International Journal of Forecasting 36 (2020) 1181–1191

Contents lists available at ScienceDirect

International Journal of Forecasting

journal homepage: www.elsevier.com/locate/ijforecast

# DeepAR: Probabilistic forecasting with autoregressive recurrent networks

David Salinas, Valentin Flunkert *, Jan Gasthaus, Tim Januschowski

*Amazon Research, Germany*

ARTICLE INFO

*Keywords:*
Probabilistic forecasting
Neural networks
Deep learning
Big data
Demand forecasting

ABSTRACT

Probabilistic forecasting, i.e., estimating a time series' future probability distribution given its past, is a key enabler for optimizing business processes. In retail businesses, for example, probabilistic demand forecasts are crucial for having the right inventory available at the right time and in the right place. This paper proposes *DeepAR*, a methodology for producing accurate probabilistic forecasts, based on training an autoregressive recurrent neural network model on a large number of related time series. We demonstrate how the application of deep learning techniques to forecasting can overcome many of the challenges that are faced by widely-used classical approaches to the problem. By means of extensive empirical evaluations on several real-world forecasting datasets, we show that our methodology produces more accurate forecasts than other state-of-the-art methods, while requiring minimal manual work.



## 1. Introduction

The majority of the forecasting methods that are in use today have been developed in the setting of forecasting individual or small groups of time series. In this approach, model parameters for each given time series are estimated independently for each time series from past observations. Although good and freely available automatic forecasting packages exist (such as that of Hyndman & Khandakar, 2008), typically models are selected manually, both in practice and in research, in order to account for different factors, such as autocorrelation structure, trend and seasonality, and in particular other explanatory variables. The fitted model is then used to forecast the time series into the future according to the model dynamics, with probabilistic forecasts possibly being admitted through simulation or closed-form expressions for the predictive distributions. Many methods in this class are based on the classical Box-Jenkins methodology, exponential smoothing techniques, or state space models (Box, Jenkins, Reinsel, & Ljung, 2015; Durbin & Koopman, 2012; Hyndman, Koehler, Ord, & Snyder, 2008).

Over the last few years, a different type of forecasting problem has gained increasing importance in many applications. Rather than predicting individual or small numbers of time series, one is faced with the need to forecast thousands or millions of related time series. Examples of such problems include forecasting the energy consumption at the level of the individual household, forecasting the load for servers in a data center, or forecasting the demand for each of the products offered by a large retailer. In each of these scenarios, a substantial amount of data on the past behaviors of similar, related time series can be used to produce a forecast for an individual time series. Using data from related time series (the energy consumption of other households, the demand for other products) not only allows more complex (and hence potentially more accurate) models to be fitted without over-fitting, but can also alleviate the human time- and labor-intensive steps of selecting and preparing covariates and selecting models that classical techniques require.

* Corresponding author.
*E-mail addresses:* dsalina@amazon.com (D. Salinas), flunkert@amazon.com (V. Flunkert), gasthaus@amazon.com (J. Gasthaus), tjnsch@amazon.com (T. Januschowski).

https://doi.org/10.1016/j.ijforecast.2019.07.001

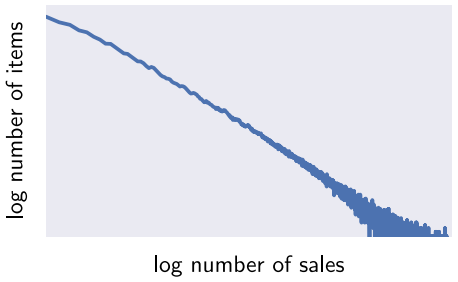


**Fig. 1.** Histogram of the average number of sales per item (in log–log scale) for 500K items of ec, showing the scale-free nature (approximately straight line) of the ec dataset (axis labels omitted due to the non-public nature of the data). This figure was created by grouping the items in the dataset into buckets according to their average weekly sales and counting the number of items in each bucket. The number of items per bucket is then plotted against the number of sales (both axes in log scale).

This work presents DeepAR, a forecasting method based on autoregressive recurrent neural networks, which learns a *global* model from historical data of *all time series* in the dataset. Our method builds upon previous work on deep learning for time series data (Graves, 2013; van den Oord et al., 2016; Sutskever, Vinyals, & Le, 2014), and tailors a similar long short-term memory (LSTM; Hochreiter & Schmidhuber, 1997) based recurrent neural network architecture to the probabilistic forecasting problem.

One challenge that is encountered often when attempting to learn from multiple time series jointly in real-world forecasting problems is that the magnitudes of the time series differ widely, and the distribution of the magnitudes is skewed strongly in practical applications. This issue is illustrated in Fig. 1, which shows a histogram (in log–log scale) of the number of sales per item for millions of items sold by Amazon. We refer to this as the *velocity* of an item. Segmenting time series according to their velocity is crude yet intuitive, is easy to convey to non-experts and suffices for our arguments and the main application further on.

The distribution of the velocity over a few orders of magnitude is an approximate power-law. This observation has fundamental implications for forecasting methods that attempt to learn a single model from such datasets. The scale-free nature of the distribution makes it difficult to divide the dataset into sub-groups of time series in a certain velocity band and learn separate models for them, as each such velocity sub-group will have a similar skew. Furthermore, group-based regularization schemes, such as that proposed by Chapados (2014), may fail, as the velocities will be differ vastly within each group. Finally, such skewed distributions make the use of certain commonly-employed normalization techniques, such as input standardization or batch normalization (Ioffe & Szegedy, 2015), less effective.

The main contributions of this paper are twofold: (1) we propose a recurrent neural network (RNN) architecture for probabilistic forecasting, which incorporates a negative binomial likelihood for count data as well as special treatment for the case where the magnitudes of the time series vary widely; and (2) we demonstrate empirically, on several real-world datasets, that this model produces accurate probabilistic forecasts across a range of input characteristics, thus showing that modern deep learning based approaches can be effective at addressing the probabilistic forecasting problem. This provides further evidence that neural networks are a useful, general-purpose forecasting technique (see e.g. Kourentzes, 2013, for a further successful application of a less deep neural network).

In addition to providing a better forecast accuracy than previous methods, our approach has a number of key advantages over classical approaches:

(i) As the model learns seasonal behaviors and dependencies on given covariates across time series, minimal manual intervention in providing covariates is needed in order to capture complex, group-dependent behavior.
(ii) DeepAR makes probabilistic forecasts in the form of Monte Carlo samples that can be used to compute consistent quantile estimates for all sub-ranges in the prediction horizon.
(iii) By learning from similar items, our method is able to provide forecasts for items that have little or no history available, a case where traditional single-item forecasting methods fail.
(vi) Our approach does not assume Gaussian noise, but can incorporate a wide range of likelihood functions, allowing the user to choose one that is appropriate for the statistical properties of the data.

Points (i) and (iii) are what sets DeepAR apart from classical forecasting approaches, while (ii) and (iv) are instrumental in the production of accurate, calibrated forecast distributions that are learned from the historical behavior of all of the time series jointly, which has not been addressed by previous related methods (see Section 2). Such probabilistic forecasts are of crucial importance in many applications, as, in contrast to point forecasts, they enable optimal decision making under uncertainty by minimizing risk functions, i.e., expectations of some loss function under the forecast distribution.

This paper is structured as follows. We begin by discussing related work in Section 2. Section 3 provides a brief overview of the key deep learning techniques on which we build in this paper. These techniques are well-established in the machine learning community, but we provide them here for convenience. Section 4 discusses the architecture of the DeepAR model in detail, and also details how the training of the model works. Section 5 provides empirical evidence for the practical usability of our method, and we conclude in Section 6.

## 2. Related work

In practical forecasting problems, especially in the demand forecasting domain, one is often faced with highly lumpy or intermittent data which violate the core assumptions of many classical techniques, such as Gaussianity, stationarity, or homoscedasticity of the time series. This has long been recognized as an important issue.

Intermittent demand can be treated using techniques that range from classical methods (Croston, 1972) to neural networks (Gutierrez, Solis, & Mukhopadhyay, 2008) directly. Data-preprocessing methods such as Box–Cox transformations (Box & Cox, 1964) or differencing (Hyndman & Athanasopoulos, 2012, provides an overview) have been proposed in order to alleviate unfavorable characteristics of time series, but with mixed results. Other approaches incorporate more suitable likelihood functions, such as the zero-inflated Poisson distribution, the negative binomial distribution (Snyder, Ord, & Beaumont, 2012), a combination of both (Chapados, 2014), or a tailored multi-stage likelihood (Seeger, Salinas, & Flunkert, 2016). We approach the (demand) forecasting problem by incorporating appropriate likelihoods and combining them with non-linear data transformation techniques, as learned by a (deep) neural network. In particular, we use a negative binomial likelihood in the case of demand forecasting, which improves the accuracy but precludes us from applying standard data normalization techniques directly. By using deeper networks than have been proposed previously in the forecasting literature, we allow the neural network to represent more complex data transformations. Goodfellow, Bengio, and Courville (2016) provides a comprehensive overview of modern deep neural networks, including justifications of why deep neural networks are preferable to shallow and wide neural networks.

When faced with time series as they occur in industrial applications, sharing information across time series is key to improving the forecast accuracy. However, this can be difficult to accomplish in practice, due to the often heterogeneous nature of the data. A prominent approach to sharing information across time series is to use clustering techniques such as *k*-means clustering to compute seasonality indices, which are then combined with classic forecasting models (see for example the *Foresight* 2007 Spring issue on seasonality for a number of examples, as well as the papers by ,Chen & Boylan, 2008 and Mohammadipour, Boylan, & Syntetos, 2012). Other examples include the explicit handling of promotional effects (see e.g. Trapero, Kourentzes, & Fildes, 2015, and references therein) via pooled principal component analysis regression. The latter is another instance of using an unsupervised learning technique as a pre-processing step. These effects need to be handled in practical applications, which leads to complex pipelines that are difficult both to tune and to maintain. The complexity of such pipelines is likely to increase when one needs to address specific sub-problems such as forecasts for new products. Effectively, one decomposes the overall forecasting problem into a number of distinct forecasting sub-problems and applies a dedicated model or even a chain of models to each one. These models can range from classical statistical models (e.g. Hyndman et al., 2008) to machine learning models (e.g. Laptev, Yosinsk, Li Erran, & Smyl, 2017; Wen, Torkkola, & Narayanaswamy, 2017) and judgmental approaches (e.g. Davydenko & Fildes, 2013). However, the forecasting problem might not lend itself to a decomposition into a sequence of distinct procedures, in which case one would have to think about model consolidation, blending, or ensembling (e.g. Oliveira & Torgo, 2014; Timmermann, 2006). These techniques increase the complexity still further. Deep neural networks offer an alternative to such pipelines. Such models require a limited amount of standardized data pre-processing, after which the forecasting problem is solved by learning an end-to-end model. In particular, data-processing is included in the model and optimized jointly towards the goal of producing the best possible forecast. In practice, deep learning forecasting pipelines rely almost exclusively on what the model can learn from the data, unlike traditional pipelines, which rely heavily on heuristics such as expert-designed components and manual covariate design.

Other approaches to the sharing of information across time series are via matrix factorization methods (e.g. the recent work of Yu, Rao, and Dhillon (2016)). We compare our approach to this method directly in Section 5, and show how we empirically outperform it. Further methods that share information include Bayesian methods that share information via hierarchical priors (Chapados, 2014) and by making use of any hierarchical structure that may be present in the data (Ben Taieb, Taylor, & Hyndman, 2017; Hyndman, Ahmed, Athanasopoulos, & Shang, 2011).

Finally, we note that neural networks have been being investigated in the context of forecasting for a long time by both the machine learning and forecasting communities (for more recent work considering LSTM cells, see for example the numerous references in the surveys by Zhang, Eddy Patuwo, & Hu, 1998, Fildes, Nikolopoulos, Crone, & Syntetos, 2008, and Gers, Eck, & Schmidhuber, 2001). Outside of the forecasting community, time series models based on RNNs have been applied very successfully to various other applications, such as natural language processing (NLP) (Graves, 2013; Sutskever et al., 2014), audio modeling (van den Oord et al., 2016) or image generation (Gregor, Danihelka, Graves, Rezende, & Wierstra, 2015). Direct applications of RNNs to forecasting include the recent papers by Wen et al. (2017) and Laptev et al. (2017). Our work differs from these in that it provides a comprehensive benchmark including publicly available datasets and a fully probabilistic forecast.[1]

Within the forecasting community, neural networks in forecasting have been applied typically to individual time series, i.e., a different model is fitted to each time series independently (Díaz-Robles et al., 2008; Ghiassi, Saidane, & Zimbra, 2005; Hyndman & Athanasopoulos, 2018; Kaastra & Boyd, 1996). Kourentzes (2013) applies neural networks specifically to intermittent data. The author uses a feed-forward neural network (which, by design, ignores

[1] Since the initial pre-print of the present work became available, neural networks have received an increasing amount of attention, see for example (Bandara, Bergmeir, & Smyl, 2017; Gasthaus, Benidis, Wang, Rangapuram, Salinas, Flunkert, et al., 2019; Oreshkin, Carpov, Chapados, & Bengio, 2019; Rangapuram, Seeger, Gasthaus, Stella, Wang, & Januschowski, 2018; Smyl, Ranganathan, & Pasqua, 2018; Toubeau, Bottieau, Vallée, & De Grève, 2018). The winning solution to the M4 competition was based on a neural network (Makridakis, Spiliotis, & Assimakopoulos, 2018; Smyl et al., 2018). Future work will address a systematic review of these methods.

the sequential nature of the data) and provides a shallow neural network that contains one hidden layer. Given the limited training data, this shallowness allows for effective training, and the author obtains promising results. However, the use of larger datasets and data augmentation techniques such as we discuss further on allows us to train deeper neural networks. Furthermore, we provide more details regarding the set-up for the training of the RNN and a methodology for obtaining probabilistic forecasts. Gutierrez et al. (2008) also use a feed-forward neural network for lumpy demand forecasting data. RNNs have the advantage for modeling the sequential nature of time series explicitly, and thus have a smaller number of parameters that need fitting. Our work differs from these papers in that it provides more details on the neural network architecture and utilizes recent advances from the machine learning community on the training of RNNs. In addition, in contrast to the aforementioned papers, our work also provides the full probability distribution of the forecasts, which is crucial for optimal decision making in downstream applications.

## 3. Background: RNNs

We assume that the reader is familiar with basic neural network nomenclature around multi-layer perceptrons (MLPs) or feed-forward neural networks, which have been applied successfully to forecasting problems by the forecasting community (e.g., Gutierrez et al., 2008; Kourentzes, 2013); in particular, modern forecasting textbooks include MLPs, e.g., (Hyndman & Athanasopoulos, 2018). We refer the interested reader to Goodfellow et al. (2016) for a comprehensive introduction to modern deep learning approaches and Faloutsos, Flunkert, Gasthaus, Januschowski, and Wang (2019) for a tutorial that focuses on forecasting. In what follows, we provide a brief introduction to recurrent neural networks (RNNs) and key techniques for handling them, as they have not been dealt with extensively in the forecasting literature (though the machine learning community have applied them to forecasting problems with some success; e.g. Laptev et al., 2017; Wen et al., 2017). We follow (Goodfellow et al., 2016) in our exposition.

A classic dynamic system driven by an external signal $x^{(t)}$ is given by

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta), \tag{1}$$

where $h^{(t)}$ is the state of the system at step $t$ and $\theta$ is a parameter of a transit function $f$. RNNs use Eq. (1) to model the values of their hidden units (recall that a hidden unit of a neural network is one that is neither the input layer nor the output layer).

This means that RNNs are deterministic, non-linear dynamic systems, in contrast to additive exponential smoothing in state space form, which can be represented as linear non-deterministic dynamic systems with a single source of error/innovation (Hyndman et al., 2008).

Fig. 2 contains a depiction of a simple RNN. The recursive structure of the RNN means that fewer parameters need to be learned than in the case of MLPs. However, a technical difficulty arises in the training of RNNs via a gradient-based optimization procedure. The recursive nature of RNNs often results in ill-conditioned optimization problems which are referred to commonly in the machine learning community as vanishing or exploding gradients. The long short-term memory (LSTM) model (Hochreiter & Schmidhuber, 1997) alleviates this problem (among other favorable properties), and it is the approach that we adopt in this paper. We do not present the full functional form of LSTMs, as this is unnecessary for our arguments, but again refer to the paper by Goodfellow et al. (2016) for an overview and a comprehensive exposition. All modern neural learning packages, such as that of Chen et al. (2015), include an implementation of LSTM-based RNNs.

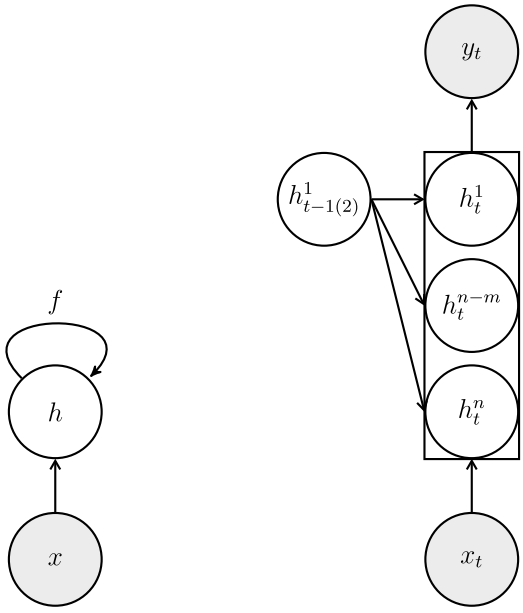


**Fig. 2.** *Left.* A RNN without an output layer. *Right.* A partially unrolled RNN with an output layer and multiple hidden units.

In addition to LSTMs, another concept from RNNs will be useful: the *encoder–decoder* framework, which allows RNNs to be used to map an input sequence $x = (x_1, \ldots, x_{n_x})$ to an output sequence $y = (y_1, \ldots, y_{n_y})$ of differing lengths. This idea is used frequently in NLP and machine translation, and works as follows. Given an input sequence, a first RNN processes this sequence and emits a so-called *context*, a vector or a sequence of vectors. In practice, this is often the last state $h_{n_x}$ of the encoder RNN. A second RNN, the decoder RNN, is conditioned on the context in order to generate the output sequence. The two RNNs are trained jointly to maximize the average of $\log P(y|x)$ over all pairs $x, y$ in the training set. Section 4 discusses the application of this concept to forecasting.

## 4. Model

Denoting the value of time series (for an item) $i$ at time $t$ by $z_{i,t}$, our goal is to model the conditional distribution

$$P(\mathbf{z}_{i,t_0:T} | \mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T})$$

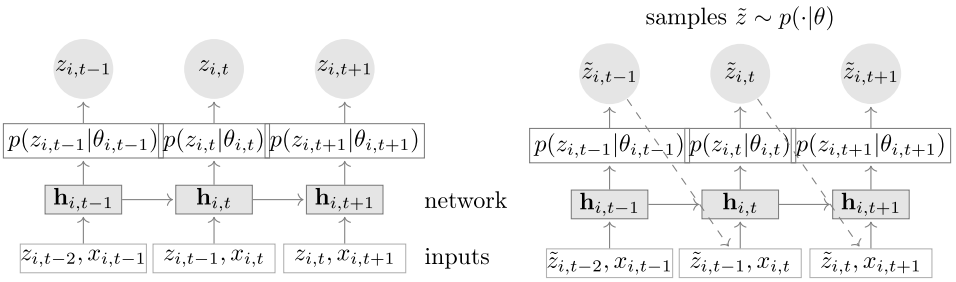


**Fig. 3.** Summary of the model. Training (left): At each time step $t$, the inputs to the network are the covariates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, and the previous network output $\mathbf{h}_{i,t-1}$. The network output $\mathbf{h}_{i,t} = h(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta)$ is then used to compute the parameters $\theta_{i,t} = \theta(\mathbf{h}_{i,t}, \Theta)$ of the likelihood $p(z|\theta)$, which is used for training the model parameters. For prediction, the history of the time series $z_{i,t}$ is fed in for $t < t_0$, then in the prediction range (right) for $t \geq t_0$ a sample $\hat{z}_{i,t} \sim p(\cdot|\theta_{i,t})$ is drawn and fed back for the next point until the end of the prediction range $t = t_0 + T$, generating one sample trace. Repeating this prediction process yields many traces that represent the joint predicted distribution.

of the future of each time series $[z_{i,t_0}, z_{i,t_0+1}, \ldots, z_{i,T}] := \mathbf{z}_{i,t_0:T}$ given its past $[z_{i,1}, \ldots, z_{i,t_0-2}, z_{i,t_0-1}] := \mathbf{z}_{i,1:t_0-1}$, where $t_0$ denotes the time point from which we assume $z_{i,t}$ to be unknown at prediction time, and $\mathbf{x}_{i,1:T}$ are covariates that are assumed to be known for all time points. To limit ambiguity, we avoid the terms "past" and "future" and will refer to the time ranges $[1, t_0 - 1]$ and $[t_0, T]$ as the *conditioning range* and *prediction range*, respectively. The conditioning range corresponds to the encoder range introduced in Section 3 and the prediction range to the decoder range. During training, both ranges have to lie in the past so that the $z_{i,t}$ are observed, but during prediction, $z_{i,t}$ is only available in the conditioning range. Note that the time index $t$ is relative, i.e. $t = 1$ can correspond to a different actual/absolute time period for each $i$.

Our model, summarized in Fig. 3, is based on an autoregressive recurrent network architecture (Graves, 2013; Sutskever et al., 2014). We assume that our model distribution $Q_\Theta(\mathbf{z}_{i,t_0:T}|\mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T})$ consists of a product of likelihood factors

$$Q_\Theta(\mathbf{z}_{i,t_0:T}|\mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T}) = \prod_{t=t_0}^{T} Q_\Theta(z_{i,t}|\mathbf{z}_{i,1:t-1}, \mathbf{x}_{i,1:T}) = \prod_{t=t_0}^{T} p(z_{i,t}|\theta(\mathbf{h}_{i,t}, \Theta)),$$

parametrized by the output $\mathbf{h}_{i,t}$ of an autoregressive recurrent network

$$\mathbf{h}_{i,t} = h\left(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta\right), \tag{2}$$

where $h$ is a function that is implemented by a multi-layer recurrent neural network with LSTM cells parametrized by $\Theta$. We provide further details of the architecture and hyper-parameters in Section 5. The model is autoregressive, in the sense that it consumes the observation at the last time step $z_{i,t-1}$ as an input, as well as recurrent, i.e., the previous output of the network $\mathbf{h}_{i,t-1}$ is fed back as an input at the next time step. The likelihood[2] $p(z_{i,t}|\theta(\mathbf{h}_{i,t}))$ is a fixed distribution with parameters that are given by a function $\theta(\mathbf{h}_{i,t}, \Theta)$ of the network output $\mathbf{h}_{i,t}$ (see below).

Information about the observations in the conditioning range $\mathbf{z}_{i,1:t_0-1}$ is transferred to the prediction range through the initial state $\mathbf{h}_{i,t_0-1}$. In the sequence-to-sequence setup, this initial state is the output of an *encoder network*. While in general this encoder network can have a different architecture, in our experiments we opt to use the same architecture for the model in both the conditioning range and the prediction range (corresponding to the *encoder* and *decoder* in a sequence-to-sequence model). Further, we share weights between them, so that the initial state for the decoder $\mathbf{h}_{i,t_0-1}$ is obtained by computing Eq. (2) for $t = 1, \ldots, t_0 - 1$, where all required quantities are observed. The initial states of both the encoder $\mathbf{h}_{i,0}$ and $z_{i,0}$ are initialized to zero.

Given the model parameters $\Theta$, we can obtain joint samples $\tilde{\mathbf{z}}_{i,t_0:T} \sim Q_\Theta(\mathbf{z}_{i,t_0:T}|\mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T})$ directly through ancestral sampling. First, we obtain $\mathbf{h}_{i,t_0-1}$ by computing Eq. (2) for $t = 1, \ldots, t_0$. For $t = t_0, t_0+1, \ldots, T$, we sample $\tilde{z}_{i,t} \sim p(\cdot|\theta(\tilde{\mathbf{h}}_{i,t}, \Theta))$, where $\tilde{\mathbf{h}}_{i,t} = h\big(\tilde{\mathbf{h}}_{i,t-1}, \tilde{z}_{i,t-1}, \mathbf{x}_{i,t}, \Theta\big)$ initialized with $\tilde{\mathbf{h}}_{i,t_0-1} = \mathbf{h}_{i,t_0-1}$ and $\tilde{z}_{i,t_0-1} = z_{i,t_0-1}$. Samples from the model obtained in this way can then be used to compute quantities of interest, e.g. quantiles of the distribution of the sum of values for some future time period.

### 4.1. Likelihood model

The likelihood $p(z|\theta)$ determines the "noise model", and should be chosen to match the statistical properties of the data. In our approach, the network directly predicts *all* parameters $\theta$ (e.g. mean *and* variance) of the probability distribution for the next time point.

For the experiments in this paper, we consider two choices: Gaussian likelihood for real-valued data, and negative-binomial likelihood for positive count data. Other likelihood models can also be used readily, e.g. a beta likelihood for data in the unit interval, a Bernoulli likelihood for binary data, or mixtures in order to handle complex marginal distributions, as long as samples from the distribution can be obtained cheaply, and the log-likelihood and its gradients with respect to the parameters can be evaluated. We parametrize the Gaussian likelihood using

[2] We refer to $p(z|\theta)$ as a *likelihood* when we think of it as a function of $\theta$ for a fixed $z$.

its mean and standard deviation, $\theta = (\mu, \sigma)$, where the mean is given by an affine function of the network output, and the standard deviation is obtained by applying an affine transformation followed by a softplus activation in order to ensure $\sigma > 0$:

$$p_G(z|\mu,\sigma) = (2\pi\sigma^2)^{-\frac{1}{2}} \exp(-(z-\mu)^2/(2\sigma^2)) ,$$
$$\mu(\mathbf{h}_{i,t}) = \mathbf{w}_\mu^T \mathbf{h}_{i,t} + b_\mu ,$$
$$\sigma(\mathbf{h}_{i,t}) = \log(1 + \exp(\mathbf{w}_\sigma^T \mathbf{h}_{i,t} + b_\sigma)) .$$

The negative binomial distribution is a common choice for modeling time series of positive count data (Chapados, 2014; Snyder et al., 2012). We parameterize the negative binomial distribution by its mean $\mu \in \mathbb{R}^+$ and a shape parameter $\alpha \in \mathbb{R}^+$,

$$p_{NB}(z|\mu,\alpha) = \frac{\Gamma(z+\frac{1}{\alpha})}{\Gamma(z+1)\Gamma(\frac{1}{\alpha})} \left(\frac{1}{1+\alpha\mu}\right)^{\frac{1}{\alpha}} \left(\frac{\alpha\mu}{1+\alpha\mu}\right)^{z} ,$$
$$\mu(\mathbf{h}_{i,t}) = \log(1 + \exp(\mathbf{w}_\mu^T \mathbf{h}_{i,t} + b_\mu)) ,$$
$$\alpha(\mathbf{h}_{i,t}) = \log(1 + \exp(\mathbf{w}_\alpha^T \mathbf{h}_{i,t} + b_\alpha)) ,$$

where both parameters are obtained from the network output by a fully-connected layer with softplus activation so as to ensure positivity. In this parameterization of the negative binomial distribution, the shape parameter $\alpha$ scales the variance relative to the mean, i.e. $\mathrm{Var}[z] = \mu + \mu^2\alpha$. While other parameterizations are possible, preliminary experiments showed this particular one to be especially conducive to fast convergence.

### 4.2. Training

Given a dataset of time series $\{\mathbf{z}_{i,1:T}\}_{i=1,\dots,N}$ and associated covariates $\mathbf{x}_{i,1:T}$, obtained by choosing a time range such that $z_{i,t}$ in the prediction range is known, the parameters $\Theta$ of the model, consisting of the parameters of both the RNN $h(\cdot)$ and $\theta(\cdot)$, can be learned by maximizing the log-likelihood

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{t=t_0}^{T} \log p(z_{i,t}|\theta(\mathbf{h}_{i,t})) . \tag{3}$$

As $\mathbf{h}_{i,t}$ is a deterministic function of the input, all quantities required for computing Eq. (3) are observed, so that, in contrast to state space models with latent variables, no inference is required, and Eq. (3) can be optimized directly via stochastic gradient descent by computing gradients with respect to $\Theta$. In our experiments, where the encoder model is the same as the decoder, the distinction between the encoder and the decoder is somewhat artificial during training, so that we also include the likelihood terms for $t = 0, \dots, t_0 - 1$ in Eq. (3) (or, equivalently, set $t_0 = 0$).

For each time series in the dataset, we generate multiple training instances by selecting from the original time series windows with different starting points. In practice, we keep both the total length $T$ and the relative lengths of the conditioning and prediction ranges fixed for all training examples. For example, if the total available period for a given time series ranges from 2013-01-01 to 2017-01-01, we can create training examples where $t = 1$ corresponds to 2013-01-01, 2013-01-02, 2013-01-03, and so on. When choosing these windows, we ensure that the entire prediction range is always covered by the available ground truth data, but we may chose to have $t = 1$ lie *before* the start of the time series, e.g. 2012-12-01 in the example above, padding the unobserved target with zeros. This allows the model to learn the behavior of "new" time series by taking into account all other available covariates. Augmenting the data using this windowing procedure ensures that information about absolute time is available to the model only through covariates, not through the relative position of $z_{i,t}$ in the time series. Fig. 4 contains a depiction of this data augmentation technique.

Bengio, Vinyals, Jaitly, and Shazeer (2015) noted that the autoregressive nature of such models means that optimizing Eq. (3) directly causes a discrepancy between the ways in which the model is used during training and when obtaining predictions from the model: during training, the values of $z_{i,t}$ are known in the prediction range and can be used to compute $\mathbf{h}_{i,t}$; however, during prediction, $z_{i,t}$ is unknown for $t \geq t_0$, and a single sample $\tilde{z}_{i,t} \sim p(\cdot|\theta(\mathbf{h}_{i,t}))$ from the model distribution is used in the computation of $\mathbf{h}_{i,t}$ according to Eq. (2) instead. While this disconnect has been shown to pose a severe problem for NLP tasks, for example, we have not observed adverse effects from it in a forecasting setting. Preliminary experiments with variants of scheduled sampling (Bengio et al., 2015) did not show any noteworthy improvements in accuracy (but did slow convergence).

### 4.3. Scale handling

Applying the model to data that exhibit a power-law of scales, as depicted in Fig. 1, presents two challenges. Firstly, the autoregressive nature of the model means that both the autoregressive input $z_{i,t-1}$ and the output of the network (e.g. $\mu$) scale with the observations $z_{i,t}$ directly, but the non-linearities of the network in between have a limited operating range. Thus, without further modifications, the network has to learn, first, to scale the input to an appropriate range in the input layer, then to invert this scaling at the output. We address this issue by dividing the autoregressive inputs $z_{i,t}$ (or $\tilde{z}_{i,t}$) by an item-dependent scale factor $\nu_i$, and conversely multiplying the scale-dependent likelihood parameters by the same factor. For instance, for the negative binomial likelihood, we use $\mu = \nu_i \log(1+\exp(o_\mu))$ and $\alpha = \log(1+\exp(o_\alpha))/\sqrt{\nu_i}$, where $o_\mu$ and $o_\alpha$ are the outputs of the network for these parameters. Note that while one could alternatively scale the input in a preprocessing step for real-valued data, this is not possible for count distributions. The selection of an appropriate scale factor might be challenging in itself (especially in the presence of missing data or large within-item variances). However, scaling by the average value $\nu_i = 1 + \frac{1}{t_0}\sum_{t=1}^{t_0} z_{i,t}$, as we do in our experiments, is a heuristic that works well in practice.

Secondly, the imbalance in the data means that a stochastic optimization procedure that picks training instances uniformly at random will visit the small number time series with large scales very infrequently, resulting

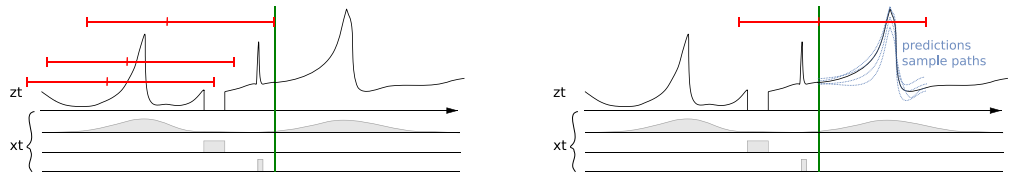


**Fig. 4.** Depiction of the data setup for training and forecasting for a single input time series $z$ with covariates $x$. The green vertical line separates the training data from the testing data, so we compute the out-of-sample accuracy for forecasts to the right of the green line; in particular, no data to the right of the green line is used in training. *Left.* The data setup during the training phase. The red lines marks the slices of $x$ that are presented to the model during training, where the left part marks the conditioning range and the right part the prediction range. Note that all windows are to the left of the green line. *Right.* During forecasting, when the model is fully trained, only the conditioning range is to the left of the green line. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

in those time series being underfitted. This could be especially problematic in the demand forecasting setting, where high-velocity items can exhibit qualitatively different behaviors from low-velocity items, and having accurate forecasts for high-velocity items might be more important for meeting certain business objectives. We counteract this effect by sampling the examples non-uniformly during training. In particular, our weighted sampling scheme sets the probability of selecting a window from an example with scale $\nu_i$ proportional to $\nu_i$. This sampling scheme is simple yet effective in compensating for the skew in Fig. 1.

### 4.4. Covariates

The covariates $\mathbf{x}_{i,t}$ can be item-dependent, time-dependent, or both. This distinction is mainly of practical importance. In theory, any covariate $\mathbf{x}_{i,t}$ that does not vary with time can be generalized trivially to be time-dependent by repeating it along the time dimension. Examples of such time-independent covariates include the categorization of item $i$, for example to denote membership to a certain group of products (e.g., product $i$ is a shoe, so $\mathbf{x}_{i,t} = s$, where $s$ is an identifier for the shoe category). Such covariates allow for the fact that time series with the same time-independent covariate may be similar. Examples of time-dependent covariates include information about the time point (e.g. week of year) of the model. They can also be used to include covariates that one expects to influence the outcome (e.g. price or promotion status in the demand forecasting setting), as long as the covariates' values are available in the prediction range as well. If these values are not available (e.g., future price changes), one option is to set them manually (e.g., assume that there are no price changes), which allows for what-if analysis. The solution based on principles is to predict these time series jointly, e.g., forecast demand and price jointly in a multivariate forecast. What-if analyses are possible when future prices become known via conditioning. We leave multivariate forecasting as a valuable direction for future work.

All of our experiments use an "age" covariate, i.e., the distance to the first observation in that time series. We also add day-of-the-week and hour-of-the-day for hourly data, week-of-the-year for weekly data and month-of-the-year for monthly data. We encode these simply as increasing numeric values, instead of using multi-dimensional binary variables to encode them. Furthermore, we include a single categorical item covariate, for which an embedding is learned by the model. In the retail demand forecasting datasets, the item covariate corresponds to a (coarse) product category (e.g. "clothing"), while in the smaller datasets it corresponds to the item's identity, allowing the model to learn item-specific behaviors. By appropriate normalization, we standardize all covariates to have a zero mean and unit variance.

## 5. Applications and experiments

We implement our model using MXNet (Chen et al., 2015), and use a single p2.xlarge AWS EC2 compute instance containing 4 CPUs and 1 GPU to run all experiments.[3] With this set-up, training and prediction on the large `ec` dataset containing 500K time series can be completed in less than 10 h. Note that prediction with a trained model is fast (in the order of tens of minutes for a single compute instance), and can be sped-up if necessary by executing prediction in parallel.

We use the ADAM optimizer (Kingma & Ba, 2014) with early stopping and standard LSTM cells with a forget bias set to 1.0 in all experiments, and 200 samples are drawn from our decoder for generating predictions.

### 5.1. Datasets

We use five datasets for our evaluations. The first three, namely `parts`, `electricity`, and `traffic`, are public datasets; `parts` consists of 1046 aligned time series of 50 time steps each, representing the monthly sales of different items by a US automobile company (Seeger et al., 2016); `electricity` contains hourly time series of the electricity consumptions of 370 customers (Yu et al., 2016); and `traffic`, also used by Yu et al. (2016), contains the hourly occupancy rates, between zero and one, of 963 car lanes of San Francisco bay area freeways. For the `parts` dataset, we use the 42 first months as training data and report the error on the remaining 8 months. For the other datasets, `electricity`, `traffic`, `ec-sub` and `ec`, the set of possible training instances is subsampled to the number indicated in Table 1. The results for `electricity` and `traffic` are computed using a rolling window of predictions, as described by Yu et al.

[3] Implementations of DeepAR are available on Amazon SageMaker (closed-source) and as part of GluonTS (Alexandrov et al., 2019).

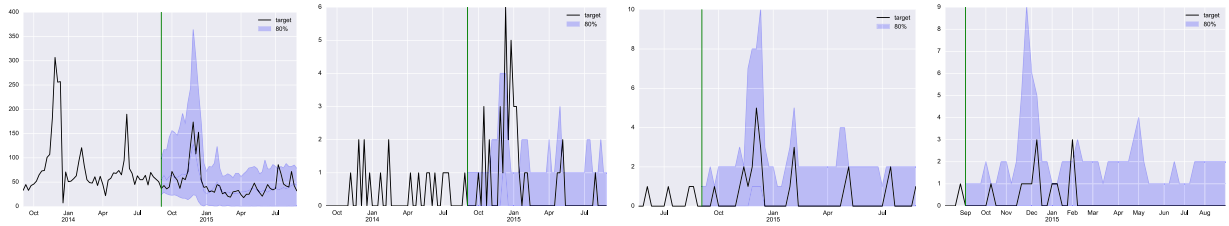

**Fig. 5.** Example time series of ec. The vertical line separates the conditioning period from the prediction period. The black line shows the true target. In the prediction range, we plot the p50 as a blue line (mostly zero for the three slow items), along with 80% confidence intervals (shaded). The model learns accurate seasonality patterns and uncertainty estimates for items of different velocities and ages. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

**Table 1**
Dataset statistics and RNN parameters.

| | parts | electricity | traffic | ec-sub | ec |
|---|---|---|---|---|---|
| # of time series | 1046 | 370 | 963 | 39700 | 534884 |
| time granularity | month | hourly | hourly | week | week |
| domain | $\mathbb{N}$ | $\mathbb{R}^+$ | [0, 1] | $\mathbb{N}$ | $\mathbb{N}$ |
| encoder length | 8 | 168 | 168 | 52 | 52 |
| decoder length | 8 | 24 | 24 | 52 | 52 |
| # of training examples | 35K | 500K | 500K | 2M | 2M |
| item input embedding dimension | 1046 | 370 | 963 | 5 | 5 |
| item output embedding dimension | 1 | 20 | 20 | 20 | 20 |
| batch size | 64 | 64 | 64 | 512 | 512 |
| learning rate | 1e−3 | 1e−3 | 1e−3 | 5e−3 | 5e−3 |
| # of LSTM layers | 3 | 3 | 3 | 3 | 3 |
| # of LSTM nodes | 40 | 40 | 40 | 120 | 120 |
| running time | 5 min | 7h | 3h | 3h | 10h |

(2016). We do not retrain our model for each window, but use a single model trained on the data before the first prediction window. The remaining two datasets, ec and ec-sub, are the weekly item sales from Amazon that were used by Seeger et al. (2016), and we predict the 52 weeks following 2014-09-07. The time series in these two datasets are very diverse and lumpy, ranging from very fast-moving to very slow-moving items, and include "new" products that were introduced in the weeks before the forecast time 2014-09-07; see Fig. 5. Further, the item velocities in these datasets have a power-law distribution, as shown in Fig. 1.

Table 1 also lists running times as measured by an end-to-end evaluation, e.g. processing covariates, training the neural network, drawing samples and evaluating the distributions produced.

For each dataset, a grid-search is used to find the best value for the hyper-parameters *item output embedding dimension* and *# of LSTM nodes* (e.g. hidden number of units). To do so, the data before the forecast start time are used as the training set and split into two partitions. For each hyper-parameter candidate, we fit our model on the first partition of the training set, containing 90% of the data, and pick the one that has the minimal negative log-likelihood on the remaining 10%. Once the best set of hyper-parameters has been found, the evaluation metrics (0.5-risk, 0.9-risk, ND and RMSE) are evaluated on the test set, that is, the data coming after the forecast start time. Note that this procedure could lead to the hyper-parameters being over-fitted to the training set, but this would also degrade the metric that we report. A better procedure would be to fit the parameters and evaluate the negative log-likelihood not only on different windows, but also on non-overlapping time intervals. We tune the learning rate manually for every dataset and keep it fixed in hyper-parameter tuning. Other parameters such as the encoder length, decoder length and item input embedding are considered to be domain-dependent, and are not fitted. The batch size is increased on larger datasets in order to benefit more from GPU's parallelization. Finally, the running time measures an end-to-end evaluation, e.g. processing covariates, training the neural network, drawing samples for the production of probabilistic forecasts, and evaluating the forecasts.

### 5.2. Accuracy comparison

For the parts and ec/ec-sub datasets, we provide comparisons with the following baselines, which represent the state-of-the-art on these datasets to the best of our knowledge:

- Croston: the Croston method developed for intermittent demand forecasting, from the R package of Hyndman and Khandakar (2008).
- ETS: the ETS model (Hyndman et al., 2008) from the R package with automatic model selection. Only additive models are used, as multiplicative models shows numerical issues on some time series.
- Snyder: the negative-binomial autoregressive method of Snyder et al. (2012).
- ISSM: the method of Seeger et al. (2016) using an innovative state space model with covariate features.

In addition, we compare our results to two baseline RNN models:

**Table 2**
Accuracy metrics relative to the strongest previously published method (baseline).

| | 0.5-risk | | | | 0.9-risk | | | | Average |
|---|---|---|---|---|---|---|---|---|---|
| | parts | | | | | | | | |
| $(L, S)$ | (0, 1) | (2, 1) | (0, 8) | all(8) | (0, 1) | (2, 1) | (0, 8) | all(8) | average |
| Snyder (baseline) | 1.00 | 1.00 | 1.00 | **1.00** | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| Croston | 1.47 | 1.70 | 2.86 | 1.83 | – | – | – | – | 1.97 |
| ISSM | 1.04 | 1.07 | 1.24 | 1.06 | 1.01 | 1.03 | 1.14 | 1.06 | 1.08 |
| ETS | 1.28 | 1.33 | 1.42 | 1.38 | 1.01 | 1.03 | 1.35 | 1.04 | 1.23 |
| rnn-gaussian | 1.17 | 1.49 | 1.15 | 1.56 | 1.02 | 0.98 | 1.12 | 1.04 | 1.19 |
| rnn-negbin | **0.95** | **0.91** | 0.95 | **1.00** | 1.10 | **0.95** | 1.06 | 0.99 | 0.99 |
| DeepAR | 0.98 | **0.91** | **0.91** | 1.01 | **0.90** | **0.95** | **0.96** | **0.94** | **0.94** |
| | ec-sub | | | | | | | | |
| $(L, S)$ | (0, 2) | (0, 8) | (3, 12) | all(33) | (0, 2) | (0, 8) | (3, 12) | all(33) | average |
| Snyder | 1.04 | 1.18 | 1.18 | 1.07 | 1.0 | 1.25 | 1.37 | 1.17 | 1.16 |
| Croston | 1.29 | 1.36 | 1.26 | 0.88 | – | – | – | – | 1.20 |
| ISSM (baseline) | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | **1.00** | 1.00 | 1.00 |
| ETS | 0.83 | 1.06 | 1.15 | 0.84 | 1.09 | 1.38 | 1.45 | 0.74 | 1.07 |
| rnn-gaussian | 1.03 | 1.19 | 1.24 | 0.85 | 0.91 | 1.74 | 2.09 | 0.67 | 1.21 |
| rnn-negbin | 0.90 | 0.98 | 1.11 | 0.85 | 1.23 | 1.67 | 1.83 | 0.78 | 1.17 |
| DeepAR | **0.64** | **0.74** | **0.93** | **0.73** | **0.71** | **0.81** | 1.03 | **0.57** | **0.77** |
| | ec | | | | | | | | |
| $(L, S)$ | (0, 2) | (0, 8) | (3, 12) | all(33) | (0, 2) | (0, 8) | (3, 12) | all(33) | average |
| Snyder | 0.87 | 1.06 | 1.16 | 1.12 | 0.94 | 1.09 | 1.13 | 1.01 | 1.05 |
| Croston | 1.30 | 1.38 | 1.28 | 1.39 | – | – | – | – | 1.34 |
| ISSM (baseline) | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | **1.00** | 1.00 | 1.00 |
| ETS | 0.77 | 0.97 | 1.07 | 1.23 | 1.05 | 1.33 | 1.37 | 1.11 | 1.11 |
| rnn-gaussian | 0.89 | 0.91 | 0.94 | 1.14 | 0.90 | 1.15 | 1.23 | **0.90** | 1.01 |
| rnn-negbin | 0.66 | 0.71 | **0.86** | **0.92** | 0.85 | 1.12 | 1.33 | 0.98 | 0.93 |
| DeepAR | **0.59** | **0.68** | 0.99 | 0.98 | **0.76** | **0.88** | **1.00** | 0.91 | **0.85** |

Note: The best results are marked in bold (lower is better).

- `rnn-gaussian` uses the same architecture as `DeepAR` with a Gaussian likelihood; however, it uses uniform sampling and a simpler scaling mechanism, where the time series $z_i$ are divided by $v_i$ and the outputs are multiplied by $v_i$.
- `rnn-negbin` uses a negative binomial distribution, but does not scale the inputs and outputs of the RNN, and the training instances are drawn uniformly rather than using weighted sampling.

We define the error metrics used in our comparisons formally below. The metrics are evaluated for certain *spans* $[L, L + S)$ in the prediction range, where $L$ is a *lead time* after the forecast start point.

#### 5.2.1. $\rho$-risk metric

Following Seeger et al. (2016), we use $\rho$-risk metrics (quantile loss) that quantify the accuracy of a quantile $\rho$ of the predictive distribution.

The aggregated target value of an item $i$ in a span is denoted by $Z_i(L, S) = \sum_{t=t_0+L}^{t_0+L+S} z_{i,t}$. For a given quantile $\rho \in (0, 1)$ we denote the predicted $\rho$-quantile for $Z_i(L, S)$ by $\hat{Z}_i^{\rho}(L, S)$. We obtain such a quantile prediction from a set of sample paths by first summing each realization in the given span. The samples of these sums then represent the estimated distribution for $Z_i(L, S)$, and we can take the $\rho$-quantile from the empirical distribution.

The $\rho$-quantile loss is then defined as

$$L_\rho(Z, \hat{Z}^\rho) = 2(\hat{Z} - Z)\left(\rho \mathrm{I}_{\hat{Z}^\rho > Z} - (1 - \rho)\mathrm{I}_{\hat{Z}^\rho \leq Z}\right).$$

We summarize the quantile losses for a given span across all items by considering a normalized sum of quantile losses $\left(\sum_i L_\rho(Z_i, \hat{Z}_i^\rho)\right) / \left(\sum_i Z_i\right)$, which we call the $\rho$-risk.

#### 5.2.2. ND and RMSE metrics

The ND and RMSE metrics are defined as

$$\mathrm{ND} = \frac{\sum_{i,t} |z_{i,t} - \hat{z}_{i,t}|}{\sum_{i,t} |z_{i,t}|} \quad \text{and}$$

$$\mathrm{RMSE} = \frac{\sqrt{\frac{1}{N(T-t_0)} \sum_{i,t} (z_{i,t} - \hat{z}_{i,t})^2}}{\frac{1}{N(T-t_0)} \sum_{i,t} |z_{i,t}|},$$

where $\hat{z}_{i,t}$ is the predicted median value for item $i$ at time $t$ and the sums are over all items and all time points in the prediction period.

#### 5.2.3. Results

Table 2 shows the 0.5-risk and 0.9-risk for different lead times and spans. Here, all($K$) denotes the average risk of the marginals $[L, L + 1)$ for $L < K$. We normalize all reported metrics with respect to the strongest previously-published method (baseline). `DeepAR` outperforms all other methods on these datasets. The results also show the importance of modeling these datasets using a count distribution, as `rnn-gaussian` leads to worse accuracies. The `ec` and `ec-sub` datasets exhibit the power-law behavior discussed above, and overall forecast accuracy is affected negatively by the absence of scaling and weighted sampling (`rnn-negbin`). On the `parts` dataset, which does not exhibit the power-law behavior, the performance of `rnn-negbin` is similar to that of `DeepAR`.

Table 3 compares the point forecast accuracies on the `electricity` and `traffic` datasets against that of the matrix factorization technique (`MatFact`) proposed by Yu et al. (2016). We consider the same metrics,

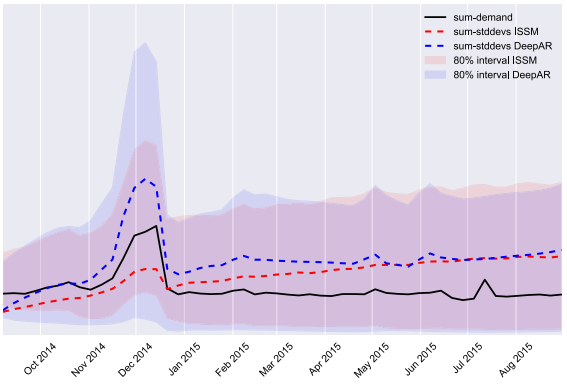


**Fig. 6.** Uncertainty growth over time for the ISSM and DeepAR models. Unlike the ISSM, which postulates a linear growth of uncertainty, the behavior of the uncertainty is learned from the data, resulting in a non-linear growth with a (plausibly) higher uncertainty around Q4. The aggregate is calculated over the entire `ec` dataset.

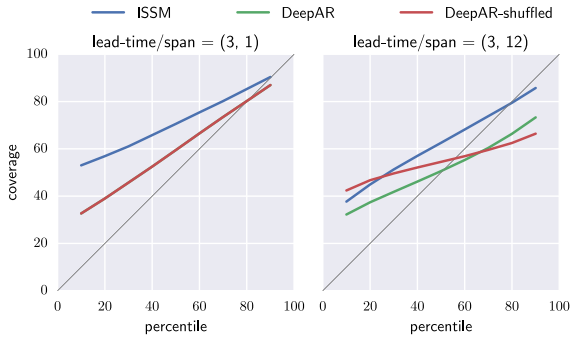


**Fig. 7.** Coverages for two spans of the `ec-sub` dataset. The left panel shows the coverage for a single time-step interval, while the right panel shows these metrics for a larger time interval with nine time-steps. When the samples for each time step are shuffled, the correlation in the prediction sample paths is destroyed and the forecast becomes less calibrated. This shuffled prediction also has a 10% higher 0.9-risk.

**Table 3**
Comparison with `MatFact`.

| | `electricity` | | `traffic` | |
|---|---|---|---|---|
| | ND | RMSE | ND | RMSE |
| `MatFact` | 0.16 | 1.15 | 0.20 | 0.43 |
| `DeepAR` | **0.07** | **1.00** | **0.17** | **0.42** |

namely the normalized deviation (ND) and normalized RMSE (NRMSE). The results show that DeepAR outperforms `MatFact` on both datasets.

### 5.3. Qualitative analysis

Fig. 5 shows example predictions from the `ec` dataset. Fig. 6 shows aggregate sums of different quantiles of the marginal predictive distribution for `DeepAR` and `ISSM` on the `ec` dataset. In contrast to ISSM models such as that of Seeger et al. (2016), where a linear growth of uncertainty is part of the modeling assumptions, the uncertainty growth pattern is learned from the data. In this case, the model does learn an overall growth of uncertainty over time. However, this is not simply linear growth: uncertainty (correctly) increases during Q4, and decreases again shortly afterwards.

The calibration of the forecast distribution is depicted in Fig. 7. Here, we show the Coverage($p$) for each percentile $p$, which is defined as the fraction of time series in the dataset for which the $p$-percentile of the predictive distribution is larger than the true target. For a perfectly calibrated prediction, it holds that Coverage($p$) $= p$, which corresponds to the diagonal. Overall, the calibration is improved compared to the `ISSM` model.

We assess the effect of modeling correlations in the output, i.e., how much they differ from independent distributions for each time-point, by plotting the calibration curves for a shuffled forecast, where the realizations of the original forecast have been shuffled for each time point, destroying any correlation between time steps. For the short lead-time span (left), which consists of just one time-point, this has no impact, because it is just the marginal distribution. However, for the longer lead-time span (right), destroying the correlation leads to a worse calibration, showing that important temporal correlations are captured between the time steps.

## 6. Conclusion

We have shown that forecasting approaches based on modern deep learning techniques can improve the forecast accuracy drastically relative to state-of-the-art forecasting methods on a wide variety of datasets. Our proposed DeepAR model is effective at learning a *global* model from related time series, can handle widely-varying scales through rescaling and velocity-based sampling, generates calibrated probabilistic forecasts with high accuracy, and is able to learn complex patterns such as seasonality and uncertainty growth over time from the data.

Interestingly, the method works on a wide variety of datasets with little or no hyperparameter tuning, and is applicable to medium-sized datasets that contain only a few hundred time series.

## References

Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., et al. (2019). *GluonTS: probabilistic time series models in Python*. ICML time series workshop, abs/1906.05264 arXiv:1906.05264.

Bandara, K., Bergmeir, C., & Smyl, S. (2017). Forecasting across time series databases using long short-term memory networks on groups of similar series. arXiv preprint arXiv:1710.03222 8 (pp. 805–815).

Ben Taieb, S., Taylor, J. W., & Hyndman, R. J. (2017). Coherent probabilistic forecasts for hierarchical time series. In *Proceedings of the 34th international conference on machine learning*.

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in neural information processing systems* (pp. 1171–1179).

Box, G. E. P., & Cox, D. R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society. Series B. Statistical Methodology*, *26*(2), 211–252.

Box, G. E., Jenkins, G. M., Reinsel, G. C., & Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Chapados, N. (2014). Effective Bayesian modeling of groups of related count time series. In *Proceedings of the 31st international conference on machine learning* (pp. 1395–1403).

Chen, H., & Boylan, J. E. (2008). Empirical evidence on individual, group and shrinkage seasonal indices. *International Journal of Forecasting*, *24*(3), 525–534.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., et al. (2015). MXNet: A Flexible and efficient machine learning library for heterogeneous distributed systems. arXiv preprint arXiv:1512.01274.

Croston, J. (1972). Forecasting and stock control for intermittent demands. *Operational Research Quarterly*, *23*, 289–304.

Davydenko, A., & Fildes, R. (2013). Measuring forecasting accuracy: the case of judgmental adjustments to SKU-level demand forecasts. *International Journal of Forecasting*, *29*(3), 510–522.

Díaz-Robles, L. A., Ortega, J. C., Fu, J. S., Reed, G. D., Chow, J. C., Watson, J. G., et al. (2008). A hybrid ARIMA and artificial neural networks model to forecast particulate matter in urban areas: the case of Temuco, Chile. *Atmospheric Enviroment*, *42*(35), 8331–8340.

Durbin, J., & Koopman, S. J. (2012). *Time series analysis by state space methods: Vol. 38*. OUP Oxford.

Faloutsos, C., Flunkert, V., Gasthaus, J., Januschowski, T., & Wang, Y. (2019). Forecasting big time series: theory and practice. In *KDD '19, Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 3209–3210). New York, NY, USA: ACM.

Fildes, R., Nikolopoulos, K., Crone, S., & Syntetos, A. (2008). Forecasting and operational research: a review. *The Journal of the Operational Research Society*, *59*(9), 1150–1172.

Gasthaus, J., Benidis, K., Wang, Y., Rangapuram, S. S., Salinas, D., Flunkert, V., et al. (2019). Probabilistic forecasting with spline quantile function RNNs. In *The 22nd international conference on artificial intelligence and statistics* (pp. 1901–1910).

Gers, F. A., Eck, D., & Schmidhuber, J. (2001). Applying LSTM to time series predictable through time-window approaches. In G. Dorffner (Ed.), *Artificial neural networks – ICANN 2001 (Proceedings)* (pp. 669–676). Springer.

Ghiassi, M., Saidane, H., & Zimbra, D. (2005). A dynamic artificial neural network model for forecasting time series events. *International Journal of Forecasting*, *21*(2), 341–362.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning: Adaptive computation and machine learning*. MIT Press.

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., & Wierstra, D. (2015). DRAW: a recurrent neural network for image generation. arXiv preprint arXiv:1502.04623.

Gutierrez, R. S., Solis, A. O., & Mukhopadhyay, S. (2008). Lumpy demand forecasting using neural networks. *International Journal of Production Economics*, *111*(2), 409–420.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Compututation*, *9*(8), 1735–1780.

Hyndman, R. J., Ahmed, R. A., Athanasopoulos, G., & Shang, H. L. (2011). Optimal combination forecasts for hierarchical time series. *Computational Statistics & Data Analysis*, *55*(9), 2579–2589.

Hyndman, R. J., & Athanasopoulos, G. (2012). *Forecasting: principles and practice*. OTexts.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Hyndman, R. J., & Khandakar, Y. (2008). Automatic time series forecasting: the forecast package for R. *Journal of Statistical Software*, *26*(3), 1–22.

Hyndman, R., Koehler, A. B., Ord, J. K., & Snyder, R. D. (2008). *Springer series in statistics, Forecasting with exponential smoothing: the state space approach*. Springer.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd international conference on machine learning* (pp. 448–456).

Kaastra, I., & Boyd, M. (1996). Designing a neural network for forecasting financial and economic time series. *Neurocomputing*, *10*(3), 215–236.

Kingma, D. P., & Ba, J. (2014). Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kourentzes, N. (2013). Intermittent demand forecasts with neural networks. *International Journal of Production Economics*, *143*(1), 198–206.

Laptev, N., Yosinsk, J., Li Erran, L., & Smyl, S. (2017). *Time-series extreme event forecasting with neural networks at Uber*. ICML time series workshop.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). The M4 Competition: results, findings, conclusion and way forward. *International Journal of Forecasting*, *34*(4), 802–808.

Mohammadipour, M., Boylan, J., & Syntetos, A. (2012). The application of product-group seasonal indexes to individual products. *Foresight: The International Journal of Applied Forecasting*, *26*, 20–26.

Oliveira, M. R., & Torgo, L. (2014). Ensembles for time series forecasting. *Journal of Machine Learning Research (JMLR)*.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., et al. (2016). Wavenet: A generative model for raw audio. arXiv preprint arXiv:1609.03499.

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: neural basis expansion analysis for interpretable time series forecasting. arXiv preprint arXiv:1905.10437.

Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y., & Januschowski, T. (2018). Deep state space models for time series forecasting. In *Advances in neural information processing systems* (pp. 7785–7794).

Seeger, M. W., Salinas, D., & Flunkert, V. (2016). Bayesian intermittent demand forecasting for large inventories. In *Advances in neural information processing systems* (pp. 4646–4654).

Smyl, S., Ranganathan, J., & Pasqua, A. (2018). M4 Forecasting competition: introducing a new hybrid ES-RNN model. https://eng.uber.com/m4-forecasting-competition.

Snyder, R. D., Ord, J., & Beaumont, A. (2012). Forecasting the intermittent demand for slow-moving inventories: a modelling approach. *International Journal of Forecasting*, *28*(2), 485–496.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Timmermann, A. (2006). Forecast combinations. In G. Elliott, C. Granger, & A. Timmermann (Eds.), *Handbook of economic forecasting: Vol. 1* (1st ed.). (pp. 135–196). Elsevier.

Toubeau, J.-F., Bottieau, J., Vallée, F., & De Grève, Z. (2018). Deep learning-based multivariate probabilistic forecasting for short-term scheduling in power markets. *IEEE Transactions on Power Systems*, *34*(2), 1203–1215.

Trapero, J. R., Kourentzes, N., & Fildes, R. (2015). On the identification of sales forecasting models in the presence of promotions. *The Journal of the Operational Research Society*, *66*(2), 299–307.

Wen, R. W., Torkkola, K., & Narayanaswamy, B. (2017). *A multi-horizon quantile recurrent forecaster*. NIPS time series workshop.

Yu, H.-F., Rao, N., & Dhillon, I. S. (2016). Temporal regularized matrix factorization for high-dimensional time series prediction. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems 29* (pp. 847–855). Curran Associates, Inc..

Zhang, G., Eddy Patuwo, B., & Hu, Y. M. (1998). Forecasting with artificial neural networks:: the state of the art. *International Journal of Forecasting*, *14*(1), 35–62.