ARTICLE IN PRESS

International Journal of Forecasting xxx (xxxx) xxx

Contents lists available at ScienceDirect

International Journal of Forecasting

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

Robust recurrent network model for intermittent time-series forecasting

Yunho Jeon *, Sihyeon Seong

mofl Inc., Daejeon, Republic of Korea[1]

ARTICLE INFO

Keywords:
M5 accuracy competition
Time-series forecasting
DeepAR
Tweedie
Ensemble
Model selection

ABSTRACT

This paper describes a deep-learning-based time-series forecasting method that was ranked third in the accuracy challenge of the M5 competition. We solved the problem using a deep-learning approach based on DeepAR, which is an auto-regressive recurrent network model conditioned on historical inputs. To address the intermittent and irregular characteristics of sales demand, we modified the training procedure of DeepAR; instead of using actual values for the historical inputs, our model uses values sampled from a trained distribution and feeds them to the network as past values. We obtained the final result using an ensemble of multiple models to make a robust and stable prediction. To appropriately select a model for the ensemble, each model was evaluated using the average weighted root mean squared scaled error, calculated for all levels of a wide range of past periods.

1. Introduction

Because of the success of deep learning in computer vision applications, its application has gradually expanded to various fields such as natural language processing (NLP) and robot control. It has become mainstream in machine learning (ML). Although time series forecasting has been studied for a long time and is one of the major applications of ML, the progress of applying deep learning is somewhat lagging behind other fields.

Initially, deep learning for time-series forecasting was applied only in the form of models from other fields applied to time-series data. In recent years, research on applying deep learning to a time series has progressed significantly (Borovykh et al., 2017; Li et al., 2019; Oreshkin et al., 2019; Rasul et al., 2021; Salinas et al., 2019a, 2019b; Sen et al., 2019; Wang et al., 2019b). Although classical forecasting models such as ARIMA (Box & Jenkins, 1968) and exponential smoothing (Hyndman et al., 2008) are used in applications that have a single or small number of time series, deep learning approaches have tackled the forecasting problem with a large number of related time series (such as, electricity usage in numerous homes and traffic congestion forecasts). Because a deep learning-based model has the advantage of extracting features from high-dimensional inputs, it has shown a meaningful performance for applications that use a large number of related time series.

However, the benchmark dataset for time series applications is still relatively small compared to that of other applications (e.g., ImageNet (Russakovsky et al., 2015) for image recognition and GLUE (Wang et al., 2019a) for NLP) and has the limitation that research has not been connected to real-life applications. Although electricity (Trindade, 2015) and traffic (Cuturi, 2011) datasets are commonly used as a benchmark, they only have a couple of hundred sequences, which is a small number when compared to real-life applications. Furthermore, the experimental protocol is not well established, and it makes a performance comparison between methods difficult to achieve.

* Corresponding author.
E-mail address: yunho.jeon@mofl.ai (Y. Jeon).
[1] URL: https://mofl.ai.

https://doi.org/10.1016/j.ijforecast.2021.07.004
0169-2070/

In this light, the M Competitions are beneficial events contributing to the advancement of the time-series forecasting field, providing real-world datasets and the opportunity to compare various methods under the same experimental conditions. In particular, the dataset used for the M5 competition has involved more realistic sales demands across various product categories. Many of these sequences include intermittent sales having many zeros as well as high-volume sales. This dataset also includes many covariates to explain the number of sales, e.g., the day, price of the items, category of the items, etc. The number of the time series in the dataset is 42,840, a large number of sequences compared to other benchmarks.

One of the challenges in M5 is to predict intermittent and irregular sales demands. Intermittent data forecasting (Croston, 1972) is concerned with sequences in which values appear sporadically. This is a challenging problem, as it involves irregular observations in time and numerous zero values. Such datasets have attracted considerable attention, as they often appear in real-world demand forecasting applications (Hyndman, 2006; Kourentzes, 2013, 2014; Seeger et al., 2016; Turkmen et al., 2020; Willemain et al., 1994). However, few studies have focused on deep-learning approaches for intermittent sequences (Kourentzes, 2013; Muhaimin et al., 2021), and more research is required in this direction.

Compared with previous datasets, the M5 dataset contains a large number of sequences of various items, as well as multiple covariates. As deep learning has outperformed other approaches in various fields when a large amount of data is used for training, we are convinced that deep learning can work well in the present context. Our goal in participating in the M5 competition was to generate a viable deep-learning solution for forecasting various types of items, including intermittent sales. This paper describes the successful application of such a method in this competition.

We selected DeepAR (Salinas et al., 2019b) for a base model, which is an auto-regressive recurrent neural network model. DeepAR has been applied successfully in various time series applications and is a representative deep-learning forecasting method (Januschowski et al., 2018). To adapt the characteristics of the M5 dataset, we modified the training procedure of DeepAR. Our method uses sampled values from the trained distribution in the training phase by rolling predictions differing from DeepAR, which uses an actual value as a past value. This can reduce the error from the discrepancy between training and inference (Bengio et al., 2015). Furthermore, this modification increases the generalization performance by injecting diverse past samples into the network. As this modification generates rolling predictions in the training phase, not only for the inference, we call this model rolled DeepAR.

We used the negative log-likelihood of the Tweedie distribution to tackle the zero-inflated distribution of sales demands for the loss function. A model with low WRMSSEs[2] for a wide range of past periods was chosen for generating a stable prediction: An average WRMSSE of these past periods is used as a metric for selecting a model. For the final prediction, an ensemble of multiple selected models was used to boost the performance.

The remainder of this paper is organized as follows. Section 2 describes the main idea of our method, and Section 3 provides more details about the implementations. Discussions and further analysis are presented in Section 4, and Section 5 provides some concluding remarks regarding this research.

2. Method

2.1. Network model

2.1.1. DeepAR

DeepAR (Salinas et al., 2019b) is a deep neural network model for time-series forecasting. Based on historical data, it produces a probability distribution for predicting future sequences. Unlike the classical autoregressive models like ARIMA, DeepAR makes a single global model (Januschowski et al., 2020) that can generate predictions for all time series. Many classical methods have been developed based on core assumptions such as stationary and white noise. However, many real-life applications easily violate these assumptions, and DeepAR does not rely on this assumption. Instead, DeepAR models the input sequence by maximizing the likelihood of a selected probability distribution; therefore, it is necessary to set an appropriate distribution of input sequences for the training.

DeepAR uses a recurrent neural network (RNN) as a basic component and accepts past sequences and its covariates as an input. Formally, denoting the ith item of sale at time t by $z_{i,t}$ and covariates $\mathbf{x}_{i,t}$, the goal of training is to predict the conditional probability P of future sales $z_{i,t_0:T}$ based on past values (length C) and covariates, where t_0 and T are the first and the last time of the future, respectively (Eq. (1)).

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

Denoting a multi-layer RNN parameterized by Θ as h, the output of a network at time t can be expressed as $\mathbf{h}_{i,t} = h(\mathbf{h}_{i,t-1}, z_{i,t-1}, \mathbf{x}_{i,t}; \Theta)$. In general, long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) cells are used for RNN blocks. DeepAR optimizes the model parameters to maximize the likelihood of the given probability distribution and the loss of DeepAR is defined as follows:

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

where θ is a linear mapping from the hidden representation $\mathbf{h}_{i,t}$ to the parameters of the given distribution, and ℓ is the likelihood of the distribution. Because there is no distinction between the past and future values in Eq. (2), DeepAR generally uses all historical sequences to calculate the loss (i.e., the loss is calculated from t_0-C+1); DeepAR is trained to predict a 1-step forwarded value regardless of whether it is of the past or future (Fig. 1(a)).

[2] Weighted root mean squared scaled error (WRMSSE) is defined in M5 guideline (Spiliotis et al., 2021).

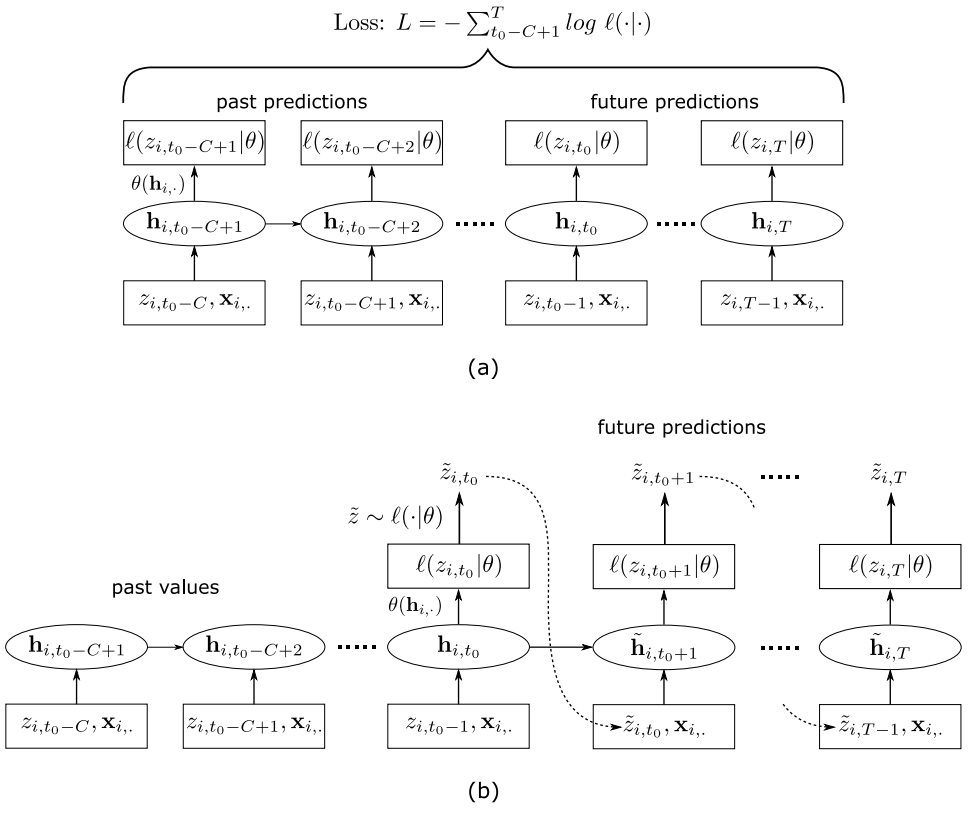

Fig. 1. Training and inference phase of DeepAR. (a) During the training, original input values are fed into the model, and the loss is calculated with all past and future predictions. (b) In the inference, past values are fed into the model, and future predictions are rolled with sampled predictions.

To predict multiple future days in the inference, the model generates the prediction of the next day repeatedly until the end of the future. First, past sequences ($t < t_0$) are fed into the trained model, and the prediction of the first day is generated by sampling from the trained probability distribution (i.e., $\tilde{z}_{t_0} \sim \ell(\cdot|\theta(\mathbf{h}_{i,t_0}))$). The prediction of the first day is fed into the model to generate the prediction of the second day, and likewise for each successive day, which is called rolling predictions (Fig. 1(b)). Because the future prediction depends on past samples from the predicted distribution, the output of the model is not deterministic and is a distribution of sampled sequences. This sampling procedure is useful because it can generate a probability distribution of predictions that can measure the reliability of the predictions.

2.1.2. Rolled DeepAR

DeepAR is trained to predict a 1-step forward value based on past sequences, and these past sequences are all actual values from input sequences. Whereas the model is only trained using actual values, sampled predictions are applied as past values for the next prediction in the inference. Because of this difference between training and inference, the model may generate erroneous predictions if an untrained past sequence pattern is seen. And a small error in each time step may accumulate over time, resulting in a significant drift of future forecast. When the input sequences are more diverse and erratic, this type of situation can occur more often.

To reduce this error, Bengio et al. (2015) proposed *scheduled sampling*, in which the network input is randomly selected between the true and a sampled value; the probability of selecting a sampled value is adjusted based on the training phase. Similarly, we modified the training phase to reduce the difference between training and inference. Our method generates rolling predictions in the training phase and feeds the sampled prediction to the network for predicting the next day. As the network also makes rolling predictions during the training phase, we called this model rolled DeepAR. This is equivalent to using sampling with 100% probability in *scheduled sampling*.

In addition to reducing the accumulated error in inference, this modification has additional benefits. By using sampled prediction instead of ground truth values for future values, the training sequences can be diverse even though the input sequence is the same; this acts as an augmentation of input sequences. Because there are many cases where the input value is probabilistic (e.g., many zeros and intermittent demands), this sampling method helps the model generate robust predictions. This is the main reason for using the sampling inputs in the entire training procedure. Additionally, as the training procedure is the same as the inference, it is easy to assess the status of the model during the training phase.

The major difference between our method and DeepAR is the method of generating the loss during the training.

Unlike DeepAR, which always uses the actual values for the training, the rolled DeepAR model uses the sampled past values to predict the future values (Fig. 2). Formally, the future values $\tilde{z}_{i,t}$ are drawn from conditional probability based on past data (i.e., $\tilde{z}_{i,t} \sim \ell(\cdot|\theta(\tilde{\mathbf{h}}_{i,t}))$), and the next prediction $\tilde{z}_{i,t+1}$ are drawn by feeding a previously sampled value for the next prediction (i.e., $\tilde{\mathbf{h}}_{i,t+1} = h(\tilde{\mathbf{h}}_{i,t}, \tilde{z}_{i,t}, \mathbf{x}_{i,t+1}; \Theta)$). After rolling all future predictions, the loss is summed only for the future predictions instead of using all sequences as follows:

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

One should note that rolled DeepAR is not necessarily better than DeepAR, and it depends on the application. Using actual values for the future in DeepAR is similar to teacher forcing (Williams & Zipser, 1989) in NLP models. Teacher forcing replaces the sampled value from the network with the actual value to prevent the drift of sequential outputs caused by previous incorrect outputs. This helps fast convergence of the training model. However, in the M5 application, the input sequence is regarded as one sample of a probability distribution, and the value is not the only correct answer. For example, suppose a person purchases one item per week, and the purchase date is randomly distributed. In that case, it is better to train with as many cases as possible rather than using only one set of samples. This is why rolled DeepAR is used for this competition.

2.2. Tweedie loss

Because DeepAR is trained to maximize the likelihood of the probability distribution, it is important to set the proper output distribution. Even though the accuracy competition is only required to generate point forecasts, training with the probability distribution facilitates the generation of long-term forecasting horizons. As DeepAR generates predictions for multiple days with rolling, previous predictions become inputs for the next forecasting. Using multiple samples from the trained distribution generates various forecasting sequences. The final forecast is an ensemble of these predictions, whereas rolling with point forecasts only produces a single forecasting sequence. Accordingly, the final forecast is more robust than the point forecast.

A Poisson or negative binomial distribution can be generally used for modeling customer demands. However, many M5 series at the lowest level (level 12) display intermittent demand, including many zero values, and these distributions can underestimate the frequency of zero values. To alleviate this problem, we used the Tweedie distribution (Tweedie, 1984) as the target distribution. Tweedie distributions include several different distributions according to the Tweedie power parameter ρ, and we use $1 < \rho < 2$, which is related to the compound Poisson–gamma distribution. This distribution is the sum of N i.i.d. random variables sampled from a gamma distribution, where the number of samples to be added (N) is a Poisson-distributed variable:

$$\begin{aligned} N &\sim \text{Poisson}(\lambda) \\ X_n &\sim \text{Gamma}(\alpha, \beta) \\ Y &= \sum_{n=1}^{N} X_n \end{aligned} \tag{4}$$

where λ, α, and β are parameters of the Poisson and gamma distributions, respectively. By definition of the distribution, the Tweedie model has a massive probability at zero, which is more suitable for M5 sequences. The probability density function of the Tweedie distribution ($1 < \rho < 2$) can be defined by re-parameterizing the distribution using Eq. (5) as follows (Zhou et al., 2020):

$$\lambda = \frac{1}{\phi}\cdot\frac{\mu^{2-\rho}}{2-\rho},\ \alpha = \frac{2-\rho}{\rho-1},\ \beta = \frac{1}{\phi}\cdot\frac{\mu^{1-\rho}}{\rho-1} \tag{5}$$

$$f(z|\mu,\phi,\rho) = a(z,\phi,\rho)\cdot\exp\left(\frac{1}{\phi}\left(z\frac{\mu^{1-\rho}}{\rho-1} - \frac{\mu^{2-\rho}}{2-\rho}\right)\right), \tag{6}$$

where $a(z,\phi,\rho)$ is a normalization factor, and μ and $\phi\cdot\mu^{\rho}$ are the mean and variance of the distribution, respectively.

Eq. (7) shows the log-likelihood of the distribution. Because the normalizing factor a is the sum of an infinite series (Zhou et al., 2020), it is not easy to calculate the partial derivative of the log-likelihood for ϕ. To simplify the problem, we assumed ϕ as a constant, and we only modeled the mean of distribution μ from a linear projection of the hidden representation $\tilde{\mathbf{h}}_{i,t}$. More precisely, because μ should always be positive, we modeled $\log(\mu)$ as a network output. Eq. (8) shows the loss function of the network.

$$\log f(z|\mu,\phi,\rho) = \log(a(z,\phi,\rho)) + \frac{1}{\phi}\left(z\frac{\mu^{1-\rho}}{\rho-1} - \frac{\mu^{2-\rho}}{2-\rho}\right), \tag{7}$$

$$\begin{aligned} L &= -\sum_{i=1}^{N}\sum_{t=t_0}^{T} \log \ell(z_{i,t}|\theta_{\mu}(\tilde{\mathbf{h}}_{i,t})), \\ &= -\frac{1}{\phi}\sum_{i=1}^{N}\sum_{t=t_0}^{T}\left(z_{i,t}\cdot\frac{\mu^{1-\rho}}{\rho-1} - \frac{\mu^{2-\rho}}{2-\rho}\right), \end{aligned} \tag{8}$$

where θ_{μ} is a linear projection for $\log(\mu)$ from the hidden representation $\tilde{\mathbf{h}}_{i,t}$.

By Eq. (4), sampling from Tweedie distribution requires two steps. First, the number of samples to be added (N) is drawn from the Poisson distribution. Subsequently, N samples drawn from the Gamma distribution are summed, resulting in a Tweedie sample. This procedure requires N iterations to draw N samples from the Gamma distribution. However, this can be simplified, as the sum of independent samples from the Gamma distribution also follows the Gamma distribution (Eq. (9)). Therefore, instead of sampling N times from the Gamma distribution, the single sample from the aggregated Gamma distribution (Gamma($N\alpha, \beta$) is the same as the

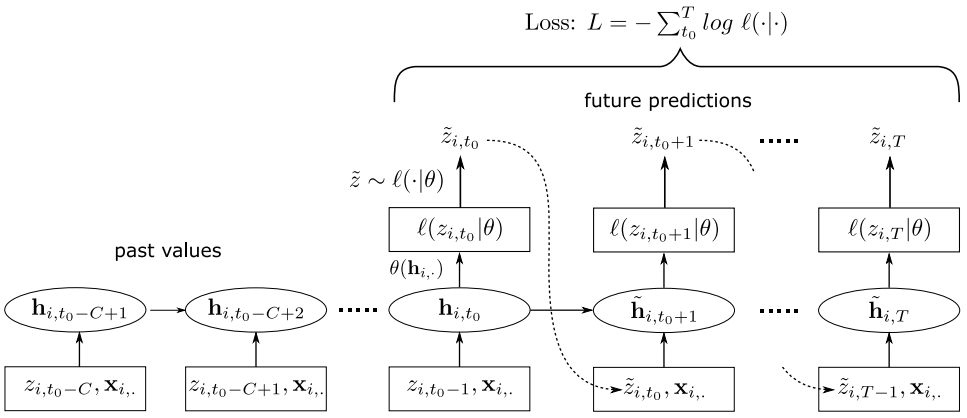

Fig. 2. The training phase of rolled DeepAR is almost the same as the inference phase of DeepAR. Past values are fed into the model, and future predictions are rolled with sampled predictions. The loss is only calculated with future predictions.

Tweedie sample.

$$\begin{aligned} X_i &\sim \text{Gamma}(\alpha, \beta) \\ \sum_{i=1}^{N} X_i &\sim \text{Gamma}\left(N\alpha, \beta\right), \end{aligned} \tag{9}$$

where all X_i are independently sampled.

2.3. Model selection

It is important to find the proper model that works well in machine learning during test time. The validation set is adopted for a model choice in general. For deep learning, in particular, the performance of a model depends on various configurations such as hyperparameters and initial weights. We did not search for hyperparameters extensively and applied generic hyperparameters to all experiments. Instead, we focused on choosing a model showing lower error at the test time from numerous trials, which are trained from random initialization.

For the time-series forecasting, the last period is commonly used for the validation; that is, if the forecast horizon is T_f and the total length of the input sequence is T_t, the model is trained within time $t \in [0, T_t - T_f)$ and the validation period is $t \in [T_t - T_f, T_t)$. However, we found that a validation period (d1914[3] $\sim$ d1941) has significantly different patterns with another period because there are many items in which sales increase suddenly from zero. We thought this was due to out of stocks situations. However, we have no priors or evidence to model these abrupt changes. Therefore, we decided not to rely heavily on the performance of this period and chose a model that shows a stable performance over a wide range of periods.

In principle, it is believed that validation data should be separated from the training data to prevent overfitting. However, there was a trade-off in that if we exclude a wide range of validation data from training, the model cannot learn the latest information. We believe that learning the latest data is more important than separating the validation data from the training data. Therefore, we used all periods of the input sequence as the training data.

Another reason why we include all periods to train the sequences is based on empirical evidence indicating that the input sequences are rarely over-fitted. As stated earlier, M5 input sequences have a probabilistic nature and thus are particularly noisy. Therefore, it is difficult to achieve a small error unless the model does not memorize sequences completely. As a result, the model was more vulnerable to an under-fitting than an over-fitting.

Based on the findings and assumptions, we evaluated our model using the previous 14 sets consisting of 28 days each (28 $\times$ 14 days, from d1550 to d1941). We selected 14 periods to include the last year of the same period as the private set (2015-05-23$\sim$2015-06-19). We calculated WRMSSEs of all levels for 14 sets and used their average value to measure model selection.

2.4. Ensemble

Ensemble methods are well-known techniques that combine multiple models to achieve a better performance than a single model. Because the weights of the network are randomly initialized in our model, the prediction of each trial can be diverse. We made an ensemble using two strategies: an ensemble of predictions from multiple epochs within a single trial and an ensemble of predictions from multiple trials.

Instead of choosing one prediction from a single trial, we chose multiple epochs to predict a trial. Because the performance of the model changes as the training progresses, it is more robust to select predictions from multiple epochs than from one epoch. During training, the model parameters were saved every ten epochs, and we chose the top-k performing epochs from the saved epoch models. The performance of each model was evaluated using the average WRMSSE of 14 sets described in the previous section. Fig. 3 shows an example of the effect of the ensemble between epochs. The three gray lines indicate the WRMSSEs of the top-3 epochs chosen from the same training trial. The green line is the ensemble of these models and shows a better WRMSSE for most periods than a single epoch prediction.

[3] 1914th date of the dataset.

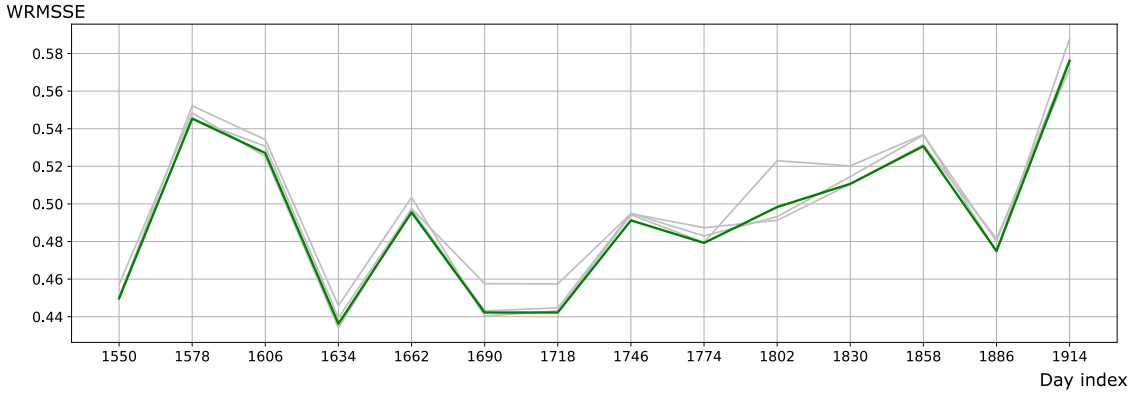

Fig. 3. Ensemble of epoch models from a single trial. Each gray line shows the WRMSSEs of the top-3 epoch models. The green line indicates an ensemble of these models and shows a better performance during most periods. The *x*-axis represents the start date of each period, that is, 1550 represents the period of d1550∼d1577.

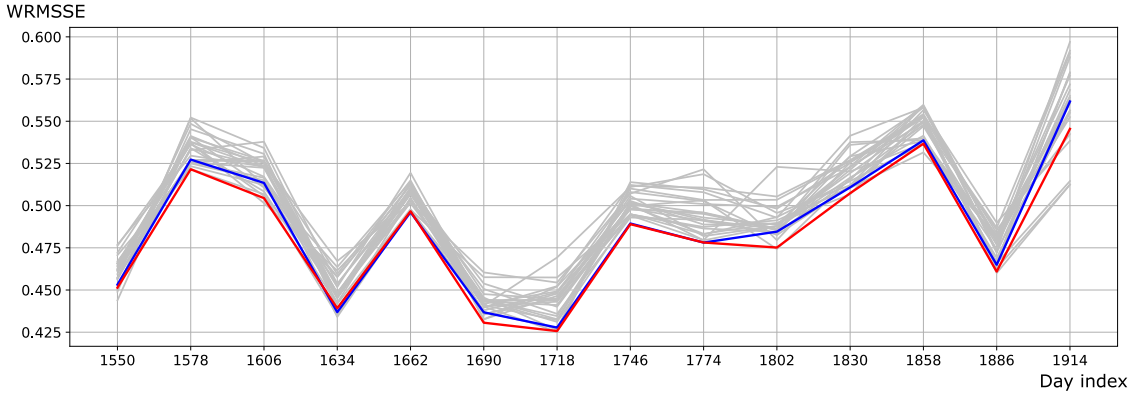

Fig. 4. An example of an ensemble of multiple trials. Gray lines show WRMSSEs of the best three epoch models from eight different trials. The blue line is an ensemble of two trials (a total of six models). The red line is an ensemble of eight trials (a total of 24 models). The *x*-axis represents the start date of each period. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The second ensemble strategy uses multiple trials. As the performance of models can differ depending on the initialization, we used multiple trials to make an ensemble; each training starts from different initial weights. An ensemble makes the representative prediction of each trial of multiple epochs described earlier. Fig. 4 shows the ensemble of eight trials. Gray lines are WRMSSEs of 24 models (eight trials $\times$ three epochs), and they have a wide band of variations. An ensemble of these eight trials (red line) shows better results than an ensemble of two trials (blue line). The red line achieved the lowest WRMSSE during most periods than any single model except for the 1914 set (d1914∼d1941).

In the first strategy, models for an ensemble are sampled in different epochs in the same training procedure, resulting in smaller model diversity than in the second strategy, in which models are trained with different initialization. Generally, as diversity is encouraged for the ensemble, the second strategy is more effective. However, as the second strategy requires training multiple times (whereas the first only requires one training pass), it is more expensive computationally.

3. Implementations

3.1. Features

Table 1 shows a list of all features used for the model input. All features are concatenated along the channel dimension and fed into the network. There are 100 feature channels, and we used the past 28 days for the network input; i.e., the dimensions of the network input are the batch size $\times$ 28 $\times$ 100. These features have various information, including sales, time, price, and events. The type of feature can be classified into two types: sequential and categorical. The sequential type is a 2-dimensional real-valued feature that spans the time and channel dimensions. This feature is fed to the network as is. The categorical type features are embedded into the given channel before feeding to the network. These embeddings are also trained from the network; an embedding layer is a lookup table that maps an input value to a trainable embedding.

We used the lowest level (level 12) of sales data for raw values of sale, which are 30,490 sales of product aggregated for each store. Including sales data of the past 28

Table 1
Features used for the network input. All of these features are concatenated along the channel dimension. Categorical features are expanded to the given channel using a trainable embedding layer.

Name	Type	#ch	Feature	#ch	Note
Sales	Sequential	3	Raw value	1	
			Moving average of past 7 days	1	Level 12
			Moving average of past 28 days	1	
Time	Sequential	6	Week day	1	
			Day	1	
			Month	1	Normalized to [−0.5, 0.5]
			Week number	1	
			Year	1	
			# of days after first sale	1	Log-scale
Price	Sequential	3	Raw price	1	
			Normalized price across time	1	
			Normalized price in the same department	1	
SNAP	Sequential	3	SNAP for CA, TX, WI	3	
Event	2d categorical	21	Event type 1	2	5 categories
			Event type 2	2	
			Event name 1	15	31 categories
			Event name 2	2	
ID	1d categorical	63	State-id	2	3 categories
			Store-id	5	10 categories
			Cat-id	2	3 categories
			Dept-id	4	7 categories
			Item-id	50	3049 categories
Zero-sales period	Sequential	1	Continuous zero-sale days	1	

days and a moving average of the sales of the past 7 and 28 days (the average value of $z_{i,t-7+1:t}$ and $z_{i,t-28+1:t}$) are also used as inputs to provide information about short- and long-term trends of sales. As a time feature, five types of values are used: weekday, day, month, week number, and year. These values are normalized to the range of [−0.5, 0.5] by subtracting 0.5 after dividing the maximum value. Regarding the year, we subtracted 2014 and divided it by 5 for normalization. The aging feature, which represents the number of days after the first sale, is used after scaling by the log function for each sequence.

For the price feature, three features are used, i.e., the raw price value and two normalized price values. We used a standard score for the normalization: All values are divided by the standard deviation after the mean value is subtracted. The first is normalized for each item across all time to utilize the disparity from the mean price. The second is normalized within the item group having the same department to compare the relative price of each item. The three values of the supplement nutrition assistance program (SNAP), which is a binary feature in M5 indicating whether the stores allow SNAP purchases on the examined date, were used as-is.

Calendar events are two-dimensional categorical features that have different values depending on the time, and these are embedded and fed into the network. The IDs are one-dimensional features that have the same value independent of time. To match the dimension with other features, IDs are repeated along the time dimension after embedding.

The final feature is the zero-sale period, which indicates the number of days in which sales are consecutively zero. This feature is used to help the network infer out-of-stock items. Because this feature is unavailable for the future, this value is also generated along with each sale prediction; if the prediction is below 0.5, the count is increased by one or is reset to zero.

3.2. Training

We implemented our own model[4] based on PyTorchTS (Rasul, 2021), which is a variant of GluonTS (Alexandrov et al., 2020) using PyTorch (Paszke et al., 2019). We stacked two LSTM layers with 120 cells for the rolled DeepAR. The network was trained to maximize the log-likelihood of the Tweedie distribution as stated in Section 2. To generate the log mean of the Tweedie distribution, we attached a linear layer following the last LSTM layer.

The model was optimized by Kingma and Ba (2015). The learning rate increased linearly from 0 to 1e−3 for the first five epochs, and cosine annealing (Loshchilov & Hutter, 2017) was applied for the rest of the epochs (Fig. 5). The batch size was 64. Training samples with a given time slice (28 past days) were randomly selected for each sequence. Because the number of all sequences is 30490, 477 ($= 30490/64 + 1$) iterations are needed to go through all sequences. We defined this number of iterations as one epoch. For every epoch, the order of sequences was shuffled, and every batch contained different sequences. We trained the network for 300 epochs without early stopping.

As the final output is the point forecast, ϕ of the Tweedie distribution only controls the deviation of rolling samples in the training phase; a large value generates more diverse samples. We used 1 for ϕ as the default and

[4] The code is available at: https://github.com/devmofl/M5_Accuracy_3rd.

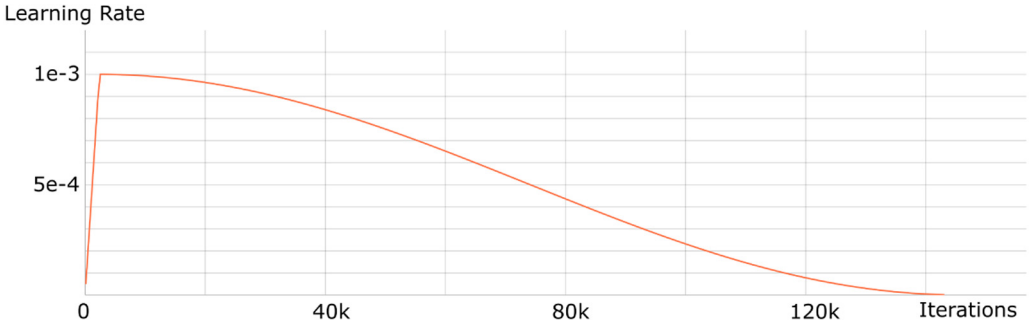

Fig. 5. Learning rate scheduling. The *x*-axis is the number of iterations, and the *y*-axis is the value of the learning rate. The learning rate increases linearly until 5 epochs and is gradually annealed.

did not search for the optimal value. Regarding training data, we used all available data (from d1 to d1941) except the zero-sale period from the beginning of each sequence; we removed zero-sale periods before the first sales to reduce samples having all zero values.

3.3. Inference

As stated earlier, the inference is performed by rolling samples of predictions drawn from the trained Tweedie distribution (Fig. 1(b)). We draw 100 samples from the distribution and feed these samples to the network to generate predictions on the next day; this is applied until the last day of the future. From this process, we can obtain sample predictions for all future days. To convert these samples to the point forecast, we used the average daily samples as the final value. Because the prediction is for the sale of level 12, we made all levels of predictions by aggregating predictions according to the aggregation levels.

Although the sales value should be an integer in the real world, a floating-point sales prediction is not problematic for M5, as the objective of the competition is to achieve the lowest WRMSSE. Accordingly, we did not apply any post-processing, such as rounding, to the samples of the Tweedie distribution. If an integer value is required for real-world applications, the fractional part of a number can be simply dropped or interpreted as the probability of the number being rounded off.

3.4. Ensemble method for submission

For the submission, we considered two configurations to improve the generality of the result. For the first configuration, the Tweedie power was 1.2, and we applied a 0.1 dropout rate on the LSTM layer. We trained eight models with different initialization and selected three epochs for each model according to the strategy stated in Section 2.3; a total of 24 (8×3) models are selected.

For the second configuration, we used a Tweedie power of 1.3 and did not apply a dropout. We trained four models with different initialization and selected 19 models out of all trained epochs. Because of the deadline, we were unable to train eight models for these configurations. Instead, we chose the top-19 models out of all epochs according to the same criteria.

By comparing these two configurations, the second exhibited slightly better performance in terms of our model selection metric. However, the first configuration includes dropout, which is commonly considered to improve generalization capability. As the series of M5 had a large dynamic range over time, the performance of the trained model exhibited wide variations on these series even under the same architecture and hyperparameters. Owing to the abrupt changes in the data and the difficulties in model selection, we opted to reduce risk by averaging both configurations; we selected 43 (24 + 19) models for the ensemble and averaged all predictions from these models with the same weighting. The score of the ensemble model was 0.52575 for the public dataset and 0.53571 for the private dataset.

4. Discussion

4.1. Choice of modeling method

Various approaches can be applied to this type of application. Gradient boosting (Chen & Guestrin, 2016; Ke et al., 2017) is one of the most successful approaches in several Kaggle competitions, and numerous participants in the M5 competition selected this approach. Even though gradient boosting has been successfully applied in various applications, we regarded deep learning more suitable for M5. It requires modeling sales considering the effect of high-dimensional inputs. Furthermore, one of the reasons for selecting deep-learning approaches for this competition was that that the size of the data required for training deep networks was sufficiently large.

Several neural networks can be used to model time series, such as DeepAR, transformer (Vaswani et al., 2017), and casual convolutions (Borovykh et al., 2019; van den Oord et al., 2016) . We conducted preliminary experiments using various architectures, and we selected DeepAR and transformer as candidate network architectures. We were not able to find clear evidence regarding which of the two models was superior, and we selected DeepAR because there are considerably more studies on DeepAR and LSTM models than on transformer-based models (Wu et al., 2020). However, further study is required to clarify which architecture is better for forecasting intermittent time series.

Table 2
Comparison of the WRMSSE score according to variants of DeepAR. Each result is an ensemble of 24 models (8 trials × 3 epochs). The value of each row is the average and standard deviation of the WRMSSE for five runs.

Network	Public score	Private score
DeepAR	0.4760 ± 0.0030	0.6586 ± 0.0153
Rolled DeepAR	0.5283 ± 0.0096	0.5454 ± 0.0028

Table 3
Comparison of the WRMSSE according to the number of models for an ensemble. The value of each row is the average and standard deviation of the WRMSSE for five runs.

# of models	Public score	Private score
1 trial × 3 epochs	0.5495 ± 0.0222	0.5544 ± 0.0117
2 trials × 3 epochs	0.5443 ± 0.0160	0.5485 ± 0.0089
4 trials × 3 epochs	0.5417 ± 0.0147	0.5480 ± 0.0045
8 trials × 3 epochs	0.5283 ± 0.0096	0.5454 ± 0.0028

4.2. Model comparisons

Table 2 shows a comparison between DeepAR and rolled DeepAR. Each result is an ensemble of 24 models (eight trials × three epochs), which is the first strategy of the submitted method. Although DeepAR achieved a better public score, the private score is much worse than our modified version.

Our method is based on an ensemble of multiple models; it requires a significant amount of time to train the network. We experimented with the performance of the ensemble model according to the number of models. Table 3 shows the results of this experiment. When increasing the number of models for an ensemble, the public and private score (WRMSSE) decreases. Furthermore, the standard deviation decreases, which means that an ensemble model outputs more stable predictions. This result shows a trade-off between the performance and usage of resources.

4.3. Model execution time

Regarding the training time, it takes approximately 6 h for training 300 epochs of one model using a Titan RTX; however, the GPU utilization is not fully optimized, and there might be the possibility to reduce the training time. As rolled DeepAR should generate future predictions sequentially, it is two or three times as slow as DeepAR, which can generate all sequences in parallel in the training phase.

The training should be run multiple times for ensemble models, and increasing the number of the ensemble models also increases the training time linearly. It is possible to lessen the training time by reducing the number of models in the ensemble, but this can reduce the accuracy of the ensemble model; this is the trade-off between model accuracy and training time.

For each model, we make predictions for 14 periods using ten models saved in different epochs (from 200 to 300 epochs). It takes approximately one hour to make predictions for each model. After making a prediction, choosing the top-k models and making an ensemble prediction takes less than 5 min. If the model is already chosen, making a final prediction might take no longer than 10 min because predictions only need to be generated for the test period.

4.4. Limitations

The major drawback of rolled DeepAR compared with DeepAR is that it requires more training time. To generate future predictions, all past sequences should be generated. As a result, the training procedure cannot be parallelized, and thus GPU utilization cannot be reduced. In the inference phase, there is no difference between DeepAR and rolled DeepAR; both generate predictions sequentially.

We used the Tweedie distribution, as several sequences in the M5 data have intermittent characteristics. However, there were also a few items with a high volume, for which the Tweedie distribution is not appropriate. If possible, other distributions should be used for this group of items to achieve better performance. Instead of using Tweedie, we attempted to use classification: each sale value was directly mapped to one class label (0–1000). However, this was not successful due to the imbalance of the class distribution; large values are rare compared to small values in the training data.

5. Conclusion

In this paper, we described a deep-learning-based method used in the M5 accuracy competition. Based on DeepAR, we proposed a rolled DeepAR that modifies the training scheme by rolling the future predictions and provides a network with more diverse input distribution samples to achieve a generality of the model. A Tweedie distribution is applied to overcome the intermittent and irregular patterns of the demands. By using the performance of a wide range of periods for the model choice, we were able to select a stable model that performs well during the test period. In addition, we proposed an application of the deep learning method for solving a practical time-series problem and hope that this paper will be a step toward its advancement.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., Maddix, D. C., Rangapuram, S., Salinas, D., Schulz, J., Stella, L., Türkmen, A. C., & Wang, Y. (2020). Gluonts: Probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, *21*(116), 1–6, http://jmlr.org/papers/v21/19-820.html, https://gluon-ts.mxnet.io.

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *NIPS'15, Proceedings of the 28th international conference on neural information processing systems. Vol. 1* (pp. 1171–1179). Cambridge, MA, USA: MIT Press.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. arXiv preprint arXiv:1703.04691.

Borovykh, A., Bohte, S., & Oosterlee, C. (2019). Dilated convolutional neural networks for time series forecasting. *Journal of Computational Finance*, *22*(4), 73–101. http://dx.doi.org/10.21314/JCF.2019.358.

Box, G., & Jenkins, G. M. (1968). Some recent advances in forecasting and control. *Journal of the Royal Statistical Society. Series C. Applied Statistics*, *17*(2), 91–109, https://EconPapers.repec.org/RePEc:bla:jorssc:v:17:y:1968:i:2:p:91-109.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd Acm sigkdd international conference on knowledge discovery and data mining* (pp. 785–794).

Croston, J. D. (1972). Forecasting and stock control for intermittent demands. *Operational Research Quarterly (1970-1977)*, *23*(3), 289–303, http://www.jstor.org/stable/3007885.

Cuturi, M. (2011). Fast global alignment kernels. In *Proceedings of the 28th international conference on machine learning (ICML-11)* (pp. 929–936).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Hyndman, R. (2006). Another look at forecast accuracy metrics for intermittent demand. *Foresight: The International Journal of Applied Forecasting*, *4*, 43–46.

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

Januschowski, T., Arpin, D., Salinas, D., Flunkert, V., Gasthaus, J., Stella, L., & Vazquez, P. (2018). Now available in amazon sagemaker: Deepar algorithm for more accurate time series forecasting. https://aws.amazon.com/ko/blogs/machine-learning/now-available-in-amazon-sagemaker-deepar-algorithm-for-more-accurate-time-series-forecasting/.

Januschowski, T., Gasthaus, J., Wang, Y., Salinas, D., Flunkert, V., Bohlke-Schneider, M., & Callot, L. (2020). Criteria for classifying forecasting methods. *International Journal of Forecasting*, *36*(1), 167–177. http://dx.doi.org/10.1016/j.ijforecast.2019.05.008, https://www.sciencedirect.com/science/article/pii/S0169207019301529, M4 Competition.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Vol. 30, Advances in neural information processing systems*. Curran Associates, Inc., https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In Y. Bengio, & Y. LeCun (Eds.), *3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, conference track proceedings*. http://arxiv.org/abs/1412.6980.

Kourentzes, N. (2013). Intermittent demand forecasts with neural networks. *International Journal of Production Economics*, *143*(1), 198–206. http://dx.doi.org/10.1016/j.ijpe.2013.01.00, https://ideas.repec.org/a/eee/proeco/v143y2013i1p198-206.html.

Kourentzes, N. (2014). On intermittent demand model optimisation and selection. *International Journal of Production Economics*, *156*, 180–190. http://dx.doi.org/10.1016/j.ijpe.2014.06.007, https://www.sciencedirect.com/science/article/pii/S092552731400190X.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., & Yan, X. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Advances in Neural Information Processing Systems*, *32*, 5243–5253.

Loshchilov, I., & Hutter, F. (2017). SGDR: stochastic gradient descent with warm restarts. In *5th international conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, conference track proceedings*. OpenReview.net, https://openreview.net/forum?id=Skq89Scxx.

Muhaimin, A., Prastyo, D. D., & Horng-Shing Lu, H. (2021). Forecasting with recurrent neural network in intermittent demand data. In *2021 11th international conference on cloud computing, data science engineering (confluence)* (pp. 802–809). http://dx.doi.org/10.1109/Confluence51648.2021.9376880.

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. In *International conference on learning representations*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., ... Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems 32* (pp. 8024–8035). Curran Associates, Inc., http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Rasul, K. (2021). PytorchTS. https://github.com/zalandoresearch/pytorch-ts.

Rasul, K., Sheikh, A.-S., Schuster, I., Bergmann, U., & Vollgraf, R. (2021). Multi-variate probabilistic time series forecasting via conditioned normalizing flows. *Proceedings of ICLR*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252. http://dx.doi.org/10.1007/s11263-015-0816-y.

Salinas, D., Bohlke-Schneider, M., Callot, L., Medico, R., & Gasthaus, J. (2019a). High-dimensional multivariate forecasting with low-rank Gaussian Copula Processes. In *Thirty-fourth conference on neural information processing systems* (p. 6824).

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2019b). Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*.

Seeger, M. W., Salinas, D., & Flunkert, V. (2016). Bayesian Intermittent demand forecasting for large inventories. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Vol. 29, Advances in neural information processing systems*. Curran Associates, Inc., https://proceedings.neurips.cc/paper/2016/file/03255088ed63354a54e0e5ed957e9008-Paper.pdf.

Sen, R., Yu, H.-F., & Dhillon, I. S. (2019). Think globally, act locally: A deep neural network approach to high-dimensional time series forecasting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, & R. Garnett (Eds.), *Vol. 32, Advances in neural information processing systems* (pp. 4837–4846). Curran Associates, Inc., https://proceedings.neurips.cc/paper/2019/file/3a0844cee4fcf57de0c71e9ad3035478-Paper.pdf.

Spiliotis, E., Makridakis, S., & Assimakopoulos, V. (2021). The M5 accuracy competition: Results, findings and conclusions. *International Journal of Forecasting*, in press.

Trindade, A. (2015). Electricityloaddiagrams20112014 data set. https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014.

Turkmen, A. C., Januschowski, T., Wang, Y., & Cemgil, A. T. (2020). Intermittent demand forecasting with renewal processes. arXiv preprint arXiv:2010.01550.

Tweedie, M. (1984). An index which distinguishes between some important exponential families. *Statistics: Applications and New Directions*.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. Arxiv https://arxiv.org/abs/1609.03499.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. https://arxiv.org/pdf/1706.03762.pdf.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., & Bowman, S. R. (2019a). GLUE: A multi-task benchmark and analysis platform for natural language understanding. In the Proceedings of ICLR..

Wang, Y., Smola, A., Maddix, D., Gasthaus, J., Foster, D., & Januschowski, T. (2019b). Deep factors for forecasting. In K. Chaudhuri, & R. Salakhutdinov (Eds.), *Proceedings of machine learning research*: *Vol. 97*, *Proceedings of the 36th international conference on machine learning* (pp. 6607–6617). Long Beach, California, USA: PMLR, http://proceedings.mlr.press/v97/wang19k.html.

Willemain, T. R., Smart, C. N., Shockor, J. H., & DeSautels, P. A. (1994). Forecasting intermittent demand in manufacturing: a comparative evaluation of Croston's method. *International Journal of Forecasting*, *10*(4), 529–538. http://dx.doi.org/10.1016/0169-2070(94)90021-3, https://www.sciencedirect.com/science/article/pii/0169207094900213.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270–280. http://dx.doi.org/10.1162/neco.1989.1.2.270.

Wu, N., Green, B., Ben, X., & O'Banion, S. (2020). Deep transformer models for time series forecasting: The influenza prevalence case. arXiv preprint arXiv:2001.08317.

Zhou, H., Qian, W., & Yang, Y. (2020). Tweedie gradient boosting for extremely unbalanced zero-inflated data. *Communications in Statistics. Simulation and Computation*, 1–23. http://dx.doi.org/10.1080/03610918.2020.1772302.