Deep State Space Models for Time Series Forecasting

Syama Sundar Rangapuram **Matthias Seeger** **Jan Gasthaus** **Lorenzo Stella**

Yuyang Wang **Tim Januschowski**

Amazon Research
{rangapur, matthis, gasthaus, stellalo, yuyawang, tjnsch}@amazon.com

Abstract

We present a novel approach to probabilistic time series forecasting that combines state space models with deep learning. By parametrizing a per-time-series linear state space model with a jointly-learned recurrent neural network, our method retains desired properties of state space models such as data efficiency and interpretability, while making use of the ability to learn complex patterns from raw data offered by deep learning approaches. Our method scales gracefully from regimes where little training data is available to regimes where data from large collection of time series can be leveraged to learn accurate models. We provide qualitative as well as quantitative results with the proposed method, showing that it compares favorably to the state-of-the-art.

1 Introduction

Time series forecasting is a key component in many industrial and business decision processes. A typical example of such tasks is demand forecasting: accurate and up-to-date models are fundamental to successful inventory planning and minimization of operational costs.

State space models [8, 13, 23] (SSMs) provide a principled framework for modeling and learning time series patterns such as trend and seasonality. Prominent examples include ARIMA models [4, 8] and exponential smoothing [13]. SSMs are particularly well-suited for applications where the structure of the time series is well-understood, as they allow for the incorporation of structural assumptions into the model. This allows for the model to be interpretable and the associated learning procedure to be data efficient, but it requires time series with enough history. In modern forecasting applications with large and diverse time series corpora, SSMs require prohibitively labor- and compute-intensive tasks such as model and covariate selection. Further, traditional SSMs cannot infer shared patterns from a dataset of similar time series, as they are fitted on each time series separately. This makes creating forecasts for time series with little or no history challenging.

Deep neural networks [12, 25, 26] offer an alternative. With their capability to extract higher order features, they can identify complex patterns within and across time series, and can do so from datasets of raw time series with considerably less human effort [9, 27, 19]. However, as these models make fewer structural assumptions, they typically require larger training datasets to learn accurate models. Additionally, these models are difficult to interpret and—often more importantly—make it difficult to enforce assumptions such as temporal smoothness.

In this paper we propose to bridge the gap between these two approaches by fusing SSMs with deep (recurrent) neural networks. We present a forecasting method that parametrizes a particular linear SSM using a recurrent neural network (RNN). The parameters of the RNN are learned jointly from a dataset of raw time series and associated covariates, allowing the model to automatically extract features and learn complex temporal patterns. At the same time, as each individual time series is modeled using an SSM, we can enforce and exploit assumptions such as temporal smoothness.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Furthermore our method is interpretable, in the sense that the SSM parameters for each individual time series can be inspected (and even changed if necessary). By incorporating prior structural assumptions, the presented method scales from small to large data regimes seamlessly. When there is little data to learn from, the structure imposed by the SSM can alleviate overfitting.

The rest of the paper is organized as follows. We first discuss related work in Section 2 and then review the general state space approach to time series forecasting in Section 3. In Section 4, we present our joint forecasting model and describe the training and inference procedure. In Section 5, we first do a qualitative analysis of our method and then present quantitative comparison against the state-of-the-art. We conclude in Section 6.

2 Related work

Hyndman et al. [13] and Durbin and Koopman [8] provide comprehensive overviews of SSMs. Recent work in the machine learning literature on linear state-space models includes [23, 22]. We follow [13] in their approach to use linear state space models. The assumption of linear dynamics consisting of interpretable components (level/trend/seasonality) makes the forecasts robust and understandable. Note that non-linear effects can still be captured via exogenous variables. In the forecasting context, non-linearities are typically associated with interventions such as promotions, so this assumption is practically reasonable.

Combining state-space models (SSM) with RNNs has been proposed before, through either or both of the following: (i) extending the Gaussian emission to complex likelihood models; (ii) making the transition equation non-linear via a multi-layer perceptron (MLP) or interlacing SSM with transition matrices temporally specified by RNNs. The so-called *Deep Markov Model* (DMM) proposed by [18, 17] keeps the Gaussian transition dynamics with mean and covariance matrix parameterized by MLPs. *Stochastic RNNs* [10] explicitly incorporate the deterministic dynamics from RNNs by interlacing them with an SSM while the dynamics of RNNs do not depend on latent variables. Compared to DMM, the difference is the added information from the deterministic state of RNNs. An alternative way to make the transition equation non-linear is to cut the ties between the latent states $\boldsymbol{l}_t$'s and associate them with deterministic states $\boldsymbol{h}_t$ of RNN. In this way, the transition from $\boldsymbol{l}_{t-1}$ to $\boldsymbol{l}_t$ is non-linearly determined by the RNN and the observation model. Chung et al. [7] first proposed such *Variational RNNs*. They were later used in *Latent LSTM Allocation* [29] and *State-Space LSTM* [30]. [15] discusses unsupervised learning of state space models from sequential data.

Arguably the most relevant to our work is [11], which aims to keep the linear Gaussian transition structure intact so that the highly efficient Kalman filter/smoother is applicable. The non-linear behavior is approximated by locally linear transition matrices. The so-called called *Kalman Variational Auto-Encoder* (KVAE) disentangles the observations (emissions) and the latent dynamics (transitions) with VAE. By making the locally linear part outside of the standard inference routine and using a fully factorized Gaussian "decoder," Kalman smoothing can be readily applied. A similar idea appeared in [14] where a recognition network is used to output conjugate graphical model potentials so that efficient structural inference is feasible. Our model differs from [11] in that instead of using an RNN to specify the linear combination of a fixed set of K parameters, we directly use RNNs to output the SSM parameters, eliminating the need to tune additional hyper-parameters.

3 Background

The general probabilistic forecasting problem is the following. Let N be a set of univariate time series $\{z_{1:T_i}^{(i)}\}_{i=1}^N$, where $z_{1:T_i}^{(i)} = (z_1^{(i)}, z_2^{(i)}, \dots, z_{T_i}^{(i)})$ and $z_t^{(i)} \in \mathbb{R}$ denotes the value of the i-th time series at time t.[1] Further, let $\{\mathbf{x}_{1:T_i+\tau}^{(i)}\}_{i=1}^N$ be a set of associated, time-varying covariate vectors with $\mathbf{x}_t^{(i)} \in \mathbb{R}^D$. Our goal is to produce a set of probabilistic forecasts, i.e. for each $i = 1, \dots, N$ we are interested in the probability distribution of future trajectories $z_{T_i+1:T_i+\tau}^{(i)}$ given the past:

$$p\left(z_{T_i+1:T_i+\tau}^{(i)} \,\middle|\, z_{1:T_i}^{(i)}, \mathbf{x}_{1:T_i+\tau}^{(i)}; \Phi\right). \tag{1}$$

[1] We consider time series where the the time points are equally spaced, but the units are arbitrary (e.g. hours, days, months). Further, the time series do not have to be aligned, i.e., the starting point $t = 1$ can refer to a different absolute time point for different time series i.

Here Φ denotes the set of learnable parameters of the model, which are shared between and learned jointly from all N time series. For any given i, we refer to time series $z_{1:T_i}^{(i)}$ as *target* time series, to time range $\{1, 2, \dots, T_i\}$ as *training range*, and to time $\{T_i + 1, T_i + 2, \dots, T_i + \tau\}$ as *prediction range*. The time point $T_i + 1$ is referred to as *forecast start time* and $\tau \in \mathbb{N}_{>0}$ is the *forecast horizon*. Note that we assume that the covariate vectors $\mathbf{x}_t^{(i)}$ are given also in the prediction range.

We make the common simplifying assumption that the time series are independent of each other when conditioned on the associated covariates $\mathbf{x}_{1:T_i}^{(i)}$ and the parameters Φ. However, in constrast to many related methods that make this assumption, in our approach the model parameters Φ are shared between *all* time series. So while this assumption precludes us from modeling correlations between time series, it does *not* mean that the proposed model is not able to share statistical strength between and learn patterns across the different time series, as we are learning the parameters Φ jointly from *all* time series.

State Space Models. SSMs model the temporal structure of the data via a *latent state* $\boldsymbol{l}_t \in \mathbb{R}^L$ that can be used to encode time series components such as level, trend, and seasonality patterns. In the forecasting setting they are typically applied to individual times series (though multivariate exensions with multi-dimensional targets exist). For this reason, we will drop the superscript i from the notation in this section. A general SSM is described by the so-called state-transition equation, defining the stochastic transition dynamics $p(\boldsymbol{l}_t|\boldsymbol{l}_{t-1})$ by which the latent state evolves over time, and an *observation model* specifying the conditional probability $p(z_t|\boldsymbol{l}_t)$ of observations given the latent state.

We consider *linear* state space models where the transition equation takes the form

$$\boldsymbol{l}_t = \boldsymbol{F}_t \boldsymbol{l}_{t-1} + \boldsymbol{g}_t \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, 1). \tag{2}$$

Here at time t, the latent state $\boldsymbol{l}_{t-1}$ maintains information about level, trend, and seasonality patterns and evolves by way of a deterministic *transition matrix* $\boldsymbol{F}_t$ and a random *innovation* $\boldsymbol{g}_t \varepsilon_t$. The structure of the transition matrix $\boldsymbol{F}_t$ and innovation strength $\boldsymbol{g}_t$ determine which kind of time series patterns are encoded by the latent state $\boldsymbol{l}_t$ (see [13] or [22] for details on possible structures; Appendix A.1 in the long version of the paper contains two example instantiations).

The probabilistic observation model then describes how the observations are generated from the latent state $\boldsymbol{l}_t$. Here we consider a univariate Gaussian observation model of the form

$$z_t = y_t + \sigma_t \epsilon_t, \qquad y_t = \boldsymbol{a}_t^\top \boldsymbol{l}_{t-1} + b_t, \qquad \epsilon_t \sim \mathcal{N}(0, 1), \tag{3}$$

where $\boldsymbol{a}_t \in \mathbb{R}^L$, $\sigma_t \in \mathbb{R}_{>0}$ and $b_t \in \mathbb{R}$ are further (time-varying) parameters of the model. Finally, the initial state $\boldsymbol{l}_0$ is assumed to follow an isotropic Gaussian distribution, $\boldsymbol{l}_0 \sim N(\boldsymbol{\mu}_0, \mathrm{diag}(\boldsymbol{\sigma}_0^2))$.

Parameter learning. The state space model is fully specified by the parameters $\Theta_t = (\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0, \boldsymbol{F}_t, \boldsymbol{g}_t, \boldsymbol{a}_t, b_t, \sigma_t)$, $\forall t > 0$. In the classical setting the dynamics are assumed to be time-invariant, that is $\Theta_t = \Theta$, $\forall t > 0$. One generic way of estimating them is by maximizing the marginal likelihood, i.e., $\Theta_{1:T}^* = \mathrm{argmax}_{\Theta_{1:T}} p_{SS}(z_{1:T}|\Theta_{1:T})$, where

$$p_{SS}(z_{1:T}|\Theta_{1:T}) := p(z_1|\Theta_1) \prod_{t=2}^{T} p(z_t|z_{1:t-1}, \Theta_{1:t}) = \int p(\boldsymbol{l}_0) \left[\prod_{t=1}^{T} p(z_t|\boldsymbol{l}_t) p(\boldsymbol{l}_t|\boldsymbol{l}_{t-1}) \right] \mathrm{d}\boldsymbol{l}_{0:T} \tag{4}$$

denotes the marginal probability of the observations $z_{1:T}$ given the parameters Θ under the state space model, integrating out the latent state $\boldsymbol{l}_t$. In the linear-Gaussian case considered here, the required integrals are analytically tractable.

Note that in the classical setting, if there is more than one time series, a separate set of parameters $\Theta^{(i)}$ is learned for each time series $z_{1:T_i}^{(i)}$ independently. This has the disadvantage that no information is shared across different time series, making it challenging to apply this approach to time series with limited historical data or high noise levels.

4 Deep State Space Models

Instead of learning the state space parameters $\Theta^{(i)}$ for each time series independently, our forecasting model instead learns a globally shared mapping from the covariate vectors $\mathbf{x}_{1:T_i}^{(i)}$ associated with each

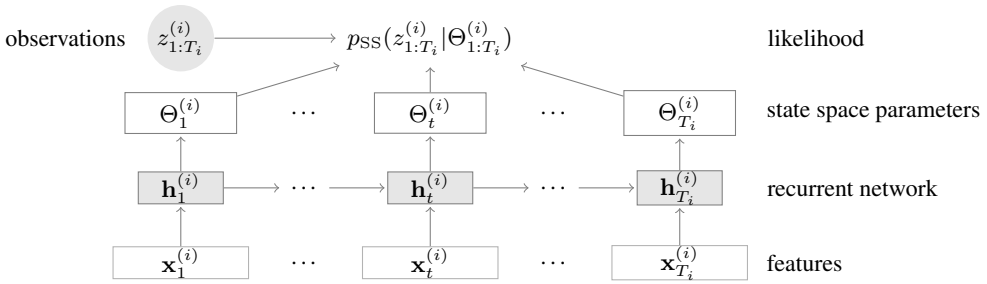

Figure 1: Summary of the model. During training, the inputs to the network are the features $\mathbf{x}^{(i)}_t$ as well as the previous network output $\mathbf{h}^{(i)}_{t-1}$ at each time step t in the training range $\{1, 2, \ldots, T_i\}$. The network output $\mathbf{h}^{(i)}_t = h(\mathbf{h}^{(i)}_{t-1}, \mathbf{x}^{(i)}_t, \Phi)$ is then used to compute the parameters of the state space model $\Theta^{(i)}_t$ after mapping it to the corresponding ranges of the parameters. Given the time series observations $z^{(i)}_{1:T_i}$ in the training range, the likelihood of the state space parameters $\Theta^{(i)}_{1:T_i}$ (which are functions of the shared network parameters Φ) are computed according to Eq. 4. The shared network parameters Φ are then learned by maximizing the likelihood.

target time series $z^{(i)}_{1:T_i}$, to the (time-varying) parameters $\Theta^{(i)}_t$ of a linear state space model for the i-th time series. This mapping,

$$\Theta^{(i)}_t = \Psi(\mathbf{x}^{(i)}_{1:t}, \Phi), \qquad i = 1, \ldots, N, \quad t = 1, \ldots, T_i + \tau, \tag{5}$$

is a function of the entire covariate time series $\mathbf{x}^{(i)}_{1:t}$ up to (and including) time t, as well as a set of shared parameters Φ. Then, given the features $\mathbf{x}^{(i)}_{1:T}$ and the parameters Φ, under our model, the data $z^{(i)}_{1:T_i}$ is distributed according to

$$p(z^{(i)}_{1:T_i}|\mathbf{x}^{(i)}_{1:T_i}, \Phi) = p_{SS}(z^{(i)}_{1:T_i}|\Theta^{(i)}_{1:T_i}), \qquad i = 1, \ldots, N. \tag{6}$$

where p_{SS} denotes the marginal likelihood under a linear state space model as defined in Eq. 4, given its (time-varying) parameters $\Theta^{(i)}_t$.

We parameterize the mapping Ψ from covariates to state space model parameters using a deep recurrent neural network (RNN). Figure 1 shows a sketch of the overall model structure, unrolled for all the time steps in the training range. Given the covariates[2] $\mathbf{x}^{(i)}_t$ associated with time series $z^{(i)}_t$, a multi-layer recurrent neural network with LSTM cells and parameters Φ computes a representation of the features via a recurrent function h,

$$\mathbf{h}^{(i)}_t = h(\mathbf{h}^{(i)}_{t-1}, \mathbf{x}^{(i)}_t, \Phi).$$

The real-valued output vector of the last LSTM layer is then mapped to the parameters $\Theta^{(i)}_t$ of the state space model, by applying affine mappings followed by suitable elementwise transformations constraining the parameters to appropriate ranges (see Appendix A.2 in the long version of the paper). Parameters $\Theta^{(i)}_t$ are then used to compute the likelihood of the given observations $z^{(i)}_t$, which is used for learning of the network parameters Φ.

4.1 Training

The model parameters Φ are learned by maximizing the probability of observing the data $\left\{z^{(i)}_{1:T_i}\right\}_{i=1}^N$ in the training range, i.e., by maximizing the (log-)likelihood: $\Phi^\star = \mathrm{argmax}_\Phi \mathcal{L}(\Phi)$, where

$$\mathcal{L}(\Phi) = \sum_{i=1}^N \log p\left(z^{(i)}_{1:T_i} \middle| \mathbf{x}^{(i)}_{1:T_i}, \Phi\right) = \sum_{i=1}^N \log p_{SS}\left(z^{(i)}_{1:T_i} \middle| \Theta^{(i)}_{1:T_i}\right). \tag{7}$$

[2]The covariates (features) can be time dependent (e.g. product price or a set of dummy variables indicating day-of-week) or time independent (e.g., product brand, category etc.).

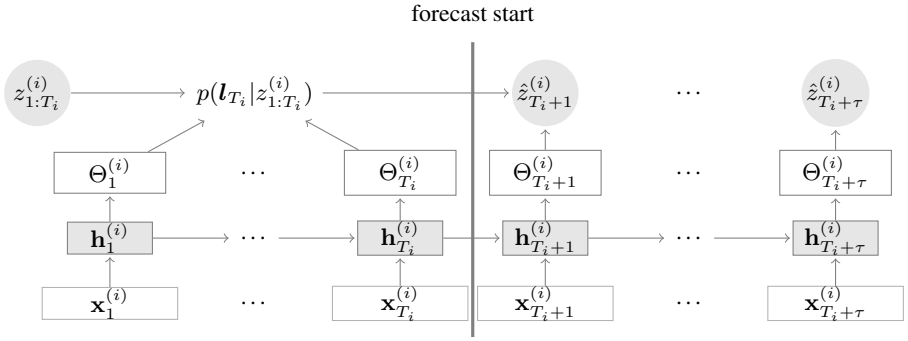

Figure 2: Illustration of the how the model is used to make forecasts after the network parameters Φ are learned. Given a time series $z^{(i)}_{1:T_i}$ in the training range $\{1, 2, \ldots, T_i\}$ (not necessarily in the training set) and associated features $\mathbf{x}^{(i)}_{1:T_i+\tau}$ for both training and prediction ranges, forecasts are produced as follows: (i) first the posterior of the latent state $p(\boldsymbol{l}_{T_i}|z_{1:T_i})$ for the last time step T_i in the training range is computed using the observations $z^{(i)}_{1:T_i}$ and the state space parameters $\Theta^{(i)}_{1:T_i}$ obtained by unrolling the RNN network in the training range; (ii) given the posterior of the latent state $p(\boldsymbol{l}_{T_i}|z_{1:T_i})$, prediction samples are generated by recursively applying the transition equation and the observation model (Eq. 8) where the state space parameters for the prediction range $\Theta^{(i)}_{T_i+1:T_i+\tau}$ are obtained by unrolling the RNN in the prediction range.

We can view each summand of $\mathcal{L}(\Phi)$ in Eq. 7 as a (negative) loss function, that measures compatibility between the state space model parameters $\Theta^{(i)}_{1:T_i}$ produced by the RNN when given input $\mathbf{x}^{(i)}_{1:T_i}$, and the true observations $z^{(i)}_{1:T_i}$. Each of these terms is a standard likelihood computation under linear-Gaussian state space model, which can be carried out efficiently via Kalman filtering (see e.g. [3, Sec. 24.3] or [22, Appendix A] for details): this involves mainly matrix-matrix and matrix-vector multiplications, which allows us to implement the overall log-likelihood computation using a neural network framework (MXNet), and use automatic differentiation to obtain gradients with respect to the parameters Φ, which are then used by a stochastic gradient descent-based optimization procedure. Note that a forward pass of our network to compute the loss (i.e., negative log-likelihood) essentially uses the same basic primitives as that of classical methods that learn parameters per time series independently. Thus, one can, in principle, extend our ideas to other instances of state space models by simply redefining their parameters as the outputs of the RNN.

4.2 Prediction

Once the network parameters Φ are learned, we can use them to address our original problem specified in Eq. 1, i.e., to make probabilistic forecasts for each given time series. Given Φ, we can compute the joint distribution over the prediction range for each time series analytically, as this joint distribution is a multivariate Gaussian. However, in practice it is often more convenient to represent the forecast distribution in terms of K Monte Carlo samples,

$$\hat{z}^{(i)}_{k,T_i+1:T_i+\tau} \sim p(z^{(i)}_{T_i+1:T_i+\tau}|z^{(i)}_{1:T_i}, \mathbf{x}^{(i)}_{1:T_i+\tau}, \Theta^{(i)}_{1:T_i+\tau}), \qquad k = 1, \ldots, K.$$

In order to generate prediction samples from a state space model, one first computes the posterior of the latent state $p(\boldsymbol{l}_T|z_{1:T})$ for the last time step T in the training range, and then recursively applies the transition equation and the observation model to generate prediction samples. More precisely, starting with sample $\ell_T \sim p(\ell_T|z_{1:T})$, we recursively apply

$$y_{T+t} = \boldsymbol{a}^\top_{T+t}\ell_{T+t-1} + b_{T+t}, \qquad t = 1, \ldots \tau, \tag{8a}$$

$$\hat{z}_{T+t} = y_{T+t} + \sigma_t\epsilon_t, \qquad \epsilon_{T+t} \sim \mathcal{N}(0, 1), \qquad t = 1, \ldots \tau, \tag{8b}$$

$$\boldsymbol{l}_{T+t} \sim \boldsymbol{F}_{T+t}\ell_{T+t-1} + \boldsymbol{g}_{T+t}\varepsilon_{T+t}, \qquad \varepsilon_{T+t} \sim \mathcal{N}(0, 1), \qquad t = 1, \ldots \tau - 1. \tag{8c}$$

In our case, we compute the posterior $p(\boldsymbol{l}^{(i)}_{T_i}|z^{(i)}_{1:T_i})$ for each of the time series $z^{(i)}_{1:T_i}$ by unrolling the RNN network in the training range to obtain $\Theta^{(i)}_{1:T_i}$ as shown in Figure 2, and then using the Kalman

filtering algorithm. Next, we unroll the RNN for the prediction range $t = T_i + 1, \ldots, T_i + \tau$ and obtain $\Theta^{(i)}_{T_i+1:T_i+\tau}$, then generate the prediction samples by recursively applying above equations K times.[3]

Remarks. Note that in our model, in contrast to classical and deep learning-based auto-regressive models (e.g. DeepAR [9]), target values are *not* used as inputs directly. This is a key feature of our method, and brings several advantages: (i) It makes the model more robust to noise, as target values are only incorporated through the likelihood term, where noise is properly accounted for; (ii) Missing target values can easily be handled by simply dropping the corresponding likelihood terms; (iii) Forecast sample path generation is computationally more efficient, as the RNN needs to be unrolled only once for the entire prediction (independent of the number of samples), whereas auto-regressive models (e.g. [9, 26]) have to be unrolled for each sample.

5 Experiments

Qualitative experiments. In our first experiment, we test whether our model effectively recovers the state space parameters if trained on synthetic data. For this, we generate five groups of time series from day-of-week seasonality model (see Appendix A.1 in the long version of the paper) but with different initial states and innovation parameters per group. For simplicity, we use the same observation noise σ_t for all time series. Each time series consists of six weeks of daily data and we use the first four weeks of all time series for training our model. We use a group identifier as an input feature. In the ideal case, for each time series the model should output the parameters of the state space model from which this time series was generated.

The state space model parameters in this case are given by $\Theta^{(i)}_t = (\boldsymbol{\mu}^{(i)}_0, \boldsymbol{\sigma}^{(i)}_0, \gamma^{(i)}_t, \sigma^{(i)}_t), \quad t = 1, \ldots, T_i + \tau$, where $T_i = 28, \forall i$ and $\tau = 14$. Note that except for $\sigma^{(i)}_t$, all the other parameters are different for each group. We encode the day-of-week seasonality using seven components of the latent state as in [22], i.e., $L = 7$ and $\boldsymbol{\mu}_0 \in \mathbb{R}^7$ (each component corresponds to a different day of the week). For simplicity, we fix the term $b^{(i)}_t = 0$ in this experiment.

To analyse how much data is required for recovering the parameters, we train three different models using $N = \{20, 40, 140\}$ examples from each group. Figure 3 shows the ground truth values of the parameters as well as the values obtained by our model for different number of training examples per group. The columns show the mean of the initial state $\boldsymbol{\mu}_0$ (seven values), innovation parameter γ_t as well as the standard deviation σ_t of the observations while the rows correspond to each of the five groups. The innovation parameter and the standard deviation of the observations are shown for the prediction range (two-weeks). The recovery of state space parameters becomes more accurate gradually as we increase the number of examples from 20 to 140. Moreover, these parameters are recovered reasonably well with $N = 200$ examples per group. In fact, the means of the initial state are exactly recovered. The standard deviation of the initial state $\boldsymbol{\sigma}_0$ (not plotted) has converged to a constant value in all cases. It turns out that the initial state means $\boldsymbol{\mu}_0$ are easy to recover but observation noise σ_t and the standard deviation of the initial state $\boldsymbol{\sigma}_0$ are the most difficult to recover.

Quantitative experiments. In our first quantitative experiment we evaluate how our model performs under small data regimes. For this, we use the publicly available datasets `electricity` and `traffic` [28]. The `electricity` dataset is a hourly time series of electricity consumption of 370 customers. The `traffic` dataset contains hourly occupancy rates (between 0 and 1) of 963 car lanes of San Francisco bay area freeways. As one expects, all the time series in these datasets exhibit hourly as well as daily seasonal patterns. As baselines we use the classical forecasting methods `auto.arima`, `ets` implemented in R's `forecast` package and a recent RNN-based method DeepAR [9]. We obtained results for DeepAR using the Amazon Sagemaker machine learning platform [1]. Since `DeepAR` and `DeepState` fit a joint model across the time series, both are given a time independent feature representing the category (i.e., the index) of the time series and time-based features like hour-of-the-day, day-of-the-week, day-of-the-month. For `DeepState` the size of SSM (i.e., latent dimension) directly depends on the granularity of the time series which determines the number of seasons. For hourly data, we use hour-of-day (24 seasons) as well as

[3] Note that the sampling procedure is trivially parallelizable over K samples once the parameters $\Theta^{(i)}_{T_i+1:T_i+\tau}$ and the distribution of the final latent state are computed.

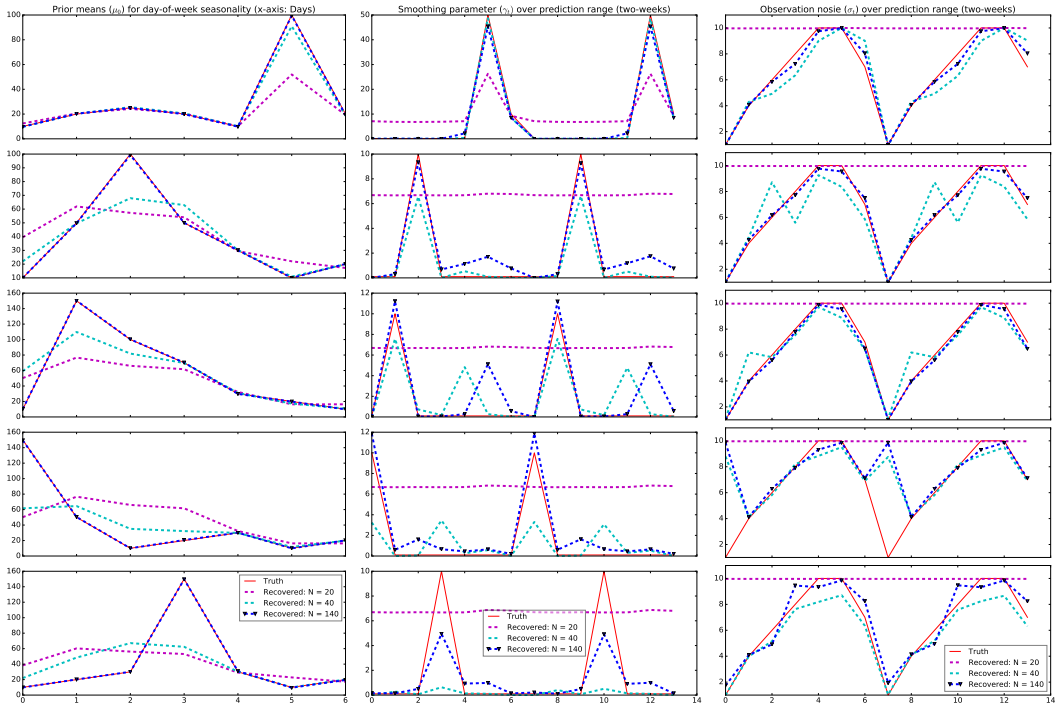

Figure 3: Recovery of state space parameters as the number of examples per group is increased. Columns show state space parameters while the rows correspond to each of the five groups. Each plot shows the true and the recovered values of the parameters with increased number of examples.

Datasets	Methods	2-weeks p50Loss	2-weeks p90Loss	3-weeks p50Loss	3-weeks p90Loss	4-weeks p50Loss	4-weeks p90Loss
`electricity`	`auto.arima`	0.283	0.109	0.291	0.112	0.30	0.11
	`ets`	0.121	0.101	0.130	0.110	0.13	0.11
	`DeepAR`	0.153	0.147	0.147	0.132	0.125	0.080
	`DeepState`	**0.087**	**0.05**	**0.085**	**0.052**	**0.085**	**0.057**
`traffic`	`auto.arima`	0.492	0.280	0.492	0.289	0.501	0.298
	`ets`	0.621	0.650	0.509	0.529	0.532	0.60
	`DeepAR`	0.177	0.153	**0.126**	**0.096**	0.219	0.138
	`DeepState`	**0.168**	**0.117**	0.170	0.113	**0.168**	**0.114**

Table 1: Data efficiency. Evaluation on `electricity` and `traffic` datasets with increasing training range. The forecast is evaluated on 7 days.

day-of-week (7 seasons) models and hence latent dimension is 31. We train each method on all time series of these datasets but vary the size of the training range $T_i \in \{14, 21, 28\}$ days. We evaluate all the methods on the next $\tau = 7$ days after the forecast start time using the standard $p50$ and $p90$- quantile losses. For a given collection of time series $\boldsymbol{z}$ and corresponding predictions $\hat{\boldsymbol{z}}$, the ρ-quantile loss for $\rho \in (0, 1)$ is defined as

$$\mathrm{QL}_\rho(\boldsymbol{z}, \hat{\boldsymbol{z}}) = 2\frac{\sum_{i,t} P_\rho(z_t^{(i)}, \hat{z}_t^{(i)})}{\sum_{i,t} |z_t^{(i)}|}, \quad P_\rho(z, \hat{z}) = \begin{cases} \rho(z - \hat{z}) & \text{if } z > \hat{z}, \\ (1-\rho)(\hat{z} - z) & \text{otherwise.} \end{cases}$$

The $p50$ and $p90$ losses are reported in Table 1. Overall our method achieves the best performance except for one case. Moreover, our method achieves very good performance even with 2-weeks data since it can explicitly incorporate seasonal structures (i.e., hour-of-day seasonality). Although `ets` and `auto.arima` incorporate such seasonal structures, their results are much worse. Inability to learn shared patterns across the time series could be a possible reason for their worse performance. DeepAR tries to learn seasonal patterns purely from the data and its performance generally improves with increased training size. We show some example predictions of our method in Appendix A.5.

Next, to compare against the matrix factorization method [28], we repeat the experiment in [28] that evaluates rolling-day forecasts for seven days (i.e., prediction horizon is one day and forecasts start time is shifted by one day after evaluating the prediction for the current day). Note that unlike MatFact, our method and DeepAR need not retrain after updating the forecast start time. We just extend the training range by one day and update the posterior of the latent state accordingly. The results are shown in Table 2. Since MatFact only produces point forecasts, we report normalized deviation as in [28], which in our case is equal to $p50$-loss. For `DeepAR` and `DeepState` we report both $p50$- and $p90$-losses. Note that our method is much better than MatFact even though the latter is retrained after each day of prediction. We get comparable results to DeepAR, which in our experience performs well with short forecast horizons.

	MatFact	DeepAR	DeepState
electricity	0.16	**0.075/0.04**	0.083/0.056
traffic	0.20	**0.161/0.099**	0.167/0.113

Table 2: Average $p50/p90$-loss for rolling-day prediction for seven days. MatFact outputs points predictions, so we only report $p50$-loss.

In the final experiment we evaluate our method on a diverse collection of publicly available datasets. We selected datasets containing time series from a single domain as our method is most suited for datasets of related time series. This includes monthly and quarterly time series from the tourism competition dataset [2] describing tourism demand, hourly time series from the M4 competition [20] and `parts` dataset [6] which contains monthly demand of spare parts at a US auto-mobile company. The number of time series in these data sets are 414 (`M4-Hourly`), 366 (`tourism-Monthly`), 427 (`tourism-Quarterly`) and 1046 (`parts`). For tourism and M4 datasets, train and test splits are already provided. The length of the training time series as well as the starting date differ for the time series in the `M4-Hourly` and tourism datasets. The prediction horizon for these data sets are 48 hours (`M4-Hourly`), 24 months (`tourism-Monthly`) and 8 quarters (`tourism-Quarterly`). For `parts` dataset we use the last 12 months as the prediction range while the training range contains 39 months. For both `tourism-Monthly` and `tourism-Quarterly` we used month-of-year seasonal model along with a trend component (to accommodate the trend visible in the training range of these time series) and for `parts` we used month-of-year seasonal model. For `M4-Hourly` we used the hour-of-day as well as day-of-week seasonal models. The $p50$ and $p90$ losses are reported in Table 3 for all the methods. These results further show that our method achieves the best performance overall.

	ets	auto.arima	DeepAR	DeepState
M4-Hourly	0.054/0.0267	0.052/0.0354	0.090/0.0304	**0.044/0.0266**
parts	1.639/1.0086	1.6444/1.0664	**1.273**/1.086	1.47/**0.935**
tourism-Monthly	**0.093/0.054**	0.0999/0.058	0.107/0.059	0.138/0.067
tourism-Quarterly	0.105/0.055	0.1241/0.062	0.11/0.062	**0.098/0.047**

Table 3: $p50/p90$-losses for datasets obtained from publicly available sources.

6 Conclusions

In this paper we propose a new approach to time series forecasting by marrying state space models with deep recurrent neural networks. This combination allows us to explicitly incorporate structural assumptions to handle small data regimes on one hand and learn complex patterns from raw time series data for larger data regimes on the other hand. Our experiments on synthetic data suggest that the model is capable of accurately recovering the parameters of the state space model from which the data is generated. We also showed, on real-world datasets, that the proposed method achieves state-of-the-art performance by comparing it against a recent RNN-based method, a matrix factorization method, as well as classical approaches such as ARIMA and ETS. Under regimes of limited data our method clearly outperforms the other methods by explicitly modelling seasonal structure. Extending our approach to other instances of state space models as well as non-Gaussian likelihoods are some of the directions we are currently pursuing. Some ideas of extending our method to non-Gaussian likelihoods are discussed in Appendix A.4 in the long version of the paper.

References

[1] Amazon Sagemaker: DeepAR Forecasting. https://docs.aws.amazon.com/sagemaker/latest/dg/deepar.html.

[2] G. Athanasopoulos, R. Hyndman, H. Song, and D. Wu. The tourism forecasting competition. *International Journal of Forecasting*, 27(3):822–844, 2011.

[3] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[4] George E. P. Box and Gwilym M. Jenkins. Some recent advances in forecasting and control. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 17(2):91–109, 1968.

[5] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[6] Nicolas Chapados. Effective bayesian modeling of groups of related count time series. In *International Conference on Machine Learning*, pages 1395–1403, 2014.

[7] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

[8] James Durbin and Siem Jan Koopman. *Time series analysis by state space methods*, volume 38. OUP Oxford, 2012.

[9] Valentin Flunkert, David Salinas, and Jan Gasthaus. DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *CoRR*, abs/1704.04110, 2017. URL http://arxiv.org/abs/1704.04110.

[10] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pages 2199–2207, 2016.

[11] Marco Fraccaro, Simon Kamronn, Ulrich Paquet, and Ole Winther. A disentangled recognition and nonlinear dynamics model for unsupervised learning. In *Advances in Neural Information Processing Systems*, pages 3604–3613, 2017.

[12] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[13] R. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder. *Forecasting with Exponential Smoothing: The State Space Approach*. Springer Series in Statistics. Springer, 2008. ISBN 9783540719182.

[14] Matthew Johnson, David K Duvenaud, Alex Wiltschko, Ryan P Adams, and Sandeep R Datta. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pages 2946–2954, 2016.

[15] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *ICLR*, 2017.

[16] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *ICLR*, 2014.

[17] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[18] Rahul G Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *AAAI*, pages 2101–2109, 2017.

[19] Nikolay Laptev, Jason Yosinsk, Li Li Erran, and Slawek Smyl. Time-series extreme event forecasting with neural networks at Uber. In *ICML Time Series Workshop*. 2017.

[20] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The M4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802–808, 2018.

[21] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

[22] Matthias Seeger, Syama Rangapuram, Yuyang Wang, David Salinas, Jan Gasthaus, Tim Januschowski, and Valentin Flunkert. Approximate Bayesian inference in linear state space models for intermittent demand forecasting at scale. *CoRR*, abs/1704.04110, 2017. URL `http://arxiv.org/abs/1704.04110`.

[23] Matthias W Seeger, David Salinas, and Valentin Flunkert. Bayesian intermittent demand forecasting for large inventories. In *Advances in Neural Information Processing Systems*, pages 4646–4654, 2016.

[24] Matthias W Seeger, David Salinas, and Valentin Flunkert. Bayesian intermittent demand forecasting for large inventories. In *Advances in Neural Information Processing Systems*, pages 4646–4654, 2016.

[25] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[26] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016.

[27] Ruofeng Wen Wen, Kari Torkkola, and Balakrishnan Narayanaswamy. A multi-horizon quantile recurrent forecaster. In *NIPS Time Series Workshop*. 2017.

[28] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 847–855. Curran Associates, Inc., 2016.

[29] Manzil Zaheer, Amr Ahmed, and Alexander J Smola. Latent LSTM allocation: Joint clustering and non-linear dynamic modeling of sequence data. In *International Conference on Machine Learning*, pages 3967–3976, 2017.

[30] Xun Zheng, Manzil Zaheer, Amr Ahmed, Yuan Wang, Eric P Xing, and Alexander J Smola. State space LSTM models with particle MCMC inference. *arXiv preprint arXiv:1711.11179*, 2017.

A Appendix

A.1 Example Instantiations of the State Space Model

For illustration, we describe a simple level-trend model as well a seasonality-based model, listing out all the parameters that need to be learned. In the level-trend model, the latent state $\boldsymbol{l}_t \in \mathbb{R}^2$ has two dimensions, one for representing the level and the other for the slope of the (linear) trend. The model is given by

$$\boldsymbol{a}_t = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \boldsymbol{F}_t = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \boldsymbol{g}_t = \begin{bmatrix} \alpha_t \\ \beta_t \end{bmatrix},$$

where $\alpha_t > 0$, $\beta_t > 0$. Both the level and slope components evolve over time by adding innovations $\alpha_t \varepsilon_t$ and $\beta_t \varepsilon_t$ respectively, so that $\alpha_t > 0$, $\beta_t > 0$ are the innovation strength for the level and slope respectively. The level at time t is the sum of level at $t-1$ and slope at $t-1$ (linear prediction). The initial state prior $P(\boldsymbol{l}_0)$ is given by $\boldsymbol{l}_0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \mathrm{diag}(\boldsymbol{\sigma}_0^2))$. We learn the state space parameters $\alpha_t > 0$, $\beta_t > 0$, $\boldsymbol{\mu}_0 > \mathbf{0}$, $\boldsymbol{\sigma}_0 > \mathbf{0}$ as well as the external contribution $b_t \in \mathbb{R}$ and the observation noise $\sigma_t > 0$. Thus, for a level-trend model, we have

$$\Theta_t^{(i)} = (\alpha_t^{(i)}, \beta_t^{(i)}, \boldsymbol{\mu}_0^{(i)}, \boldsymbol{\sigma}_0^{(i)}, b_t^{(i)}, \sigma_t^{(i)}), \quad t = 1, \ldots, T_i + \tau.$$

Note that these parameters vary for different time series according to $\Theta_t^{(i)} = \Psi(\mathbf{x}_{1:t}^{(i)}, \Phi)$ but all of them are parametrized by the common parameters Φ of the global RNN model.

In case of seasonality-based models, each seasonality pattern can be described by a set of seasonal factors (or seasons) associated with it. For example, in the day-of-week pattern there are seven factors, one for each day of the week. We can represent each factor as a component of the latent state $\boldsymbol{l}_t \in \mathbb{R}^7$ (see [22] for details). Then, for the day-of-week seasonality model, we have

$$\boldsymbol{a}_t = 1_{\{\text{day}(t)=j\}_{j=1}^7}, \quad \boldsymbol{F}_t = \boldsymbol{I}, \quad \boldsymbol{g}_t = \gamma_t \boldsymbol{a}_t,$$

where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{a}_t$ is an indicator vector specifying *when a factor is used*. The parameters to be learned in this case are

$$\Theta_t^{(i)} = (\gamma_t^{(i)}, \boldsymbol{\mu}_0^{(i)}, \boldsymbol{\sigma}_0^{(i)}, b_t^{(i)}, \sigma_t^{(i)}), \quad t = 1, \ldots, T_i + \tau.$$

A.2 Encoding of State Space Model Parameters

In order to constrain the real-valued output values of the RNN to the parameter domains of the state space model, we use the following transformations. Denote by $\mathbf{o}_t \in \mathbb{R}^H$ the output of the RNN at time t. For any state space model parameter θ_t, we first compute the affine transformation $\tilde{\theta}_t = \mathbf{w}_\theta^\top \mathbf{o}_t + b_\theta$ with separate weights $\mathbf{w}_\theta \in \mathbb{R}^H$ and biases $b_\theta \in \mathbb{R}$ for each parameter θ. All of these weights and biases are included in Φ and learned. We then transform $\tilde{\theta}_t$ to the domain of the parameter by applying:

- for real-valued parameters, e.g. b_t: no transformation,
- for positive parameters $\theta > 0$: the softplus function $\theta_t = \log(1 + \exp(\tilde{\theta}_t))$,
- for bounded parameters $\theta \in [a, b]$: a scaled and shifted sigmoid $\theta_t = (b-a)\frac{1}{1+\exp(-\tilde{\theta}_t)} + a$.

In practice, it is often advisible to impose stricter bounds than theoretically required, e.g. enforcing an upper bound on the observation noise variance σ_t or a lower bound on the innovation strengths can stabilize the training procedure in the presence of outliers.

A.3 Likelihood computation

In this section we provide the technical details for computing the likelihood of our model needed for its training. Recall that the log-likelihood of the observations $\{\boldsymbol{z}_i\}_{i=1}^N$, under our model, is given by

$$\sum_{i=1}^{N} \log p_{SS}(z_{1:T_i}^{(i)} | \Theta_{1:T_i}^{(i)}), \ \Theta_t^{(i)} = \Psi(\mathbf{x}_{1:t}^{(i)}, \Phi).$$

Each of these terms here can be further decomposed as

$$p_{SS}(z^{(i)}_{1:T_i}|\Theta^{(i)}_{1:T_i}) = \prod_{t=1}^{T_i} p(z^{(i)}_t|z^{(i)}_{1:t}, \Theta^{(i)}_{1:T_i}).$$

The factors here can be computed using the Kalman filtering algorithm [3]. Here filtering refers to finding the distribution $p(\boldsymbol{l}^{(i)}_{t-1}|z^{(i)}_{1:t}), t = 1, \ldots, T_i$ of the latent state given all the observations up to the current time step. These filtered distributions are Gaussians $p(\boldsymbol{l}^{(i)}_{t-1}|z^{(i)}_{1:t}) = \mathcal{N}(\boldsymbol{l}^{(i)}_{t-1}|\boldsymbol{f}^{(i)}_t, \boldsymbol{S}^{(i)}_t)$ in our case. The mean $\boldsymbol{f}^{(i)}_t$ and covariance $\boldsymbol{S}^{(i)}_t$ of these filtered distributions are found using Kalman filtering algorithm. Since in our case observations at each time point are scalars, the updates in Kalman filtering algorithm involve mainly matrix-matrix and matrix-vector multiplications.

Once we have the filtered distributions, the factors in the likelihood for each of the observations $z^{(i)}$ can be computed as

$$p(z^{(i)}_t|z^{(i)}_{1:t-1}, \Theta^{(i)}_{1:T_i}) = \mathcal{N}(z^{(i)}_t|\boldsymbol{\mu}^{(i)}_t, \boldsymbol{\Sigma}^{(i)}_t),$$

where

$$\begin{aligned}
&\boldsymbol{\mu}^{(i)}_1 = \boldsymbol{a}^{(i)^\top}_1 \boldsymbol{\mu}^{(i)}_0, && \boldsymbol{\Sigma}^{(i)}_1 = \boldsymbol{a}^{(i)^\top}_1 \boldsymbol{\Sigma}^{(i)}_0 \boldsymbol{a}^{(i)}_1 + \sigma^{(i)^2}_1 && t = 1,\\
&\boldsymbol{\mu}^{(i)}_t = \boldsymbol{a}^{(i)^\top}_t \boldsymbol{F}^{(i)}_t \boldsymbol{f}^{(i)}_{t-1}, && \boldsymbol{\Sigma}^{(i)}_t = \boldsymbol{a}^{(i)^\top}_t \left(\boldsymbol{F}^{(i)}_t \boldsymbol{S}^{(i)}_t \boldsymbol{F}^{(i)^\top}_t + \boldsymbol{g}^{(i)}_t \boldsymbol{g}^{(i)^\top}_t\right) \boldsymbol{a}^{(i)}_t + \sigma^{(i)^2}_t, && t > 1.
\end{aligned}$$

A.4 Non-Gaussian Likelihoods

A common assumption in classical forecasting methods is normally distributed data. When the data deviates from this assumption, a widely-used approach is to apply a (power) transformation to the data to make them well-behaved, e.g.,via the widely used Box-Cox transformation [5].

In one alternative, Seeger et al. [24, 22] propose a Bayesian latent state forecaster, which, instead of using Gaussian likelihood, admits a Poisson and a *multi-stage* likelihood that is tailored for intermittent sales patterns for large inventory. Laplace approximation in particular ensures the scalability of this approach. However, deriving the Laplace approximation for each likelihood function is cumbersome, and more importantly requires careful numerical attention, especially for non-logconcave likelihoods such as negative binomial.

Here, we briefly discuss a possible approach to handle arbitrary likelihood functions based on *Variational Auto-Encoder* [16, 21] (VAE) framework in the lines of [11]. For this, we first extend the emission equation (3) to

$$z_t \sim \Pr(\cdot|u_t), \qquad u_t = y_t + \sigma_t \epsilon_t,$$

where we have both u_t and l_t as latent variables. The general idea of VAE is to use neural networks to parameterize the observation model $\Pr(z_t|u_t)$ and a variational posterior distribution. The parameters of the neural network are learned jointly to maximize a stochastic approximation of the evidence lower bound (ELBO). In the forecasting case, the typical assumption is that the likelihood function is known in advance. Next we discuss about the inference and model learning. Our goal is to optimize the marginal likelihood of $\boldsymbol{z} := z_{1:T}$, which is intractable without Gaussian emission. We therefore optimize a variational lower bound,

$$\log p(\boldsymbol{z}) = \log \int p(\boldsymbol{z}, \boldsymbol{u}, \boldsymbol{l}) d\boldsymbol{u}\, d\boldsymbol{l} \geqslant \mathbb{E}_{q_\phi(\boldsymbol{u},\boldsymbol{l})} \log \left[\frac{p(\boldsymbol{z}, \boldsymbol{u}, \boldsymbol{l})}{q_\phi(\boldsymbol{u}, \boldsymbol{l})}\right], \tag{9}$$

where $\boldsymbol{u} = u_{1:T}$, $\boldsymbol{l} = l_{1:T}$ and $q_\phi(\cdot)$ is an NN parameterized by ϕ to approximate the variational posterior, also known as the *recognition network*. We use the following variational posterior to approximate the true posterior $p(\boldsymbol{u}, \boldsymbol{l}|\boldsymbol{z})$,

$$q_\phi(\boldsymbol{u}, \boldsymbol{l}|\boldsymbol{z}) = q_\phi(\boldsymbol{u}|\boldsymbol{z}) p(\boldsymbol{l}|\boldsymbol{u}),$$

where the key is to make the second term coming from the exact conditional posterior from Linear Dynamical System (LDS) such that given $\boldsymbol{u}$, the inference becomes the standard Kalman filtering (see also [11]). Thus, the variational lower bound can be simplified to

$$\begin{aligned}
\mathbb{E}_{q_\phi(\boldsymbol{u},\boldsymbol{l})} \log \left[\frac{p(\boldsymbol{z}, \boldsymbol{u}, \boldsymbol{l})}{q_\phi(\boldsymbol{u}, \boldsymbol{l})}\right] &= \mathbb{E}_{q_\phi(\boldsymbol{u})} \left(\log \left[\frac{p(\boldsymbol{z}|\boldsymbol{u})}{q_\phi(\boldsymbol{u})}\right] + \log p(\boldsymbol{u})\right)\\
&\approx \frac{1}{L} \left(\log \left[\frac{p(\boldsymbol{z}|\widetilde{\boldsymbol{u}}_j)}{q_\phi(\widetilde{\boldsymbol{u}}_j)}\right] + \log p(\widetilde{\boldsymbol{u}}_i)\right), \qquad \widetilde{\boldsymbol{u}}_j \sim q_\phi(\boldsymbol{u}), \quad j = 1, \cdots, L.
\end{aligned}$$

The first term gives rise to the likelihood of the observations. For some cases (e.g., Poisson), we can analytically integrate it out. The second term is the marginal likelihood that can be computed with Kalman Filtering. Different NN structures can be selected for the variational encoder $q_\phi(\boldsymbol{u}|\boldsymbol{z})$. In particular, we consider the bi-directional LSTM to be the desired structure, given that it considers information from both the past and the future, analogously to *backwards messages* in Kalman smoothing. Note that the recognition network is *only* used in the training phrase to better approximate the posterior distribution of the latent Poisson rate (intensity function). Thus, there is no information leak for the desired forecasting use case.

A.5 Visualization of Forecasts

Here we show example predictions of our model by plotting the median, $p10$ and $p90$ forecasts for two randomly selected time series from `electricity` and `traffic` datasets.

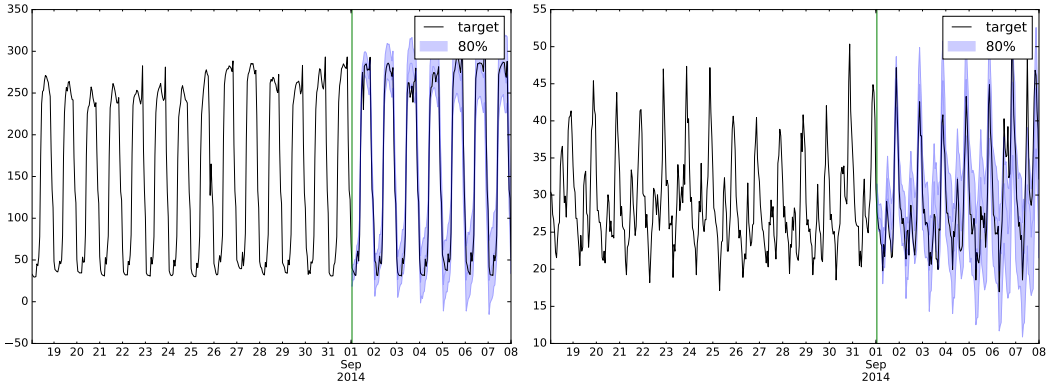

Figure 4: 80% prediction intervals obtained by our method on two randomly selected time series from `electricity` dataset. The training range is two weeks while the prediction range is one week. The forecasts start date is indicated by the green vertical line.

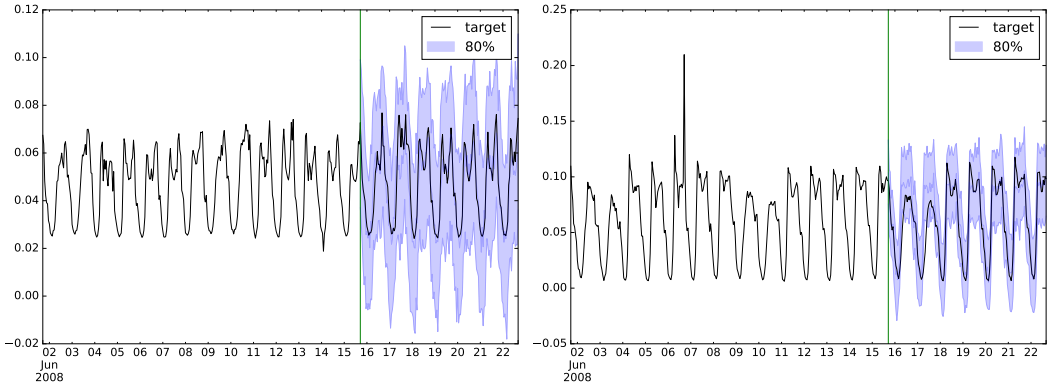

Figure 5: 80% prediction intervals obtained by our method on two randomly selected time series from `traffic` dataset. The training range is two weeks while the prediction range is one week. The forecasts start date is indicated by the green vertical line.