DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks

Valentin Flunkert [*1] **David Salinas** [*1] **Jan Gasthaus** [1]

Abstract

A key enabler for optimizing business processes is accurately estimating the probability distribution of a time series future given its past. Such probabilistic forecasts are crucial for example for reducing excess inventory in supply chains. In this paper we propose DeepAR, a novel methodology for producing accurate probabilistic forecasts, based on training an auto-regressive recurrent network model on a large number of related time series. We show through extensive empirical evaluation on several real-world forecasting data sets that our methodology is more accurate than state-of-the-art models, while requiring minimal feature engineering.

1. Introduction

Forecasting is a central problem in many businesses. Demand forecasting, in particular, plays a key role in the operational, tactical and strategical decisions of most businesses. For example, probabilistic forecasts are necessary for inventory management, staff scheduling and topology planning (Larson et al., 2001). More generally speaking, they are a crucial technology for most aspects of supply chain optimization.

Traditionally, most forecasting methods have been developed in the setting of forecasting individual time series. In this approach, one independently estimates model parameters from past observations for each given time series. The model typically accounts for different factors such as trend and seasonality (Hyndman et al., 2008). The fitted model then forecasts the time series into the future according to the model dynamics, possibly accounting for uncertainty e.g. by generating many realizations.

In recent years, a new type of forecasting problem has become increasingly important in many applications. Instead of needing to predict individual or few time series, one is faced with forecasting thousands or millions of related time series. Examples include forecasting the energy consumption of individual households, forecasting the load for servers in a data center, or forecasting the demand for items that a large retailer offers. In all these scenarios, a substantial amount of data on past behavior of similar time series can be leveraged for making a forecast for an individual time series.

In this work we present DeepAR, a forecasting method based on autoregressive recurrent networks. DeepAR learns from historical of *all time series* in the data set jointly. We show in our experiments that DeepAR outperforms state of the art forecasting methods on a several real-world forecasting problems. It is noteworthy that DeepAR performs well on both intermediate and large datasets.

In addition to providing better forecast accuracy than previous methods, our approach has a number key advantages: (i) As the model learns seasonal behavior and dependencies on given covariates across time series, minimal manual feature engineering is needed to capture complex, group-dependent behavior; (ii) 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; (iii) Our method is able to provide forecasts for items with little or no history at all, a case where traditional single-item forecasting methods fail; (iv) Our method 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. Accurate, calibrated forecast distributions are learned from the historical behavior of all of the time series jointly. Such a probabilistic forecast is of crucial importance in many applications, as it—in contrast to a point forecast—enables optimal decision making under uncertainty by minimizing risk functions, i.e. expectations of some loss function under the forecast distribution.

In order to make our method effective for real-world forecasting problems, we address the problem of jointly learning from time series with values that span a very wide range, or where the amplitude distribution in the dataset exhibits a strong skew. This problem is illustrated in Fig. 1.

[*]Equal contribution [1]Amazon Development Center Germany. Correspondence to: <{dsalina,flunkert, gasthaus}@amazon.com>.

Under review for the 34^{th} International Conference on Machine Learning, Sydney, Australia, 2017.

The left panel shows the distribution of demand velocity (i.e. average weekly sales of an item) across millions of items sold by Amazon. The distribution is over a few orders of magnitude an approximate power-law. This observation is to the best of our knowledge new (although maybe not surprising) and has fundamental implications for forecasting methods that deal with such datasets. The scale-free nature of the distribution makes it difficult to consider subgroups of time series with a certain velocity band, as each such velocity sub-group would have a similar skew. Further, group-based regularization schemes such as the one proposed by Chapados (2014) may fail, as the velocities will be vastly different within each group. Finally, such skewed distributions make the use of certain commonly employed deep-learning methodologies, such as batch normalization (Ioffe & Szegedy, 2015), less effective. The right panel of Fig. 1 shows a histogram of the electricity consumption per user in log scale in the `electricity` dataset (see Sec. 4.1 for details). This plot indicates that the electricity consumption approximately follows a log-normal distribution, i.e. it is also a heavy-tailed distribution. In both these cases, the heavy tail means that the network has to be able to learn from examples in the tail and generalize to more extreme cases. For instance, we want to ensure that the model is able to predict a time series with values that are larger than all examples that it saw during training.

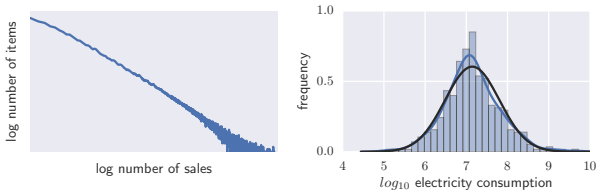

Figure 1. (Left) Log-log histogram of the number of items versus number of sales for the 500K time series of `ec`, showing the scale-free nature (approximately straight line) present in the `ec` dataset (axis labels omitted due to the non-public nature of the data). (Right) histogram of users' electricity consumption (in log scale) for the `electricity` dataset.

2. Related Work

Due to the immense practical importance of forecasting, a vast variety of different forecasting methods have been developed. Prominent examples of methods for forecasting individual time series include ARIMA models (Box & Jenkins, 1968) and exponential smoothing methods; Hyndman et al. (2008) provide a unifying review of these and related techniques.

In the demand forecasting domain, one is often faced with highly erratic, intermittent or bursty data which violate core assumptions of many classical techniques such as Gaussian errors, stationarity or homoscedasticity of the time series. Since data preprocessing methods (e.g. (Box & Cox, 1964)) often do not alleviate these conditions, forecasting methods have also incorporated more suitable likelihood functions, such as the zero-inflated Poisson distribution, the negative binomial distribution (Snyder et al., 2012), a combination of both (Chapados, 2014), or a tailored multi-stage likelihood (Seeger et al., 2016).

Sharing information across time series constitutes another challenge: information sharing should improve the forecast accuracy, but it is difficult to accomplish in practice, because of the often heterogeneous nature of the data. Matrix factorization methods (e.g. the recent work of Yu et al. (2016)), as well as Bayesian methods that share information via hierarchical priors (Chapados, 2014) have been proposed as mechanisms for learning across multiple related time series and leveraging hierarchical structure in time series (Hyndman et al., 2011). In contrast to existing work, our approach allows full flexibility in the use of likelihoods. Any of the afore-mentioned likelihoods can for instance be used. Information sharing is accomplished by a neural network that is trained on all time-series.

Neural networks have been investigated in the context of forecasting for a long time (see e.g. the numerous references in the survey (Zhang et al., 1998), or (Gers et al., 2001) for more recent work considering LSTM cells). More recently, Kourentzes (2013) applied neural networks specifically to intermittent data but obtained mixed results. Neural networks in forecasting have been typically applied to individual time series, i.e. a different model is fitted to each time series independently (Kaastra & Boyd, 1996; Ghiassi et al., 2005; Díaz-Robles et al., 2008). On the other hand, outside of the forecasting community, time series models based on recurrent neural networks have been very successfully applied to other applications, such as natural language processing (Graves, 2013; Sutskever et al., 2014), audio modeling (van den Oord et al., 2016) or image generation (Gregor et al., 2015). There are two main difference to the forecasting setting that we consider here. First, in the other applications the data that is predicted at each step is either discrete and finite, such as the set of characters or words, or lives in a bounded domain that can be bucketized (van den Oord et al., 2016) and thus a softmax likelihood is used. Second, the goal is to generate one or few samples of the most likely realization e.g. the best translations from one language to another or a realistic image or sound, while in forecasting the task is to estimate a full distribution and one thus needs to draw many samples.

Our work draws on the aforementioned recurrent neural network approaches and adds crucial extensions that enable state-of-the-art accuracy performance in the forecasting domain.

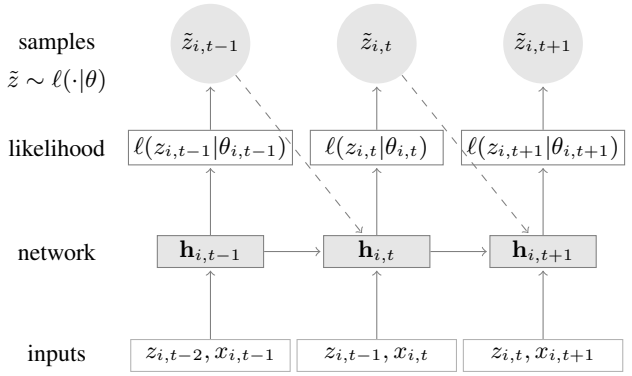

Figure 2. Summary of the model: At each time step t, the inputs to the network are the covariates $x_{i,t}$, the target value at the previous time step $z_{i,t-1}$, as well as the previous network output $\mathbf{h}_{i,t-1}$. The network output $\mathbf{h}_{i,t} = h(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}, \Theta)$ is then used to compute the parameters $\theta_{i,t} = \theta(\mathbf{h}_{i,t}, \Theta)$ of the likelihood $\ell(z|\theta)$, which is used for training the model parameters. When $z_{i,t}$ is not known (e.g. during prediction), a sample $\hat{z}_{i,t} \sim \ell(\cdot|\theta_{i,t})$ is fed back to the next step instead of the true value.

3. Model

Denoting the value of time series i at time t by $z_{i,t}$, our goal is to model the conditional distribution of the future of each time series $[z_{i,t_0}, z_{i,t_0+1}, \dots, z_{i,T}] := \mathbf{z}_{i,t_0:T}$ given its past $[z_{i,1}, \dots, z_{i,t_0-2}, z_{i,t_0-1}] := \mathbf{z}_{i,1:t_0-1}$,

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

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 prevent confusion we avoid the ambiguous terms "past" and "future" and will refer to time ranges $[1, t_0 - 1]$ and $[t_0, T]$ as the conditioning range and prediction range, respectively.[1]

Our model, summarized in Fig. 2, is based on an autoregressive recurrent network architecture, similar to the sequence-to-sequence models proposed in (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

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

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

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

where h is a function implemented by a multi-layer recurrent neural network whose exact architecture will be described in the experiments section. 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 $\ell(z_{i,t}|\theta(\mathbf{h}_{i,t}))$ is a fixed distribution whose parameters 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 for using the same architecture for the model in 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 (1) for $t = 1, \dots, t_0 - 1$, where all required quantities are observed.[2]

Given the model parameters Θ, we can directly obtain joint samples $\tilde{\mathbf{z}}_{i,t_0:T} \sim Q_\Theta(\mathbf{z}_{i,t_0:T}|\mathbf{z}_{i,1:t_0-1}, \mathbf{x}_{i,1:T})$ through ancestral sampling: First, we obtain $\mathbf{h}_{i,t_0-1}$ by computing (1) for $t = 1, \dots, t_0$. For $t = t_0, t_0 + 1, \dots, T$ we sample $\tilde{z}_{i,t} \sim \ell(\cdot|\theta(\tilde{\mathbf{h}}_{i,t}, \Theta))$ where $\tilde{\mathbf{h}}_{i,t} = h\left(\mathbf{h}_{i,t-1}, \tilde{z}_{i,t-1}, \mathbf{x}_{i,t}, \Theta\right)$ 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 time range in the future.

3.1. Likelihood model

The likelihood $\ell(z|\theta)$ determines the "noise model", and should be chosen to match the statistical properties of the data. We consider two choices, Gaussian likelihood for real-valued data, and negative-binomial likelihood for positive count data, though the model can easily be used with other likelihoods, e.g. Beta likelihood for data in the unit interval or Bernoulli likelihood for binary data, as long as samples from the distribution can cheaply be obtained, and the log-likelihood and its gradients wrt. the parameters can be evaluated. We parametrize the Gaussian likelihood using 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

[1] 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 time period for each i.

[2] The initial state of the encoder $\mathbf{h}_{i,0}$ as well as $z_{i,0}$ are initialized to zero.

transformation followed by a softplus activation in order to ensure $\sigma > 0$:

$$\begin{aligned}\ell_{\mathrm{G}}(z|\mu,\sigma) &= (2\pi\sigma^2)^{-\frac{1}{2}}\exp(-(z-\mu)^2/(2\sigma^2))\\ \mu(\mathbf{h}_{i,t}) &= \mathbf{w}_\mu^T\mathbf{h}_{i,t}+b_\mu\\ \sigma(\mathbf{h}_{i,t}) &= \log(1+\exp(\mathbf{w}_\sigma^T\mathbf{h}_{i,t}+b_\sigma)).\end{aligned}$$

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

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

where both parameters are obtained from the network output by a fully-connected layer with softplus activation to ensure positivity. In this parameterization of the negative binomial distribution the shape parameter α scales the variance relative to the mean, i.e. $\mathrm{Var}[z] = \mu + \mu^2\alpha$.

3.2. Training

Given a data set 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 Θ of the model, consisting of the parameters of the RNN $h(\cdot)$ as well as the parameters of $\theta(\cdot)$, can be learned by maximizing the (log-)likelihood

$$\mathcal{L} = \sum_{i=1}^{N}\sum_{t=t_0}^{T}\log\ell(z_{i,t}|\theta(\mathbf{h}_{i,t}))\,. \tag{2}$$

As $\mathbf{h}_{i,t}$ is a deterministic function of the input, all quantities required to compute (2) are observed, so that—in contrast to e.g. state space models with latent variables—no inference is required, and (2) can be optimized directly via stochastic gradient descent by computing gradients with respect to Θ. In our experiments, where the encoder model is the same as the decoder, the distinction between encoder and decoder is somewhat artificial during training, so that we also include the likelihood terms for $t = 0, \dots, t_0 - 1$ in (2) (or, equivalently, set $t_0 = 0$).

Bengio et al. (2015) noted that, due to the autoregressive nature of such models, optimizing (2) directly causes a discrepancy between how the model is used during training and when obtaining samples from the model: during training the values of $z_{i,t}$ are known in the prediction range and can be used to compute $\mathbf{h}_{i,t}$; during prediction however, $z_{i,t}$ is unknown for $t \geq t_0$, and a single sample $\tilde{z}_{i,t} \sim \ell(\cdot|\theta(\mathbf{h}_{i,t}))$ from the model distribution is used in the computation of $\mathbf{h}_{i,t}$ according to (1) instead. However, contrary to what has been observed in other sequence-to-sequence models, we did not notice any adverse effects, such as diverging sample paths, due to this discrepancy. We experimented with variants of scheduled sampling (Bengio et al., 2015), i.e. occasionally feeding back samples from the model instead of the true value during training, but did not observe any improvements.

3.3. Data augmentation

For each time series in the dataset, we generate multiple training instances by selecting windows with different starting points from the original time series. In practice, we keep the total length T as well as the relative length of the conditioning and prediction ranges fixed for all training examples. For example, if the total available range for a given time series ranges from 2013-01-01 to 2017-01-01, we may create training examples with $t = 1$ corresponding to 2013-01-01, 2013-01-02, 2013-01-03, and so on. When choosing these windows we ensure that entire prediction range is always covered by the available ground truth data, but we may chose $t = 1$ to lie *before* the start of the time series, e.g. 2012-12-01 in the example above, padding the unobserved part with zeros. By augmenting the data in this way we ensure that information about absolute time is only available to the model through covariates, but not through the relative position of $z_{i,t}$ in the time series.

3.4. Scale handling

Applying the model to data that exhibits a power-law of scales as depicted in Fig. 1 presents two challenges: Firstly, due to the autoregressive nature of the model, both the autoregressive input $z_{i,t-1}$ as well as the output of the network (e.g. μ) directly scale with the observations $z_{i,t}$, but the non-linearities of the network in between have a limited operating range. Without further modifications, the network thus has to learn to scale the input to an appropriate range in the input layer, and 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 ν_i, and conversely multiplying the scale-dependent likelihood parameters by the same factor. Choosing an appropriate scale factor might in itself be challenging (especially in the presence of missing data or large within-item variances). However, scaling by the average value $\nu_i = \frac{1}{t_0}\sum_{t=1}^{t_0} z_{i,t}$ (as we do in our experiments) is a heuristic that seems to work well in practice.

Secondly, due to the imbalance in the data, a stochastic optimization procedure that picks training instances uniformly at random will visit the small number time series with a large scale very infrequently, which can result in un-

derfitting on those time series unless the model is trained for a large number of epochs. This could be especially problematic in the demand forecasting setting, where high-velocity items can exhibit qualitatively different behavior than low-velocity items, and having an accurate forecast for high-velocity items might be more important for meeting certain business objectives. One strategy to counteract this effect is to sample the examples non-uniformly during training, e.g. with a weight proportional to the inverse of the power-law shown in Fig. 1. While we observed some benefits from applying such weighted sampling schemes, they are not essential for achieving good performance without excessive amounts of training even on very skewed data sets, which is why the results we report in the experiments section do not make use of this.

3.5. Missing Observations

In some forecasting settings, the target values $z_{i,t}$ might be missing (or unobserved) for a subset of the time points. For instance, in the context of demand forecasting, an item may be out-of-stock at a certain time, in which case the demand for the item cannot be observed. Not explicitly modeling such missing observations (e.g. by assuming that the observed sales correspond to the demand even when an item is out of stock), can, in the best case, lead to systematic forecast underbias, and, in a worst case in the larger supply chain context, can lead to a disastrous downward spiral where an out-of-stock situation leads to a lower demand forecast, lower re-ordering and more out-of-stock-situations. In our model, missing observations can easily be handled in a principled way by replacing each unobserved value $z_{i,t}$ by a sample $\tilde{z}_{i,t} \sim \ell(\cdot|\theta(\mathbf{h}_{i,t}))$ from the conditional predictive distribution when computing (1), and excluding the likelihood term corresponding to the missing observation from (2).[3]

3.6. Features

The model can make use of covariates (features) $\mathbf{x}_{i,t}$ that can depend be item-dependent, time-dependent, or both.[4] Such features can be used to provide additional information about the item or the time point to the model. For example, in single-item forecasting methods, time-dependent features are often used for modeling (non-periodic) seasonality, by using time period indicator features (e.g. "day of the week") and kernels centered around (moving) holidays, such as "distance to Thanksgiving" (Seeger et al., 2016). 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 features' values are available also in the prediction range.

In our experiments we include the following time-dependent features. The first feature called "age" (the distance to the first observation in that time series) is used in all experiments. We also add day-of-the-week and hour-of-the-day for hourly data, week-of-year for weekly data and month-of-year for monthly data. Further, we include a single categorical item feature, for which an embedding is learned by the model. In the retail demand forecasting data set we consider, the item feature corresponds to a (coarse) product category (e.g. "clothing"), while in the smaller data sets it corresponds to the item's identity, allowing the model to learn item-specific behavior.

While including additional features such as distances to moving holidays or covariates such as price improves the forecast accuracy, we choose not to do so in the experiments, as the fact that the model outperforms state-of-the art methods without them is a major advantage of our approach: linear per-item methods such as the one presented (Seeger et al., 2016) require laborious manual feature engineering in order to model group-dependent seasonal patterns, whereas in our approach they are simply learned from the data.

4. Applications and Experiments

4.1. Datasets

We use six datasets for our evaluations. The first three: `parts`, `electricity` and `traffic` are public datasets. `parts` consists of 1046 aligned time series of 50 time steps each representing monthly sales for different items of a US automobile company (Seeger et al., 2016). `electricity` contains hourly time series of the electricity use of 370 customers which was used in (Yu et al., 2016). The scales of time series vary widely in this dataset as shown in Fig. 1 (right). `traffic` was used in the same publication and contains the hourly occupancy rate, between 0 and 1, 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 error on the remaining 8. The results for `electricity` and `traffic` are computed using a rolling window of predictions as described in (Yu et al., 2016).We do not retrain our model for each window, but use a single model trained on the data before the first prediction window for making all predictions.

The remaining two datasets `ec` and `ec-sub` are item sales time series from the Amazon US marketplace and were

[3] We omitted experimental results in this setting from the paper, as doing a proper evaluation in the light of missing data in the prediction range requires non-standard adjusted metrics that are hard to compare across studies (see e.g. (Seeger et al., 2016)).

[4] In the model description we only explicitly cover covariates $\mathbf{x}_{i,t}$ that depend on both the item and time. Purely item-dependent or time-dependent features can be handled by repeating them across the missing dimension, thus turning them into item-time dependent features.

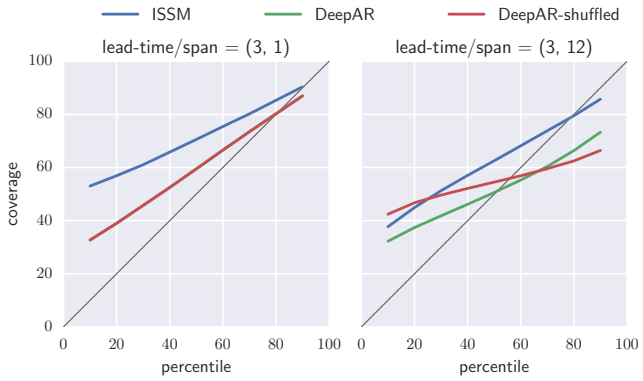

Figure 3. Coverage for several percentiles on `ec-sub` dataset. The left part gives the coverage metrics over a single time-step interval while the right part shows these metrics a larger time interval with 9 time-steps. In the latter case, modelling correlation is necessary to get accurate predictions. Indeed, the forecast becomes less calibrated when shuffling the samples which destroy their correlation, this also increases the 0.9-risk by 10% for the lead-time/span (3, 12).

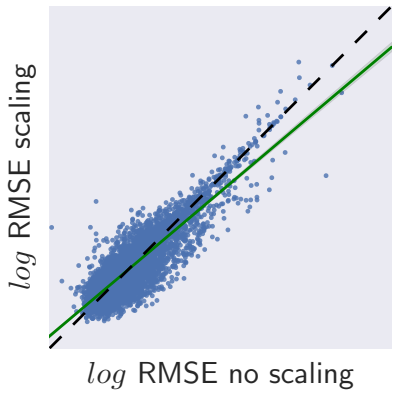

Figure 4. RMSE of the forecast with velocity scaling (y-axis) and without velocity scaling (x-axis) in logarithmic scale. The solid straight line shows a linear fit of the data. The tilting with respect to the diagonal (dashed line) indicates the increased error for higher velocities.

used precedently in (Seeger et al., 2016). The data is taken at a weekly grain, we predict 52 weeks and evaluation is done on the year following `2014-09-07`. Some of the time series in the dataset correspond to items that were already sold before the start of the data set (i.e. have been truncated) and others start at some point in the data window, i.e., they correspond to new products (however no new items start in the evaluation window). Note that the time series in these datasets are very diverse and erratic (see Fig. 6), ranging from slow moving items to very fast items, and the item velocities have a power-law distribution as shown in Fig. 1.

4.2. Accuracy comparison

For the `parts` and `ec/ec-sub` datasets we compare with the negative-binomial autoregressive method of Snyder et al. (2012) (`NegBin`) and the `ISSM` method (Seeger et al., 2016), respectively, which where the best-performing methods on these data sets in the comparison of Seeger et al. (2016).

As both these methods provide probabilistic forecasts, we use metrics that quantify the accuracy of different quantiles of the predicted distributions. These metrics are evaluated for a certain *spans* $[L, L+S)$ in the prediction range, where L is a *lead time* after the forecast start point. The aggregated target value of an item i in a span is denoted as $Z_i(L,S) = \sum_{t=t_0+L}^{t_0+L+S} z_{i,t}$. For a given quantile $\rho \in (0,1)$ we denote the predicted ρ-quantile for $Z_i(L,S)$ by $\hat{Z}_i^\rho(L,S)$. To obtain such a quantile prediction from a set of sample paths, each realization is first summed in the given span. The samples of these sums then represent the estimated distribution for $Z_i(L,S)$ and we can take the ρ-quantile from the empirical distribution.

The ρ-quantile loss is then defined as

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

In order to summarize the quantile losses for a given span across all items, we consider 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 ρ-risk. Table 1 shows the risks for $\rho = 0.5$ and $\rho = 0.9$ and 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 accuracy of `NegBin` for `parts` and `ISSM` for `ec/ec-sub`.

In Table 2 we compare point forecast accuracy on the `electricity` and `traffic` datasets against the matrix factorization technique (`MatFact`) proposed in (Yu et al., 2016). We consider the same metrics as in (Yu et al., 2016), namely Normalized Deviation (ND) and Normalized RMSE (NRMSE), defined as

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

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

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.

The results shown in Table 1 and 2 show that DeepAR significantly outperforms the state-of-the art methods `ISSM`

	$\rho = 0.5$ risk				$\rho = 0.9$ risk				average
				parts					
(L, S)	$(0, 1)$	$(2, 1)$	$(0, 8)$	all(8)	$(0, 1)$	$(2, 1)$	$(0, 8)$	all(8)	average
DeepAR	**0.98**	**0.92**	**0.98**	1.01	**0.96**	**0.99**	1.02	**0.97**	**0.98**
NegBin	1.00	1.00	1.00	**1.00**	1.00	1.00	**1.00**	1.00	1.00
				ec-sub					
(L, S)	$(0, 2)$	$(0, 8)$	$(3, 12)$	all(33)	$(0, 2)$	$(0, 8)$	$(3, 12)$	all(33)	average
DeepAR	**0.66**	**0.81**	**0.98**	**0.76**	**0.73**	**0.87**	**0.97**	**0.56**	**0.79**
ISSM	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
				ec					
(L, S)	$(0, 2)$	$(0, 8)$	$(3, 12)$	all(33)	$(0, 2)$	$(0, 8)$	$(3, 12)$	all(33)	average
DeepAR	**0.59**	**0.68**	**0.99**	**0.98**	**0.76**	**0.88**	1.00	**0.91**	**0.85**
ISSM	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 1. Accuracy metrics for distribution forecasts, all scores are relative to a baseline method.

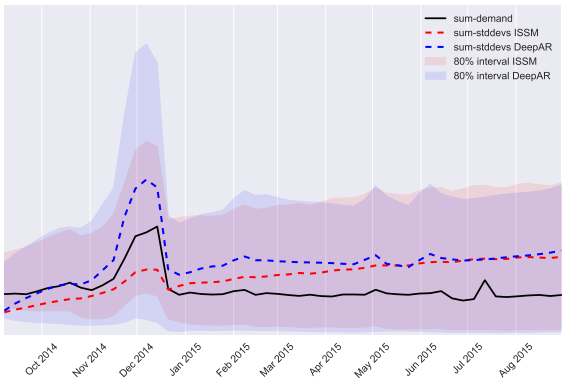

Figure 5. Uncertainty growth over time: unlike ISSM models where the uncertainty is typically assumed to grow linearly over time, DeepAR learns it in a data-driven way. Regions such as Christmas that are harder to predict exhibits a burst of uncertainty that drops after the region is passed. Overall we still learn a uncertainty trend but it is not linear.

and MatFact on medium and large datasets.

On the traffic dataset, MatFact outperforms DeepAR; we speculate that this is due to MatFact effectively exploiting the fact that the time series in this data set are very similar to each other. Futher, DeepAR's accuracy could probably be improved on this dataset by using the more suitable Beta likelihood, as the data lies in unit interval.

4.3. Qualitative analysis

We first show the effect of our scaling correction in Fig. 4. This scatter plot depicts the RMSE in log scale with and without scaling correction. Without scaling, faster items exhibit larger error than slower, items which is highly problematic for demand forecasting. The difficulty for the neural network to learn from few high velocity examples also results in a higher overall error if the scaling correction is omitted.

	electricity		traffic	
	ND	RMSE	ND	RMSE
DeepAR	**0.08**	**0.49**	0.27	0.56
MatFact	0.25	1.40	**0.19**	**0.42**

Table 2. Comparison with the global forecasting method MatFact

In Fig. 5 we show different quantiles of the marginal predicted distribution for DeepAR as well as the ISSM model. This gives an indication of how both models handle the growth of uncertainty over time. Compared to ISSM models such as (Seeger et al., 2016) where a linear growth of uncertainty is postulated and included in the a model assumption, DeepAR learns the uncertainty from the data. In this case, the model does learn an overall growth of uncertainty over time, but this is not merely a linear growth. In this case, for instance, the model learned that during the Christmas period uncertainty is higher, but after Christmas the uncertainty decreases again. Such time-dependent variance is one of the reasons why the predicted distributions are more accurate.

The calibration of the forecast distribution is depicted in Fig.3. Here we show the coverage metric Coverage(p), where p is a percentile of the distribution. This metric is defined for a given lead-time span as the fraction of time series in the dataset, for which the p percentile of the predicted distribution is larger than the the true target. For a perfectly calibrated prediction it holds that Coverage$(p) = p$, which corresponds to the diagonal. Compared to the ISSM model DeepAR has a better calibration overall. To estimate how far the predictions of the recurrent neural network (RNN) are correlated, i.e., how much they differ from

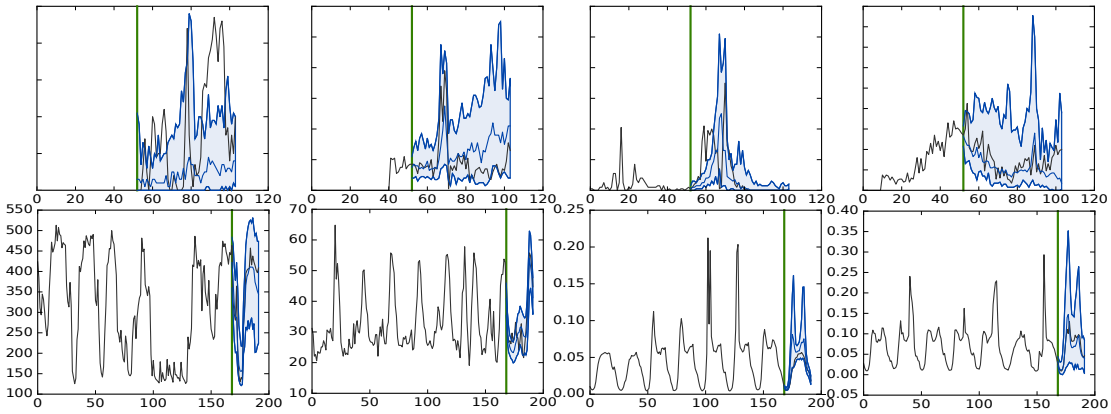

Figure 6. Time series with conditioning period to the left of the vertical green line and prediction period to the right. Observed data is black and forecasts are in blue for p10, p50 and p90. Top: Demand time series together for `ec`. Note that the forecasts are reasonable even though none of the individual time series has relevant information in the conditioning period. Bottom: Time series for `electricity` (first two) and for `traffic` (last two).

independent distributions for each time-point, we plot the calibration curves for a randomized forecast where for each time-point the realizations of the original forecast have been shuffled, thus destroying any correlation between different time-points. For the short lead-time span (left) that consists of just one time-point, this has no impact, because it is just the marginal distribution. For the longer lead-time span (right), however, re-shuffling the samples leads to a worst calibration. This shows that the RNN model is not merely predicting independent distributions for each point. In particular, to generate such a correlation the model has to change its forecast for the next time-point based on what was sampled in the previous time-point for that particular realization.

Finally, example predictions of time series are shown in Fig. 6 for different datasets. Both for highly irregular time series with little relevant data in the conditioning period and for more regular time series, DeepAR's predictions resemble typical behavior in the dataset and one can see how aspects like seasonality, trend and uncertainty are reflected in the forecast.

5. Conclusion and future work

We have presented DeepAR, a new probabilistic method for time series forecasting based on recurrent neural networks. Our experiment show a drastic improvement in forecasts accuracy over the state of the art. DeepAR learns from historical behavior of items in a dataset and gives considerably better forecasts than state of the art forecasting models for large enough historical data sets.

There are many future directions for extensions and improvements. A potential shortcoming of using LSTM for forecasting arises when considering finer time-grains such as seconds where seasonality in larger time-frames. This necessitates further attention and more elaborated LSTM gates such as the one proposed in (Neil et al., 2016).

Specifying different likelihoods is straight-forward in DeepAR. The only prerequisite are functions for sampling and for the log-likelihood. Therefore, more complex likelihoods (e.g. (Seeger et al., 2016)) can be incorporated easily. Another direction would be to not postulate a likelihood, but work with other generative models, that learn a distribution from the data such as (Goodfellow et al., 2014).

Finally, an exciting avenue for future work is transfer learning. Initial experiments revealed that DeepAR is robust across time within a fixed domain. It would be worthwhile to understand if the same holds across domains.

Acknowledgements

The authors wish to thank the Lutter team and in particular Tim Januschowski for his continous support and faith in the project.

References

Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

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

Box, George E. P. and Jenkins, Gwilym M. Some recent advances in forecasting and control. *Journal of the Royal*

Statistical Society. Series C (Applied Statistics), 17(2): 91–109, 1968.

Chapados, Nicolas. Effective bayesian modeling of groups of related count time series. In *Proceedings of The 31st International Conference on Machine Learning*, pp. 1395–1403, 2014.

Chen, Tianqi, Li, Mu, Li, Yutian, Lin, Min, Wang, Naiyan, Wang, Minjie, Xiao, Tianjun, Xu, Bing, Zhang, Chiyuan, and Zhang, Zheng. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015.

Díaz-Robles, Luis A, Ortega, Juan C, Fu, Joshua S, Reed, Gregory D, Chow, Judith C, Watson, John G, and Moncada-Herrera, Juan A. A hybrid arima and artificial neural networks model to forecast particulate matter in urban areas: the case of temuco, chile. *Atmospheric Environment*, 42(35):8331–8340, 2008.

Gers, F. A., Eck, D., and Schmidhuber, J. Applying LSTM to time series predictable through time-window approaches. In Dorffner, Georg (ed.), *Artificial Neural Networks – ICANN 2001 (Proceedings)*, pp. 669–676. Springer, 2001.

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

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014.

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

Gregor, Karol, Danihelka, Ivo, Graves, Alex, Rezende, Danilo Jimenez, and Wierstra, Daan. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

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

Hyndman, Rob J., Ahmed, Roman A., Athanasopoulos, George, and Shang, Han Lin. Optimal combination forecasts for hierarchical time series. *Computational Statistics & Data Analysis*, 55(9):2579 – 2589, 2011.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015.

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

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Kourentzes, Nikolaos. Intermittent demand forecasts with neural networks. *International Journal of Production Economics*, 143(1):198–206, 2013. ISSN 09255273. doi: 10.1016/j.ijpe.2013.01.009.

Larson, Paul D., Simchi-Levi, David, Kaminsky, Philip, and Simchi-Levi, Edith. Designing and managing the supply chain: Concepts, strategies, and case studies. *Journal of Business Logistics*, 22(1):259–261, 2001. ISSN 2158-1592. doi: 10.1002/j.2158-1592.2001.tb00165.x. URL http://dx.doi.org/10.1002/j.2158-1592.2001.tb00165.x.

Neil, Daniel, Pfeiffer, Michael, and Liu, Shih-Chii. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3882–3890. Curran Associates, Inc., 2016.

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

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

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

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

Yu, Hsiang-Fu, Rao, Nikhil, and Dhillon, Inderjit S. Temporal regularized matrix factorization for high-dimensional time series prediction. In Lee, D. D.,

Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 847–855. Curran Associates, Inc., 2016.

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

Supplementary material

Experiment details

We use MxNet as our neural network framework (Chen et al., 2015). Experiments are run on a laptop for `parts` and with a single AWS p2.xlarge instance (four core machine with a single GPU) for other datasets. Note that predictions can be done in all datasets end to end in a matter of hours even with a single machine. We use ADAM optimizer (Kingma & Ba, 2014) with early stopping and standard LSTM cells with a forget bias set to 1.0 in all experiment and 200 samples are drawn from our decoder to generate predictions.

	`parts`	`electricity`	`traffic`	`ec-sub`	`ec`
# 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
# 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	0.01	0.01
# LSTM layers	3	3	3	3	3
# LSTM nodes	40	40	40	150	150
running time	5min	7h	3h	3h	10h

Table 3. Datasets statistics and parameters

For `parts` dataset, we use the 42 first months as training data and report error on the remaining 8. For the other datasets `electricity`, `traffic`, `ec-sub` and `ec` the set of possible training instances is sub-sampled to the number indicated in table 3. The scores of `electricity` and `traffic` are reported using the rolling window operation described in (Yu et al., 2016), note that we do not retrain our model but reuse the same one for predicting across the different time windows instead. Finally, running times measures an end to end evaluation, e.g. processing features, training the neural network, drawing samples and evaluating produced distributions.