Probabilistic Forecasting with Temporal Convolutional Neural Network

Yitian Chen[a], Yanfei Kang[b,*], Yixiong Chen[c], Zizhuo Wang[d]

[a]*Bigo Beijing R&D Center, Bigo Inc., Beijing 100191, China.*
[b]*School of Economics and Management, Beihang University, Beijing 100191, China.*
[c]*IBM China CIC, KIC Technology Center, Shanghai 200433, China.*
[d]*Department of Industrial and Systems Engineering, University of Minnesota, Minneapolis, MN 55455.*

Abstract

We present a probabilistic forecasting framework based on convolutional neural network for multiple related time series forecasting. The framework can be applied to estimate probability density under both parametric and non-parametric settings. More specifically, stacked residual blocks based on dilated causal convolutional nets are constructed to capture the temporal dependencies of the series. Combined with representation learning, our approach is able to learn complex patterns such as seasonality, holiday effects within and across series, and to leverage those patterns for more accurate forecasts, especially when historical data is sparse or unavailable. Extensive empirical studies are performed on several real-world datasets, including datasets from JD.com, China's largest online retailer. The results show that our framework outperforms other state-of-the-art methods in both accuracy and efficiency.

Keywords: Neural network, Dilated causal convolution, Probabilistic forecasting

1. Introduction

Time series forecasting plays a key role in many business decision-making scenarios, such as managing limited resources, optimizing operational processes, among others. Most existing forecasting methods focus on point forecasting, i.e., forecasting the conditional mean or median of future observations. However, probabilistic forecasting becomes increasingly important as it is able to extract richer information from historical data and better capture the uncertainty of the future. In retail business, probabilistic forecasting of product supply and demand is fundamental for successful procurement process and optimal inventory planning. Also, probabilistic shipment forecasting, i.e., generating probability distributions of the delivery volumes of packages, is the key component of the consequent logistics operations, such as labor resource planning and delivery vehicle deployment.

[*] Corresponding author

Email addresses: `chenyitian@bigo.sg` (Yitian Chen), `yanfeikang@buaa.edu.cn` (Yanfei Kang), `chenyixiong516@msn.com` (Yixiong Chen), `zwang@umn.edu` (Zizhuo Wang)

In such circumstances, instead of predicting individual or a small number of time series, one needs to predict thousands or millions of related series. Real-world applications are far more complicated. For instance, new products emerge weekly on retail platforms. Forecasting the demand of products without historical shopping festival data (e.g., Black Friday in North America, "11.11" shopping festival in China) is another challenge. Furthermore, forecasting often requires the consideration of exogenous variables that have significant influence on future demand (e.g., promotion plans provided by operations teams, accurate weather forecasts for brick and mortar retailers). Such forecasting problems can be extended to a variety of domains. Examples include forecasting the web traffic for internet companies kaggle (2017), the energy consumption for individual households, the load for servers in a data center Flunkert et al. (2017) and traffic flows in transportation domain Lv et al. (2015).

Classical forecasting methods, such as ARIMA Box et al. (2015) and exponential smoothing Hyndman et al. (2008), are widely employed for univariate base-level forecasting. To incorporate exogenous covariates, several extensions of these methods have been proposed, such as ARIMAX and dynamic regression models Hyndman and Athanasopoulos (2018). These models are well-suited for applications in which the structure of the data is well understood and there is sufficient historical data. However, working with thousands or millions of series requires prohibitive labor and computing resources for parameter estimation. Moreover, they are not applicable in situations where historical data is sparse or unavailable.

Recurrent neural network (RNN) Graves (2013) and the sequence to sequence (Seq2Seq) framework Cho et al. (2014); Sutskever et al. (2014) have achieved great success in many different sequential tasks such as machine translation Sutskever et al. (2014), language modeling Mikolov et al. (2010) and recently found applications in the field of time series forecasting Laptev et al. (2017); Wen et al. (2017); Flunkert et al. (2017); Rangapuram et al. (2018). For example, in the forecasting competitions community, a gated recurrent unit (GRU) Cho et al. (2014) based Seq2Seq model won the Kaggle web traffic forecasting competition Suilin (2017). A hybrid model that combines exponential smoothing method and RNN won the M4 forecasting competition, which consists of 100,000 series with different seasonal patterns (Makridakis et al., 2018). However, training with back propagation through time (BPTT) algorithm often hampers efficient computation. In addition, training RNN can be remarkably difficult Werbos (1990); Pascanu et al. (2013). Dilated causal convolutional architectures, e.g., Wavenet Van Den Oord et al. (2016), offer an alternative for modeling sequential data. By stacking layers of dilated casual convolutional nets, receptive fields can be increased, and the long-term correlations can be captured without violating the temporal orders. In addition, in dilated causal convolutional architectures, the training process can be performed in parallel, which guarantees computation efficiency.

Most Seq2Seq frameworks or Wavenet Van Den Oord et al. (2016) are autoregressive generative models that factorize the joint distribution as a product of the conditionals. In this setting, a one-step-ahead prediction approach is adopted, i.e., first a prediction is generated by using the past observations, and the generated result is then fed back as the ground truth to make further forecasts. More recent research shows that non-autoregressive approaches or direct prediction strategy, predicting observations of all time steps directly, can achieve better performances Gu et al. (2017); Bai et al. (2018); Wen et al. (2017). In particular, Non-autoregressive models are more robust to mis-specification by avoiding error accumulation and thus yield better prediction accuracy. Moreover, training over all the prediction horizons can be parallelized.

Having reviewing all these challenges and developments, in this paper, we propose the deep temporal convolutional network (DeepTCN), a non-autoregressive probabilistic forecasting framework for large collections of related time series.

The main contributions of the paper are as follows:

- We propose a convolutional-based forecasting framework that provides both parametric and non-parametric approaches for probability density estimation.
- The framework, being able to learn latent correlation among series and handle complex real-world forecasting situations such as data sparsity and cold starts, shows high scalability and extensibility.
- Extensive empirical studies show our framework outperforms other state-of-the-art methods on both point forecasting and probabilistic forecasting.
- Compared to recurrent architectures, the computation of convolutional models can be fully parallelized and thus high training efficiency can be achieved. Meanwhile, the optimization is much easier. In our cases, the training time is up to 1/8 of that of the recurrent models reported in the literature Flunkert et al. (2017).
- The model is very flexible and can include exogenous covariates such as an additional promotion plan or weather forecasts.

The rest of this paper is organized as follows. Section 2 provides a brief review of related works on time series forecasting and deep learning methods for forecasting. In Section 3, we describe the proposed forecasting method, including the probabilistic forecasting framework, the neural network architectures and the input features. We demonstrate the superiority of the proposed approach via extensive experimental results in Section 4 and conclude the paper in Section 5.

2. Related Work

Earlier studies on time series forecasting are mostly based on statistical models, which are mainly generative models based on state space framework such as exponential smoothing, ARMA models, their integrated versions (ARIMA) and several other extensions. For these methods, Hyndman et al. (2008) and Box et al. (2015) provide a comprehensive overview in the context of univariate forecasting.

In recent years, large quantities of related series are emerging in the routine functioning of many companies. Not sharing information from other time series, traditional univariate forecasting methods fit a model for each individual time series and thus can not learn across similar time series. Therefore, methods that can provide forecasting on multiple series jointly have received increasing attention in the last few years Yu et al. (2016).

Both RNNs and CNNs have been shown to be able to model complex nonlinear feature interactions and yield substantial forecasting performances, especially when many related time series are available Smyl (2016); Bandara et al. (2017); Laptev et al. (2017); Wen et al. (2017); Flunkert et al. (2017); Rangapuram et al. (2018). For example, Long Short-Term Memory (LSTM), one type of RNN architecture, won the CIF2016 forecasting competition for monthly time series Stepnicka and Burda (2016). Bianchi et al. (2017) compare a variety of RNNs in their performances in the Short Term Load Forecasting problem. Borovykh et al. (2017) investigate the application of CNNs to financial time series forecasting.

To better understand the uncertainty of the future, probabilistic forecasting with deep learning models has attracted increasing attention. DeepAR Flunkert et al. (2017), which trains an auto-regressive RNN model on a rich collection of similar time series, produces more accurate probabilistic forecasts on several real-world data sets. The deep state space models (DeepState), presented by Rangapuram et al. (2018), combine state space models with deep learning and can retain data efficiency and interpretability while learning the complex patterns from raw data. Under a similar scheme, Maddix et al. (2018) propose the combination of deep neural networks and Gaussian Process.

Most of these probabilistic forecasting frameworks are autoregressive models, which uses recursive strategy to generate multi-step forecasts. In neural machine translation, non-autoregressive translation (NAT) models have achieved significantly inference spee-dup at the cost of slightly inferior accuracy compared to autoregressive translation models Gu et al. (2017). Bai et al. (2018) propose a non-autore

gressive framework based on dilated causal convolution and the empirical study on multiple datasets shows the framework outperforms generic recurrent architectures such as LSTMs and GRUs. In forecasting applications, non-autoregressive approaches have also been shown to be less biased and more robust. Recently, Wen et al. (2017) present a multi-horizon quantile recurrent forecaster to combine sequential neural nets and quantile regression Koenker and Bassett Jr (1978). By training on all time points at the same time, their framework can significantly improve the training stability and the forecasting performances of recurrent nets.

Our method differs from the aforementioned approaches in the following ways. Firstly, instead of applying gating mechanism used in Wavenet Van Den Oord et al. (2016), residual blocks are applied to stabilize the training of the network and help achieve superior forecasting accuracy. Inspired by the models such as ARIMAX, a novel decoder based on a variant of the residual neural network is designed to incorporate information from past observations and exogenous covariates. Finally, our model enjoys the flexibility to embrace a variety of probability density estimation approaches. We demonstrate that our method indeed has the potential to solve those more challenging forecast tasks with great efficiency.

3. Method

We start by describing a general probabilistic forecasting problem for multiple related time series. Given a set of time series $\{y^{(i)}\}_{i=1}^{N}$, where N is the number of the series, the goal is to model the conditional distribution of the future time series $y^{(i)}_{(t+1):(t+\Omega)}$ for each $i = 1, \ldots, N$:

$$P\left(y^{(i)}_{(t+1):(t+\Omega)} | y^{(i)}_{1:t}, X^{(i)}_{1:t}, X^{(i)}_{(t+1):(t+\Omega)}\right), \tag{1}$$

where Ω denotes the length of the forecasting horizon; $y^{(i)}_{1:t}$ are the historical observations of the ith series; $X^{(i)}_{1:t}$ is a set of covariate vectors which can be static (e.g., product id) or time-varying (e.g., price of the product or promotion information); $X^{(i)}_{(t+1):(t+\Omega)}$ are covariates representing the corresponding information about the future. Under the Seq2Seq framework Cho et al. (2014); Sutskever et al. (2014), the input sequences including $y^{(i)}_{1:t}$ and $X^{(i)}_{1:t}$ can be encoded into latent variables $h^{(i)}_t$, and hence the conditional distribution of future observations $y^{(i)}_{(t+1):(t+\Omega)}$ can be reformulated by using direct prediction strategy:

$$\prod_{\omega=1}^{\Omega} P(y^{(i)}_{t+\omega} | h^{(i)}_t, X^{(i)}_{t+\omega}). \tag{2}$$

In the following sections, we describe the probabilistic forecasting framework, the neural network architecture, and some practical considerations of input features.

3.1. Probabilistic forecasting framework

We consider two probabilistic forecasting frameworks in this paper. The first one is a parametric framework, in which probabilistic forecasts of future observations can be achieved by directly predicting the parameters (e.g., the mean and the standard deviation for Gaussian distribution) of the hypothetical distribution based on maximum likelihood estimation. The second one is non-parametric, which produces a set of forecasts corresponding to quantile points of interest Koenker and Bassett Jr (1978).

Neural networks enjoy the flexibility to produce multiple outputs for each future observation: $Z = (z^1, \ldots, z^m)$, where Z represents the parameter set of the hypothetical distribution for the parametric framework, and the quantile forecasts for the non-para- metric framework.

In practice, whether to choose the parametric approach or the non-parametric approach depends on the application context. The parametric approach requires the assumption of a specific probability distribution while the non-parametric approach is distribution-free and thus is usually more robust. However, a decision-making scenario may rely on the sum of probabilistic forecasts for a certain period. For example, an inventory replenishment decision may depend on the distribution of the sum of demand for the next few days. In such cases, the non-parametric approach will not work since the output (e.g., the quantiles) is not additive over time and the parametric approach will have its advantage of being flexible in obtaining such information by sampling from the estimated distributions.

3.1.1. Non-parametric approach

In the non-parametric framework, the set of forecasts can be obtained by quantile regression. In quantile regression Koenker and Bassett Jr (1978), denoting the observation and the prediction for a specific quantile level q as y and $\hat{y}^q$ respectively, models are trained to minimized the quantile loss :

$$\boldsymbol{L}_q(y, \hat{y}^q) = q(y - \hat{y}^q)^+ + (1 - q)(\hat{y}^q - y)^+,$$

where $(y)^+ = \max(0, y)$ and $q \in [0, 1]$. Given a set of quantile levels $Q = (q_1, ..., q_m)$, the m corresponding forecasts can be obtained by minimizing the total quantile loss:

$$L_Q = \sum_{j=1}^{m} \boldsymbol{L}_{q_j} \left(y, \hat{y}^{q_j}\right).$$

3.1.2. Parametric approach

For the parametric approach, given the predetermined distribution (e.g., Gaussian distribution), the maximum likelihood estimation is applied to estimate the corresponding parameters. Take Gaussian distribution as an example: for each target value y, the network outputs the parameters of the distribution, namely the mean and the standard deviation, denoted by μ and σ, respectively. The negative log-likelihood function is then constructed as the loss function:

$$\begin{aligned} L_G &= -\log \ell(\mu, \sigma \,|y) \\ &= -\log \left((2\pi\sigma^2)^{-1/2} \exp \left[-(y-\mu)^2/(2\sigma^2) \right] \right) \\ &= \frac{1}{2}\log(2\pi) + \log(\sigma) + \frac{(y-\mu)^2}{2\sigma^2}. \end{aligned}$$

We can extend this approach to a variety of probability distribution families. For example, we can choose negative-binomial distribution for long-tail products.

It is worth mentioning that some parameters of a certain distribution (e.g., σ in Gaussian distribution) must satisfy the condition of positivity. To accomplish this, we apply "Soft ReLU" activation, namely the transformation $\hat{z} = \log(1 + \exp(z))$, to ensure positivity Flunkert et al. (2017).

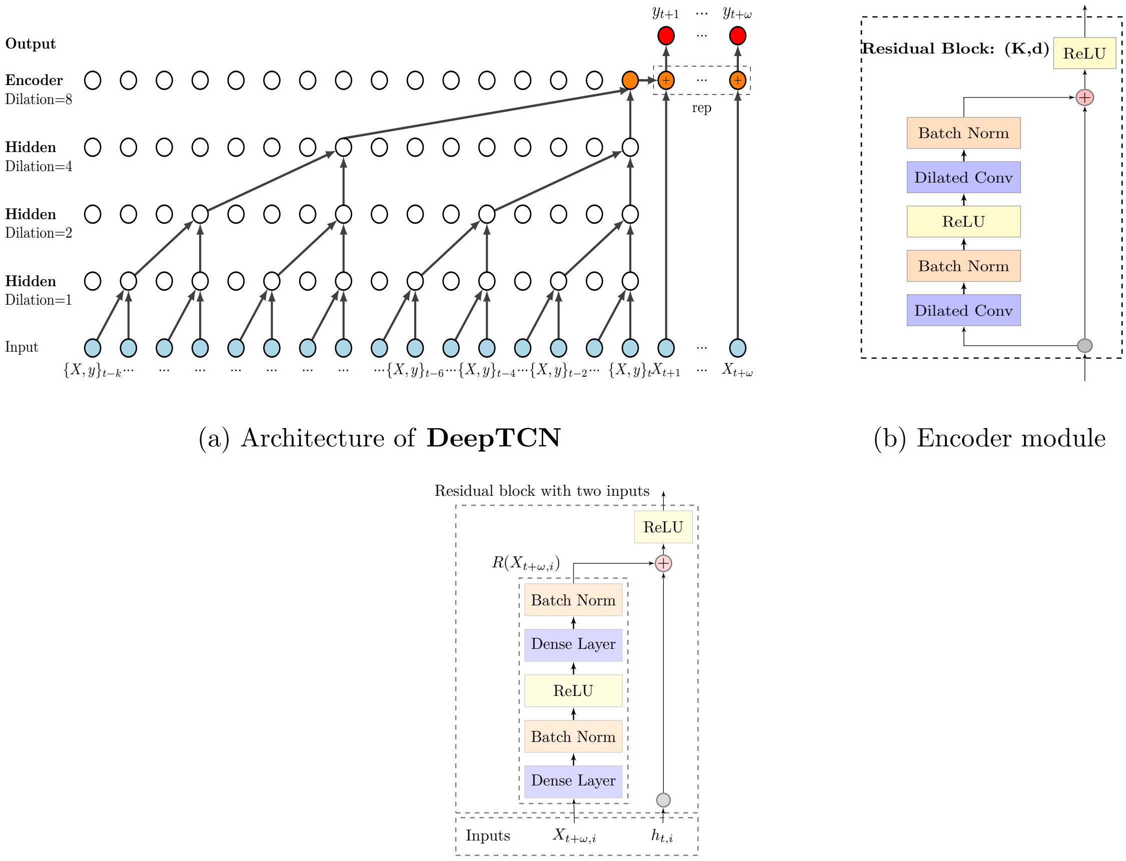
3.2. Neural network architecture

The architecture of DeepTCN is illustrated in Figure 1a. The high-level architecture is similar to the classical Seq2Seq framework. For the encoder, stacked dilated causal convolutions are constructed to capture the temporal dependencies. For the decoder, a variant of residual block (a block with two inputs) is applied instead of original RNN or dilated causal convolutions. The decoder is designed in such a way for two reasons: 1) such a framework can naturally cooperate two parts of inputs: the outputs of encoder and the future covariates; 2) from the perspective of time series modeling, a future observation can be considered to be composed of an auto-correlation component determined by past covariates and a nonlinear component determined by the future knowledge. In other words, the residuals between the future observations and predictions solely determined by the historical covariates can be explained as the function of future covariates. And a variant of residual block naturally captures such relationships between these two inputs.

3.2.1. Encoder: Dilated causal convolutions

Causal convolutions are convolutions where an output at time t can be only obtained from inputs that are no later than t. Dilation causal convolutions allow the filter to be applied over an area larger than its length by skipping input values with a certain step Van Den Oord et al. (2016). In the case of univariate series, given a 1-D input sequence x, the output (feature map) s at location t of a dilated convolution with kernel w can be expressed as:

$$s(t) = (x *_d w)(t) = \sum_{k=0}^{K-1} w(k) x(t - d \cdot k), \tag{3}$$



(a) Architecture of **DeepTCN**

(b) Encoder module

(c) Decoder module

Figure 1: (a) Architecture of DeepTCN. Encoder part: stacked dilated causal convolutions are constructed to capture the long-term temporal dependencies; Decoder part: a variant of residual block is designed to cooperate both historical covariates and future covariates. (b) Ingredient for each layer of encoder, a residual module based on dilated causal convolutions. (c) Decoder module: $h_t^{(i)}$ is the output of encoder, $X_{t+\omega}^{(i)}$ are the future covariates. R is the nonlinear function applied on $X_{t+\omega}^{(i)}$.

where d is the dilation factor, and K is the size of the kernel. Stacking multiple dilated convolutions enable networks to have very large receptive fields and to capture long-range temporal dependencies with a smaller number of layers. The left part of Figure 1a is an example of dilated casual convolutions with dilation factors $d = \{1, 2, 4, 8\}$, where the filter size $K = 2$ and a receptive field of size 16 is reached by staking four layers.

Figure 1b shows the basic module for each layer of the encoder, where both of two dilated convolutions inside the module have the same kernel size K and dilation factor d. Instead of implementing the classical gating mechanism in Wavenet Van Den Oord et al. (2016), in which a dilated convolution is followed by a gating activation, residual blocks are taken as the ingredient. As shown in Figure 1b, each residual block consists of two layers of dilated causal convolutions, first of which is followed by a batch normalization and rectified nonlinear unit (ReLU) Nair and Hinton (2010) and second of which is followed by another batch normalization Ioffe and Szegedy (2015). The output after the second batch normalization layer is added to the input of the residual block and the addition is then followed by a second ReLU. Residual blocks have be proven to help efficient training and stabilize the network, especially when the input sequence is very long. More importantly, non-linearity gained by rectified linear unit (ReLU) achieves better prediction accuracy in our most of forecasting empirical study. Similar conclusions can also be found in various NLP tasks Bai et al. (2018).

3.2.2. Decoder: Residual neural network

Figure 1c shows the structure of the decoder. $X^{(i)}_{t+1:t+\omega}$ are the future covariates and $h^{(i)}_t$ is the latent variable output by the encoder. R is the residual function applied on $X^{(i)}_{t+1:t+\omega}$ to explain the residuals between ground truth and predictions solely determined by the encoder part. For the residual function $R(\cdot)$, we first apply a dense layer and a batch normalization to project the future covariates. Then a ReLU activation is applied followed by another dense layer and batch normalization. Such a decoder also enjoys the flexibility to include additional features (e.g., a promotion plans provided by operation teams or weather forecast for brick and mortar retailers). In the end, the decoder part produces the final output Z that corresponds to the probabilistic estimation of interest.

3.3. Input features

There are typically two kinds of input features: time-dependent features (e.g., product price, a set of dummy variables like day-of-the-week) and time-independent features (e.g., product_id, product brand, category etc). Time-independent covariates such as product_id contain series-specific information. Including these covariates help capture the scale level and seasonality for

	`JD-demand`	`JD-shipment`	`electricity`	`traffic`	`parts`
Number	50,000	1,450	370	963	1,406
Length	$[0, 1800]$	$[0, 1800]$	26,304	10,560	51
Domain	$\mathbb{N}$	$\mathbb{N}$	$\mathbb{R}^+$	$[0, 1]$	$\mathbb{N}$
Granularity	daily	daily	hourly	hourly	monthly

Table 1: Summary of the datasets used in the experiments.

each specific series.

To capture seasonality, we use hour-of-the-day, day-of-the-week, day-of-the-month for hourly data, day-of-the-year for daily data and month-of-year for monthly data. Besides, we use hand-crafted holiday indicators for shopping festival such as “11.11”, which enable the model to learn planned event spikes.

Dummy variables such as product_id and day-of-the-week are mapped to dense numeric vectors via embedding Mikolov, Sutskever, Chen, Corrado and Dean (2013); Mikolov, Chen, Corrado and Dean (2013). We find that the model is able to learn more similar patterns across series by representation learning and thus improve the forecasting accuracy for related time series, which is especially useful for series with little or without historical data. In the case of new products or new warehouses without sufficient historical data, we perform zero padding to ensure the desired length of the input sequence.

4. Experiments

In this section, we perform empirical studies on five datasets. The information of the datasets is given in Table 1. `JD-demand` and `JD-shipment` are two datasets from JD.com, which correspond to two forecasting tasks for online retailers, demand forecasting of regional product sales and shipment forecasting of the daily delivery volume of packages for retailers' warehouses. Since it is inevitable for new products or warehouses to emerge, the training periods for these two datasets can range from zero to several years and the corresponding forecasting tasks involve situations such as cold-starts and data sparsity. `Electricity` [1], `traffic` [2] and `parts` [3] are three public datasets which have been widely used in various time series forecasting evaluation studies. A more detailed description of these datasets can be found in Appendix A.

The baseline methods evaluated on JD.com's datasets are presented in Section 4.1. For

[1] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

[2] https://archive.ics.uci.edu/ml/datasets/PEMS-SF

[3] http://www.exponentialsmoothing.net/supplements#data

public datasets, the proposed DeepTCN framework is compared with published state-of-the-art methods.

4.1. Experimental settings

4.1.1. Baselines

Current baseline models for JD.com's datasets include JD-online, seasonal ARIMA (SARIMA) and XGBoost.These models are deployed and continuously improved to provide more accurate forecasts and to better serve the consequent business operations. More detailed description including feature lists, parameters can be found Appendix B.

- SARIMA: Seasonal ARIMA (SARIMA) is a widely used time series forecasting model which extends the ARIMA model by including additional seasonal term and is capable of modeling seasonal behaviors from the data Box et al. (2015).

- XGBoost: Gradient boosting tree method has been empirically proven to be a highly effective approach in predictive modeling. As one of efficient implementation of the gradient boosting tree algorithm, XGBoost has gained popularity of being the winning algorithm in numerous machine learning competitions, like Kaggle Competition Chen and Guestrin (2016).

- JD-online: JD-online is the current model used in production which produces probabilistic forecasts by combining results from time series models such as SARIMA and results inferred from the residuals between point forecasts of machine learning models and ground truth.

4.1.2. Evaluation metrics

The evaluation metrics used in our experiments for point forecasting include Symmetric Mean Absolute Percent Error (SMAPE), Root Mean Squared Logarithmic Error (RMLSE), Normalized Deviation (ND) and Normalized RMSE (NRMSE). These metrics are defined as follows:

$$
\begin{aligned}
SMAPE &= \frac{1}{N}\sum\left|\frac{2\left(y_t^{(i)}-\hat{y}_t^{(i)}\right)}{y_t^{(i)}+\hat{y}_t^{(i)}}\right| \\
RMLSE &= \sqrt{\frac{1}{N}\sum\left(\log\left(y_t^{(i)}+1\right)-\log\left(\hat{y}_t^{(i)}+1\right)\right)^2} \\
ND &= \frac{\sum_{i,t}\left|y_t^{(i)}-\hat{y}_t^{(i)}\right|}{\sum_{i,t}\left|y_t^{(i)}\right|} \\
NRMSE &= \frac{\sqrt{\frac{1}{N}\sum_{i,t}\left(y_t^{(i)}-\hat{y}_t^{(i)}\right)^2}}{\frac{1}{N}\sum_{i,t}\left|y_t^{(i)}\right|}
\end{aligned}
$$

where $y_t^{(i)}$ is the true value of series i at time step t, $\hat{y}_t^{(i)}$ is the corresponding prediction value and N is the number of all points in the testing periods.

For the evaluation of probabilistic forecasting, given a set of time series $\boldsymbol{y}$ and corresponding predictions $\hat{\boldsymbol{y}}$, we use ρ-quantile loss, $\rho \in (0,1)$:

$$
QL_\rho(\boldsymbol{y},\hat{\boldsymbol{y}}) = 2\frac{\sum_{i,t} P_\rho(y_t^{(i)},\hat{y}_t^{(i)})}{\sum_{i,t}|y_t^{(i)}|},
$$

where

$$
P_\rho(y,\hat{y}) = \begin{cases} \rho(y-\hat{y}) & \text{if } y > \hat{y}, \\ (1-\rho)(\hat{y}-y) & \text{otherwise.} \end{cases}
$$

4.2. Results on JD.com's datasets

4.2.1. Accuracy comparison

We begin with comparing the probabilistic forecasting results of `DeepTCN` against `JD-online` over two testing periods: Oct 2018 and Nov 2018. In particular, China's largest shopping festival "11.11" lasts from November 1 to November 12, during which November 11 is the biggest promotion day. we choose the standard ρ50 and ρ90-quantile losses as the evaluation

Method	`JD-demand`		`JD-shipment`	
	Oct 2018	Nov 2018	Oct 2018	Nov 2018
`JD-online`	0.719/0.592	0.764/0.958	0.270/0.169	0.388/0.258
`TCN-Quantile`	**0.653/0.528**	**0.698/0.701**	**0.173/0.100**	**0.247/0.160**
`TCN-Gaussian`	0.697/0.588	0.720/0.873	0.188/0.105	0.326/0.219

Table 2: Comparison of probabilistic forecasts on `JD-demand` and `JD-shipment` datasets. The quantile losses $\rho50/\rho90$ are evaluated against online models over two testing periods – Oct 2018 and Nov 2018.

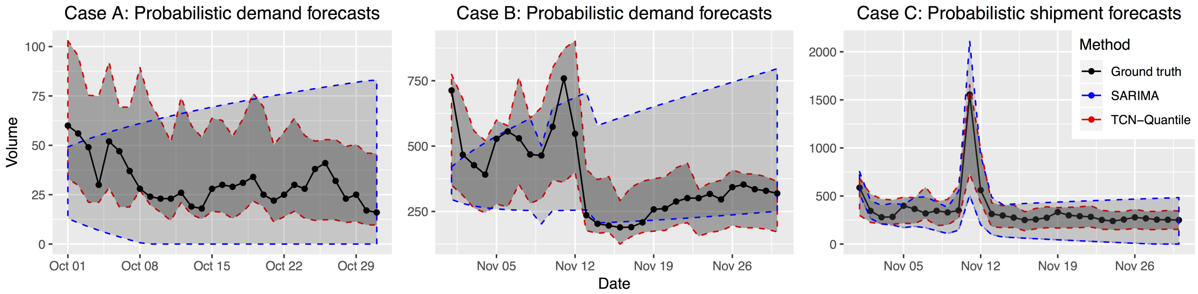


Figure 2: Probabilistic forecasts of `SARIMA` and `TCN-Quantile` for three cases (randomly chosen for illustration purposes). Case A and Case B show the forecasting results of two fast-moving products; Case C shows the forecasting results of the daily delivery volume of packages from one warehouse. The ground truth, and the [10%, 90%] prediction intervals of `SARIMA` and `TCN-Quantile` are shown in different colors.

metrics. We consider, within the `DeepTCN` framework, two models for probabilistic forecasting , the non-parametric model which predicts the quantiles and Gaussian likelihood model (we refer to them as `TCN-Quantile` and `TCN-Gaussian`, respectively, for the rest of the paper). More specifically, `TCN-Quantile` is trained to predict ρ-quantiles with $\rho \in \{0.1, 0.5, 0.9\}$, and `TCN-Gaussian` estimates the mean and standard deviation for each future observation. The quantiles of `TCN-Gaussian` are obtained by calculating the percent point function of Gaussian distribution (the inverse of cumulative density function) at 0.5 and 0.9 quantile points.

The comparison results of `JD-demand` and `JD-shipment` are illustrated in Table 2. As we can see, both `TCN-Quantile` and `TCN-Gaussian` perform better than online results. In particular, `TCN-Quantile` performs the best. There are two possible reasons for that. First, `TCN-Gaussian` is constructed based on the gaussian likelihood but `JD-demand` dataset does not necessarily follow the assumption of normal distribution. Second, `TCN-Quantile`, in light of the distribution-free nature, generates the quantile forecasts by minimizing the quantile loss functions which correspond to our evaluation metrics directly.

4.2.2. Uncertainty estimation

In Figure 2, we show three cases of probabilistic forecasts generated by `SARIMA` and `TCN-Quantile`. Case A and case B are two demand forecasting examples of Oct 2018 and Nov 2018, respectively, while case C is an example of shipment forecasting of Nov 2018. It is shown that for tasks of both `JD-demand` and `JD-shipment`, `TCN-Quantile` generates more accurate uncertainty estimation. Moreover, `SARIMA` postulates increasing uncertainty over time while the uncertainty estimation of `DeepTCN` is learned from the data. For example, the uncertainty during the shopping festival period is huge due to both promotion activities and intense market competitions.

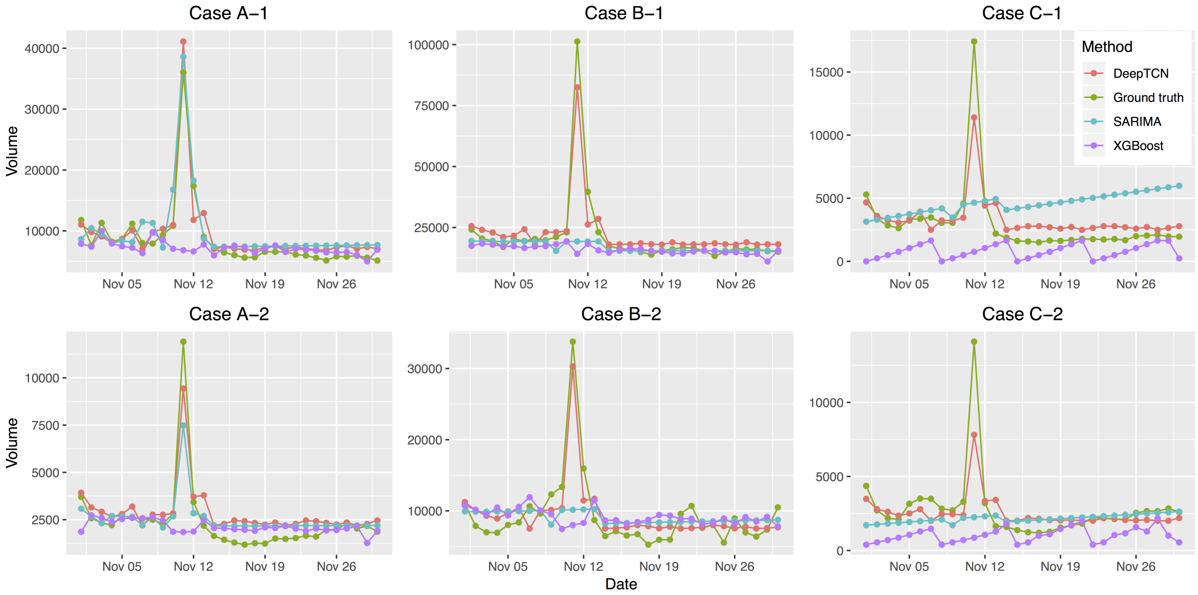


Figure 3: Point forecasts of `DeepTCN`, `SARIMA` and `XGBoost` for six cases (randomly chosen from `JD-shipment` for illustration purposes). Cases A-1 and A-2 are examples with historical data of more than two years; cases B-1 and B-2 show instances without previous shopping festival data; cases C-1 and C-2 illustrate cold-start forecasting namely the forecasting of time series with little historical data, e.g., less than three days. Note that Nov 11 is one of China's biggest promotion days.

Data-group	Method	SMAPE	RMSLE
`All-data`	`SARIMA`	0.369	0.789
	`XGBoost`	0.430	0.820
	`DeepTCN`	**0.284**	**0.497**
`Group-1`	`SARIMA`	0.323	0.644
	`XGBoost`	0.312	0.630
	`DeepTCN`	**0.268**	**0.460**
`Group-2`	`SARIMA`	0.430	0.832
	`XGBoost`	0.457	0.967
	`DeepTCN`	**0.354**	**0.532**

Table 3: Point forecasting accuracy comparison on SMAPE and RMSLE of different subgroups of `JD-shipment` in Nov. 2018. `All-Data` represents all series with the length of training periods ranging from zero to four years; `Group-1` includes the warehouses with historical data of more than two years; `Group-2` indicates series starting after 2018-01-01, namely those with no historical shopping festival data.

4.2.3. Data sparsity

Next, we perform a qualitative analysis on JD-shipment dataset over the testing period of November, for the purpose of gaining a deeper understanding of the performance improvement exhibited by DeepTCN, as compared with other baseline models. We choose this data because 1) it consists of series whose magnitudes of volume are high and stable, and 2) The testing period involves China's biggest shopping festival "11.11". As mentioned before, the occurrence of this festival will result in a spike for the shipment volume and make the forecasting task more challenging.

We first present in Table 3 an accuracy comparison of point forecasting between our model and other two baseline models including SARIMA and XGBoost. The point forecasting results of DeepTCN is achieved by directly predicting the 0.5 quantiles. All-Data consists of all series in the dataset; Group-1 includes series with historical data longer than two years; Group-2 is chosen as those series starting after 2018-01-01. We can see from Table 3 that DeepTCN achieves consistently the best accuracy with regard to both metrics across all data groups. In particular, when historical shopping festival data is not available, the performance of SARIMA and XGBoost became much worse (the result of Group-2), while DeepTCN maintains the same performance level.

In Figure 3, we illustrate cases of point forecasting under three different scenarios."11.11" is the major promotion day and we can observe a spike in the true volume. In cases A-1 and A-2 , where historical data of more than two years is available, all models can learn a similar volume pattern, including the spike on "11.11". However, SARIMA and XGBoost in cases B-1 and B-2 fail to capture the spike on "11.11" due to lack of sufficient training data such as historical festivals. Finally, cases C-1 and C-2 are selected to demonstrate how these models handle cold-start forecasting. It turns out that DeepTCN stands out for this situation as it is able to capture both scale and curve pattern of the new warehouses by learning data from those old warehouses with similar store-specific features.

4.3. Results on the public datasets

In this section, we evaluate our method on three public datasets – electricity, traffic and parts. The electricity dataset contains hourly time series of the electricity consumption of 370 customers; the traffic dataset is a collection of the occupancy rates (between 0 and 1) of 963 car lanes from San Francisco bay area freeways; the parts dataset is comprised of 1,046 time series representing monthly demand of spare parts at a US car company. We compare DeepTCN against MatFact Yu et al. (2016), DeepAR Flunkert et al. (2017) and DeepState Rangapuram et al. (2018), which got the strongest published results on these datasets. We also report the results of classical forecasting methods including `auto.arima` and `ets`. Both methods are implemented in **R**'s `forecast` package Hyndman and Khandakar (2008).

4.3.1. Probabilistic forecasting

We start with conducting the experiments of probabilistic forecasting. For `electricity` and `traffic` dataset, we implement a 24-hour ahead forecasting task for last seven days based on a rolling-window approach as described in Flunkert et al. (2017). It is worth noting that we use the same model trained on the data before the first prediction window rather than retraining the model after updating the forecast point. For `parts` dataset, we evaluate the performance for last 12 months. In all forecasting experiments, we train the `TCN-Quantile` models to predict ρ-quantiles with $\rho \in \{0.1, 0.5, 0.9\}$.

Table 4 illustrates the probabilistic forecasting results obtained by these models. We use the same evaluation metrics as in Rangapuram et al. (2018). For `DeepState` and `DeepAR`, we report the results obtained based on the 2-week training range, while we show the result of `DeepTCN` achieved by using one week as the training range. As shown in Table 4, the probabilistic forecasting results of `TCN-Quantile` and `TCN-Gaussian` outperform other state-of-the-art models on both `traffic` and `parts` datasets. For `electricity` dataset containing series that are not so related, `DeepState` achieves the best results and the performance of `DeepTCN` is slightly worse. We believe that models such as `DeepState` and `ES-RNN` Makridakis et al. (2018) have more advantages on situations where time series are not highly correlated as they specify

Dataset	`ets`	`auto.arima`	`DeepAR`	`DeepState`	`TCN-Quantile`	`TCN-Gaussian`
`electricity`	0.121/0.101	0.283/0.109	0.153/0.147	**0.087/0.050**	0.114/0.058	0.124/0.078
`traffic`	0.621/0.650	0.492/0.280	0.177/0.153	0.168/0.117	**0.115/0.079**	**0.141/0.097**
`parts`	1.639/1.009	1.644/1.066	1.273/1.086	1.470/0.935	**1.066/0.923**	**1.245/0.930**

Table 4: ρ50/ρ90-losses evaluation on public datasets.

Method	`electricity`		`traffic`	
	ND	NRMSE	ND	NRMSE
`MatFact`	0.25	1.40	0.19	0.42
`DeepAR`	**0.08**	**0.49**	0.27	0.56
`DeepTCN`	0.11	0.51	**0.12**	**0.36**

Table 5: Accuracy comparison of point forecasting.

unique parameters for each series.

4.3.2. Point forecasting

Table 5 reports the point forecasting results of `DeepTCN` (the quantile prediction with quantile point 0.5) compared against `MatFact` Yu et al. (2016) and `DeepAR` Flunkert et al. (2017). The results are similar with probabilistic forecasting comparison. For `traffic` dataset with highly correlated series, `DeepTCN` achieves more accurate forecasting by learning across the series and significantly outperforms the other two methods while the performance of `DeepTCN` on `electricity` dataset is slightly worse than `DeepAR`.

4.3.3. Run-time efficiency

Finally, we demonstrate in Table 6 a comparison with respect to run-time efficiency between `DeepTCN` and `DeepAR`. Running times are obtained from the measurement of an end-to-end evaluation on datasets `electricity`, `traffic` and `parts`, including processing features, training the model, and producing the corresponding results. For `DeepTCN`, we show the run-time result of `TCN-Quantile`. For `DeepAR`, we report the running time presented in Flunkert et al. (2017). Both models are trained on the same GPU service `Tesla P40`. As shown in Table 6, `DeepTCN`, due to its capability of performing the convolutions in parallel, has a clear advantage on the run-time efficiency.

5. Conclusion

We present a convolutional-based probabilistic forecasting framework for multiple related time series and show both non-parametric and parametric approaches to model the probabilistic distribution based on neural networks. Our solution can help in the design of practical large-scale forecasting applications, which involves situations such as cold-starts and data sparsity. Results from both industrial datasets and public datasets shows the framework yields superior performance compared to other state-of-the-art methods on both accuracy and efficiency.

Dataset	`DeepTCN`	`DeepAR`
`electricity`	50m	7h
`traffic`	30m	3h
`parts`	40s	5m

Table 6: Computation time comparison on public datasets.

References

References

Bai, S., Kolter, J. Z. and Koltun, V. (2018), 'An empirical evaluation of generic convolutional and recurrent networks for sequence modeling', *arXiv preprint arXiv:1803.01271* .

Bandara, K., Bergmeir, C. and Smyl, S. (2017), 'Forecasting across time series databases using recurrent neural networks on groups of similar series: A clustering approach', *arXiv preprint arXiv:1710.03222* .

Bianchi, F. M., Maiorino, E., Kampffmeyer, M. C., Rizzi, A. and Jenssen, R. (2017), 'An overview and comparative analysis of recurrent neural networks for short term load forecasting', *arXiv preprint arXiv:1705.04378* .

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

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

Chen, T. and Guestrin, C. (2016), Xgboost: A scalable tree boosting system, *in* 'Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining', ACM, pp. 785–794.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C. and Zhang, Z. (2015), 'Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems', *arXiv preprint arXiv:1512.01274* .

Cho, K., Van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014), 'Learning phrase representations using RNN encoder-decoder for statistical machine translation', *arXiv preprint arXiv:1406.1078* .

Flunkert, V., Salinas, D. and Gasthaus, J. (2017), 'Deepar: Probabilistic forecasting with autoregressive recurrent networks', *arXiv preprint arXiv:1704.04110* .

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

Gu, J., Bradbury, J., Xiong, C., Li, V. O. and Socher, R. (2017), 'Non-autoregressive neural machine translation', *arXiv preprint arXiv:1711.02281* .

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

Hyndman, R. and Khandakar, Y. (2008), 'Automatic time series forecasting: The forecast package for r', *Journal of Statistical Software, Articles* **27**(3), 1–22.

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

Ioffe, S. and Szegedy, C. (2015), Batch normalization: Accelerating deep network training by reducing internal covariate shift, *in* 'International Conference on Machine Learning', pp. 448–456.

kaggle (2017), 'Web traffic time series forecasting', `https://www.kaggle.com/c/web-traffic-time-series-forecasting`.

Koenker, R. and Bassett Jr, G. (1978), 'Regression quantiles', *Econometrica: journal of the Econometric Society* pp. 33–50.

Laptev, N., Yosinski, J., Li, L. E. and Smyl, S. (2017), Time-series extreme event forecasting with neural networks at uber, *in* 'International Conference on Machine Learning'.

Lv, Y., Duan, Y., Kang, W., Li, Z. and Wang, F.-Y. (2015), 'Traffic flow prediction with big data: a deep learning approach', *IEEE Transactions on Intelligent Transportation Systems* **16**(2), 865–873.

Maddix, D. C., Wang, Y. and Smola, A. (2018), 'Deep factors with gaussian processes for forecasting', *arXiv preprint arXiv:1812.00098* .

Makridakis, S., Spiliotis, E. and Assimakopoulos, V. (2018), 'The m4 competition: Results, findings, conclusion and way forward', *International Journal of Forecasting* .

Mikolov, T., Chen, K., Corrado, G. and Dean, J. (2013), 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781* .

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J. and Khudanpur, S. (2010), Recurrent neural network based language model, *in* 'Eleventh Annual Conference of the International Speech Communication Association'.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J. (2013), Distributed representations of words and phrases and their compositionality, *in* 'Advances in neural information processing systems', pp. 3111–3119.

Nair, V. and Hinton, G. E. (2010), Rectified linear units improve restricted boltzmann machines, *in* 'Proceedings of the 27th international conference on machine learning (ICML-10)', pp. 807–814.

Pascanu, R., Mikolov, T. and Bengio, Y. (2013), On the difficulty of training recurrent neural networks, *in* 'International Conference on Machine Learning', pp. 1310–1318.

Rangapuram, S. S., Seeger, M. W., Gasthaus, J., Stella, L., Wang, Y. and Januschowski, T. (2018), Deep state space models for time series forecasting, *in* 'Advances in Neural Information Processing Systems', pp. 7795–7804.

Smith, T. G. (2017), 'pmdarima: Arima estimators for python', `https://www.alkaline-ml.com/pmdarima/`.

Smyl, S. (2016), 'Forecasting short time series with lstm neural networks', `https://gallery.azure.ai/Tutorial/Forecasting-Short-Time-Series-with-LSTM-Neural-Networks-2`.

Stepnicka, M. and Burda, M. (2016), Computational intelligence in forecasting (cif) 2016 time series forecasting competition, *in* 'IEEE WCCI 2016, IJCNN-13 Advances in Computational Intelligence for Applied Time Series Forecasting (ACIATSF)', IEEE.

Suilin, A. (2017), '1st place solution of kaggle web traffic time series forecasting', `https://github.com/Arturus/kaggle-web-traffic`.

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

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W. and Kavukcuoglu, K. (2016), Wavenet: A generative model for raw audio., *in* 'SSW', p. 125.

Wen, R., Torkkola, K. and Narayanaswamy, B. (2017), 'A multi-horizon quantile recurrent forecaster', *arXiv preprint arXiv:1711.11053* .

Werbos, P. J. (1990), 'Backpropagation through time: what it does and how to do it', *Proceedings of the IEEE* **78**(10), 1550–1560.

Yu, H.-F., Rao, N. and Dhillon, I. S. (2016), Temporal regularized matrix factorization for high-dimensional time series prediction, *in* 'Advances in neural information processing systems', pp. 847–855.

Appendices

A. Dataset

1. `JD-demand`. The `JD-demand` dataset is a collection of 50,000 time series of regional demand which involves around 6,000 products of `3C` (short for communication, computer and consumer electronics) category from seven regions of China. The dataset is gathered from 2014-01-01 to 2018-12-01. The features set for `JD-demand` includes historical demand and the product-specific information (e.g., region_id, product categories, brand, the corresponding product price and promotions).

2. `JD-shipment`. The `JD-shipment` dataset includes about 1450 time series from 2014-10-01 to 2018-12-01, including new series (warehouses) that emerge with the development of the companies' business. The covariates consist of historical demand, the warehouse specific info including geographic and metropolitan informations (e.g., geo_region, city) and warehouse categories (e.g. food, fashion, appliances).

3. `Electricity`. The `electricity` dataset describes the series of the electricity consumption of 370 customers. The electricity usage values are recorded per 15 minutes from 2011 to 2014. We select the data of the last three years. By aggregating the records of the same hour, we finally get the hourly consumption data of size $N \times T = 370 \times 26304$, where N is the number of time series and T is the length Yu et al. (2016).

4. `Traffic`. The `traffic` dataset describes the occupancy rates (between 0 and 1) of 963 car lanes from San Francisco bay area freeways. The measurements are carried out over the period from 2008-01-01 to 2009-03-30 and are sampled every 10 minutes. The original dataset was split into training and test parts, and the daily order was shuffled. The total datasets were merged and rearranged to make sure it followed the calendar order. Hourly aggregation was applied to obtain hourly traffic data Yu et al. (2016). Finally, we get the dataset of size $N \times T = 963 \times 10560$, with the occupancy rates at each station described by a time series of length $10,560$.

5. `Parts`. The `parts` dataset includes 2,674 time series supplied by a US car company, which represents the monthly sales for slow-moving parts and covers a period of 51 months. After applying two filtering rules as follows:

- Removing series possessing fewer than ten positive monthly demands.
- Removing series having no positive demand in the first 15 and final 15 months.

There are finally 1,046 time series left and a more detailed description can be find in Hyndman et al. (2008).

B. Baselines

Forecasting in industrial applications often relies on a combination of univariate forecasting models and machine learning methods.

1. SARIMA model is applied to `JD-shipment` dataset and fast-moving products with historical data of length more than 14 in `JD-demand` dataset. The model is implemented with Python's package `pmdarima` Smith (2017) and the best parameters are automatically select based on the criterion of minimizing the AICs Hyndman and Athanasopoulos (2018). The predictions at confidence level {10%, 90%} are taken as the probabilistic forecasts in our experiments.

2. `XGBoost` is also applied to both `JD-demand` dataset and `JD-shipment` dataset. The features for forecasting on `JD-shipment` are presented in Table 7. A grid-search is used to find the best values of parameters like learning rate, the depth-of-tree based on the offline evaluation on data from both last month and the same month of last year.

3. `JD-online`. As mentioned before, the probabilistic results of `JD-online` include two parts. The results of time series models like SARIMA are presented in the previous list. Gaussian distribution assumption is taken to generate the probabilistic forecasting for machine learning models. The bagging of several models' results is taken as the mean. The standard deviation of residuals between predictions and ground truth of last month's data are taken as the forecasted deviation. These two parts are re-bagged to produce final forecasts.

C. Experiment details

The current model is implemented with `Mxnet` Chen et al. (2015) and its new high-level interface Gluon. We trained our model on a `GPU` server with one `Tesla P40` and 16 CPU (3.4 `GHz`). Multiple-GPU can be applied to speed up and achieve better training efficiency in real industrial application. The codes for pubiic datasets are released at https://github.com/oneday88/kdd2019deepTCN.

For the JD.com's datasets, the training range and prediction horizon are both 31 days. We implement two models for both JD-demand and JD-shipment datasets. One model is trained on the data before Oct 2018 and produces forecasting on Oct 2018; the other one is trained on the data before Nov 2018 and produces forecasting on Nov 2018.

For the parts dataset, we use the first 39 months as training data and the last 12 months for evaluation. A rolling window approach with window size =4 is adopted. The training and prediction range are both 12 months and a rolling window approach with window size 4 is adopted. For both electricity and traffic datasets, the training range and prediction range are selected as 168 hours and 24 hours respectively. For electricity dataset, we use only samples taken in December of 2011, 2012 and 2013 as training data, as we assume that this small data set is sufficient for the task of forecasting electricity consumption during the last seven days of December 2014. For traffic dataset, we train models on all the data before last

Table 7: XGBoost feature lists

Feature type	Details
Category	region_id, city_id, warehouse_type, holiday_indicators, is-weekend, etc.
Stats of Warehouse level	summary (mean,median) of last week and last two weeks, summary(median, SD) of last four weeks, etc .
Stats of city level	summary (mean,median) of last week and last two weeks, summary(median, SD) of last four weeks, etc .
Stats of Warehouse-type level	summary (mean,median) of last week and last two weeks, summary(median, SD) of last four weeks, etc.

Table 8: TCN parameters

	JD-demand	JD-shipment	electricity-quantile	traffic	parts
number of time series	50,000	1450	370	963	1406
input-output length	31-31	31-31	168-24	168-24	12-12
dilation-list	[1,2,4,8]	[1,2,4,8]	[1,2,4,8,16,20,32]	[1,2,4,8,16,20,32]	[1,2]
number of training samples	200k	40k	30k	26k	4k
batch size	16	512	512	128	8
learning rate	1e-2	5e-2	5e-2	1e-2	1e-4

seven days.

For each dataset, we fit the model on the training data and evaluate the corresponding metrics on the testing data after every epoch. When the training process is complete, we pick the model that gains the best evaluation results on the test set.

Convolution-related hyper-parameters, such as kernel size, number of channels and dilation length, are selected according to different tasks and datasets. The most important principle for choosing kernel size and dilation length is to make sure that the encoder (stacked residual blocks) has sufficiently large receptive field, namely long effective history of the time series. The number of channels at each convolution layer is determined by the number of input features and is kept fixed for all residual blocks. We manually tune for each dataset training-related hyper-parameters, including batch size and learning rate, in order to achieve the best performance on both evaluation metrics and running time. A more detailed description of parameters is presented in Table 8.