steps.

Following the approaches in (Pearlmutter, 1994; Agarwal et al., 2016; Koh & Liang, 2017), we address the computational challenges above by developing a stochastic estimation approach with blockwise batch sampling to estimate the hessian-vector product (HVP) $H^{-1}_{\boldsymbol{w}}\, g_{\boldsymbol{w}}(\boldsymbol{w}')$, without explicitly inverting or forming the Hessian. The pseudo-code for BLOCKWISEINFLUENCE, which computes $\mathcal{I}_{\boldsymbol{w}}(\boldsymbol{w}^{-i}_{-j[K]})$, is:

- Randomly sample $v \leq n-1$ sub-sequences $m_1, \ldots, m_v$, each represented as a tuple $\{\mathcal{S}^{(m_s)}_{1:Kj}, \mathcal{S}^{(m_s)}_{K(j+1):T}\}^{v}_{s=1}$, with the j-th interval block deleted, then from each tuple select $\mathcal{S}^{(m_s)}_{1:Kj}$. In doing so, we only include temporally contiguous data in the sampled batch, hence retaining the RNN hidden states computed in training. (See Figure 3 for an illustration.)

- For all j and $s \in \{1, \ldots, v\}$, recursively compute

$$\widetilde{H}^{-1}_{s,\boldsymbol{w}}\, g = g + (\boldsymbol{I} - \nabla^2_{\theta} L(\mathcal{S}^{(m_s)}_{1:Kj}; \hat{\boldsymbol{\theta}}(\boldsymbol{w}), \boldsymbol{w}'))\, \widetilde{H}^{-1}_{s-1,\boldsymbol{w}}\, g,$$

where $g = g_{\boldsymbol{w}}(\boldsymbol{w}^{-i}_{-j[K]})$ is the loss gradient after j-th interval and i-th sequence deletion, $\widetilde{H}^{-1}_{s,\boldsymbol{w}} = \sum^{s}_{v=0}(\boldsymbol{I} - \widetilde{H}_{s,\boldsymbol{w}})^{v}$,

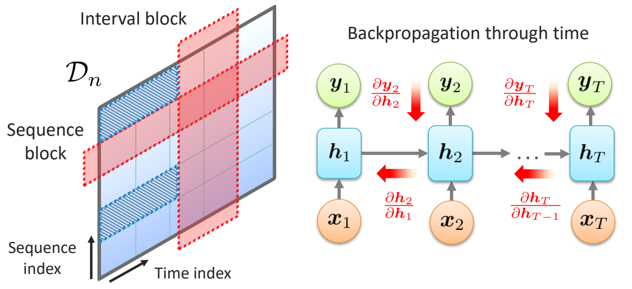
Interval block
$\mathcal{D}_n$
Sequence block
Sequence index
Time index
Backpropagation through time
$\boldsymbol{y}_1$
$\boldsymbol{y}_2$
$\boldsymbol{y}_T$
$\frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{h}_2}$
$\frac{\partial \boldsymbol{y}_T}{\partial \boldsymbol{h}_T}$
$\boldsymbol{h}_1$
$\boldsymbol{h}_2$
$\ldots$
$\boldsymbol{h}_T$
$\frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1}$
$\frac{\partial \boldsymbol{h}_T}{\partial \boldsymbol{h}_{T-1}}$
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_T$

Figure 3. Illustration for blockwise batch sub-sampling in our HVP stochastic estimation algorithm. **Left:** Red-shaded areas are the deleted sequence and interval blocks and blue-shaded areas are the sampled blocks for gradient computation. **Right:** Gradient computation in BPTT — due to parameter sharing, gradients at time t depend on gradients at all preceding time steps $t' < t$.

and $\widetilde{H}$ is the stochastic estimate of the Hessian H. The recursive procedure is initialized with $\widetilde{H}^{-1}_{0,\boldsymbol{w}}\, g = g$, and our final estimate of the HVP is $\widetilde{H}^{-1}_{v,\boldsymbol{w}}\, g$. The recursion above follows from the power series expansion of a matrix inverse, which converges when $H_{\boldsymbol{w}} \succ 0$ and all its eigenvalues are less than 1. Since restricting our sampled sub-sequences to the interval $[1, Kj]$ enables reusing the loss gradients evaluated in training, the overall complexity of our procedure is $\mathcal{O}(nP + vP)$. Further discussion on the complexity and stability of the procedure above is given in Appendix B.

4.4. Theoretical Guarantees

We conclude this Section with a Theorem that characterizes the time-dependent coverage performance of the BJ-based confidence intervals constructed via Algorithm 1.

Theorem 1. *Let $\beta_T(\tau)$ be the β-mixing coefficient of the stochastic processes drawn from $(\Omega, \mathcal{H}, \mathbb{P})$, and let $T_0 \leq T$ be the time interval for which $\beta_T(T_0) = 0$. For an interval block length of $K = T_0$, the time-dependent coverage achieved by the BJ procedure with exact re-training is*

$$\mathbb{P}(\varphi(y_{t+t'}) \in \mathcal{C}_{t,t'}(\mathcal{H}_t; \alpha)) \geq (1 - 2\alpha),\ \forall 1 \leq t \leq T,$$

for prediction horizon t', and coverage $\alpha \in (0, 1)$. □

Theorem 1 shows that if the interval block length K matches the inherent correlation structure of the data, i.e., K is equal to the "mixing-time" T_0, then confidence intervals generated by exact BJ resampling always achieve a time-dependent coverage of $(1 - 2\alpha)$ — as we show in the next Section, BJ resampling usually achieves the target $(1 - 2\alpha)$ in practice (Barber et al., 2019b). The extent to which the approximate (influence function-based) BJ is capable of achieving the target coverage depends on the quality of estimating the re-trained parameters, which in turn depends on various properties of the model's loss function (e.g., convexity and differentiability (Giordano et al., 2018; Koh et al., 2019)).

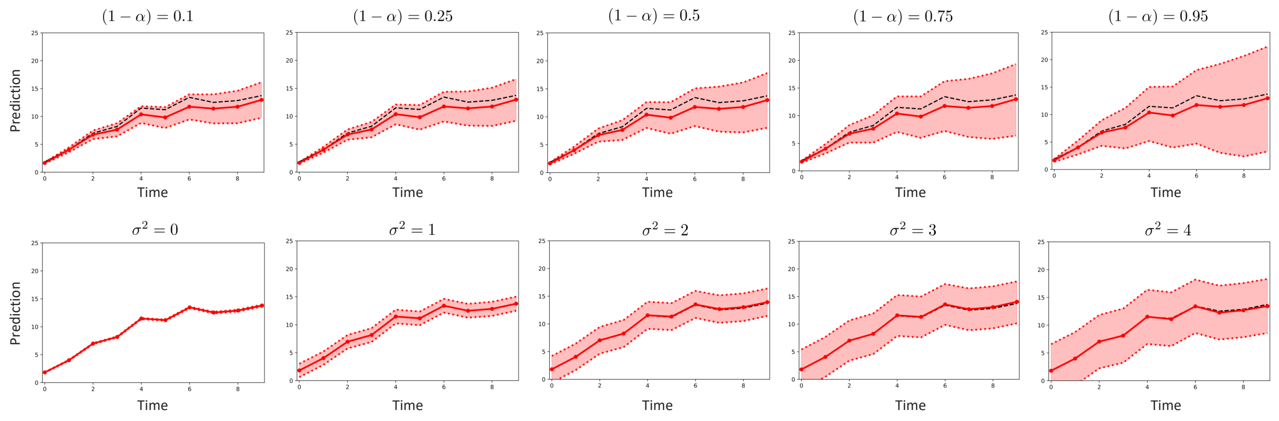

Figure 4. **Sequential confidence intervals.** (a) *Time-dependent aleatoric uncertainty.* In the top panels, we display the sequential confidence intervals $\{\mathcal{C}_{t,1}\}_t$ issued on top of a vanilla RNN model over $T = 10$ time steps within a single sequence. Here, the model is trained on data generated by the process in (9) using the time-dependent noise profile in (10). The red lines correspond to the RNN predictions and the dashed black lines correspond to the true labels $\{y_t\}_t$. We display the confidence intervals at different coverage levels: as we can see, the confidence intervals get wider over time to cover the progressively increasing aleatoric noise. The confidence intervals get wider for higher coverage levels. (b) *Static aleatoric uncertainty.* In the bottom panels, we display the confidence intervals for the same test sequence in the top panels but with the static noise profile in (10). Here, we trained 5 RNN models with varying levels of the static noise variance $\bar{\sigma}^2 \in \{0, 1, 2, 3, 4\}$. As we can see, the sequential confidence intervals are fixed over time, reflecting the uniform aleatoric noise across all time steps. The confidence intervals vanish when no noise is added ($\bar{\sigma}^2 = 0$) and gets increasingly wide as the noise variance increases.

5. Experiments

In this Section, we evaluate our method quantitatively and qualitatively through synthetic simulations (Section 5.1) and experiments on real-world data (Section 5.2). Throughout this Section, we set $K = T$ in all experiments.

5.1. Synthetic Simulations

We consider a synthetic model for generating sequential data through the following autoregressive process:

$$y_t = \sum_{k=0}^{t} a^k \cdot x_k + \epsilon_t,\ k \in \{1, \ldots, T\}, \tag{9}$$

where $a = 0.9$ and $\epsilon_k \sim \mathcal{N}(0, \sigma_k^2)$ is a noise process, with a variance σ_t^2 at time step t. Throughout our experiments, we consider the following noise profiles:

$$\begin{aligned} \textit{Static noise variance}: \sigma_t^2 &= \bar{\sigma}^2, \\ \textit{Time-dependent noise variance}: \sigma_t^2 &= t/10. \end{aligned} \tag{10}$$

The first noise profile induces a fixed level of aleatoric uncertainty across all time steps, whereas the second noise profile induces an increasing level of aleatoric uncertainty in the data as time progresses within the same sequence.

Baselines. We applied our BJ procedure (in a post-hoc manner) to standard RNN trained to learn one-step-ahead forecasts ($T' = 1$). We compared uncertainty estimates obtained by Algorithm 1 with 2 baselines that cover the Bayesian and Frequentist uncertainty estimation methods discussed in Section 1. These included the multi-horizon quantile recurrent forecaster (MQ-RNN) in (Wen et al., 2017) and the Monte Carlo dropout-based RNNs (DP-RNN) (Gal & Ghahramani, 2016b). To ensure a fair comparison, we fix the hyperparameters of the underlying RNNs in all baselines. Details on hyperparameter tuning and uncertainty calibration in all baselines are provided in Appendix D. Unless otherwise stated, confidence intervals by all baselines were estimated for a target coverage of 90% ($\alpha = 0.1$).

Results. We generated 1000 training sequences using the model in (9) with $T = 10$ time steps and trained a vanilla RNN model to predict the label y_t on the basis of the input sequence $\boldsymbol{x}_{1:t}$. We conducted this experiment using the time-dependent noise variance profile in (10), in addition to multiple selections of the noise variance for the static noise profile. In Figure 4, we display the sequential confidence intervals $\{\mathcal{C}_{t,1}\}_t$ issued by the BJ procedure under different simulation scenarios. As we can see, the sequential BJ confidence intervals reflect the ground-truth aleatoric uncertainty profiles dictated by the underlying noise processes. That is, the confidence intervals (1) are wider within a sequence as the noise variance increases over time for a time-dependent noise profile, (2) are of fixed width over time within a sequence for a static noise profile, and (3) have their width increase with the target coverage and the noise variance.

In Figure 5, we examine the performance of all baselines at varying levels of aleatoric and epistemic uncertainty. Here we use the static noise profile in (10). Aleatoric uncertainty

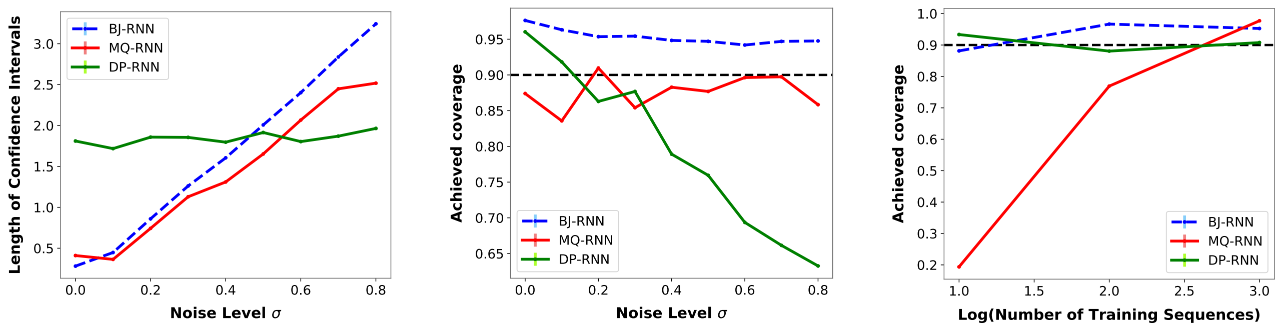

Figure 5. **Comparison between baselines at varying levels of aleatoric and epistemic uncertainty.** Black dashed lines are the target coverage rates. **(Left)** Average length of confidence intervals issued by all baselines at different noise levels. **(Middle)** Average coverage for all baselines at different noise levels. **(Right)** Average coverage for all baselines for different numbers of training sequences.

is controlled by varying the level of noise $\bar{\sigma}^2$, whereas epistemic uncertainty is controlled by varying the number of training sequences. As we can see, all baselines display an increasing length of confidence intervals with increasing aleatoric uncertainty, except for DP-RNN which does not exhibit sensitivity to the increasing ground-truth uncertainty (Figure 5, left). Our model (BJ-RNN) achieves the target coverage for all levels of aleatoric uncertainty, whereas DP-RNN tends to under-cover when the noise levels are large (Figure 5, middle). With respect to epistemic uncertainty, we note that since MQ-RNN "learns" the prediction intervals, its ability to achieve the target coverage depends on the amount of training data — MQ-RNN under-covers the true labels when the number of training sequences is small. On the contrary, since BJ-RNN is a post-hoc procedure, it achieves the target coverage irrespective of the size of training data (Figure 5, right).

5.2. Experiments on Real Data

How can uncertainty quantification help us conquer practical (sequential) decision-making problems? In this Section, we evaluate the utility of our procedure in guiding therapeutic decisions for critical care medicine.

Data. We conducted experiments on data from the Medical Information Mart for Intensive Care (MIMIC-III) (Johnson et al., 2016) database, which comprises the electronic health records for critically-ill patients admitted to an intensive care unit (ICU). From MIMIC-III, we extracted sequential data for patients on antibiotics, with trajectories up to $T = 10$ days (time steps) — this resulted in a data set of 5,833 sequences. For all patients, we extracted 5 variables (lab tests and vital signs) collected during hospitalization. More details on data processing can be found in Appendix D.

Clinical setup. Based on each patient's 5 features (in the sequence $\boldsymbol{x}$), critical care teams need to (repeatedly) decide whether or not to administer *antibiotics* for a given patient to regulate their white blood cell count (WBCC). WBCC levels are clinically categorized into: Leucopenic ($< 4 \times 10^9/L$), normal ($4 - 10 \times 10^9/L$), and Leucemoid ($> 10 \times 10^9/L$) ranges (Waheed et al., 2003). High (Leucemoid) WBCC is associated with severe illness and poor outcomes (Shuman et al., 2012) — antibiotics can prevent WBCC from hitting the Leucemoid range, but also entail the risk of plummeting WBCC to the Leucopenic range. Thus, decisions on antibiotic administration should balance risks againts benefits.

Methods	**RMSE**	**Coverage**	**CI length**	**Audit Acc.** (P_4, P_{10})
MQ-RNN	22.39	83.6%	8.29	(66%, 42%)
DP-RNN	20.32	49.2%	3.99	(66%, 50%)
BJ-RNN	20.24	94.4%	13.18	(67%, 48%)

Table 2. Performance comparison between the different baselines.

Evaluation. We train various RNN-based baselines in order to sequentially predict next-day's WBCC for every patient, and quantify the uncertainty in these predictions. (Figure 6 demonstrates the RNN predictions along with the BJ-based confidence intervals for one patient in MIMIC-III.) We evaluate the uncertainty estimates by the different baselines under study with respect to various performance metrics: the predictive accuracy of the underlying RNN model, the achieved coverage rate, the average length of the confidence interval and the accuracy of the uncertainty estimates in *auditing* model reliability. We use two metrics to evaluate the auditing accuracy of uncertainty estimates: the *precision* of predicting whether the true WBCC is in the Leucopenic range (P_4) and the precision of predicting whether WBCC is in the Leucemoid range (P_{10}). P_4 is evaluated using the lower confidence limit f_L as the prediction and the indicator of whether WBCC $< 4 \times 10^9/L$ as the label, whereas P_{10} is evaluated with f_U as the prediction and the indicator of whether WBCC $> 10 \times 10^9/L$ as the label.

Results. In Table 2, we summarize the performance of all baselines. The first thing to notice is that since our BJ procedure is post-hoc, it was capable of providing accurate model

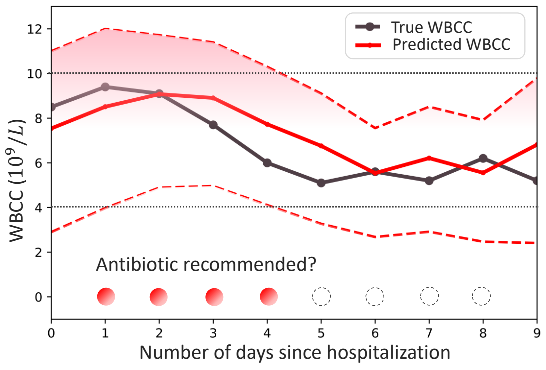

Figure 6. WBCC predictions issued by a standard RNN (▬) for one patient over time, along with the BJ-based confidence intervals (dashed limits ┅). An antibiotic is safely administered (red circle) on a given day if the upper confidence limit crosses the Leucopenic range, and the lower limit is above the Leucopenic range.

audits without compromising the model's accuracy; the root mean square error (RMSE) of the vanilla RNNs were better than those of the baselines with built-in approaches to uncertainty quantification (MQ-RNN). In terms of the audit accuracy, BJ-based confidence intervals perform competitively with MQ-RNN — which is specifically trained to optimize prediction intervals — and the DP-RNN baseline. Moreover, because MQ-RNN is optimized to learn prediction intervals, it incurs a loss in predictive accuracy, whereas our model is applied in a post-hoc fashion without compromising the accuracy of the model's predictions. Notably, our procedure is the only baseline that is capable of achieving the desired coverage rate, whereas the two competing baselines undercover the true labels. The target coverage is achieved by our procedure without trivially exaggerating the lengths of the confidence intervals — this is crucial for confidently informing clinicians on the true hematologic range for the patient at hand.

6. Conclusion

In this paper, we introduced one of the first approaches for estimating frequentist confidence intervals on the predictions of Recurrent neural networks (RNNs). Our method does not interfere with model training or compromise its accuracy, applies to any RNN architecture, and provides theoretical coverage guarantees on the estimated uncertainty intervals. The key idea behind our method is to re-sample "blocks" of temporally-correlated training data, and collect the predictions of the RNN re-trained on the remaining data. We used influence functions to approximate the re-trained RNN model parameters, thus eliminating the need for exhaustive re-training. Future extensions of our work would include more computationally efficient approaches for evaluating influence functions in RNN models with a large number of parameters, and analyzing the accuracy of blockwise influence functions as approximations of re-trained RNN model parameters.

Acknowledgments

The authors would like to thank the reviewers for their helpful comments. This work was supported by the US Office of Naval Research (ONR) and the National Science Foundation (NSF grants 1524417 and 1722516).

References

Agarwal, N., Bullins, B., and Hazan, E. Second-order stochastic optimization in linear time. *stat*, 1050:15, 2016.

Alaa, A. M. and Van Der Schaar, M. A hidden absorbing semi-markov model for informatively censored temporal data: Learning and inference. *The Journal of Machine Learning Research*, 19(1):108–169, 2018.

Alaa, A. M. and van der Schaar, M. Attentive state-space modeling of disease progression. In *Advances in Neural Information Processing Systems*, pp. 11334–11344, 2019.

Auger, A. and Roy, J. Expression of uncertainty in linguistic data. In *2008 11th International Conference on Information Fusion*, pp. 1–8. IEEE, 2008.

Avati, A., Duan, T., Jung, K., Shah, N. H., and Ng, A. Countdown regression: sharp and calibrated survival predictions. *arXiv preprint arXiv:1806.08324*, 2018.

Bakker, B. Reinforcement learning with long short-term memory. In *Advances in neural information processing systems*, pp. 1475–1482, 2002.

Bao, W., Yue, J., and Rao, Y. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

Barber, R. F., Candes, E. J., Ramdas, A., and Tibshirani, R. J. Conformal prediction under covariate shift. *arXiv preprint arXiv:1904.06019*, 2019a.

Barber, R. F., Candes, E. J., Ramdas, A., and Tibshirani, R. J. Predictive inference with the jackknife+. *arXiv preprint arXiv:1905.02928*, 2019b.

Bayarri, M. J. and Berger, J. O. The interplay of bayesian and frequentist analysis. *Statistical Science*, pp. 58–80, 2004.

Chien, J.-T. and Ku, Y.-C. Bayesian recurrent neural network for language modeling. *IEEE transactions on neural networks and learning systems*, 27(2):361–374, 2015.

Dusenberry, M. W., Tran, D., Choi, E., Kemp, J., Nixon, J., Jerfel, G., Heller, K., and Dai, A. M. Analyzing the role of model uncertainty for electronic health records. *arXiv preprint arXiv:1906.03842*, 2019.

Efron, B. Jackknife-after-bootstrap standard errors and influence functions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 54(1):83–111, 1992.

Fedorov, V. V. *Theory of optimal experiments*. Elsevier, 2013.

Fortunato, M., Blundell, C., and Vinyals, O. Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*, 2017.

Gal, Y. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning (ICML)*, pp. 1050–1059, 2016a.

Gal, Y. and Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pp. 1019–1027, 2016b.

Gasthaus, J., Benidis, K., Wang, Y., Rangapuram, S. S., Salinas, D., Flunkert, V., and Januschowski, T. Probabilistic forecasting with spline quantile function rnns. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1901–1910, 2019.

Giordano, R., Stephenson, W., Liu, R., Jordan, M. I., and Broderick, T. A swiss army infinitesimal jackknife. *arXiv preprint arXiv:1806.00550*, 2018.

Gollier, C. *The economics of risk and uncertainty*. Edward Elgar Publishing Limited, 2018.

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. 2006.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. *Robust statistics: the approach based on influence functions*, volume 196. John Wiley & Sons, 2011.

Hernández-Lobato, J. M. and Adams, R. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *International Conference on Machine Learning (ICML)*, pp. 1861–1869, 2015.

Hron, J., Matthews, A. G. d. G., and Ghahramani, Z. Variational gaussian dropout is not bayesian. *arXiv preprint arXiv:1711.02989*, 2017.

Johnson, A. E., Pollard, T. J., Shen, L., Li-wei, H. L., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Celi, L. A., and Mark, R. G. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

Kalchbrenner, N. and Blunsom, P. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1700–1709, 2013.

Koenker, R. and Hallock, K. F. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1885–1894. JMLR. org, 2017.

Koh, P. W., Ang, K.-S., Teo, H. H., and Liang, P. On the accuracy of influence functions for measuring group effects. *arXiv preprint arXiv:1905.13289*, 2019.

Krishnan, R. G., Shalit, U., and Sontag, D. Structured inference networks for nonlinear state space models. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Kunsch, H. R. The jackknife and the bootstrap for general stationary observations. *The annals of Statistics*, pp. 1217–1241, 1989.

Lakshminarayanan, B., Pritzel, A., and Blundell, C. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6402–6413, 2017.

Lawless, J. and Fredette, M. Frequentist prediction intervals and predictive distributions. *Biometrika*, 92(3):529–542, 2005.

Lim, B., Alaa, A., and Schaar, M. v. d. Forecasting treatment responses over time using recurrent marginal structural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 7494–7504, 2018.

Luong, M.-T., Le, Q. V., Sutskever, I., Vinyals, O., and Kaiser, L. Multi-task sequence to sequence learning. *International Conference on Representation Learning (ICLR)*, 2016.

Maddox, W., Garipov, T., Izmailov, P., Vetrov, D., and Wilson, A. G. A simple baseline for bayesian uncertainty in deep learning. *arXiv preprint arXiv:1902.02476*, 2019.

Malinin, A. and Gales, M. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, pp. 7047–7058, 2018.

Marra, G. and Wood, S. N. Coverage properties of confidence intervals for generalized additive model components. *Scandinavian Journal of Statistics*, 39(1):53–74, 2012.

Mcdonald, D., Shalizi, C., and Schervish, M. Estimating beta-mixing coefficients. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 516–524, 2011.

Mentch, L. and Hooker, G. Quantifying uncertainty in random forests via confidence intervals and hypothesis tests. *The Journal of Machine Learning Research (JMLR)*, 17(1):841–881, 2016.

Miller, R. G. The jackknife-a review. *Biometrika*, 61(1): 1–15, 1974.

Mirikitani, D. T. and Nikolaev, N. Recursive bayesian recurrent neural networks for time-series modeling. *IEEE Transactions on Neural Networks*, 21(2):262–274, 2009.

Osband, I. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS Workshop on Bayesian Deep Learning*, 2016.

Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J. V., Lakshminarayanan, B., and Snoek, J. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Quenouille, M. H. Notes on bias in estimation. *Biometrika*, 43(3/4):353–360, 1956.

Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y., and Januschowski, T. Deep state space models for time series forecasting. In *Advances in Neural Information Processing Systems*, pp. 7785–7794, 2018.

Ritter, H., Botev, A., and Barber, D. A scalable laplace approximation for neural networks. 2018.

Selvin, S., Vinayakumar, R., Gopalakrishnan, E., Menon, V. K., and Soman, K. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1643–1647. IEEE, 2017.

Shickel, B., Tighe, P. J., Bihorac, A., and Rashidi, P. Deep ehr: a survey of recent advances in deep learning techniques for electronic health record (ehr) analysis. *IEEE journal of biomedical and health informatics*, 22(5):1589–1604, 2017.

Shuman, M., Demler, T. L., Trigoboff, E., and Opler, L. A. Hematologic impact of antibiotic administration on patients taking clozapine. *Innovations in clinical neuroscience*, 9(11-12):18, 2012.

Sundermeyer, M., Schlüter, R., and Ney, H. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

Sutskever, I., Vinyals, O., and Le, Q. Sequence to sequence learning with neural networks. *Advances in NIPS*, 2014.

Taylor, J. W. A quantile regression neural network approach to estimating the conditional density of multiperiod returns. *Journal of Forecasting*, 19(4):299–311, 2000.

Tukey, J. Bias and confidence in not quite large samples. *Ann. Math. Statist.*, 29:614, 1958.

Wager, S. and Athey, S. Estimation and inference of heterogeneous treatment effects using random forests. *Journal of the American Statistical Association*, 113(523):1228–1242, 2018.

Wager, S., Hastie, T., and Efron, B. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *The Journal of Machine Learning Research (JMLR)*, 15(1):1625–1651, 2014.

Waheed, U., Williams, P., Brett, S., Baldock, G., and Soni, N. White cell count and intensive care unit outcome. *Anaesthesia*, 58(2):180–182, 2003.

Wang, L., Zhang, W., He, X., and Zha, H. Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2447–2456. ACM, 2018.

Welling, M. and Teh, Y. W. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688, 2011.

Wen, R., Torkkola, K., Narayanaswamy, B., and Madeka, D. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

Zhu, L. and Laptev, N. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 103–110. IEEE, 2017.

Appendix

Appendix A: Literature Survey

Driven by the desire to suppress the "overconfidence" exhibited by poorly calibrated — but highly discriminative — deep learning models, various methods for uncertainty estimation in standard feed-forward neural networks have been recently proposed (Hernández-Lobato & Adams, 2015; Gal & Ghahramani, 2016a; Lakshminarayanan et al., 2017; Malinin & Gales, 2018). However, the literature on uncertainty quantification in RNNs is rather scarce. In what follows, we provide a detailed overview of uncertainty quantification methods in RNNs, other methods that have been developed for feed-forward networks, background on jackknife resampling, and the connections between our work and modern applications of influence functions.

Uncertainty quantification in RNNs

We identified the following three strands of literature relating to uncertainty quantification in RNNs.

Bayesian RNNs. The most prevalent approaches for RNN uncertainty estimation hinge on Bayesian modeling (Fortunato et al., 2017; Chien & Ku, 2015; Zhu & Laptev, 2017; Mirikitani & Nikolaev, 2009). By specifying a prior over the RNN parameters, these approaches estimate a model's uncertainty via its posterior credible intervals. However, exact inference in Bayesian deep learning is generally intractable or computationally infeasible. Hence, practical Bayesian RNNs rely on dropout-based approximate inference (Gal & Ghahramani, 2016a;b), which is often hard to calibrate, and is generally ill-posed as its posterior distributions do not concentrate asymptotically (Osband, 2016; Hron et al., 2017). Moreover, Bayesian methods require significant modifications to model architecture and training which limits their versatility. Our paper addresses these issues by developing a frequentist post-hoc alternative to Bayesian methods.

Quantile RNNs. Similar to quantile regression (Koenker & Hallock, 2001), quantile RNNs (Q-RNNs) learn prediction intervals by explicitly modeling the cumulative distribution function of the prediction targets (Taylor, 2000; Wen et al., 2017; Gasthaus et al., 2019). The key difference between these approaches and ours is that Q-RNNs learn uncertainty jointly with the main prediction task, which as we have shown in Section 5, can compromise their predictive accuracy. Moreover, Q-RNNs are bespoke models that must be retailored for each new prediction task, whereas our approach can be applied in a post-hoc manner to any RNN architecture.

Probabilistic RNNs. This class of models capture the structural uncertainty within the stochastic dynamics of sequential data. Existing variants of probabilistic RNNs are mainly based on deep state-space models (Krishnan et al., 2017; Rangapuram et al., 2018; Alaa & van der Schaar, 2019). Probabilistic models are bespoke to the application at hand, and may not be appropriate for some application domains. Moreover, most probabilistic models entail restrictive assumptions on sequence dynamics, such as state Markovianity (Alaa & Van Der Schaar, 2018).

Uncertainty quantification in feed-forward networks

Bayesian models are the dominant tools for uncertainty quantification in feed-forward neural networks (Welling & Teh, 2011; Hernández-Lobato & Adams, 2015; Ritter et al., 2018; Maddox et al., 2019). Post-hoc application of Bayesian methods is not possible since Bayesian models require major modifications to the underlying predictive models (by specifying priors over model parameters). Moreover, Bayesian credible intervals do not guarantee frequentist coverage, and more crucially, the achieved coverage can be very sensitive to hyper-parameter tuning (Bayarri & Berger, 2004). Finally, a major limitation to exact Bayesian inference is that they are computationally prohibitive, and their alternative approximations — e.g., (Gal & Ghahramani, 2016a) — may induced posteriors that do not concentrate asymptotically (Osband, 2016; Hron et al., 2017).

Deep ensembles (Lakshminarayanan et al., 2017) are regarded as the most competitive alternative non-Bayesian benchmark for uncertainty estimation (Ovadia et al., 2019). These methods repeatedly re-trains the model on sub-samples of the data (using adversarial training), and then estimates uncertainty through the variance of the aggregate predictions. However, while applying deep ensemble methods may be feasible in feed-forward neural networks, creating large ensembles of RNN models is computationally infeasible, especially for long time series.

Jackknife resampling

Jackknife resampling, originally developed by Quenouille in (Quenouille, 1956) and refined by Tukey in (Tukey, 1958), is a general methodology for estimating sampling distributions. Our work builds on this literature by applying the jackknife to RNN predictions in order to estimate its predictive uncertainty through the sampling variance. The key difference between our setup and that of the standard jackknife is that RNNs do not operate on i.i.d data — they operate on temporally correlated sequences. In (Kunsch, 1989), Künsch proposed a variant of the jackknife for general stationary observations rather than i.i.d data. As we have shown in Section 5, the sampling procedure therein is a special case of ours when the RNN is trained with a single data sequence (i.e., a single realization of the underlying stochastic process).

The (infinitesimal) jackknife method was previously used for quantifying the predictive uncertainty in random forests (Wager et al., 2014; Mentch & Hooker, 2016; Wager & Athey, 2018). In these works, however, the developed jackknife estimators are bespoke to bagging predictors, and cannot be straightforwardly extended to deep neural networks. More recently, general-purpose jackknife estimators were developed in (Barber et al., 2019b), where two exhaustive leave-one-out procedures: the *jackknife+* and the *jackknife-minmax* where shown to have assumption-free worst-case coverage guarantees of $(1-2\alpha)$ and $(1-\alpha)$, respectively. However, our work is the first to develop such methods for time-series data.

Previous application of influence functions has been limited to model interpretability by learning sample importance (Koh & Liang, 2017; Koh et al., 2019). Our work uses the same machinery of influence function computation via Hessian-vector products, but for the purpose of estimating leave-one-out model parameters.

Appendix B: Stochastic Estimation of HVPs via Blockwise Batch Sampling

The bottleneck in computing the influence function in (8) is the inversion of the Hessian matrix $H_{\boldsymbol{w}}$; for P RNN parameters in $\hat{\boldsymbol{\theta}}$, the complexity of computing $H_{\boldsymbol{w}}^{-1}$ is $\mathcal{O}(P^3)$. Moreover, because the RNN parameters are shared across time steps, the complexity of gradient computation using backpropagation through time (BPTT) is $\mathcal{O}(PT)$. This computational burden can be significant when performing interval block deletion on long sequences, since deleting the j-th interval block entails altering the RNN hidden states in time steps $t \geq K \cdot (j+1)$ and re-computing all gradients for those time steps.

We start by the following well-known fact about the inverse of a matrix H with $\|H\| \leq 1$ and $H \succ 0$. The inverse of such matrix can be evaluated through the following power series expansion:

$$H^{-1} = \sum_{i=0}^{\infty} (I - H)^i. \tag{1}$$

Following the approaches in (Pearlmutter, 1994; Agarwal et al., 2016; Koh & Liang, 2017), we address the computational challenges above by developing a stochastic estimation approach with blockwise batch sampling to estimate the hessian-vector product (HVP) $H_{\boldsymbol{w}}^{-1}\, g_{\boldsymbol{w}}(\boldsymbol{w}')$, without explicitly inverting or forming the Hessian. The pseudo-code for BLOCKWISEINFLUENCE, which computes $\mathcal{I}_{\boldsymbol{w}}(\boldsymbol{w}_{-j[K]}^{-i})$, is:

- Randomly sample $v \leq n-1$ sub-sequences $m_1, \ldots, m_v$, each represented as a tuple $\{\mathcal{S}_{1:Kj}^{(m_s)}, \mathcal{S}_{K(j+1):T}^{(m_s)}\}_{s=1}^{v}$, with the j-th interval block deleted, then from each tuple select $\mathcal{S}_{1:Kj}^{(m_s)}$. In doing so, we only include temporally contiguous data in the sampled batch, hence retaining the RNN hidden states computed in training. (See Figure 3 for an illustration.)
- For all j and $s \in \{1, \ldots, v\}$, recursively compute

$$\widetilde{H}_{s,\boldsymbol{w}}^{-1}\, g = g + (I - \nabla_\theta^2\, L(\mathcal{S}_{1:Kj}^{(m_s)}; \hat{\boldsymbol{\theta}}(\boldsymbol{w}), \boldsymbol{w}'))\, \widetilde{H}_{s-1,\boldsymbol{w}}^{-1}\, g,$$

where $g = g_{\boldsymbol{w}}(\boldsymbol{w}_{-j[K]}^{-i})$ is the loss gradient after j-th interval and i-th sequence deletion, $\widetilde{H}_{s,\boldsymbol{w}}^{-1} = \sum_{v=0}^{s}(I - \widetilde{H}_{s,\boldsymbol{w}})^v$, and $\widetilde{H}$ is the stochastic estimate of the Hessian H. The recursive procedure is initialized with $\widetilde{H}_{0,\boldsymbol{w}}^{-1}\, g = g$, and our final estimate of the HVP is given by $\widetilde{H}_{v,\boldsymbol{w}}^{-1}\, g$, which converges to the true inverse for $v \to \infty$.

Since restricting our sampled sub-sequences to the interval $[1, Kj]$ enables reusing the loss gradients evaluated in training, the overall complexity of our procedure is $\mathcal{O}(nP + vP)$, this can be broken down as follows. The complexity of multiplying $\nabla_\theta^2\, L(\mathcal{S}_{1:Kj}^{(m_s)}; \hat{\boldsymbol{\theta}}(\boldsymbol{w}), \boldsymbol{w}'))$ with $\widetilde{H}_{s-1,\boldsymbol{w}}^{-1}\, g$ is $\mathcal{O}(P)$ per iteration which renders the overall complexity of computing this Hessian-product term as $\mathcal{O}(vP)$. The complexity of computing the initial gradient terms is $\mathcal{O}(nP)$.

The procedure above converges only if the norm of the Hessian satisfies $\|H\| \leq 1$. Thus, if the largest eigenvalue σ_{max} of H is greater than 1, we normalize the loss to keep the largest eigenvalue less than 1. Moreover, we add a dampening term λ to the Hessian in order to keep it invertible (positive definite). This results in the following recursion:

$$\widetilde{H}_{s,\boldsymbol{w}}^{-1}\, g = g + (1-\lambda)\, \widetilde{H}_{s-1,\boldsymbol{w}}^{-1}\, g - \frac{1}{\sigma_{max}} \nabla_\theta^2\, L(\mathcal{S}_{1:Kj}^{(m_s)}; \hat{\boldsymbol{\theta}}(\boldsymbol{w}), \boldsymbol{w}')\, \widetilde{H}_{s-1,\boldsymbol{w}}^{-1}\, g,$$

which converges with the appropriate selection of σ_{max} and λ.

Appendix C: Proof of Theorem 1

Given a mixing-time of T_0, we can segment the data set $\mathcal{D}_n = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^n$ into evenly spaced segments of data

$$\widetilde{\mathcal{D}}_n = \{(\boldsymbol{x}^{(i)}_{j\cdot T_0:j\cdot T_0+T_0}, y^{(i)}_{j\cdot T_0:j\cdot T_0+T_0})\}_{i=1}^n,$$

which we can re-write as a new data set with $\tilde{n} = n \cdot T/T_0$ samples each with $d \cdot T_0$ dimensional features as follows:

$$\widetilde{\mathcal{D}}_n = \{(\tilde{\boldsymbol{x}}^{(j)}, \tilde{y}^{(j)})\}_{j=1}^{\tilde{n}},$$

where all the data points $\widetilde{\mathcal{D}}_n$ are independent. By treating each RNN prediction at time step t as a new regression problem, it follows that the exact BJ procedure achieves coverage by directly applying Theorem 1 in (Barber et al., 2019a) to the modified data set $\widetilde{\mathcal{D}}_n$.

Appendix D: Experiments

Experimental Setup

Data. We used data from the Medical Information Mart for Intensive Care (MIMIC-III) (Johnson et al., 2016) database, which comprises the electronic health records for critically-ill patients admitted to an intensive care unit (ICU). From MIMIC-III, we extracted sequential data for patients on antibiotics, with trajectories up to $T = 10$ time steps — this resulted in a data set with $n = 5,833$ sequences. For each patient, we extracted 30 variables (lab tests and vital signs) listed in Table 1, based on which predictions are issued.

Variable Names	
Age	Weight
Temperature	Heart rate
Systolic blood pressure	Diastolic blood pressure
Mean blood pressure	SpO2
FiO2	Respiratory rate
Glucose	Bicarbonate (high)
Bicarbonate (low)	Creatinine (high)
Creatinine (low)	Hematocrit (high)
Hematocrit (low)	Hemoglobin (high)
Hemoglobin (low)	Platelet (high)
Platelet (low)	Potassium (high)
Potassium (low)	Billirubun (high)
Billirubun (low)	Weight blood cell (high)
Weight blood cell (low)	Antibiotics
Norepinephrine	Mechanical ventilator

Table 1. Vital signs and lab tests in MIMIC-III.

The follow-up times were varying from one patient to the other, ranging from 4 days to 10 days. This results in a learning problem with variable-length sequences. Clinical care teams decide *when* to administer an antibiotic for each patient on a daily basis — the frequency of antibiotic treatments over time for the entire population is provided in Figure 1. The patient outcomes under consideration are the white blood cell count (WBCC) — a high white blood cell count is associated with severe illness and poor outcome for ICU patients (Shuman et al., 2012). The WBCC averaged for all patients over time are depicted in Figure 2.

Clinical setup. The WBCC levels are medically categories into four levels as follows (Waheed et al., 2003):

- Leucopenic ($< 4 \times 10^9/L$).
- Normal ($4 - 10 \times 10^9/L$).
- Leucemoid ($10 - 25 \times 10^9/L$).

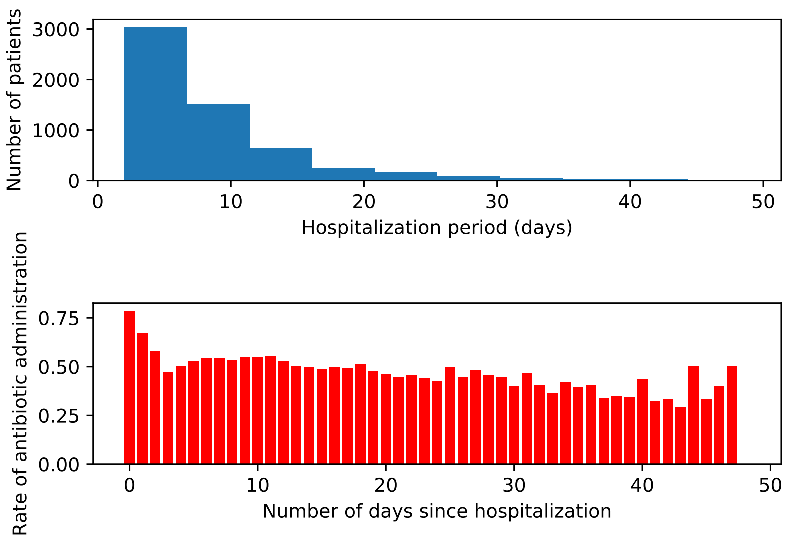

Figure 1. Statistics of hospitalization and treatment data.

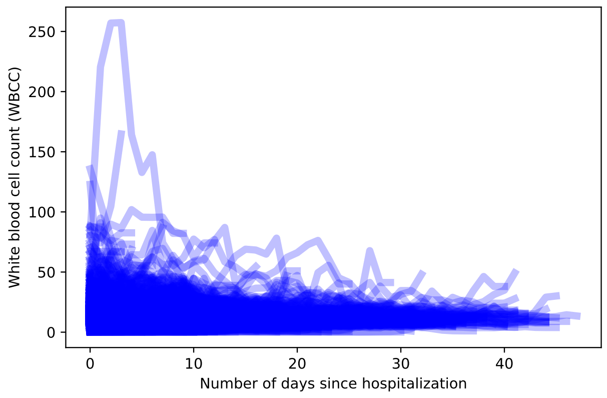

Figure 2. Distribution of white blood cell counts over time.

- Exaggerated Leucemoid ($> 25 \times 10^9/L$).

Antibiotic administration in the ICU aims at keeping the WBCC within the normal range. However, the extent by which an antibiotic affects a patient's WBCC — if the WBCC reduces significantly and enters the Leucopenic level, then the patient might encounter further health risks. Thus, we need to predict a patient's outcome with confidence intervals; if those intervals overlaps with the Leucopenic and Leucemoid regions, then we should abstain from administering an antibiotic.

We link this setup with our formulation as follows. Each patient's feature sequence $\boldsymbol{x}$ is a collection of 30 clinical variables measured repeatedly over T time steps ($T \leq 10$). We selected the most relevant 5 among these features, and trained an RNN model to predict the next-day WBCC for every patient, and use our procedure to construct confidence intervals on these predictions.

Baselines. We conduct our main experiment with the simple RNN architecture, in addition to LSTMs and GRUs. Further architectures are also explored later in this Section. We compared our method with 5 baselines for predictive uncertainty estimation in RNNs that cover the different modeling categories presented in Section 1. The modeling approaches under consideration and their respective baselines are:

(a) Bayesian RNNs. We implemented the Monte Carlo dropout-based Bayesian RNNs (DP-RNNs) proposed in (Gal & Ghahramani, 2016b) using the approximate variational inference scheme described in Sections 3 and 4 in (Gal & Ghahramani, 2016b). Dropout is applied to all RNN layers during training and then Monte Carlo dropout is applied for each new test

sequence to collect samples of the model predictions. We compute the confidence intervals by estimating the posterior variance of a Gaussian distribution through the Monte Carlo samples and then constructing confidence intervals through the tails of the Gaussian.

(b) Quantile RNNs. We implemented an quantile RNN forecaster similar to the one proposed in (Wen et al., 2017) using the quantile loss:

$$\ell(y, f) = q\,(y - f^{(q)})_+ + (1-q)\,(f^{(q)} - y)_+,$$

where $(.)_+ = \max(0, .)$ and $y^{(q)}$ is the q-th quantile of the prediction target. The parameter q is selected so that the learned prediction intervals achieve a coverage probability of $(1-\alpha)$. In order to construct confidence intervals with α-level coverage, we train the RNN decoder to predict $f^{(1-\alpha/2)}$ and $f^{(1-\alpha/2)}$ in order to recover the lower and upper limits of the confidence intervals, respectively.

Hyperparameter Tuning and Uncertainty Calibration

To ensure a fair comparison, we fix the hyperparameters of the underlying RNNs in all baselines to the following:

- Number of layers: 1
- Number of hidden units: 20
- Learning rate: 0.01
- Batch size: 150
- Number of epochs: 10
- Number of steps: 1000

The hyperparameters above where tuned to optimize the RMSE performance of the vanilla RNN model. There was no improvement in the out-of-sample RMSE or discriminative accuracy of all baselines by tweaking these hyperparameters. For the DP-RNN model, we tuned the dropout rate to optimize its RMSE given the values above for the other hyperparameters, the optimal dropout value was 0.45.

The $(1-\alpha)$ confidence intervals for all baselines were computed as follows:

- **MQ-RNN:** The confidence intervals are obtained straightforwardly as the interval bounded by the model's quantile outputs $[y^{(\alpha)}, y^{(1-\alpha)}]$. Note that the MQ-RNN is explicitly trained to predict these values.
- **DP-RNN:** Because DP-based inference approximates a (deep) Gaussian process prosterior, we construct the $(1-\alpha)$ confidence intervals through the tails of the Gaussian, i.e., the confidence interval is given by $[y^{(\alpha)}, y^{(1-\alpha)}]$, where the limits of the interval are constructed as $y^{(q)} = z_c(q) \cdot \sqrt{\mathbb{V}_n[\{y_i\}_i]}$, where $z_c(q)$ is the critical value of the Gaussian at the q-th quantile, and $\mathbb{V}_n[\{y_i\}_i]$ is the empirical variance of the Monte Carlo samples of the model.