Departmental Coversheet

MSc in Computer Science 2020-2021

Project Dissertation

Project Dissertation title: Conformal time-series forecasting

Term and year of submission: Trinity 2021

Candidate Number: 1047602

Conformal time-series forecasting

Candidate number: 1047602

University of Oxford

A dissertation submitted in partial fulfilment of
the requirements for the degree of

Master of Science

Trinity 2021

Abstract

Current approaches for (multi-horizon) time-series forecasting using recurrent neural networks (RNNs) focus on issuing point estimates, which are insufficient for informing decision-making in critical application domains wherein uncertainty estimates are also required. Existing methods for uncertainty quantification in RNN-based time-series forecasts are limited as they may require significant alterations to the underlying model architecture, may be computationally complex, may be difficult to calibrate, may incur high sample complexity, and may not provide theoretical guarantees on the frequentist coverage of the issued uncertainty intervals. In this work, we extend the inductive conformal prediction framework to the time-series forecasting setup, and propose a lightweight uncertainty estimation procedure to address the above limitations. With minimal exchangeability assumptions, the proposed approach provides uncertainty intervals with theoretical guarantees on frequentist coverage for *any* multi-horizon forecast predictor and *any* dataset. We demonstrate the effectiveness of the conformal forecasting framework by comparing it against existing baselines on a variety of synthetic and real-world datasets.

Contents

1 Introduction

Time-series forecasting tasks are central to a broad range of application domains, including stock price predictions [1, 2], service demand forecasting [3, 4], and medical prognoses [5, 6, 7]. Recurrent neural networks (RNNs) and their variants (e.g., LSTM, GRU, etc.) constitute an instrumental class of models that are most commonly used to carry out time-series forecasting tasks [8, 9]. These models, however, are usually used to issue *point* predictions—i.e., singular estimates of the future values of a time-series. In many high-stakes applications—such as finance and medicine—these are not enough; we also need estimates of uncertainty that would be used for accurate risk assessment when using a model's forecasts to inform decisions [10]. For example, clinical practitioners need to make treatment decisions accounting for all potential scenarios, where less likely scenarios may have graver consequences and require more care compared to the more likely scenarios [11, 12].

While various methods for uncertainty estimation in standard feed-forward neural networks have been recently proposed [13, 14, 15], equivalent methods for RNN-based time-series models are still under-explored. Existing solutions include Bayesian recurrent neural networks [16, 17, 18], quantile regression models [3, 19], latent variable models with deep state-space architectures [6, 20], and post-hoc uncertainty estimates using bootstrapping or jackknife procedures [21, 22]. Each of these solutions has its own limitations: Bayesian models may be difficult to calibrate, bootstrapping methods scale poorly for RNNs with large number of parameters, and quantile predictors may "overfit" their uncertainty estimates. Almost all existing methods share at least one of the two major drawbacks: (1) they require substantial modifications to the underlying model architecture, and (2) they provide no theoretical guarantees on frequentist coverage, with the exceptions being computationally intractable.

In this work, we aim to address the above limitations by extending *conformal prediction* (CP) [23, 24]—a framework designed to construct prediction intervals in classification and regression problems with *guaranteed* finite-sample frequentist coverage rates from scalar, exchangeable observations—to the time-series forecasting setup, where the observations and predictions involve temporally dependent, potentially multivariate sequences that and are not, in general, directly comparable due to differences observation lengths, irregular observation frequencies, and variations in temporal dynamics (whereas comparison between training points is a key step in conformal prediction). We derive a novel, computationally efficient *conformal forecasting* framework that can leverage *any* underlying point-forecasting model to produce upper and lower bounds for multi-step predictions, so that frequentist coverage holds jointly across the entire prediction horizon. We focus on RNN-based conformal forecasting architectures, which we call *conformal recurrent neural networks*, and explore their effectiveness in providing *valid* and *efficient* coverage intervals.

Outline

The rest of this work is organised as follows. In Chapter 2, we formalise the uncertainty quantification in multi-horizon time-series forecasting problem and discuss current deep learning literature for deriving the uncertainty intervals. In Chapter 3, we introduce the conformal prediction framework and show how it can be extended to time-series forecasts, proposing a novel conformal recurrent neural network architecture. In Chapters 4 and 5, we present the implementations of baseline and proposed architectures, and compare performance of predicted coverage intervals in synthetic and real-world settings. In Chapter 6, we briefly explore a technique for improving conformal forecasting performance. We conclude with Chapter 7.

2 Background and related work

In this chapter, we formalise the multi-horizon time-series forecasting problem and its uncertainty quantification in terms of frequentist multi-horizon coverage. We then discuss the broad classes of approaches that have been proposed in the previous deep learning literature in order to tackle this problem; we summarise the relative advantages and limitations of the most popular models, thereby motivating the need for a more powerful framework for uncertainty quantification in time-series forecasting, which is the subject of this work.

2.1 Multi-horizon time-series forecasting

Let $y_{t:t'} = (y_t, y_{t+1} \ldots, y_{t'})$ be a time-series of d-dimensional observations $y_t, \ldots, y_{t'} \in \mathbb{R}^d$ that start at time step t and end at time step t'. The goal of the *multi-horizon time-series forecasting problem* is to predict future values

$$\hat{y}_{(t'+1):(t'+H)} = (\hat{y}_{t'+1}, \ldots, \hat{y}_{t'+H}) \in \mathbb{R}^{H \times d}, \tag{2.1}$$

where H is the number of steps to be predicted (the *prediction horizon*), given the history of observed values $y_{1:t'}$.

2.2 Frequentist coverage

For critical applications, we are interested in the uncertainty associated with the forecast—for each time step h in the prediction horizon, we would like to obtain prediction intervals of the form $[\hat{y}^L_{t+h}, \hat{y}^U_{t+h}],\ h \in \{1, \ldots, H\}$, so that the ground truth value y_{t+h} is contained in the interval with a sufficiently high probability. This is illustrated in Figure 2.1 below.

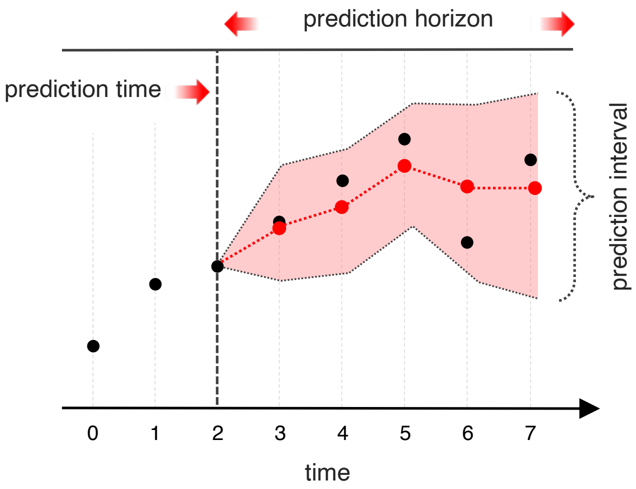


Figure 2.1: **Depiction of the forecasting setup with uncertainty intervals.** Black dots indicate the ground truth observations and red dots represent the predictions of the point forecaster; the shaded region is the associated uncertainty interval.

One way to quantify how well the predicted forecast intervals capture the true series is to fix a desired *frequentist joint coverage level* $1 - \alpha$ (with the corresponding *error rate* or *significance level* α), such that the ground-truth values of the *entire time-series trajectory* are contained within the intervals; i.e.,

$$\mathbb{P}\left[y_{t+h} \in [\hat{y}^L_{t+h}, \hat{y}^U_{t+h}],\ \forall h \in \{1, \ldots, H\}\right] \geq 1 - \alpha. \tag{2.2}$$

Alternatively, we can define *independent coverage* for each time-step in the prediction horizon separately; for a coverage level $1 - \alpha$,

$$\mathbb{P}\left[y_{t+h} \in [\hat{y}^L_{t+h}, \hat{y}^U_{t+h}]\right] \geq 1 - \alpha,\ \forall h \in \{1, \ldots, H\}. \tag{2.3}$$

2.3 Bayesian recurrent neural networks

In this and following sections, we present the broad classes of approaches that have been previously proposed in the deep learning literature to tackle the uncertainty estimation problem in the time-series setting. We focus on recurrent neural network (RNN)-based architectures as they are adapted for sequential data processing and are therefore the most commonly applied architecture type for time-series forecasting purposes.

We start with **Bayesian recurrent neural networks** [16, 17, 18], which extend the ideas of Bayesian inference for standard feed-forward neural networks to the RNN architectures. The key idea of Bayesian neural networks (BNNs) is to express the model (*epistemic*[1]) uncertainty through distributions on model parameters [25, 26]. More formally, given a dataset $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$ consisting of observations $\mathbf{x}^{(i)}$ and targets $y^{(i)}$, and a feed-forward neural network parameterised by weights $\mathbf{w}$, the *learning* process involves the computation of a *posterior* distribution of the weights having observed the dataset:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})p(\mathbf{w})}{p(\mathcal{D})}, \tag{2.4}$$

where $p(\mathbf{w})$ is the *prior* distribution of the weights, $p(\mathcal{D}|\mathbf{w})$ is the *likelihood* of the observed dataset for a given model parameterisation, and the term $p(\mathcal{D})$ represents the *evidence*. We capture the model uncertainty for a new, previously unobserved example $\mathbf{x}'$ using a *posterior predictive probability distribution* over the output space y, denoted $p(y|\mathbf{x}', \mathcal{D})$. In BNNs, this distribution can be obtained by marginalising

---

[1] Contrast with *aleatoric* uncertainty on the uncertainty of the data [12].

out the parameters:

$$p(y|\mathbf{x}', \mathcal{D}) = \mathbb{E}_{\sim p(\mathbf{w}|\mathcal{D})}[p(y|\mathbf{x}', \mathbf{w}, \mathcal{D})] = \int p(y|\mathbf{x}', \mathbf{w}, \mathcal{D}) p(\mathbf{w}|\mathcal{D}) \, \mathrm{d}\mathbf{w}. \tag{2.5}$$

However, even with moderate size neural networks containing thousands of parameters and trained on datasets of similar orders of magnitude, exact Bayesian inference quickly becomes infeasible; various approximation techniques become necessary to obtain the posterior weight distributions and predictions in practice. Two broad classes of solutions to this include Monte Carlo sampling and variational inference. *Markov chain Monte Carlo* (MCMC) approach approximates the posterior through sampling methods where subsequent samples eventually become marginally distributed according to the target distribution—for example, through simulating the Hamiltonian dynamics of the system [27, 28], with additional techniques [29, 30] for improving state exploration or reducing the computational costs for large datasets. *Variational inference* is another general approach which works by approximating the intractable posterior distribution $p(\mathbf{w}|\mathcal{D})$ with a simpler distribution $q_\lambda(\mathbf{w})$ that is characterised by its variational parameters λ. The variational parameters are then optimised to minimise the *variational free energy*, which is a cost function corresponding to the distance between the variational and true posterior distributions [31]:

$$\mathcal{F}(\mathcal{D}, \lambda) = \mathrm{KL}[q_\lambda(\mathbf{w}) \parallel p(\mathbf{w})] - \mathbb{E}_{q_\lambda(\mathbf{w})}[\log p(\mathcal{D}|\mathbf{w})], \tag{2.6}$$

where KL denotes the Kullback-Leibler divergence. Hinton and Van Camp [32] and Graves [33] presented gradient-based optimisation procedures by reformulating $\mathcal{F}$ in terms of the minimum description length (MDL) loss function and applying that to limited classes of prior and posterior distributions; Kingma and Welling [34], Hoffman et al. [35] have introduced *stochastic* techniques to variational inference. Blundell et al. [31] improved and generalised the previous approaches with a more efficient procedure known as Bayes by Backprop, where optimisation of $\mathcal{F}$ builds directly on the standard back-propagation algorithm and can be used with more choices of distributions; Fortunato et al. [16] extended Bayes by Backprop to recurrent neural networks. In another line of research, Gal and Ghahramani [36] have shown that the existing *dropout* technique, where the weights of the neural network are randomly zeroed out (dropped) with each forward pass, corresponds to an approximation of a probabilistic deep Gaussian process [37]. This motivates the use of dropout as a Bayesian technique for representing model uncertainty: the posterior distributions on outputs are obtained by using dropout to first train the neural network, and then to obtain multiple output samples as different units are dropped at the evaluation stage (the technique known as *MC dropout*). Gal and Ghahramani [38] further show a theoretically grounded application of dropout in recurrent neural networks. The main advantage of MC dropout is that it bypasses the need to double the number of training parameters (as each weight of a standard neural network needs to be represented by two parameters of a probability distribution in a BNN) and the only modification to the training procedure is the addition to dropout layers; however, while this is computationally efficient, the calibration of MC dropout models is challenging [21]; as such models account for *risk* (inherent stochasticity of the model) rather than the *uncertainty* over its parameters [39]. In addition, even the exact Bayesian inference does not result in intervals with the *frequentist* coverage as described in Equations (2.2) and (2.3) [21, 40].

2.4 Quantile recurrent neural networks

Another line of research concerning uncertainty estimation in time-series forecasts is based on **quantile recurrent neural networks** [19, 3]. Quantile RNNs could be thought of as a deep neural network extension of *quantile regression* [41] for sequential inputs—for each horizon h and some quantile a, the quantile regression model would predict such $\hat{y}_{t+h}^{(a)}$ that

$$\mathbb{P}[y_{t+h} \leq \hat{y}_{t+h}^{(a)} \mid y_{1:t}] = a. \tag{2.7}$$

In other words, instead of returning a series of point estimates across the prediction horizon, quantile RNNs learn the prediction intervals directly, learning both the upper and lower bounds of the forecast (i.e. $[\hat{y}_{t+h}^{L}, \hat{y}_{t+h}^{U}]$ from Section 2.2) as separate targets. While standard quantile regression can be solved analytically, in deep neural networks this can only be done via optimisation (gradient descent) methods. We replace the standard mean squared error (MSE) loss function with the generalised *quantile* (or *pinball*) *loss* function, defined as:

$$\mathcal{L}_a(y, \hat{y}) = a(y - \hat{y})_+ + (1 - a)(\hat{y} - y)_+, \tag{2.8}$$

where $(\cdot)_+ = \max(0, \cdot)$ (see e.g. [3] for details). A recent quantile RNN multi-horizon time-series forecasting model is the *multi-horizon quantile RNN* (MQ-RNN) model developed by Wen et al. [3], which has been successfully applied for large-scale forecasting in several real-world domains. At each step, MQ-RNN combines the RNN-generated state vector with some time-step specific static data to generate 1) a new state vector (using a global MLP) and 2) sets of quantiles, one for each time-step in the horizon (using horizon-specific local MLPs). However, since the quantiles are obtained separately from each other and are optimised using gradient descent rather than analytical methods, they are not mathematically rigorous. By having no mathematical constraints on the quantiles, MQ-RNNs are susceptible to problems such as *quantile crossing*, such that

$$\exists a < b.\, \hat{y}_{t+h}^{(a)} > \hat{y}_{t+h}^{(b)},$$

resulting in less extreme quantiles returning wider intervals compared to the more extreme quantiles. A more recent *spline quantile function RNN* (SQF-RNN) model, developed by Gasthaus et al. [19], attempts to avoid this problem, as well as the general limitation of learning different quantiles separately. SQF-RNN learns to model the *quantile function*—mapping each quantile a to its corresponding $\hat{y}_{t+h}^{(a)}$—directly, by representing the quantile function with mathematically constrained spline curves and training the RNN to return the *parameters* defining those curves rather than the target quantiles themselves. Instead of the quantile loss function, the training objective is now the continuous-ranked probability score (CPRS), which measures the compatibility of the spline curve-defined quantile function F^{-1} with an observation y, and corresponds to the quantile loss aggregated over all quantiles:

$$\mathrm{CPRS}(F^{-1}, y) = \int_0^1 2\mathcal{L}_a(F^{-1}(a), y)\,\mathrm{d}a. \tag{2.9}$$

However, both MQ-RNN and SQF-RNN do not provide any *theoretical* guarantees on the accuracy of the learnt quantile distribution; they are still at risk of *overfitting the quantiles* themselves and be confident about their (wrong) quantile estimates. This is especially true considering their *low sample efficiency* [21], which means that in low data settings—where precise uncertainty estimation arguably matters the most—the quality of the learnt quantiles suffers.

2.5 Deep state-space models

State-space models (SSMs) represent the time-series in terms of its *latent state*, which in turn captures important information about the structural components of the series, such as its level, trend, and seasonality [42]. The evolution of the latent states in an SSM is determined by its *transition matrix* (which is deterministic), the *innovation matrix* (which is stochastic), as well as any other model-specific parameters, while the target values are determined from the latent state using a probabilistic *observation model* [20]. This is a mathematically principled approach to parameterise the dynamics of a single time-series; to extend this to sets of *multiple* time-series (see observation paradigms in Figure 2.2, discussed in more detail in Section 2.7), *deep* state-space models [20] use RNNs to obtain the SSM that is *shared* between the series in the dataset—in other words, the underlying RNN learns a *single* parameterisation of transition and innovation matrices, the observation model, and other parameters. Most of the SSM-based methods are limited to various structural assumptions about the time-series dynamics, such as Markovianity, where it is assumed that the transition dynamics stay constant over time. In some other cases the problem domain does not allow for decoupling of observations and state transitions; Alaa and van der Schaar [6] attempt to address these limitations through *attentive* state-space models (ASSMs), where the state transitions depend on the *entire* history, rather than just the previous latent state. The transition dynamics between the latent states $p(\boldsymbol{l}_t|\boldsymbol{l}_{t-1})$ as well as the observations $p(y_t|\boldsymbol{l}_t)$ are probabilistic—in case of the ASSM, the posterior distribution is determined through variational inference; in simpler models the distribution is generally approximated through Monte Carlo methods. Sampling potential trajectories can therefore represent the uncertainty of the model. However, most SSM-based models have important limitations of requiring simplifying assumptions (such as Markovian dynamics), or being tailored to a specific application domain (like ASSMs are), which limits their application to general, assumption-free time-series modelling and forecasting problems.

2.6 Frequentist uncertainty estimators

To tackle the difficulties in calibration of Bayesian models, the lack of coverage guarantees of quantile models, and the limiting assumptions of deep state-space models, more recent works have explored the potential of **frequentist uncertainty estimators** that are based on variations of jackknife resampling [43, 44]. Some notable models in this direction include the *discriminative jackknife* introduced in Alaa and van der Schaar [22] and the *blockwise jackknife* introduced in Alaa and van der Schaar [21]. The former is a general procedure to make the confidence intervals produced by classical jackknife satisfy both coverage and discrimination (adaptiveness to high and low-confidence prediction) requirements [44]. It is based on leave-one-out resampling; however, rather than retraining the models on each resampled dataset (which would be too expensive for deep neural networks), it instead uses *influence functions* to determine the *change* in neural network weights that training on a resampled dataset would result in. To make the intervals adaptive, confidence intervals are constructed accounting both for the marginal model generalisation error (based on leave-one-out residuals) and for the local prediction variability (based on the effects a particular training point has on prediction errors in the test dataset). The *blockwise jackknife recurrent neural network* (BJ-RNN) [21] model is based on a similar procedure that is adapted to RNNs trained on sequential (time-series) data. Here, the "perturbed" RNN models are again computed using influence functions, but alternative datasets are generated not only by removing individual observations of the entire time-series, but also by removing intervals of particular time-steps *across* the time-series observations. At test time, all the perturbed models are run on a testing point, thereby obtaining a distribution of predictions that represent the coverage interval. The BJ-RNN model has been shown to outperform the MQ-RNN (Section 2.4) and MC dropout RNN (Section 2.3) models while having theoretical guarantees, better sample efficiency and additionally being a *post-hoc* procedure (i.e. it can be applied on top of *any* RNN model *without* changing the underlying architecture or compromising its performance). However, it has the key limitation of a prohibitive time and space complexity—for P parameters of the underlying RNN, the exact procedure involves estimation of the inverse Hessian matrix, which takes

$O(P^3)$ time, while available approximations, such as stochastic Hessian-vector product estimation used in the paper—following the methods of Pearlmutter [45], Agarwal et al. [46]—may compromise the correctness of results. Moreover, since all perturbed models have to be evaluated for every example even at the evaluation stage, and because of the large number of possible blockwise perturbations, the BJ-RNN model does not scale beyond small datasets containing few short time-series.

2.7 Conformal prediction-inspired frameworks

Our proposed method builds on conformal prediction (CP), a general framework aiming to determine the levels of confidence for *any* predictive model using past experience [23, 24]. For a given confidence or coverage level $1 - \alpha$ and a learning algorithm producing a point estimate $\hat{y}$, the CP algorithm returns a *prediction region* that is *guaranteed* to contain the true value y with probability of at least $1 - \alpha$ (thereby providing frequentist coverage). While it has been initially developed for an *online* prediction setting of *accumulating* datasets, it can also work in the *batch* setting when a separate *calibration* dataset is available (an approach called *inductive* conformal prediction; ICP [47, 48]). The conformal prediction framework will be discussed in more detail in Chapter 3.

To our knowledge, little work has been done towards applying CP methods for time-series forecasting. The main challenge is that conformal prediction assumes *exchangeability*, so that any permutation of the observations in the dataset is equally likely. However, within a *single* time-series, the time-steps $(y_1, \ldots, y_t)$ are inherently non-exchangeable, since their ordering and temporal dependencies is exactly what constitutes a time-series (see the left panel in Figure 2.2 below). Therefore, naïvely applying conformal prediction to a single time-series to derive prediction intervals for future observations is not methodologically valid, and any frequentist coverage guarantees are lost (although there have been works, such as Kowalczewski [49], that have nevertheless attempted to do this). One notable exception is the recent *Ensemble Batch Prediction Interval* (EnbPI) algorithm, introduced in Xu and Xie [50]. EnbPI bypasses the exchangeability assumption required by the standard CP framework—while still providing *approximately* valid prediction intervals—by using an ensemble of bootstrapped estimators, each of them trained on a fixed-length section of the time-series with one of the time-steps removed. However, this method is still limited and does not fully account for the complexity of the time-series forecasting problem. First, as discussed in Section 4.1 of Xu and Xie [50], EnbPI introduces certain assumptions—such as the strongly mixing error process—which may limit its applicability to some types of forecasting problems. In addition, we argue that the paradigm of learning from a *single* time-series—while useful in cases where indeed only one time-series is available—may not be optimal in settings where datasets contain *multiple* time-series, the *shared* patterns of which could potentially be exploited. The distinction between the two paradigms is illustrated in Figure 2.2. To the best of our knowledge, no existing method has applied conformal prediction to the latter setting; yet the datasets of multiple time-series are very common in domains such as healthcare, where each independent patient would have their own time-series of clinical encounters or vital sign measurements. While methods such as EnbPI would have to learn a *separate* bootstrapped ensemble for *each* new patient, being able to learn from *all* patients at once is arguably more useful—in this way, information in one time-series can be leveraged to produce better forecasts in another.

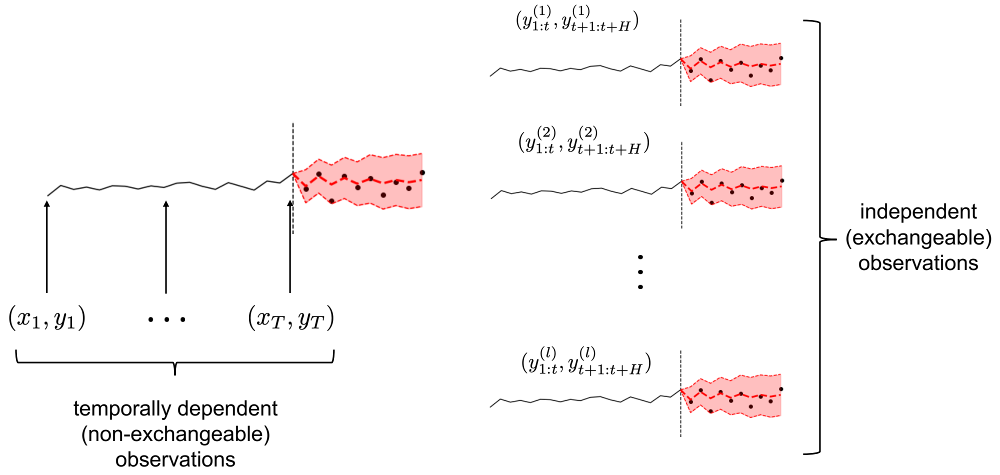

(x_1, y_1)
$\cdots$
(x_T, y_T)
temporally dependent (non-exchangeable) observations
$(y^{(1)}_{1:t}, y^{(1)}_{t+1:t+H})$
$(y^{(2)}_{1:t}, y^{(2)}_{t+1:t+H})$
$(y^{(l)}_{1:t}, y^{(l)}_{t+1:t+H})$
independent (exchangeable) observations

Figure 2.2: **Time-series observation paradigms.** (Left) The dataset is assumed to comprise a *single* time-series, with observations being individual time-steps within the time-series. These observations are temporally dependent. (Right) The dataset consists of a *set* of independent time-series, where the entire series is treated as an observation. Independence of time-series implies their exchangeability.

Summary

We summarise the most popular RNN-based probabilistic forecasting methods in Table 2.1 below. The proposed *conformal recurrent neural network* (CoRNN) is introduced and explained in more detail in the following Chapter.

Table 2.1: **Overview of the most popular RNN-based probabilistic forecasting methods.** For reference we include the proposed CoRNN model.

Method	**Paradigm**	**Architecture**	**Time-series observations**	**Frequentist coverage**
Bayesian RNNs [16]	Bayesian	Built-in	Multiple	—
MC dropout [36]	Bayesian (approx.)	Built-in	Multiple	—
MQ- [3], SQF-RNN [19]	—	Built-in	Multiple	—
ASSM [6]	—	Built-in	Multiple	—
BJ-RNN [21]	Frequentist	Post-hoc	Multiple	$1-2\alpha$
EnbPI [51]	Frequentist	Built-in	Single	$1-\alpha$
CoRNN (proposed)	Frequentist	Post-hoc	Multiple	$1-\alpha$

3 Conformal recurrent neural networks

In this chapter, we introduce the conformal recurrent neural network (CoRNN) architecture. We start off by providing further background on conformal prediction, and then extend the framework to the time-series forecasting setup while maintaining joint coverage guarantees.

3.1 Conformal prediction

In the classical *conformal prediction* (CP) framework [23, 24, 52], given a set of observations $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{l}$, a new example $\mathbf{x}^{(l+1)}$, and the error rate α (equivalently, coverage level $1 - \alpha$), the aim is to return a *prediction region* (a prediction *set* for classification or a prediction *interval* for regression) Γ^{α} such that, with probability of at least $1 - \alpha$, the true label $y^{(l+1)}$ is contained within Γ^{α}:

$$\mathbb{P}[y^{(l+1)} \in \Gamma^{\alpha}] \geq 1 - \alpha. \tag{3.1}$$

The conformal prediction framework is *distribution-free* (i.e. it does not have any assumptions on the distribution of the underlying data), and applies to *any* underlying predictive model, as long as the *exchangeability* assumption is satisfied:

Assumption 1. (*Exchangeability*) In a dataset of l observations $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{l}$, any of its $l!$ permutations are equiprobable. Note that independent identically distributed (iid) observations satisfy exchangeability.

The CP framework has originally been designed for the *online* setting, where data is observed continuously and prediction regions for the next observation are recomputed whenever new data becomes available. The regions are estimated as follows. For a new example $\mathbf{x}^{(l+1)}$ and its *potential* label y, we estimate how unusual (*nonconforming*) would the full observation $(\mathbf{x}^{(l+1)}, y)$ be compared to the the already observed dataset $\mathcal{D}$ (defined as before). Following the notation in Zeni et al. [52], such *nonconformity score* for the new example $(\mathbf{x}^{(l+1)}, y)$ is denoted as

$$R_{l+1} = A(\mathcal{D}, (\mathbf{x}^{(l+1)}, y)), \tag{3.2}$$

and, in the more general case,

$$R_i = A\left(\mathcal{D} \setminus \{(\mathbf{x}^{(i)}, y^{(i)})\}, (\mathbf{x}^{(i)}, y^{(i)})\right). \tag{3.3}$$

The alternative of including the new example in the dataset and computing $R_{l+1} = A(\mathcal{D} \cup \{(\mathbf{x}^{(l+1)}, y)\}, (\mathbf{x}^{(l+1)}, y))$ and $R_i = A\left(\mathcal{D}, (\mathbf{x}^{(i)}, y^{(i)})\right)$ is also possible. The choice for which alternative is used depends on the application—the latter will be used for regression-based conformal predictors in the next section. Theoretical guarantees remain in either case. See Zeni et al. [52] for details.

Now, since observations are exchangeable, we can compute the *empirical nonconformity score distribution* for each example in the dataset, $\{R_i\}_{i=1}^{l}$, as well as the nonconformity core for the new example R_{l+1}. We compute the *p-value* of the nonconformity of the new example,

$$p_{(\mathbf{x}^{(l+1)}, y)} = \frac{|\{i = 1, \ldots, l+1 : R_i \geq R_{l+1}\}|}{l+1}, \tag{3.4}$$

which intuitively represents the fraction of examples within the observed dataset $\mathcal{D}$ that are more nonconforming than the new example $\mathbf{x}^{(l+1)}$ with the assumed label y. A small p-value implies that the potential label y is very unusual and is unlikely to be in the prediction set; a large p-value means that the potential label y is typical and should be included in the prediction set. The threshold of the p-value depends on the desired coverage level; we include in the prediction set all labels y the p-values of which are larger than the target *error rate* α:

$$\Gamma^{\alpha}(\mathbf{x}^{(l+1)}) = \left\{ y : p_{(\mathbf{x}^{(l+1)},y)} > \alpha \right\} . \tag{3.5}$$

3.2 Regression nonconformity scoring

The key property of conformal prediction is that, as long as the assumption of exchangeability is satisfied, the prediction regions are always *valid*, *regardless of the choice of the nonconformity score.* Following Proposition 2.1 in Zeni et al. [52]:

Property 1. (*Validity*) Under the exchangeability assumption, any conformal predictor will return the prediction region $\Gamma^{\alpha}(\mathbf{x}^{(i)})$ such that the probability of error $y^{(l+1)} \notin \Gamma^{(\alpha)}$ is not greater than α. Equivalently, Equation (3.1) is satisfied.

We note that this probability refers to the correctness of the conformal prediction *procedure*, rather than that of the *prediction region*; i.e. it is the procedure that has the probability $1 - \alpha$ of being correct (before any data is observed), rather than the returned prediction region Γ^{α}. For further discussion of this see Shafer and Vovk [24].

While validity is achieved by *any* conformal predictor (including one where nonconformity scores are obtained using a random number generator), the intervals of some conformal predictors will be wider—and less informative for real-world use cases—than others (infinite-width intervals or sets containing all classes will always achieve perfect coverage). We therefore wish to obtain intervals that are also *efficient*, i.e. they achieve the desired coverage with *minimal interval width.* Finding the most efficient conformal predictor (the corresponding nonconformity score) is in general difficult, as this may depend on the exact properties of the dataset. The most sensible and widely used choice for regression problems—which the time-series forecasting problem is an example of—is the nonconformity score of the form

$$A(\mathcal{D}, (\mathbf{x}^{(i)}, y^{(i)})) = \Delta(M(\mathbf{x}^{(i)}|\mathcal{D}), y^{(i)}), \tag{3.6}$$

where M is the *underlying* (auxiliary) model trained on the dataset $\mathcal{D}$ (applied to the example $\mathbf{x}^{(i)}$ to give a prediction $\hat{y}^{(i)}$), and Δ is some *distance metric.* The choice for M depends on the dataset and the problem; although any regression model, including neural network architectures, is compatible with the framework. As for the distance metric, when $\Delta(\hat{y}, y) = |\hat{y} - y|$ we have

$$R_i = |\hat{y}^{(i)} - y^{(i)}| \tag{3.7}$$

which corresponds to the *residual error* between the prediction of the underlying model and the true label, which is particularly useful for *inductive* conformal prediction, as we discuss below.

3.3 Regression inductive conformal prediction

While the standard conformal (*transductive*[1]) predictors can work well for small accumulating datasets with *discrete* labels y (such as in classification problems), this is not computationally feasible for *real-valued* dependent variables (in regression problems, including time-series forecasting), as all $y \in \mathbb{R}$ cannot be individually tested for their p-value. In addition, the underlying model M cannot be efficiently retrained with each new available example, especially when M is a complex model

---

[1] Alternative definitions for "transductive" predictors exist. We refer to the standard setting as *transductive* in contrast to the *inductive* conformal prediction, following the categorisation in Zeni et al. [52]. For further discussion see Vovk [53].

such as a neural network. To tackle these challenges, the procedure is modified to work in the inductive setting, where instead of computing the prediction intervals directly from the dataset and a single example, considering each possible label, a general rule is derived according to which every $y \in \mathbb{R}$ can be correctly assigned to be either within or outside of the prediction interval.

In this new *inductive conformal prediction* (ICP) setup [54, 55], the training set is split into the *proper training set* of size n and a *calibration set* of size m: $\mathcal{D} = \mathcal{D}_{\text{train}} \cup \mathcal{D}_{\text{cal}}$. The true training set is used to train the underlying predictive model M, and the calibration set is used to obtain the nonconformity scores for each calibration example $\mathbf{x}^{(n+i)}$ with label $y^{(n+i)}$:

$$R_i = |y^{(n+i)} - M(\mathbf{x}^{(n+i)})|, \quad \forall i \in 1, \dots, m, \tag{3.8}$$

Now, instead of computing the distribution of p-values, we use the distribution of the calibration nonconformity scores (or residuals) $\{R_i\}_{i=1}^m$. We observe that the choice of the distance metric $\Delta(\hat{y}, y) = |\hat{y} - y|$ allows us to derive the set of values of y that will satisfy Equation (3.5): in particular, we compute a *critical nonconformity score* $\hat{\varepsilon}$ that is the $\lceil (m+1)(1-\alpha) \rceil$-th smallest residual [52] (or, equivalently, the $\min((m+1)(1-\alpha)/m, 1))$-th quantile of the calibration nonconformity score distribution; with $(n+1)/n$ term correcting for the finite sample). Our prediction intervals are thus given by:

$$\Gamma^\alpha(\mathbf{x}^{(l+1)}) = [\, \hat{y}^{(l+1)} - \hat{\varepsilon}, \hat{y}^{(l+1)} + \hat{\varepsilon}\,], \tag{3.9}$$

where $\hat{y}^{(l+1)} = M(\mathbf{x}^{(l+1)})$.

3.4 ICP for time-series forecasting

So far we have considered the case when the labels $y \in \mathbb{R}$ are scalar, but multi-horizon time-series forecasts return H (d-dimensional) values (see Section 2.1). We now extend the ICP framework to handle the multi-horizon forecasting setup, while maintaining the validity of the resulting multi-horizon forecast intervals. In this work, we focus on $d = 1$.

Let $\mathcal{D}$ be the set of exchangeable observations of the form $(y_{1:T}, y_{T+1:T+H})$, where $y_{1:T}$ is the time-series consisting of T observed steps, and $y_{T+1:T+H}$ is the H-step forecast. Note that the label $y_{T+1:T+H}$ is now a H-dimensional value, in contrast with the scalar y value from before. Due to the sequential nature of our task and complex multidimensional inputs, we will use an RNN as the underlying model M. We set M to produce multi-horizon forecasts *directly*—where at each time step t the model predicts all values of the H-step target $y_{t+1:t+H}$ directly from a single embedding—rather than *recursively*—where a single prediction is obtained at a time, and successive values are obtained by iteratively feeding the predictions back into the RNN. This distinction is illustrated in Figures 3.1 and 3.2.

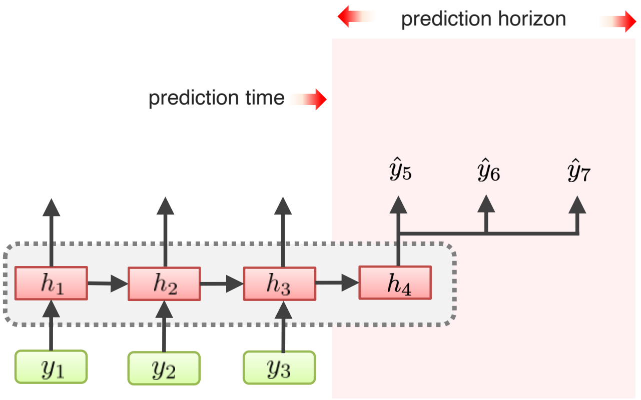


Figure 3.1: **Direct multi-horizon forecasting strategy.** The values for the 3-step forecast $\hat{y}_{5:7}$ are obtained at the same time from a single hidden state h_4.

We motivate our choice of the direct strategy by it being more robust to error accu-

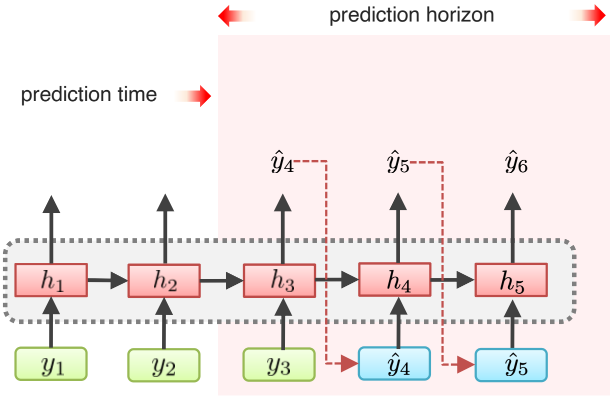


Figure 3.2: **Recursive multi-horizon forecasting strategy.** The forecast $\hat{y}_{4:6}$ is obtained by successively feeding in the previous predictions back into the RNN.

mulation [56, 3], and the predictions being independent conditional on the internal time-series embedding stored in M—this will allow us to maintain the theoretical guarantees for joint prediction interval coverage. Now we replace the single-dimensional nonconformity score defined earlier by its H-dimensional counterpart,

$$R_i = \left[|y^{(i)}_{t+1} - \hat{y}^{(i)}_{t+1}|, \ldots, |y^{(i)}_{t+H} - \hat{y}^{(i)}_{t+H}| \right]^\top \tag{3.10}$$

where $\left[\hat{y}^{(i)}_{t+1}, \ldots, \hat{y}^{(i)}_{t+H}\right]^\top = M(y^{(i)}_{1:t})$. Since we produce H predictions that are conditionally independent given the internal state of the underlying model, we can apply Bonferroni correction to the obtained critical calibration scores by dividing the desired error rate α by H, so that the critical nonconformity scores $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_H$ are the $\min((m+1)(1-\alpha/H)/m, 1)$-th quantiles in the corresponding nonconformity score distributions. The resulting set of prediction intervals is then

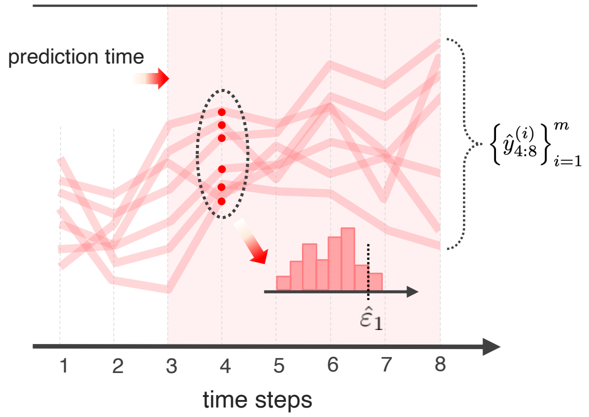
$$\Gamma^\alpha_1\left(y^{(l+1)}_{(1:t)}\right), \ldots, \Gamma^\alpha_H\left(y^{(l+1)}_{(1:t)}\right), \tag{3.11}$$

where

$$\Gamma_h^{\alpha}\left(y_{(1:t)}^{(l+1)}\right) = \left[\hat{y}_{t+h}^{(l+1)} - \hat{\varepsilon}_h, \hat{y}_{t+h}^{(l+1)} + \hat{\varepsilon}_h\right] \quad \forall h \in \{1, \ldots, H\}. \tag{3.12}$$

In summary, the resulting conformal recurrent neural network (CoRNN) model is an RNN that issues a point prediction for its forecast, along with a prediction region that covers the forecast trajectory. The entire procedure for constructing prediction intervals in CoRNN is illustrated in Figure 3.3 and summarized in Algorithm 1.

(a) Construction of critical nonconformity scores



(b) Forecasting on a new time-series

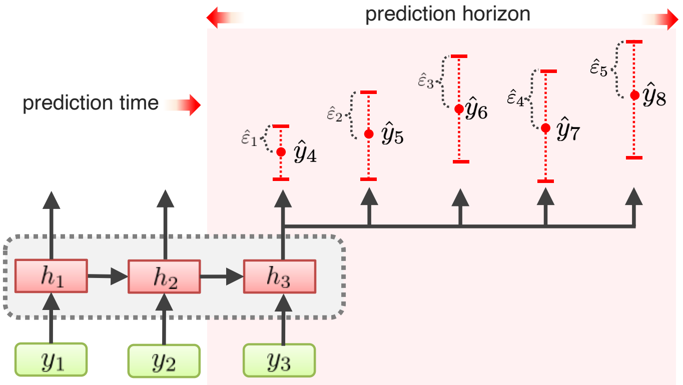


Figure 3.3: **CoRNN uncertainty estimation procedure**. (a) The calibration set is used to obtain the empirical distribution of nonconformity scores $\hat{\varepsilon}_h$, and its appropriate quantile is selected depending on the desired target coverage level. (b) Critical nonconformity scores are used to obtain the prediction interval.

Algorithm 1 Conformal recurrent neural network (CoRNN)

```
Input: A trained model $M$ producing $H$-step forecasts,
          calibration dataset $\mathcal{D}_{\text{cal}} = \left\{(y_{1:t}^{(i)}, y_{t+1:t+H}^{(i)})\right\}_{i=1}^{m}$,
          target error rate $\alpha$, the size of the training set $n$.
Output: Critical nonconformity scores $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_H$.
------------------------------------------------------------
Initialize $\boldsymbol{\varepsilon}_1 = \{\}, \ldots, \boldsymbol{\varepsilon}_H = \{\}$.
for $i = 1$ to $m$ do
    $\hat{y}_{t+1:t+H}^{(i)} \leftarrow M(y_{1:t}^{(i)})$.
    for $h = 1$ to $H$ do
        $\boldsymbol{\varepsilon}_h \leftarrow \boldsymbol{\varepsilon}_h \cup \{|\hat{y}_{t+h}^{(i)} - y_{t+h}^{(i)}|\}$.
    end for
end for
for $h = 1$ to $H$ do
    // Bonferroni and finite sample correction
    $\hat{\varepsilon}_h \leftarrow \min((m+1)(1-\alpha/H)/m, 1)$-th quantile of $\boldsymbol{\varepsilon}_h$.
end for
return $\hat{\varepsilon}_1, \ldots, \hat{\varepsilon}_H$.
------------------------------------------------------------
For a new time-series example $y_{1:t}^{*}$:
    $\hat{y}_{t+1:t+H}^{*} \leftarrow M(y_{1:t}^{*})$.
    return intervals $\hat{y}_{t+1}^{*} \pm \hat{\varepsilon}_1, \ldots \hat{y}_{t+H}^{*} \pm \hat{\varepsilon}_H$.
```

3.5 CoRNN validity

Finally, we show the theoretical motivations behind our approach via the following Theorem, which provides validity for intervals obtained from Algorithm 1.

Theorem 1. *Let* $\mathcal{D} = \left\{\left(y_{1:t}^{(i)}, y_{t+1:t+H}^{(i)}\right)\right\}_{i=1}^{l}$ *be the dataset of exchangeable time-series observations and their H-step forecasts obtained from the same underlying probability distribution. Let M be the recurrent neural network predicting H-step forecasts using the direct strategy. For any significance level $\alpha \in [0,1]$, the intervals obtained by the ICP-based conformal forecasting algorithm will have the error rate of at most α; alternatively,*

$$\mathbb{P}\left(\forall h \in \{1, \ldots, H\}.\ y_{t+h} \in [\hat{y}_{t+h} - \hat{\varepsilon}_h, \hat{y}_{t+h} + \hat{\varepsilon}_h]\right) \geq 1 - \alpha. \tag{3.13}$$

Proof. Due to the direct forecasting strategy, every step in the horizon can be treated as a separate inductive conformal predictor that uses the same underlying model M (with the final predictions derived from the internal state being independent) and the same dataset $\mathcal{D}$. Independent validity of each of the H ICPs follows from Proposition 1 in Vovk [57]. Setting the error rate of each of the H ICPs to α/H and applying Boole's inequality we obtain that the combined error rate of the H-step forecaster is α as required. $\square$

Summary

In this chapter, we presented the main contribution of this work, which is an extension of the ICP framework to the multi-horizon time-series forecasting setting. We view the time-series forecasting problem as a multi-target regression problem, where at each time-step the input is the state of a recurrent neural network, and the output is the (direct) multi-horizon forecast. We use the RNN as the underlying model to the ICP procedure, which now returns multiple prediction intervals, one for every step of the horizon. Similar to the multiple hypothesis testing problem, to maintain validity of the predictions across the entire horizon simultaneously, we use the Bonferroni correction scheme to adjust the critical values of the empirical nonconformity score distributions. Our proposed CoRNN architecture addresses the limitations of related work discussed in Chapter 2: it is computationally efficient (as the only bottleneck is the training time for the underlying model), it does not require any modifications to the underlying predictive model, and it provides theoretical validity guarantees for any underlying multi-horizon forecast predictor and any dataset with minimal exchangeability assumptions.

4 Implementation details

We now briefly present the most important implementation aspects of the baseline CoRNN architecture and selected competing baselines.

4.1 Implementation of the CoRNN architecture

The proposed CoRNN architecture is an extension of the underlying RNN model with a conformal forecasting procedure, where the trained model is calibrated as described in Algorithm 1, Section 3.4 of the previous Chapter. We chose to implement the proposed architecture using PyTorch [58], one of the most popular deep learning frameworks with tensor-based computation, automatic differentiation and hardware acceleration support. PyTorch relies on Python, a language popular in the machine learning community especially for its variety of widely supported open source scientific computing and data science libraries, such as NumPy [59] and Pandas [60]. This choice of the language and framework, as we will elaborate further below, will make it easier to compare the proposed architecture against the open source PyTorch implementations of the competing baselines: with the same internal framework and data processing methods, the comparison can be more consistent and fair.

We implement `CoRNN` as a single class extended from `torch.nn.Module`; its arguments are described in Table 4.1 and the key methods are summarised in Table 4.2. In what follows, we describe additional considerations for the training procedure.

Table 4.1: **Summary of the inputs to the CoRNN architecture.**

Argument	**Description**
`embedding_size`	(Hyperparameter) Size of internal RNN state embeddings.
`input_size`	Number of observed time-series features (default = 1).
`output_size`	Number of features in the forecast d (default = 1).
`horizon`	Length of the prediction horizon H.
`error_rate`	Target error rate α.
`mode`	Internal RNN implementation: LSTM (default), GRU, RNN.

Table 4.2: **Key components of the CoRNN structure.**

Method	**Description**
`fit`	Trains the underlying RNN model on the training dataset.
`calibrate`	Calibrates the model by computing the nonconformity score distributions and critical scores.
`predict`	Returns upper and lower bounds of the multi-horizon forecast.
`evaluate_coverage`	Computes the coverage statistics for evaluation.

Training procedure Follows Algorithm 1 in the previous chapter. The underlying RNN is first trained on the training dataset, where the model is optimised via gradient descent methods (using the mean squared error loss and the Adam optimiser [61]) to return good direct multi-horizon point forecasts. Then the optimised RNN is run on the calibration dataset, where for every calibration example the nonconformity score (in this case, the absolute error between the true time-series and the RNN forecast) is computed. Critical calibration scores are derived. For each test example, the RNN is run to return the point forecast, and the critical values are added to either side of the point forecast at each step of the horizon to obtain the uncertainty interval.

Handling training sequences of different lengths CoRNN is designed to train on sequences of different lengths in order to make use of all the available history of observations (i.e. the t in each example $y_{1:t}^{(i)}$ might be different). However, the forecast length (horizon) remains fixed. It may be the case that the length of a particular time-series example is shorter or equal to the length of prediction horizon; then the correctness of part of the forecast (which has the length of the full horizon) is not possible to check against any ground truth value. To mitigate this discrepancy, the observation length is additionally tracked and the predictions are masked wherever they are unverifiable.

4.2 Implementation of baseline architectures

We compare the performance of the CoRNN model against three baselines: the Monte Carlo dropout-based RNN (DP-RNN) [36], the multi-quantile RNN (MQ-RNN) [3], and the frequentist blockwise jackknife RNN (BJ-RNN) [21]. We use these baselines as the most popular and representative examples of the different paradigms for uncertainty estimation (Bayesian, quantile and frequentist for DP-RNN, MQ-RNN and BJ-RNN respectively), and because these are the exact baselines that have been compared in the Alaa and van der Schaar [21] paper, where BJ-RNN has achieved state-of-the-art time-series uncertainty estimation. The baseline implementations have been adapted from the above-mentioned paper to maintain as much consistency as possible[1]. As part of this work, the most important changes that have been implemented include adaptations required to produce direct multi-horizon forecasts (as original implementations would predict one-step ahead only); in addition, the evaluation procedures have been modified to compute the joint coverage.

Hyperparameter selection In both CoRNN and competing baselines, the hyperparameters have been selected to be the same as those in the reference implementation [21]. For all the experiments discussed in the next Chapter, the hyperparameters are detailed in the Appendix A.

---

[1] The original code is available at https://github.com/ahmedmalaa/rnn-blockwise-jackknife.

4.3 Supplementary code

The code supplemented with this work contains three directories; `influence`, `models` and `utils`. The `influence` directory contains the additional methods required for BJ-RNN computation and have not been changed from the implementation in Alaa and van der Schaar [21]. The `models` directory contains all the baselines that will be compared in the following Chapter. The `models/cornn.py` file in particular contains the main contribution of this work; other models contain the necessary adaptations for the direct multi-horizon forecasting setting. Finally, the `utils` directory contains all the necessary code for synthetic dataset generation, preprocessing code of the real-world datasets to the different formats used by the various models, the training routines of synthetic and medical datasets, and code for the forecast evaluation. We do not include the saved models, results or datasets due to size constraints.

5 Experiments

In this chapter, we showcase the performance of the proposed architecture and competing baselines on a variety of synthetic and real-world medical datasets. We first present the quantitative and qualitative performance of CoRNNs against competing baselines on small synthetic datasets with controlled properties that allow for an in-depth comparison. We then apply the baselines to three real-world medical datasets.

5.1 Experiments on synthetic data

We first generate the synthetic time-series consisting of three components: the *autoregressive* process determining the trend of the time-series, the *seasonal* process representing periodic fluctuations in the time-series values, and the *noise* process representing the inherent (aleatoric) uncertainty of the dataset.

For a time-series of length T, this is expressed mathematically as:

$$y_t = \sum_{k=0}^{t} a^k \cdot x_k + \gamma_t + \epsilon_t, \quad \forall k \in \{1, \ldots, T\} \tag{5.1}$$

where $x_t \sim \mathcal{N}(\mu_x, \sigma_x^2)$, $a = 0.9$ is the memory parameter, γ_t is the stochastic seasonal process representing periodic fluctuations, and $\epsilon_t \sim \mathcal{N}(0, \sigma_t^2)$ is the noise process. The periodic component is defined following the quasi-random walk model in Equations (3.7) and (3.8) of Durbin and Koopman [62]: we define

$$\gamma_t = \sum_{j=1}^{\lfloor s/2 \rfloor} \gamma_{jt}, \tag{5.2}$$

where $\gamma_{j,t+1} = \gamma_{jt} \cos \lambda_j + \gamma^*_{jt} \sin \lambda_j + w_{jt}$ and $\gamma^*_{j,t+1} = -\gamma_{jt} \sin \lambda_j + \gamma^*_{jt} \cos \lambda_j + w^*_{jt}$; s is the period length, $\lambda_j = 2\pi/s$ and $w_{jt}, w^*_{jt} \sim \mathcal{N}(0, u)$ for some amplitude u.[1].

For the noise component ϵ_t, we consider five *time-dependent* noise variance profiles:

$$\sigma_t^2 = 0.1tn \tag{5.3}$$

and five *static* noise variance profiles,

$$\sigma_t^2 = 0.1n, \tag{5.4}$$

for $n = \{1, \ldots, 5\}$.

Results

We first demonstrate the *qualitative* performance of the CoRNN architecture on two synthetic datasets containing seasonal components of different periodicities $s = 2$ and $s = 10$ but having the same noise profile. These are illustrated in Figure 5.1. In these two examples, the higher frequency series on the left have larger errors in point predictions (black dots vs the dashed red line) compared to the low frequency series on the right; however, the prediction intervals (PIs) still cover the ground truth values in both cases. We note that, due to the same noise profile in both datasets, the prediction interval widths are similar; their shape differs, however, because of the difference in patterns learnt by the underlying recurrent neural network.

---

[1] For implementation details, see also https://www.statsmodels.org/devel/examples/notebooks/generated/statespace_seasonal.html

In Figure 5.2, we show the qualitative changes in prediction interval widths as time-dependent variance increases. In the panels from left to right, the variances increase across time-steps at increasing rate; accordingly, the PI widths become larger at a higher rate, reflecting increasing uncertainty.

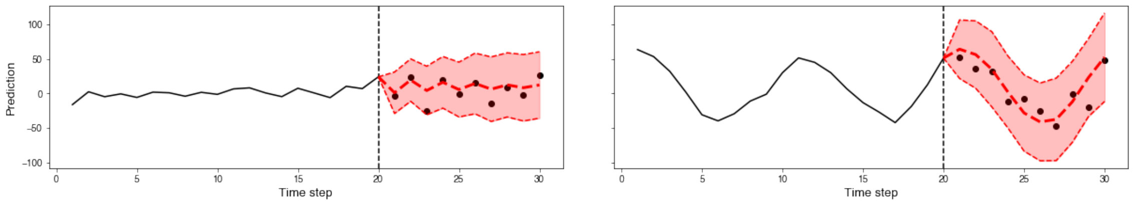


Figure 5.1: Example prediction intervals generated by the CoRNN model for datasets with periodicities 2 and 10 and the same noise profile.

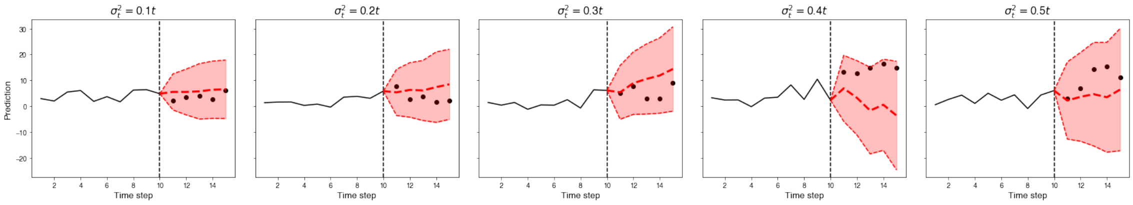


Figure 5.2: Example prediction intervals generated by the CoRNN model in the *time-dependent* noise setting. Each panel uses a different time-dependent noise variance profile, $\sigma_t^2 = 0.1tn$, for $n = \{1, \ldots, 5\}$.

Next, we train the baselines on synthetic datasets of 1000 training sequences (in case of CoRNN, the dataset is split in half for proper training and calibration datasets) for the two noise variance profiles described above. We aim to forecast prediction intervals for H future values $y_{T+1:T+H}$ for a target coverage of 90% ($\alpha = 0.1$). In these experiments, $T = 10$ and $H = 5$; we select short time-series examples due to the computational limitations of the baseline BJ-RNN model. The RNN hyperparameters for the networks underlying the two post-hoc uncertainty estimation models are fixed in order to ensure fair comparison, and largely follow those provided in previous work [21]. These are detailed in the Appendix A along with the time-series model parameters. Where possible, we repeat the experiments five times with a new randomly generated dataset; however, due to high time and space complexity, BJ-RNN was run on a single random seed only.

Table 5.1: **Empirical joint coverage of the different baselines run on autoregressive synthetic datasets** (averaged across prediction horizons); reported as mean ± std over five random seeds (excluding two unstable seeds); except for the BJ-RNN model which was run for a single random seed only.

		Empirical joint coverage			
Noise mode		CoRNN	BJ-RNN	MQ-RNN	DP-RNN
Static $\sigma_t^2 = 0.1n$	$n = 1$	93.3 ± 2.0%	100.0%	62.8 ± 2.2%	5.3 ± 1.4%
	$n = 2$	93.0 ± 1.2%	100.0%	61.9 ± 2.8%	4.8 ± 1.2%
	$n = 3$	93.4 ± 1.5%	100.0%	65.8 ± 0.4%	5.3 ± 1.2%
	$n = 4$	94.8 ± 0.9%	100.0%	65.5 ± 1.2%	5.5 ± 1.6%
	$n = 5$	94.9 ± 0.7%	100.0%	63.0 ± 4.7%	5.3 ± 1.7%
Time-dependent $\sigma_t^2 = 0.1tn$	$n = 1$	92.9 ± 1.5%	99.4%	61.7 ± 3.9%	3.6 ± 0.5%
	$n = 2$	91.2 ± 0.8%	100.0%	58.0 ± 2.1%	1.5 ± 0.5%
	$n = 3$	92.1 ± 1.2%	100.0%	59.4 ± 3.1%	0.4 ± 0.2%
	$n = 4$	91.4 ± 1.8%	97.0%	57.9 ± 1.7%	0.3 ± 0.2%
	$n = 5$	89.7 ± 0.8%	99.4%	57.2 ± 1.5%	0.2 ± 0.3%

Table 5.2: **Mean interval widths of the different baselines run on autoregressive synthetic datasets** (reported as mean ± std over the prediction horizon). This reports results for one of five available random seeds, with the remaining seeds for the CoRNN model available in Appendix A. Empty result means that training of the underlying RNN was unstable for the seed.

		Interval widths			
Noise mode		CoRNN	BJ-RNN	MQ-RNN	DP-RNN
Static $\sigma_t^2 = 0.1n$	$n = 1$	—	98.45 ± 25.95	9.36 ± 1.83	3.10 ± 0.17
	$n = 2$	18.78 ± 5.10	32.53 ± 2.92	9.44 ± 1.81	2.94 ± 0.29
	$n = 3$	18.22 ± 4.51	35.82 ± 1.59	9.65 ± 1.84	2.94 ± 0.20
	$n = 4$	17.72 ± 4.25	33.83 ± 2.49	9.75 ± 1.75	3.13 ± 0.18
	$n = 5$	20.06 ± 5.54	51.23 ± 3.21	9.69 ± 1.57	3.16 ± 0.21
Time-dependent $\sigma_t^2 = 0.1tn$	$n = 1$	20.42 ± 3.93	27.09 ± 1.16	10.57 ± 1.70	3.11 ± 0.20
	$n = 2$	21.84 ± 4.31	104.85 ± 5.68	12.54 ± 1.91	2.88 ± 0.23
	$n = 3$	28.08 ± 5.48	36.45 ± 1.25	15.71 ± 2.03	3.46 ± 0.16
	$n = 4$	33.20 ± 6.30	33.24 ± 2.32	19.45 ± 2.34	3.80 ± 0.18
	$n = 5$	37.96 ± 6.73	51.45 ± 5.37	22.82 ± 3.03	3.96 ± 0.16

Tables 5.1 and 5.2 compare the empirical joint coverage and the PI widths of the baseline models, respectively. Both CoRNN and BJ-RNN empirically surpass the target joint coverage rate of 90% (and error rate $\alpha = 0.1$) in both static and time-dependent noise settings, satisfying the theoretical coverage guarantees. We note that the *empirical* coverage rates are indeed expected to vary around the target rate but will in practice vary around the theoretically guaranteed values, as discussed in more detail in Vovk [57]. We observe that, to a certain degree, CoRNN adapts to the properties of the underlying time-series distribution: when the noise is static (and therefore the time-series more predictable), CoRNN PI widths remain similar across

the datasets while maintaining the same coverage level. For the time-dependent noise profile, uncertainty in data increases with further time steps and larger base variance, so PI widths also in monotonically increase for different n in order to maintain the coverage level. While the intervals for the BJ-RNN model also provide very high (or even perfect) coverage, the intervals are in general much less efficient (wider); however, ideally we would like to have the *minimal* interval width as long as it provides target coverage, rather than overcovering with very wide intervals. In addition, the BJ-RNN model takes prohibitively long to compute (the reason for which it has only been tested on a single random seed and will also be excluded from the comparisons on real data); conversely, the conformal forecasting procedure only requires running the trained RNN model on a calibration set, at which point adding uncertainty intervals to a prediction takes constant time. Regarding the remaining MQ-RNN and DP-RNN baselines, these fail to generate reliable prediction intervals altogether; and while the intervals are much shorter (more *efficient*), they are not *valid*, which is arguably the more important property for downstream tasks of critical nature. We find that the CoRNN model Pareto dominates all baselines in PI width/coverage trade-off, and outperforms BJ-RNNs in computational efficiency.

Finally, the experiments on data with controlled properties provide insight on the trade-offs between the desired coverage rate and how far into the future can the predictions be reliably made. For a dataset with a time-dependent noise profile $\sigma_t^2 = 0.1t$, we fix a prediction interval and for each coverage level compute the furthest horizon H at which the coverage can be maintained. As Figure 5.3 illustrates, low target coverage levels allow us to make valid predictions far into the future, and ideal coverage levels can only be achieved with horizons near the prediction point. We additionally observe that the underlying recurrent neural network model M has little effect on the overall trend.

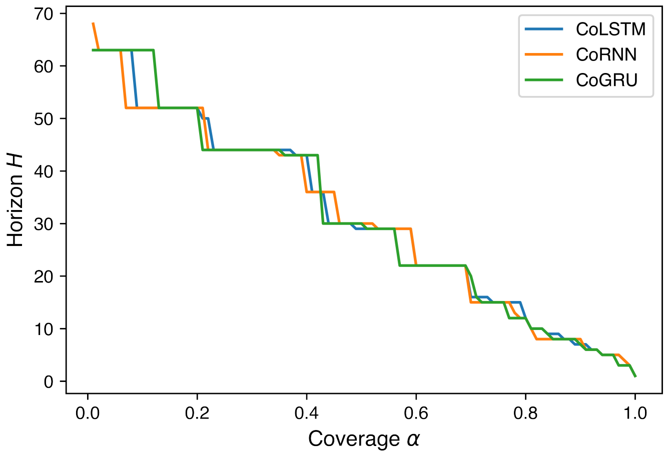


Figure 5.3: **The trade-off between the target coverage rate and prediction horizon for a fixed prediction interval width.** With increasing target coverage, the furthest horizon that can be predicted to without exceeding a certain fixed interval width decreases.

5.2 Experiments on real-world data

We now demonstrate the effectiveness of our procedure by forecasting real-world time-series. We train the proposed CoRNN architecture as well as the MQ-RNN and DP-RNN baselines on three datasets summarised in Table 5.3. (We do not train BJ-RNN as it generally does not scale to the real-world datasets). For the first task, we use the data from the Medical Information Mart for Intensive Care (MIMIC-III) [63] dataset, where we forecast daily observations of white blood cell counts of varying lengths. For the second task, we use the electroencephalography (EEG) dataset from the UCI machine learning repository [64], where we forecast the trajectories of downsampled EEG signals obtained from healthy subjects exposed to three types of visual stimuli. For the final task, we forecast the daily COVID-19 cases within the United Kingdom local authority districts [65]. All datasets are publicly available and the medical data is anonymised. We selected these datasets to represent a variety of scenarios of real time-series: the datasets have different orders of magnitude of available training instances (from hundreds to tens of thousands), have varying observation sequence lengths, have different stationarity properties (e.g. the COVID-19 time-series are synchronous—each time step representing the same point in time—the others are not), have different noise profiles (e.g. EEG signal data will inherently have higher frequencies than MIMIC-III white blood cell count data), have different target prediction horizons. Full details on preprocessing are provided in Appendix A.

Table 5.3: **Real-world medical dataset properties.** The number in parentheses under the training example column indicates how many examples were used for training when a calibration set was required, such as in the case of CoRNNs.

Dataset	**# Training sequences**	**Window length T**	**Prediction horizon H**
MIMIC-III [63]	3823 (2000)	[3, 47]	2
EEG [64]	19200 (15360)	40	10
COVID-19 [65]	300 (200)	100	50

Table 5.4: **Performance of the uncertainty forecasters on three real-world datasets**. Coverage refers to empirical joint coverage (higher is better), and is averaged over five random splits of the dataset and training seeds. Prediction interval lengths (lower is better) are averaged over the prediction horizons of a single dataset realisation.

	MIMIC-III		**EEG**		**COVID-19**	
Model	Coverage	CI/PI lengths	Coverage	CI/PI lengths	Coverage	CI/PI lengths
DP-RNN	39.0 ± 5.4%	3.71 ± 0.00	8.3 ± 10.3%	7.91 ± 0.55	0.0 ± 0.0%	51.46 ± 24.02
MQ-RNN	87.6 ± 2.7%	17.33 ± 0.31	49.1 ± 3.0%	21.42 ± 2.27	14.0 ± 7.1%	122.06 ± 57.98
CoRNN	94.6 ± 1.0%	22.24 ± 0.04	96.7 ± 0.4%	60.87 ± 13.98	89.2 ± 6.4%	635.71 ± 400.18

Results

Table 5.5: Example ranges of independent coverage and MAEs averaged over the horizon steps.

	MIMIC-III		**EEG**		**COVID-19**	
Model	Coverage range	MAE	Coverage range	MAE	Coverage range	MAE
DP-RNN	[44.6%, 48.8%]	2.07 ± 2.22	[35.9%, 54.8%]	5.60 ± 6.29	[22.5%, 67.5%]	28.90 ± 41.53
MQ-RNN	[88.8%, 90.0%]	4.87 ± 5.16	[85.3%, 90.2%]	5.50 ± 6.26	[80.0%, 93.8%]	34.71 ± 43.87
CoRNN	[94.2%, 94.4%]	4.47 ± 5.44	[98.6%, 99.7%]	5.23 ± 5.51	[97.5%, 100.0%]	45.21 ± 66.39

Performance of the models is summarised in Table 5.4. We first note that, similar to the synthetic data experiments, the underlying LSTM model of the proposed CoRNN architecture had half as many training instances available as the competing baselines—some of the training instances had to be used for the conformal calibration procedure—while having the same hyperparameters. However, CoRNN still obtains the highest coverage for all datasets, and is the only model to empirically achieve the target joint coverage rates. While this seems to disproportionately affect the *efficiency* of CoRNN intervals, which seem to be wide, we see that the predictions are indeed *valid* across the range of scenarios. In other words, CoRNN adapts its PI widths to reliably match the required target coverage, increasing PI width for unpredictable datasets. On the other hand, the remaining baselines revert to unreliable predictions, especially in less certain scenarios (COVID-19), and even the dataset/model combination that has achieved the highest coverage with competitive interval width (i.e. MQ-RNN on MIMIC-III), the intervals still arguably lack validity.

We additionally show the details on *independent* coverage rates for different prediction horizons (rather than *joint* coverage of the entire trajectory) in Table 5.5: here we observe that CoRNN outperforms the baselines on prediction of both individual and joint coverage levels.

Table 5.6: Bonferroni-corrected and uncorrected empirical coverages of the CoRNN model, as run on a single realisation of the dataset split.

	MIMIC-III		**EEG**		**COVID-19**	
Model	Joint	Independent	Joint	Independent	Joint	Independent
CoRNN	93.4%	[94.2%, 94.4%]	97.1%	[98.6%, 99.7%]	92.5%	[97.5%, 100.0%]
Uncorrected	88.0%	[89.8%, 90.0%]	57.2%	[85.8%, 90.7%]	68.8%	[87.5%, 98.8%]

Finally, we briefly explore the importance of and motivation behind the Bonferroni correction procedure for the CoRNN model, where the original significance level α is divided by horizon length H in order to maintain *joint* coverage rates across the forecast horizon. Table 5.6 shows that uncorrected calibration scores generally lead to poor *joint* coverage yet similar *independent* coverage (as the Bonferroni procedure corrects for multiple comparisons across the horizon, without affecting individual coverage rates). We again note that the estimates in this case are noisy due to small testing and empirical calibration set sizes, especially for noisy COVID-19 data.

Summary

In this chapter, we have compared the effectiveness of the proposed CoRNN model against a set of alternative uncertainty forecasters, each from a different class of uncertainty estimation paradigm (Bayesian, quantile, frequentist). We have demonstrated that CoRNN (Pareto) dominated all baselines; in case of synthetic datasets, it had *the best validity and efficiency trade-off* (i.e. had the closest to target coverage with the minimal interval widths) while being computationally efficient (in this way outperforming the state-of-the-art BJ-RNN baseline). In case of real-world datasets, CoRNN was the *only* model to reliably achieve coverage validity. We additionally demonstrated qualitative properties of CoRNN uncertainty forecasting, and the importance of the Bonferroni correction step in our proposed conformal forecasting procedure in order to achieve *joint* coverage across the prediction horizon.

6 Extensions

In this chapter, we briefly explore a further extension of our work, which is the *normalisation* technique used in ICP to derive *adaptive* (and hopefully more efficient) interval widths.

6.1 Normalised inductive conformal prediction

In standard conformal prediction, once the underlying model is trained on the training dataset and calibrated on the calibration datasetm, the prediction interval width, $2\hat{\varepsilon}$ (see Section 3.4 in Chapter 3), will be the same for *every* subsequent example (in case of CoRNN, the intervals will be horizon-specific, but not vary across examples). While this gives *valid* intervals, they are not as efficient as the widths are determined by the residuals of the most difficult examples with the largest residuals in the dataset. *Normalisation* [54], then, tries to return the interval widths that are *example-specific*; i.e. if the model knows that the example is "simple" to forecast, the intervals will be more narrow; if the example is very unusual, the intervals will be wider.

This is achieved through a modification to the nonconformity score R_i as follows:

$$R_i = \frac{|y^{(i)} - M(\mathbf{x}^{(i)})|}{\exp(\mu^{(i)}) + \beta}, \tag{6.1}$$

where the numerator is as before, and the denominator captures the "difficulty" of the example through an estimate of the residual error:

$$\mu^{(i)} = \log |y^{(i)} - M(\mathbf{x}^{(i)})|, \tag{6.2}$$

and β is the sensitivity parameter. The estimates $\mu^{(i)}$ are obtained through training another model (often a neural network such as a multilayer perceptron) on the examples in the proper training set and their log residuals: $\left\{(\mathbf{x}^{(i)}, \log |y^{(i)} - \hat{y}^{(i)}|)\right\}_{i=1}^{n}$. We learn the logarithms of errors for them to both have a smaller range across examples, and to enforce the errors to be positive once they are raised to the exponent as R_i is computed. As the difficulty score $\mu^{(i)}$ is example-specific, for the new example $\mathbf{x}^{(l+1)}$ the new interval obtained is

$$\Gamma^{\alpha}(\mathbf{x}^{(l+1)}) = [\hat{y}^{(l+1)} - \hat{\varepsilon}(\exp(\mu^{(l+1)}) + \beta), \hat{y}^{(l+1)} + \hat{\varepsilon}(\exp(\mu^{(l+1)}) + \beta)]. \tag{6.3}$$

This is analogously extended to the forecast horizon-specific set of $\hat{\varepsilon}_h$ in conformal time-series forecasting procedure.

6.2 Experiments

We carry out the experiments on the synthetic datasets discussed in Chapter 5 to investigate the effects of the normalised nonconformity scores on the quality of prediction intervals. Since CoRNN is designed to work on time-series of different lengths, the normalisation network is also trained on a recurrent neural network, contrary to the recommendation in literature (see e.g. Papadopoulos and Haralambous [54]) to use simple predictors. We use the same parameters for the normalisation RNN as the underlying model M; we set the sensitivity parameter $\beta = 1$.

The results of these experiments are shown in Tables 6.1 and 6.2. We observe that, with the simplistic hyperparameter setting, we achieve the *opposite* effect from the

Table 6.1: **Empirical joint coverage of CoRNN and normalised CoRNN (CoRNN-n) baselines run on autoregressive synthetic datasets** (averaged across prediction horizons); reported as mean ± std over five random seeds (excluding unstable seeds).

		Empirical joint coverage	
Noise mode		CoRNN	CoRNN-n
Static $\sigma_t^2 = 0.1n$	$n = 1$	93.3 ± 2.0%	93.0 ± 0.8%
	$n = 2$	93.0 ± 1.2%	93.0 ± 1.2%
	$n = 3$	93.4 ± 1.5%	93.7 ± 1.4%
	$n = 4$	94.8 ± 0.9%	93.8 ± 0.5%
	$n = 5$	94.9 ± 0.7%	95.2 ± 0.6%
Time-dependent $\sigma_t^2 = 0.1tn$	$n = 1$	92.9 ± 1.5%	93.0 ± 0.8%
	$n = 2$	91.2 ± 0.8%	91.7 ± 1.4%
	$n = 3$	92.1 ± 1.2%	91.9 ± 0.5%
	$n = 4$	91.4 ± 1.8%	90.6 ± 1.1%
	$n = 5$	89.7 ± 0.8%	90.7 ± 1.0%

Table 6.2: **Mean interval widths of CoRNN and normalised CoRNN (CoRNN-n) baselines run on autoregressive synthetic datasets** (reported as mean ± std over the prediction horizon). This reports results for one of five available random seeds. Empty results mean that training of the underlying RNN was unstable for the seed.

		Interval widths	
Noise mode		CoRNN	CoRNN-n
Static $\sigma_t^2 = 0.1n$	$n = 1$	—	—
	$n = 2$	18.78 ± 5.10	20.02 ± 5.10
	$n = 3$	18.22 ± 4.51	19.73 ± 4.59
	$n = 4$	17.72 ± 4.25	20.05 ± 5.58
	$n = 5$	20.06 ± 5.54	22.59 ± 6.90
Time-dependent $\sigma_t^2 = 0.1tn$	$n = 1$	20.42 ± 3.93	22.77 ± 4.96
	$n = 2$	21.84 ± 4.31	25.65 ± 4.91
	$n = 3$	28.08 ± 5.48	30.62 ± 6.23
	$n = 4$	33.20 ± 6.30	35.61 ± 7.17
	$n = 5$	37.96 ± 6.73	40.70 ± 7.23

intended one: while the intervals stay valid, they become *less*, rather than *more*, efficient. One reason for this is that the underlying and normalisation RNNs are both noisy, which makes the residuals difficult to learn. This results in noisy normalisation estimates that do not help with reducing the interval widths. Another reason, as discussed in Romano et al. [66], is that the standard normalisation procedure is not adaptive to *heteroscedastic* datasets, where the variance of the data depends on the time-step, which is the case in these synthetic datasets. Making the interval widths more robust to the noisy underlying and normalising RNNs as well as more adaptive to heteroscedasticity is an interesting problem for future work.

7 Conclusion

In this work, we proposed a novel technique for multi-horizon time-series forecast uncertainty estimation. Unlike the previous works, the new *conformal forecasting* framework—an adaptation of inductive conformal prediction to the multi-horizon time-series forecasting setting—is not only very lightweight, simple to implement, sample efficient, without requiring any alterations alterations (such as reparameterisation of the weights to probability distributions) to the underlying point forecaster, but importantly has theoretical guarantees on the validity of joint coverage across the prediction horizon. We proposed a concrete instantiation of the framework—the conformal recurrent neural network (CoRNN)—and compared it to existing baselines of different paradigms in a variety of synthetic and real-world settings. We showed that CoRNN outperformed all baselines—Pareto-dominating the state-of-the-art while being more computationally efficient and scalable to large datasets. We note that, while we proposed an RNN-based conformal forecaster, the general conformal forecasting framework is applicable to *any* underlying model that can produce direct multi-horizon forecasts.

Future work will focus on increasing the overall efficiency of prediction intervals by reducing their width and making them more adaptive to individual observations, including a multi-variate time-series setting.

A Additional experiment details

A.1 Synthetic data

The hyperparameters are provided in Table A.1 and the prediction interval widths for different realisations of randomly generated datasets are given in Table A.2.

Table A.1: Training hyperparameters.

Parameter	**Value**
Training samples	1000
Calibration samples	1000
Test samples	500
Sequence length L	10
Prediction horizon S	5
Autoregressive mean μ_x	1
Autoregressive variance σ_x^2	2
Periodicity s	None
Amplitude u	5
Epochs	1000
Batch size	100
Embedding size	20
Learning rate	0.01
Underlying RNN type	LSTM
Target coverage $1 - \alpha$	90%

A.2 Real-world datasets

MIMIC-III We collect the data of patients on antibiotics from the MIMIC-III dataset [63], and filter out the sequences of total length at least 5, resulting in 4323 sequences. From these sequences we pick out the white blood cell (high) count as the feature for the univariate time-series. We split the sequences randomly into training,

Table A.2: CoRNN prediction interval lengths on synthetic dataset experiments for the two different noise settings and different random seeds. Reported as mean ± std over the forecast horizon.

Noise mode		**CoRNN PI lengths**				
Static $\sigma_t^2 = 0.1n$	$n = 1$	–	–	17.28 ± 4.59	18.10 ± 4.59	19.21 ± 4.98
	$n = 2$	18.78 ± 5.10	19.14 ± 4.71	17.82 ± 4.73	19.20 ± 4.98	18.66 ± 5.15
	$n = 3$	18.22 ± 4.51	18.23 ± 4.44	18.21 ± 4.64	19.61 ± 5.11	17.55 ± 4.11
	$n = 4$	17.72 ± 4.25	19.17 ± 5.08	19.50 ± 4.99	19.62 ± 4.88	19.19 ± 4.55
	$n = 5$	20.06 ± 5.54	18.65 ± 4.55	19.07 ± 4.74	19.41 ± 4.34	16.71 ± 4.02
Time-dependent $\sigma_t^2 = 0.1tn$	$n = 1$	20.42 ± 3.93	19.68 ± 4.47	19.70 ± 4.90	20.37 ± 4.90	18.98 ± 3.98
	$n = 2$	21.84 ± 4.31	23.88 ± 4.20	22.90 ± 3.83	23.58 ± 4.39	22.70 ± 4.99
	$n = 3$	28.08 ± 5.48	28.14 ± 5.32	28.93 ± 5.97	27.75 ± 4.72	27.14 ± 4.64
	$n = 4$	33.20 ± 6.30	33.99 ± 6.33	33.76 ± 5.01	34.35 ± 5.68	33.98 ± 5.63
	$n = 5$	37.96 ± 6.73	39.09 ± 6.87	39.51 ± 7.32	39.36 ± 6.73	40.09 ± 8.11

calibration and test datasets. We pick the constant time horizon of 2, which is to account for the shortest sequences being of length 5, and use the rest of the sequence as model input.

EEG The EEG dataset, available at https://archive.ics.uci.edu/ml/datasets/EEG+Database, was used as the source for the EEG signal time-series. The dataset contains the data for control and alcoholic subjects responding to a visual stimulus of three types. We used the medium version of the dataset, involving 10 control and 10 alcoholic subjects, though for the experiments we only used the control subjects—from the summaries provided, control subject EEG responses seemed to be more difficult to predict. Each subject had repeated trials for every type of stimulus, and each trial had a time-series for the 64 channels obtained from their corresponding sensor. We treated every individual trial and each of the 64 channels as a separate time-series example, resulting in 19200 sequences in the training set. To keep training efficient, we downsampled the sequences (normally of length 255) to a total length 50, which we further split into the training sequence of 40 and prediction horizon 10. The training and test dataset splits are readily provided in the UCI repository, and for repeated trials we used different subsets for calibration and different model training seeds.

COVID-19 The data is available at https://coronavirus.data.gov.uk/. We picked the data of different regions of the same country in order to follow the exchangeability assumption as closely as possible, while the data from different countries risks having much larger distribution shifts due to a large variation of factors like government lockdown policies. Given the setup of the conformal prediction framework, we looked for the data that would have a sufficiently large number of independent sequences—the lower tier local authority split gives a total number of 380 sequences, which over repeated trials we would randomly split into the test set of 80 sequences and the the rest between the training and calibration sets. We picked the data of daily new cases over the course of 150 days starting mid-September 2020 and ending mid-February 2021, which we further split into the input sequence of 100 examples (ending Christmas 2020) and using the remaining 50 days as the testing sequence. We chose these dates to capture interesting properties of changing government lockdown policies and so that the two waves are separated between the observed and the to-be-predicted sequence.

Hyperparameters The training hyperparameters (Table A.3) mostly follow those provided in previous work [21] and are kept the same for new experiments in order to ensure fair comparison between the baselines. For the CoRNN model, the total training data available was split between the training and calibration sets, and the other baselines used all available training data to train the underlying RNN model.

Table A.3: Training hyperparameters for the real-world datasets.

Parameter	**MIMIC-III**	**EEG**	**COVID-19**
Training samples	3823 (2000)	19200 (15360)	300 (200)
Calibration samples	1823	3840	100
Test samples	500	19200	80
Sequence length L	[3, 47]	40	100
Prediction horizon S	2	10	50
Epochs	1000		
Batch size	150		
Embedding size	20		
Learning rate	0.01		
Dropout probability (for DP-RNN)	0.5		
Underlying RNN type	LSTM		
Target coverage $1-\alpha$	90%		

Bibliography

[1] Sreelekshmy Selvin, R Vinayakumar, EA Gopalakrishnan, Vijay Krishna Menon, and KP Soman. Stock price prediction using lstm, rnn and cnn-sliding window model. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1643–1647. IEEE, 2017.

[2] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[3] Ruofeng Wen, Kari Torkkola, Balakrishnan Narayanaswamy, and Dhruv Madeka. A multi-horizon quantile recurrent forecaster. *arXiv preprint arXiv:1711.11053*, 2017.

[4] Lingxue Zhu and Nikolay Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110. IEEE, 2017.

[5] Bryan Lim, Ahmed Alaa, and Mihaela van der Schaar. Forecasting treatment responses over time using recurrent marginal structural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 7494–7504, 2018.

[6] Ahmed M Alaa and Mihaela van der Schaar. Attentive state-space modeling of disease progression. In *Advances in Neural Information Processing Systems*, pages 11334–11344, 2019.

[7] Benjamin Shickel, Patrick James Tighe, Azra Bihorac, and Parisa Rashidi. 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.

[8] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[9] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013.

[10] Christian Gollier. *The economics of risk and uncertainty.* Edward Elgar Publishing Limited, 2018.

[11] Michael W Dusenberry, Dustin Tran, Edward Choi, Jonas Kemp, Jeremy Nixon, Ghassen Jerfel, Katherine Heller, and Andrew M Dai. Analyzing the

role of model uncertainty for electronic health records. In *Proceedings of the ACM Conference on Health, Inference, and Learning*, pages 204–213, 2020.

[12] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *arXiv preprint arXiv:1703.04977*, 2017.

[13] Wesley Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *arXiv preprint arXiv:1902.02476*, 2019.

[14] Andrey Malinin and Mark Gales. Predictive uncertainty estimation via prior networks. In *Advances in Neural Information Processing Systems*, pages 7047–7058, 2018.

[15] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6402–6413, 2017.

[16] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *arXiv preprint arXiv:1704.02798*, 2017.

[17] Jen-Tzung Chien and Yuan-Chu Ku. Bayesian recurrent neural network for language modeling. *IEEE transactions on neural networks and learning systems*, 27(2):361–374, 2015.

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

[19] Jan Gasthaus, Konstantinos Benidis, Yuyang Wang, Syama Sundar Rangapuram, David Salinas, Valentin Flunkert, and Tim Januschowski. Probabilistic forecasting with spline quantile function rnns. In *The 22nd international conference on artificial intelligence and statistics*, pages 1901–1910. PMLR, 2019.

[20] Syama Sundar Rangapuram, Matthias W Seeger, Jan Gasthaus, Lorenzo Stella, Yuyang Wang, and Tim Januschowski. Deep state space models for time series forecasting. *Advances in neural information processing systems*, 31:7785–7794, 2018.

[21] Ahmed Alaa and Mihaela van der Schaar. Frequentist uncertainty in recurrent neural networks via blockwise influence functions. In *International Conference on Machine Learning*, pages 175–190. PMLR, 2020.

[22] Ahmed Alaa and Mihaela van der Schaar. Discriminative jackknife: Quantifying uncertainty in deep learning via higher-order influence functions. In *International Conference on Machine Learning*, pages 165–174. PMLR, 2020.

[23] Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic learning in a random world.* Springer Science & Business Media, 2005.

[24] Glenn Shafer and Vladimir Vovk. A tutorial on conformal prediction. *Journal of Machine Learning Research*, 9(3), 2008.

[25] David JC MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

[26] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[27] Radford M Neal. Bayesian learning via stochastic dynamics. In *Advances in neural information processing systems*, pages 475–482, 1993.

[28] Radford M Neal et al. Mcmc using hamiltonian dynamics. *Handbook of markov chain monte carlo*, 2(11):2, 2011.

[29] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688. Citeseer, 2011.

[30] Tianqi Chen, Emily Fox, and Carlos Guestrin. Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

[31] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

[32] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13, 1993.

[33] Alex Graves. Practical variational inference for neural networks. *Advances in neural information processing systems*, 24, 2011.

[34] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[35] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

[36] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[37] Andreas Damianou and Neil D Lawrence. Deep gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR, 2013.

[38] Yarin Gal and Zoubin Ghahramani. A theoretically grounded application of dropout in recurrent neural networks. *Advances in neural information processing systems*, 29:1019–1027, 2016.

[39] Ian Osband. Risk versus uncertainty in deep learning: Bayes, bootstrap and the dangers of dropout. In *NIPS workshop on bayesian deep learning*, volume 192, 2016.

[40] M Jésus Bayarri and James O Berger. The interplay of bayesian and frequentist analysis. *Statistical Science*, 19(1):58–80, 2004.

[41] Roger Koenker and Kevin F Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

[42] James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*. Oxford university press, 2012.

[43] Rupert G Miller. The jackknife-a review. *Biometrika*, 61(1):1–15, 1974.

[44] Rina Foygel Barber, Emmanuel J Candes, Aaditya Ramdas, and Ryan J Tibshirani. Predictive inference with the jackknife+. *The Annals of Statistics*, 49 (1):486–507, 2021.

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

[46] Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *The Journal of Machine Learning Research*, 18(1):4148–4187, 2017.

[47] Anastasios Angelopoulos, Stephen Bates, Jitendra Malik, and Michael I Jordan. Uncertainty sets for image classifiers using conformal prediction. *International Conference on Learning Representations*, 2021.

[48] Harris Papadopoulos. Inductive conformal prediction: Theory and application to neural networks. In *Tools in artificial intelligence*. Citeseer, 2008.

[49] Jakub Kowalczewski. Normalized conformal prediction for time series data. *Degree project*, 2019.

[50] Chen Xu and Yao Xie. Conformal prediction interval for dynamic time-series. In *International Conference on Machine Learning*, pages 11559–11569. PMLR, 2021.

[51] Chen Xu and Yao Xie. Conformal prediction interval for dynamic time-series. *International Conference on Machine Learning*, 2021.

[52] Gianluca Zeni, Matteo Fontana, and Simone Vantini. Conformal prediction: a unified review of theory and new challenges. *arXiv preprint arXiv:2005.07972*, 2020.

[53] Vladimir Vovk. Transductive conformal predictors. In *IFIP International Conference on Artificial Intelligence Applications and Innovations*, pages 348–360. Springer, 2013.

[54] Harris Papadopoulos and Haris Haralambous. Reliable prediction intervals with regression neural networks. *Neural Networks*, 24(8):842–851, 2011.

[55] Harris Papadopoulos, Kostas Proedrou, Volodya Vovk, and Alex Gammerman. Inductive confidence machines for regression. In *European Conference on Machine Learning*, pages 345–356. Springer, 2002.

[56] Souhaib Ben Taieb and Amir F Atiya. A bias and variance analysis for multistep-ahead time series forecasting. *IEEE transactions on neural networks and learning systems*, 27(1):62–76, 2015.

[57] Vladimir Vovk. Conditional validity of inductive conformal predictors. In *Asian conference on machine learning*, pages 475–490. PMLR, 2012.

[58] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[59] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[60] Wes McKinney et al. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, volume 445, pages 51–56. Austin, TX, 2010.

[61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[62] J Durbin and SJ Koopman. Linear state space models. *Time Series Analysis by State Space Methods*, pages 43–75, 2012.

[63] Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-Wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[64] Catherine Blake. Uci repository of machine learning databases. *http://www. ics. uci. edu/˜ mlearn/MLRepository. html*, 1998.

[65] Coronavirus (COVID-19) in the UK. https://coronavirus.data.gov.uk/, 2021. Accessed: 2021-05-25.

[66] Yaniv Romano, Evan Patterson, and Emmanuel Candes. Conformalized quantile regression. *Advances in Neural Information Processing Systems*, 32:3543–3553, 2019.