International Journal of Forecasting 39 (2023) 332–345

Contents lists available at ScienceDirect

International Journal of Forecasting

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

Parameter-efficient deep probabilistic forecasting

Olivier Sprangers [a,*], Sebastian Schelter [b], Maarten de Rijke [b]

[a] AIRLab, University of Amsterdam, Netherlands
[b] University of Amsterdam, Netherlands

ARTICLE INFO

Keywords:
Probabilistic forecasting
Temporal convolutional network
Efficiency in forecasting methods
Large-scale forecasting Forecasting with neural networks

ABSTRACT

Probabilistic time series forecasting is crucial in many application domains, such as retail, ecommerce, finance, and biology. With the increasing availability of large volumes of data, a number of neural architectures have been proposed for this problem. In particular, Transformer-based methods achieve state-of-the-art performance on real-world benchmarks. However, these methods require a large number of parameters to be learned, which imposes high memory requirements on the computational resources for training such models. To address this problem, we introduce a novel bidirectional temporal convolutional network that requires an order of magnitude fewer parameters than a common Transformer-based approach. Our model combines two temporal convolutional networks: the first network encodes future covariates of the time series, whereas the second network encodes past observations and covariates. We jointly estimate the parameters of an output distribution via these two networks. Experiments on four real-world datasets show that our method performs on par with four state-of-the-art probabilistic forecasting methods, including a Transformer-based approach and WaveNet, on two point metrics (sMAPE and NRMSE) as well as on a set of range metrics (quantile loss percentiles) in the majority of cases. We also demonstrate that our method requires significantly fewer parameters than Transformer-based methods, which means that the model can be trained faster with significantly lower memory requirements, which as a consequence reduces the infrastructure cost for deploying these models.

1. Introduction

Time series forecasting is crucial to many application domains, such as retail, ecommerce, finance, and biology. The classical approach to forecasting is to learn a single model for each time series separately. However, creating such models often requires careful manual intervention from forecasting practitioners, which becomes impractical when the number of time series to be forecast is large (e.g., when the task is to forecast store demand for each product of a retail chain (Böse et al., 2017; Taylor & Letham, 2018)). Moreover, decision makers often prefer probabilistic forecasts to point forecasts because they are interested in a quantification of the uncertainty of the forecast. However, probabilistic forecasts may require a set of models for each time series, which further increases the disadvantages of using a single model for each time series. In this work, we address this problem of probabilistic forecasting for multiple time series.

Recently, a number of neural network architectures have been proposed (Fischer & Krauss, 2018; Laptev, Yosinski, Li, & Smyl, 2017; Li, Yu, Shahabi, & Liu, 2018) to address the disadvantages of classical forecasting methods. These neural architectures leverage the increasing availability of data in the aforementioned application domains, and enable the training of a single model that

* Corresponding author.
E-mail address: o.r.sprangers@uva.nl (O. Sprangers).

https://doi.org/10.1016/j.ijforecast.2021.11.011

can produce forecasts for multiple time series over multiple timesteps. The Transformer (Vaswani et al., 2017) is one such neural architecture that has recently shown state-of-the-art performance on a set of real-world forecasting datasets (Li et al., 2019). However, it requires a large number of parameters to achieve these state-of-the-art results compared to existing neural architecture-based probabilistic forecasting methods.

In this work, we demonstrate that it is possible to achieve on-par or better results than a Transformer with an architecture that requires an order of magnitude fewer parameters. We achieve this by

(1) smartly encoding available future information;
(2) applying a simple temporal convolutional architecture; and
(3) leveraging the Student's t(3)-distribution as a loss function to optimize the parameters of our method.

Our bidirectional temporal convolutional network (BiTCN) is based on two temporal convolutional networks (TCNs), as detailed in Section 3. The first network encodes future covariates of the time series, whereas the second network encodes past observations and covariates. This method allows us to preserve the temporal information of sequence data, and is computationally more efficient by requiring fewer sequential operations than the commonly used bidirectional LSTM. The output of both our networks is used to estimate the parameters of a Student's t(3)-distribution of our forecast.

Next to introducing BiTCN, the contributions of this paper are threefold:

- We empirically verify that we achieve state-of-the-art forecasting performance with BiTCN on four real-world datasets (Sections 4 and 5). We evaluate BiTCN using point forecast error metrics (sMAPE and NRMSE) as well as probabilistic forecast error metrics (quantile loss percentiles).
- We show how BiTCN is designed (1) to require an order of magnitude fewer parameters than the second-best-scoring Transformer-based method, and (2) to employ a simpler architecture than WaveNet (van den Oord et al., 2016)—an autoregressive neural network based on a TCN, which placed second in a Kaggle forecasting competition (Kechyn, Yu, Zang, & Kechyn, 2018). The result is that our model can be trained using cheaper hardware because it requires less memory (approximately two to four times less GPU memory, depending on the batch size). Moreover, our method requires at least 20% less energy during training, thereby reducing the cost to train the model.
- We demonstrate the benefits of choosing a Student's t(3)-distribution for the probabilistic forecasting setting (Section 5.3), which enables us to eliminate a training hyperparameter compared to using a Gaussian distribution for probabilistic forecasting, and it results in a more stable training regime.

2. Related work

Time series forecasting is a broad topic that is studied in various scientific disciplines, such as econometrics, economics, and machine learning. Traditional forecasting models such as ARIMA (Box & Pierce, 1970) and exponential smoothing (Hyndman, Koehler, Ord, & Snyder, 2008) rely on considering the time series individually, and thus create separate models for each time series (typically referred to as local methods (Montero-Manso & Hyndman, 2021)). These local methods may be more difficult to apply in contemporary large-scale forecasting applications, as maintaining individual models per time series may be impractical and these methods typically struggle to forecast unseen time series with little or no past observations. Moreover, Montero-Manso and Hyndman (2021) demonstrated that such local methods are typically outperformed by global methods (i.e., methods that create a single global model across all time series).

Recently, neural networks (in the form of sequence-to-sequence models) have been successfully applied to various autoregressive problems, such as speech generation with WaveNet (van den Oord et al., 2016) and probabilistic forecasting with DeepAR (Salinas, Flunkert, Gasthaus, & Januschowski, 2019). We refer the reader to Mariet and Kuznetsov (2019) for a more formal introduction to sequence-to-sequence modeling for time series, and to Montero-Manso and Hyndman (2021) for a study of the benefits of global models compared to local models for time series.

Recurrent neural networks. RNNs, usually in the form of long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) modules, have been widely applied to forecasting problems; Fischer and Krauss (2018), Laptev et al. (2017) and Li et al. (2018) provide recent examples of applications to financial markets, taxi services, and traffic forecasting, respectively. Moreover, RNN models have been successful at winning several forecasting competitions, such as the LSTM-based ES-RNN method that won the M4 forecasting competition (Makridakis, Spiliotis, & Assimakopoulos, 2020a), and the LSTM-based method that won the Kaggle Webtraffic competition.[1] The downside of employing RNN models in forecasting is that their recurrent nature leads to slow training times, and long-term dependencies may not be properly captured. To improve the long-term memory of RNNs, attention mechanisms (Bahdanau, Cho, & Bengio, 2015) were combined with RNNs in Chen et al. (2018) and Lai, Chang, Yang, and Liu (2018). However, these methods still rely on the sequential training of RNN modules. Hewamalage, Bergmeir, and Bandara (2021) recently provided an extensive study comparing RNN-based forecasting methods to classical approaches such as ARIMA and ETS. They found that RNN-based forecasting methods can compete with classical approaches in many scenarios.

[1] https://github.com/Arturus/kaggle-web-traffic.

Attention-only models. The Transformer (Vaswani et al., 2017) was introduced in the domain of natural language processing (NLP) to enable efficient parallel training of sequence-to-sequence problems. Recently, the Transformer was adopted to the problem of probabilistic forecasting in Li et al. (2019), who employed a decoder-only model with convolutional sparse attention to reduce the memory consumption of the base Transformer and enhance its forecasting performance. This method currently achieves state-of-the-art results on the various public datasets commonly used in probabilistic forecasting papers.

Temporal convolutional networks. TCNs employ stacked dilated convolutional neural networks (CNNs) to overcome the limitations of RNNs (Bai, Kolter, & Koltun, 2018). The benefit of this architecture is that it allows for fast training like the Transformer model, whilst having significantly lower memory consumption. This enables larger batch sizes during training, which, in turn, speed up training. Temporal convolutional networks have been applied to autoregressive problems in speech generation with Wavenet (van den Oord et al., 2016), and achieved second place in a Kaggle sales forecasting competition (Kechyn et al., 2018). A more recent method closely related to ours is introduced in Sen, Yu, and Dhillon (2019), where a matrix factorization method is combined with a standard TCN to provide point forecasts.

Our method is different in that we do not employ matrix factorization, and we study the problem of probabilistic forecasting instead of point forecasts. Probabilistic forecasting with TCNs has recently been conducted by Chen, Kang, Chen, and Wang (2019). Our work differs from this work in that (1) our core temporal module is simpler as it uses fewer components, (2) our focus is on parameter-efficient probabilistic forecasting, and (3) we introduce a forward-looking module to encode future covariates.

3. Methodology

Here, we introduce the problem setting of probabilistic forecasting and describe the core components of our method. We end the section by detailing the choice of our loss function.

Probabilistic forecasting. A time series is a sequence of ordered measurements $\{y_t, y_{t+1}, \ldots\}$, in which we assume the timestep t to be constant (e.g., a day or an hour). Denote the set of N time series as $\{\vec{y}_{i,1:t_0}\}_{i=1}^{N}$ and $\{\vec{a}_{i,1:t_0}\}_{i=1}^{N}$ as a set of additional attributes. We are interested in modeling the conditional distribution

$$\begin{aligned} p(\vec{y}_{i,t_0:T}|\vec{y}_{i,1:t_0}, \vec{a}_{i,1:T}; \mu_\theta, \sigma_\theta) \\ &= \prod_{t=t_0}^{T} p\left(\vec{y}_{i,t}|\vec{y}_{i,1:t-1}, \vec{a}_{i,1:T}; \mu_\theta, \sigma_\theta\right) \\ &= \prod_{t=t_0}^{T} p\left(\vec{y}_{i,t}|\vec{x}_{i,1:T}; \mu_\theta, \sigma_\theta\right), \end{aligned} \tag{1}$$

where t_0 and T denote the start and end of the forecast, respectively, and $(\mu_\theta, \sigma_\theta)$ the location and scale parameters of a distribution parameterized by θ, which we learn with our model. We estimate separate output distribution parameters for each timestep in the forecast window. The input to our network consists of the concatenation of lagged target variables $\vec{y}_{lag}$ and additional attributes in the form of numerical covariates $\vec{a}_{cov}$ (e.g., a day-of-the-week indicator) and categorical covariates $\vec{a}_{cat}$ (e.g., a time series identifier). We denote this concatenation by $\vec{x}$.

Dilated convolutions. We observe that future covariates on time series are usually available at the time of forecasting, such as item information or time indicators (e.g., the day of the week or indicators for promotion days or holidays). We would like to encode all such knowledge of the future into a latent state on which the forecast of the current timestamp can be conditioned. To achieve this, we create a TCN consisting of dilated convolutions that look *forward* in time, instead of backward. More formally, the *forward* dilated convolution operation F on an element s of a sequential input $\mathbf{x} \in \mathbb{R}^n$ and a filter $f \in \mathbb{R}^k$ can be defined as follows (Bai et al., 2018):

$$F(s) = \sum_{j=0}^{k-1} f(j) \cdot \mathbf{x}_{s+d \cdot j}, \tag{2}$$

with filter size k, dilation factor d, and $s + d \cdot j$ indicating the forward steps. For a backward (causal) dilated convolution, $s + d \cdot j$ in (2) becomes $s - d \cdot j$. We stack N layers with dilation 2^{i-1} for layer i to obtain a 'receptive field' for the TCN of size $1 + (k-1)(2^N - 1)$. The receptive field is the effective sequence length the network can condition its forecast on. For example, if a forecasting problem requires taking into account sequences of a length up to 500 timesteps, valid options for the kernel size and number of layers N would include $(k = 3, N = 8)$, $(k = 7, N = 7)$ or $(k = 11, N = 6)$. It is necessary to add right (left) padding of size $(k-1) \cdot 2^{i-1}$ for the forward (backward) convolution at each ith layer to maintain a constant sequence length throughout the network. Many existing forecasting methods already incorporate future covariate information. However, our approach is novel in the following ways:

(1) The temporal structure of the covariates is maintained vis-a-vis feed-forward networks to incorporate such information.
(2) Dilated convolutions enable more efficient training than, e.g., a bidirectional LSTM, which requires more sequential operations during training (refer to Section 5 for a discussion of the computational efficiency).

We considered an alternative design with an unmasked multi-head attention module (Vaswani et al., 2017) as a 'look-forward' layer. We decided against this design, however, as preliminary experiments indicated relatively poor results in terms of the balance between parameters and performance (with many additional parameters required for incremental accuracy gains). See Section 5 for a discussion of the complexity of multi-head attention layers.

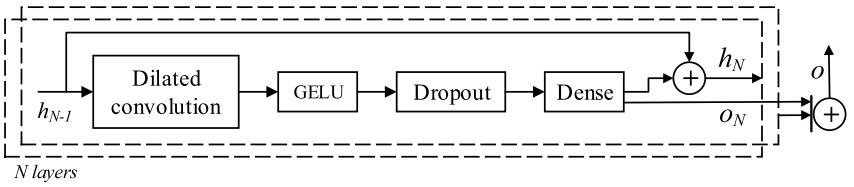

Fig. 1. Temporal block.

Temporal blocks. Inspired by the work of Bai et al. (2018) and van den Oord et al. (2016), we construct temporal blocks by stacking dilated convolutions. A single layer of our TCN is displayed in Fig. 1. Each TCN layer in our network consists of a block that contains a dilated convolution, Gaussian error linear unit (GELU) activation (Hendrycks & Gimpel, 2018), dropout, and dense layer. A GELU multiplies an input x with the cumulative distribution function $\Phi(x)$ of the Gaussian distribution, i.e.,

$$GELU(x) = xP(X \leq x) = x\Phi(x) \approx 0.5x\left(1 + \tanh\left[\sqrt{2/\pi}\left(x + 0.044715x^3\right)\right]\right) \tag{3}$$

and provides a 'softer' activation than the commonly used rectified linear unit (ReLU), which generally improves performance (Hendrycks & Gimpel, 2018).

Each layer produces a hidden state h_N and an output o_N; the latter is summed over all layers to provide the output of the network. Weight normalization (Salimans & Kingma, 2016) is applied to the dilated convolution and dense modules. Our temporal block is simpler than the ones used in WaveNet, as our temporal block does not require a gated activation unit, but instead relies solely on the GELU activation as non-linearity. Section 5.1 shows that this simplification has a positive effect on performance.

Our method employs two temporal blocks. The first block consists of N layers and encodes the past observations and covariates of time series with backward dilated convolutions, whereas the second block consists of $N+1$ layers and encodes future covariates with forward dilated convolutions. The additional layer in the second block is required to enlarge the receptive field of the forward-looking module, as the sequence length of the covariates and categorical inputs exceeds the dimension of the input and output sequence length in order to facilitate sufficient forward-looking for later timesteps in the forecast. This is to ensure that later timesteps of the forecast also receive sufficient future covariate information to condition the forecast on. A three-layer example of how these temporal blocks employ backward and forward dilated convolutions to condition the forecast is illustrated in Fig. 2.

Within the forward TCN, grouped convolutions are employed to reduce the number of parameters needed. The intuition behind this choice is that the future covariates usually contain less information than the past covariates and lags, and thereby require fewer parameters for encoding. We employ dense layers with dropout before the temporal blocks to scale the input dimensions to the hidden dimension of the network. Finally, dense output layers provide the location μ and scale σ of the output distribution. These layers are activated with a softplus activation to ensure that their output remains positive (this activation may be removed in case the possibility of a negative output is desired), and a small number ϵ is added to ensure numerical the stability of the scale output. The pseudocode of our resulting method, which we call a bidirectional temporal convolutional network (BiTCN), is shown in Algorithm 1 and graphically depicted in Fig. 3.

Algorithm 1 BiTCN pseudocode.

Input: $\vec{y}_{lag} \in \mathbb{R}^{T \times d_{batch} \times d_{input}}$, $\vec{a}_{cov} \in \mathbb{R}^{T_c \times d_{batch} \times d_{cov}}$, $\vec{a}_{cat} \in \mathbb{R}^{T_c \times d_{batch} \times d_{cat}}$
Output: $\vec{\mu}, \vec{\sigma}$

$\vec{a}_{emb} = \texttt{Embedding}(\vec{a}_{cat})$
$\vec{x}_{lag} = \texttt{Concat}(\vec{y}_{lag}, \vec{a}_{cov}[:d_l], \vec{a}_{emb}[:d_l])$
$\vec{x}_{cov} = \texttt{Concat}(\vec{a}_{cov}, \vec{a}_{emb})$
$\vec{h}_{lag} = \texttt{Drop}(\texttt{Dense}(\vec{x}_{lag}))$
$\vec{h}_{cov} = \texttt{Drop}(\texttt{Dense}(\vec{x}_{cov}))$
$\vec{o}_{lag} = \texttt{Backward Temporal Block}(\vec{h}_{lag})$
$\vec{o}_{cov} = \texttt{Forward Temporal Block}(\vec{h}_{cov})$
$\vec{o} = \texttt{Concat}(\vec{o}_{cov}[:d_l], \vec{o}_{lag})$
$\vec{\mu} = \texttt{SoftPlus}(\texttt{Dense}(\vec{o}))$
$\vec{\sigma} = \texttt{SoftPlus}(\texttt{Dense}(\vec{o})) + \epsilon$
return $\vec{\mu}, \vec{\sigma}$

Output distribution. Although BiTCN allows for the use of many parametric (e.g., Gaussian) or non-parametric (e.g., quantile) distribution functions, we use a Student's t-distribution with three degrees of freedom in our experiments as a fixed parameterized distribution. The probability density function of this distribution for a variable $\vec{y}$ is given by

$$f(\vec{y}) = \frac{2}{\pi\sqrt{3}\left(1 + \frac{\vec{y}^2}{3}\right)^2} \tag{4}$$

and we aim to minimize the log-likelihood loss

$$\mathcal{L}(\theta|\vec{y}) = \log\left(f\left(\frac{\vec{y} - \mu_\theta}{\sigma_\theta}\right)\right), \tag{5}$$

where μ and σ are the outputs of our network for a given time series at a certain timestep. There are several benefits of using this distribution. First, the Student's t-distribution can be seen as a fat-tailed version of the Gaussian distribution, which makes it a natural candidate for estimating real-valued quantities without prior information on the true distribution. The benefit of a fat-tailed distribution is twofold:

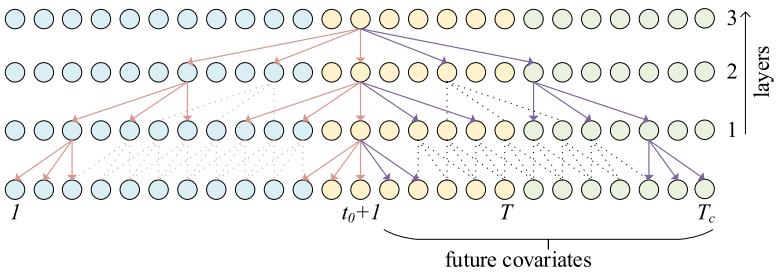

Fig. 2. Illustration of how three stacked TCN layers enable conditioning the forecast at $t = t_0 + 1$ on both past and future information using forward and backward dilated convolutions with kernel size 3 and dilation 2^{i-1} for the ith layer. The blue dots represent the input sequence, the yellow dots the output sequence, and the green dots the additional future covariates on which the forecast can be conditioned. The red connections indicate the *backward*-looking convolutions, and the purple connections the *forward*-looking convolutions. For clarity purposes, some inner convolutional connections are shown with dashed lines. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

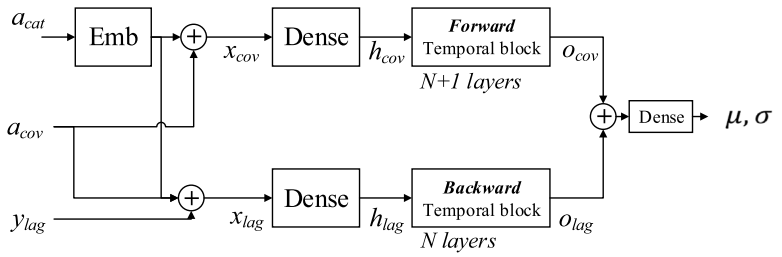

Fig. 3. Overall network of BiTCN. First, in the top half, the categorical inputs a_{cat} are embedded using an embedding layer. The result is combined with the available covariates a_{cov} to result in the input X_{cov}, which includes all available (covariate) information about the past and future. This is then led through a linear layer and a temporal block that looks forward across all the future information. In the bottom half, the lagged target y_{lag} is combined with the available historical covariate and categorical information to result in the input X_{lag}. This is then led through a linear layer and an autoregressive temporal block (i.e., standard dilated convolutions). The outputs of both halves are joined by adding them together at the right timestamp, and a final linear layer provides the location and scale of our output distribution.

(1) it allows our model to capture rare events better, and
(2) the log-likelihood loss is numerically stable around a large range of input values.

The disadvantage of fixing a distribution a priori is the imposition of a defined distribution on the output, which may not reflect the true underlying distribution properly. However, we observed this to be no issue in practice.

To facilitate a better comparison between architectures, our loss function is used for each method in our experimental section, comparing forecasting accuracy across competing methods. We demonstrate the benefits of choosing the Student's t(3)-distribution over the Gaussian distribution in Section 5.3.

4. Experimental setup

Datasets. We employ four real-world datasets to investigate the probabilistic forecasting performance of the BiTCN architecture.

- `Electricity` (UCI Machine Learning Repository: ElectricityLoadDiagrams20112014 Data Set, 2015) Hourly electricity consumption for 370 clients over a three-year period. The task is to forecast electricity consumption per client for the next 24 h given the previous seven days. The training/validation/test split is 80/10/10 based on the date.
- `Traffic` (UCI Machine Learning Repository: PEMS-SF Data Set, 2011) 15 months of hourly data describing the occupancy rate, between 0 and 1, of different car lanes of the San Francisco Bay Area freeways. We forecast the occupancy rate per lane for the next 24 h given the previous seven days. The training/validation/test split is 80/10/10 based on the date.
- `Favorita` (Corporación Favorita Grocery Sales Forecasting, 2018) A dataset from Kaggle that contains four years of daily sales data for store-product combinations taken from a retail chain. The task is to forecast the log sales for all product-store combinations for the next 30 days based on the previous 90 days. The training data consist of data between 01-01-2013 and 02-09-2013. We validate on samples between the dates 03-09-2013 and 03-10-2013, and we test on the remaining data up to 31-03-2014. The target and lagged input variables are log-transformed.
- `WebTraffic` (Web Traffic Time Series Forecasting, 2017) 145k time series of daily Wikipedia pageviews in the period from 2015–2017. The task is to forecast the number of pageviews for the next 30 days given the previous 90 days. We use a selection of 10k time series with the largest number of pageviews in the training period (which is up to 31-12-2016).

We provide more detailed descriptions of the datasets in Table A.7 in Appendix A.

The `Electricity` and `Traffic` datasets provide an indication of forecasting performance on relatively regular time series, with a small number of future covariates available to condition our forecast on (i.e., a few time-based features such as the day of the week and the hour of the day). `Favorita` and `WebTraffic` contain more irregular time series, and the former also contains a rich set of covariates. For each dataset, we add categorical covariates and numerical covariates consisting mainly of time series identifiers, time indicators (day of the week or month) and other indicators (e.g., holidays). The time indicators are represented by two Fourier terms (Hyndman, 2018) to represent the periodic nature of, e.g., days or months (i.e., the first day of the week should be close to the last day of the week) as follows:

$$\text{covariate}_{\sin} = \sin\left(\text{covariate} \cdot \left(2 \cdot \frac{\pi}{\text{period}}\right)\right) \tag{6}$$

$$\text{covariate}_{\cos} = \cos\left(\text{covariate} \cdot \left(2 \cdot \frac{\pi}{\text{period}}\right)\right). \tag{7}$$

The categorical and numerical covariates are assumed to be known a priori, i.e., we include these variables in the forward-looking of BiTCN. For all datasets except `Favorita`, we employ the scaling mechanism from Salinas et al. (2019) for normalizing the outputs and lagged outputs to our network:

$$\vec{\bar{y}} = \frac{\vec{y}}{1 + \frac{1}{t_0}\sum_{i=0}^{t_0} \vec{y}_i}. \tag{8}$$

Again, we refer the reader to Table A.7 in the Appendix for further details on the datasets.

Baseline models. We compare BiTCN against five state-of-the-art forecasting methods of different neural architectures:

- DeepAR (Salinas et al., 2019). An LSTM-based generic probabilistic forecasting framework.
- ML-RNN (Wen, Torkkola, Narayanaswamy, & Madeka, 2018). An encoder–decoder probabilistic forecasting framework that uses an LSTM to encode past observations and covariates, and two MLP decoders to generate probabilistic forecasts. This method also uses future covariate information to condition its forecast on.
- TransformerConv (Li et al., 2019). A Transformer-based forecasting model augmented with causal convolutions, which has shown state-of-the-art results on the `Electricity` and `Traffic` dataset.
- TCN (Bai et al., 2018). A standard TCN, as employed for probabilistic forecasting by Chen et al. (2019).
- WaveNet (van den Oord et al., 2016), which placed second in a Kaggle sales forecasting competition (Kechyn et al., 2018).

In addition, we compare BiTCN against three traditional forecasting methods and one popular machine learning package:

- Seasonal naive. The seasonal naive baseline, where we simply take the observation from the last period as our forecast. Since our datasets exhibit clear seasonality (either daily or weekly), this provides a sensible baseline.
- ETS (Holt, 2004). The exponential smoothing method, including Holt–Winters' seasonal and trend components.
- Theta (Assimakopoulos & Nikolopoulos, 2000). The theta method, a univariate forecasting method that uses the local curvature of the time series.
- LightGBM (Ke et al., 2017). A highly popular gradient boosting package on which the winning solution of the M5 forecasting competition (Makridakis, Spiliotis, & Assimakopoulos, 2020b) was based.

Training and optimization. We implemented, trained, and evaluated BiTCN and each neural network method using PyTorch (Paszke et al., 2019) and we intend to open-source the code and scripts we used to run the experiments upon publication of the paper. For DeepAR, we used the hyperparameters from Salinas et al. (2019), and for ML-RNN we used the hyperparameters from Wen et al. (2018). For TransformerConv, we used the hyperparameters from Li et al. (2019). For TCN, WaveNet, and BiTCN we selected a kernel size and hidden size to establish a comparable parameter budget as DeepAR, as this is the state of the art in terms of the parameter budget compared against in this study. An overview of the key model hyperparameters is given in Table 1. Finally, we optimized the learning rate and batch size for each dataset and method by performing a limited grid search using the following settings:

- Learning rate: {0.001, 0.0005, 0.0001}.
- Batch size: {128, 256, 512}, except for DeepAR and ML-RNN, for which we used {64, 128, 256}.

We ran each experiment for 100 epochs with an early stopping criterion of five epochs. We trained, validated, and tested each method for five different random seeds for the neural network weight initialization. We optimized the parameters of each method using Adam (Kingma & Ba, 2015). The results of the limited grid search for each method and dataset are given in Figures A.6–A.9 in Appendix. For the forecasting performance evaluation, we report the evaluation metrics on the test set of the best performing {(learning rate, batch size)} combination according to Figures A.6–A.9.

For the traditional forecasting methods, we used the `statsmodels` Python package and fit a model on each input sequence of our test set. For the probabilistic forecasts, we used the generated prediction intervals if the method provided these. For LightGBM, we created nine separate models for each quantile $0.1, 0.2, \ldots, 0.9$ using quantile regression, and recursively applied each model to forecast each timestep in our forecast. We used Optuna (Akiba, Sano, Yanase, Ohta, & Koyama, 2019) to find the best hyperparameters for LightGBM, which we applied to all datasets.

Table 1
Key model hyperparameters.

	DeepAR	ML-RNN	TransformerConv	TCN	WaveNet	BiTCN
State size	40	30	$d_{emb} + d_{cov} + d_{lag}$	20	20	12
Layers	3	1	3	5	5	5
Kernel size	n.a.	n.a.	9	9	9	9
Heads	n.a.	n.a.	8	n.a.	n.a.	n.a.
Dropout	0.1	n.a.	0.1	0.1	n.a.	0.1
Receptive field	∞	∞	∞	249	249	497

Evaluation. We employed a set of point accuracy metrics and range accuracy metrics to evaluate the forecasting performance. For point accuracy, we used the symmetric mean absolute percentage error (sMAPE) and normalized root mean squared error (NRMSE) (Alexandrov et al., 2020):

$$sMAPE = \frac{1}{n}\sum_{t=t_0}^{n}\frac{2|\vec{y}_t - \hat{\vec{y}}_t|}{|\vec{y}_t| + |\hat{\vec{y}}_t|} \tag{9}$$

$$NRMSE = \frac{\sqrt{\frac{1}{n}\sum_{t=t_0}^{n}\left(\vec{y}_t - \hat{\vec{y}}_t\right)^2}\cdot \mathbb{1}_{\vec{y}_t \neq 0}}{\sum_{t=t_0}^{n}|\vec{y}_t| + \mathbb{1}_{\vec{y}_t=0}}, \tag{10}$$

where $n = T - t_0$ denotes the number of forecast steps, and $\mathbb{1}_{\vec{y}_t=0}$ is an indicator function to scale the metric when the observed target value equals zero. Note that the sMAPE ranges from 0% to 200%. The NRMSE is essentially the normal root mean squared error, but normalized to account for differences in the values of each individual time series. Each forecasting metric has its advantages and disadvantages; we also considered using a popular one-step-ahead metric such as the MASE (Hyndman, 2006), but considered this inappropriate for our task (which is multistep forecasting). For range accuracy, the normalized quantile loss function (Salinas et al., 2019) was used for the quantiles $p = \{0.1, 0.5, 0.9\}$:

$$Q(\vec{y}_i, \hat{\vec{y}}_i, p) = 2 \cdot |(\vec{y} - \hat{\vec{y}}) \cdot (\mathbb{1}_{\vec{y} \leq \hat{\vec{y}}} - p)| \tag{11}$$

$$Q(\vec{y}, \hat{\vec{y}}, p) = \frac{\sum_i Q(\vec{y}_i, \hat{\vec{y}}_i, p)}{\sum_i \vec{y}_i}. \tag{12}$$

Finally, we show the mean quantile performance over the nine quantiles in the range $p = \{0.1, 0.2, \ldots, 0.9\}$.

5. Results and discussion

First, we demonstrate the forecasting performance of BiTCN on a set of real-world datasets to substantiate our claim that BiTCN achieves state-of-the-art forecasting performance with fewer parameters than competing methods (Section 5.1). Second, we show the benefit of our architecture by studying the forecasting efficiency in terms of model complexity, training time, and energy cost (Section 5.2). Finally, we study the impact of our design choices, such as employing the Student's t(3)-distribution (Section 5.3), the effect of our forward-looking module (Section 5.4), and how BiTCN's performance is impacted by its hyperparameters (Section 5.5).

5.1. Forecasting effectiveness

We report the forecasting performance in Table 2. Although similar model dimensions are used for each method across the experiments, parameter counts for each model may be different per experiment, due to the size of the embedding dimension required to embed the categorical input vector $\vec{a}_{cat}$, which has a different dimension for each dataset.

We observe the following per dataset:

- `Electricity`: BiTCN performs on par with the best methods, WaveNet and TransformerConv.
- `Traffic`: BiTCN performs on par with the best methods.
- `Favorita`: BiTCN outperforms or performs similar to competing methods on all metrics.
- `WebTraffic`: BiTCN performs similar to the best-performing method, TCN, on mean quantile loss, and in line with other methods on the other metrics.

On average, BiTCN ranks highest, at an overall average ranking of 1.75. Given these observations, we conclude that BiTCN can achieve state-of-the-art results on a set of real-world probabilistic forecasting tasks whilst using significantly fewer parameters than the second-best-performing method, TransformerConv (average ranking of 2.25).

Secondly, even though not a primary objective of our paper, we confirm the findings from Li et al. (2019) and Salinas et al. (2019) that traditional methods such as seasonal naive, ETS, and theta are outperformed by neural network methods on the task of multistep probabilistic forecasting.

Finally, we find that LightGBM performs reasonably, but not as well as expected. We attribute this to our experimental setup. First, to facilitate a like-for-like comparison to neural network-based methods, we use the same features for LightGBM as we provide to our neural networks. In practice, practitioners often spend a lot of time engineering features that provide a better signal to a LightGBM model, which typically improves performance. Second, we apply the LightGBM model recursively, which means that any bias and variance that enters the model may be propagated to future timesteps. This issue is confirmed from the results of the M5 competition, where most participants created separate LightGBM models for every timestep to avoid this bias/variance propagation. However, for our setting this would imply creating not only a separate model per quantile, but also per timestep. This would give LightGBM a somewhat unfair advantage compared to neural network-based methods.

Table 2
Forecasting results on various point and range accuracy metrics. For the neural network methods, we report mean metrics over five different seeds of parameter initializations per method, with the standard deviation of the metric shown in brackets. Lower is better, and bold indicates the best method for the metric. The rank denotes the rank of the methods across the four metrics for each dataset (lower is better).

Dataset/Method	No. parameters	Point metrics		Range metrics		Rank
		sMAPE	NRMSE	Q(0.5)	mQ	
`Electricity`						
Seasonal naive	0	0.112	0.719	0.078	0.078	8
ETS	100k	0.201	1.755	0.152	0.136	11
Theta	20k	0.177	1.575	0.141	0.164	10
LightGBM	2M	0.089	0.679	0.065	0.077	5
DeepAR	45k	0.099 (0.0044)	0.671 (0.0142)	0.069 (0.0010)	0.057 (0.0009)	7
ML-RNN	2.9M	0.115 (0.0029)	0.829 (0.0555)	0.086 (0.0030)	0.068 (0.0024)	9
TCN	46k	0.103 (0.0017)	0.668 (0.0121)	0.068 (0.0011)	0.056 (0.0010)	6
WaveNet	48k	0.093 (0.0026)	**0.646 (0.0095)**	0.063 (0.0005)	0.052 (0.0005)	2
TransformerConv	415k	**0.083 (0.0014)**	0.658 (0.0237)	**0.062 (0.0012)**	0.052 (0.0010)	1
BiTCN	49k	0.089 (0.0009)	0.648 (0.0090)	0.063 (0.0003)	**0.052 (0.0003)**	2
`Traffic`						
Seasonal naive	0	0.351	0.667	0.285	0.285	9
ETS	100k	0.483	0.693	0.371	0.348	10
Theta	20k	0.512	3.413	0.980	0.970	11
LightGBM	2M	0.128	0.358	0.111	0.114	4
DeepAR	57k	0.179 (0.0182)	0.408 (0.0274)	0.131 (0.0141)	0.111 (0.0126)	8
ML-RNN	2.4M	0.140 (0.0021)	0.387 (0.0026)	0.123 (0.0017)	0.102 (0.0013)	7
TCN	57k	0.150 (0.0128)	0.373 (0.0074)	0.117 (0.0027)	0.098 (0.0023)	6
WaveNet	60k	0.165 (0.0054)	0.357 (0.0028)	0.108 (0.0018)	0.091 (0.0015)	3
TransformerConv	372k	0.130 (0.0035)	**0.353 (0.0020)**	**0.105 (0.0020)**	**0.089 (0.0014)**	1
BiTCN	61k	**0.127 (0.0005)**	0.372 (0.0142)	0.108 (0.0005)	0.091 (0.0004)	2
`Favorita`						
Seasonal naive	0	1.001	4.541	1.053	1.053	10
ETS	100k	1.024	4.297	1.021	0.842	8
Theta	20k	1.060	2.369	0.844	0.922	8
LightGBM	2M	0.879	1.562	0.489	0.396	5
DeepAR	61k	**0.574 (0.1823)**	1.680 (0.1040)	0.543 (0.0216)	0.422 (0.0150)	5
ML-RNN	1.9M	0.689 (0.0132)	1.406 (0.0483)	0.461 (0.0079)	0.362 (0.0064)	4
TCN	61k	0.612 (0.0989)	1.323 (0.0562)	0.440 (0.0210)	0.350 (0.0134)	3
WaveNet	66k	0.711 (0.0261)	1.623 (0.1773)	0.512 (0.0427)	0.405 (0.0319)	7
TransformerConv	210k	0.673 (0.0046)	1.319 (0.0602)	0.439 (0.0179)	**0.346 (0.0129)**	1
BiTCN	66k	0.674 (0.0015)	**1.317 (0.0179)**	**0.432 (0.0049)**	0.347 (0.0033)	1
`WebTraffic`						
Seasonal naive	0	0.357	4.709	0.414	0.414	10
ETS	100k	0.311	4.096	0.350	0.359	8
Theta	20k	0.338	4.132	0.340	0.476	9
LightGBM	2M	0.260	4.022	0.273	1.002	6
DeepAR	238k	0.279 (0.0054)	4.003 (0.2526)	0.282 (0.0060)	0.244 (0.0050)	5
ML-RNN	2.4M	0.246 (0.0064)	4.027 (0.0269)	0.270 (0.0048)	0.234 (0.0051)	4
TCN	239k	**0.234 (0.0034)**	**3.718 (0.1692)**	**0.253 (0.0048)**	**0.231 (0.0046)**	1
WaveNet	241k	0.236 (0.0037)	3.953 (0.3083)	0.268 (0.0129)	0.244 (0.0107)	2
TransformerConv	630k	0.246 (0.0065)	4.191 (0.3368)	0.301 (0.0225)	0.273 (0.0198)	6
BiTCN	242k	0.254 (0.0062)	3.988 (0.2177)	0.267 (0.0057)	0.232 (0.0041)	2

5.2. Forecasting efficiency

Model complexity. As can be seen from Table 2, BiTCN requires almost an order of magnitude fewer parameters than the TransformerConv. This is an indication that our architecture is more efficient, and possibly a better choice for the task of probabilistic forecasting, compared to existing neural architectures. To understand the computational complexity, we are mainly interested in the size of the following parameters (we study the impact of varying these parameters in Section 5.2):

- N, the number of layers of a neural network;
- k, the kernel size of a convolution;
- T, the sequence length used in the network;
- d_h, the hidden dimension of the network; and
- d_{out}, the number of output channels of a convolutional layer.

The computational complexity of each of our TCN layers can be computed by adding the computational complexity of the convolution layer, the activation, and the dense output layer:

$$O(k \cdot T \cdot d_h \cdot d_{out} + T^2 + d_{out} \cdot d_h \cdot T^2) = O(k \cdot T \cdot d_h^2 + d_h \cdot T^2), \quad (13)$$

where in BiTCN, $d_{out} = 4 \cdot d_h$. Hence, the complexity for an N-layer network of our architecture is $O(N \cdot (k \cdot T \cdot d_h^2 + d_h \cdot T^2))$ with $O(1)$ sequential operations per layer. In comparison, an LSTM-based architecture such as DeepAR has a computational complexity of $O(N \cdot (T \cdot {d_h}^2))$, but

requires $O(T)$ sequential operations, which significantly slows down training when sequence length increases.

Finally, Transformer-based architectures have a computational complexity of $O(N \cdot (T \cdot {d_h}^2 + d_h \cdot T^2))$ and require $O(1)$ sequential operations (Vaswani et al., 2017). But through the use of dilated convolutions in the self-attention mechanism, the computational complexity of the Transformer architecture of Li et al. (2019) becomes $O(N \cdot (k \cdot T \cdot {d_h}^2 + d_h \cdot T^2))$, with k the kernel size of the dilated convolutions. Hence, the computational complexity of the TransformerConv architecture is equal to that of BiTCN. In practice, however, this leads to different outcomes. Even though BiTCN requires more layers to ensure a sufficient receptive field, it requires a much smaller hidden dimension throughout the network. In contrast, the TransformerConv network requires fewer layers but a higher hidden dimension in order to support sufficient model capacity for each of the attention heads.

Model complexity compared to non-neural methods. We find that BiTCN requires significantly fewer parameters than LightGBM. What causes this? Suppose we would like to have a probabilistic forecast for 28 days, for nine quantiles. This requires at least nine models in the GBT setting. In our setting, each GBT consists of approximately 2000 trees, and is trained with a maximum number of leaf nodes of 127. Hence, this yields 254k parameters. Then, we have nine such models, yielding approximately 2M parameters. If we opt for separate models for each forecast day, we need $28 \cdot 2M = 64M$ parameters, which is orders of magnitude more than comparable NN-based solutions. What causes this difference in model size? Ultimately, this is due to the fact that NN methods are better representational learning methods. Stated differently, these methods learn to compress the data better. In an NN-based solution, an existing set of parameters is modified during learning. However, GBTs work incrementally, and add parameters for each iteration. Hence, these models do not compress the data as much as the NN does, yielding models with higher complexity.

Training time. One of the benefits of a model with fewer parameters is that *ceteris paribus* it can be trained faster. Therefore, we compare training times for each method in the left plane of Fig. 4. For each method, the left side of Fig. 4 shows the mean quantile loss on the test set as compared against the training time. The training times are normalized against a DeepAR baseline. We observe that BiTCN is the only algorithm that sits in the lower-left corner—which implies the smallest training time with the least quantile loss on the test set—for each of the datasets. Also, we observe that LightGBM's training time is comparable to those of NN methods, which is mostly due to its requirement to create separate models for each individual quantile in the forecast. Note that the latter finding is subject to hardware considerations, as the LightGBM models are trained on a CPU whilst the neural models are trained on a GPU.

Energy cost. Even though one method may train faster than the other, the energy cost for training may still exceed the energy cost of training other models, due to increased resource consumption. Therefore, we analyze the consumed energy for training each model in the right-half plane of Fig. 4. For this experiment, we ran each method separately and sequentially for a single epoch, and measured the average GPU board power drawn from an nVidia GTX 1080Ti using GPU-Z. During each experiment, we fixed the amount of CPU threads available to the training process, and there were no other processes running on the GPU used for measuring energy cost. The obtained result indicates the total average amount of energy in Joules that is required to compute a single epoch for each method. This energy consumption is then multiplied with the average number of epochs (across the five seeds) required to train each method to obtain the overall GPU energy cost. Finally, we normalize this energy cost against a DeepAR baseline. BiTCN is the strongest performer across all the datasets, as it sits in the lower-left corner on each graph. We now see a more clear distinction between TransformerConv and the other methods, and its performance is now less favorable, as it requires a higher energy consumption on every dataset compared to the other methods. We see energy consumption differences of 20%, up to an order of magnitude compared to BiTCN. In practice, this result is beneficial for practitioners who are interested in optimizing electricity and cooling costs, which commonly represent a significant portion of data-center operating costs (Strubell, Ganesh, & McCallum, 2020).

Memory usage. The Transformer's memory consumption scales quadratically with sequence length (Li et al., 2019), as all activations of the multi-head attention layers need to be stored during a forward pass of the network for use during the backward pass. In contrast, for temporal convolutional architectures, memory consumption predominantly scales with (1) the number of layers required to obtain a sufficient receptive field, and (2) the number of parameters of these layers. This enables training these models using less memory, which in turn enables the use of cheaper resources. In Table 3 we show the GPU memory consumption during training compared between BiTCN and TransformerConv across datasets. When comparing similar batch sizes, we observe a difference in memory consumption of two to four times. In practice, this means that BiTCN models can be trained on much cheaper GPUs. For example, a TransformerConv model with a batch size of 512 requires a high-end video card such as the nVidia RTX 2080Ti (which is equipped with 11 GB of GPU memory) to train the model on the `Electricity` and `Traffic` datasets, whereas a similar batch-size BiTCN model would only require a low-end RTX 2060. The first card typically retails for over USD $1000, whereas the latter can be acquired starting from USD $300 on Amazon.

5.3. Effect of Student's t(3)-distribution

To illustrate the benefits of using a Student's t(3)-distribution for probabilistic forecasting, we re-ran the experiments from Section 5.1 on the `Electricity` and

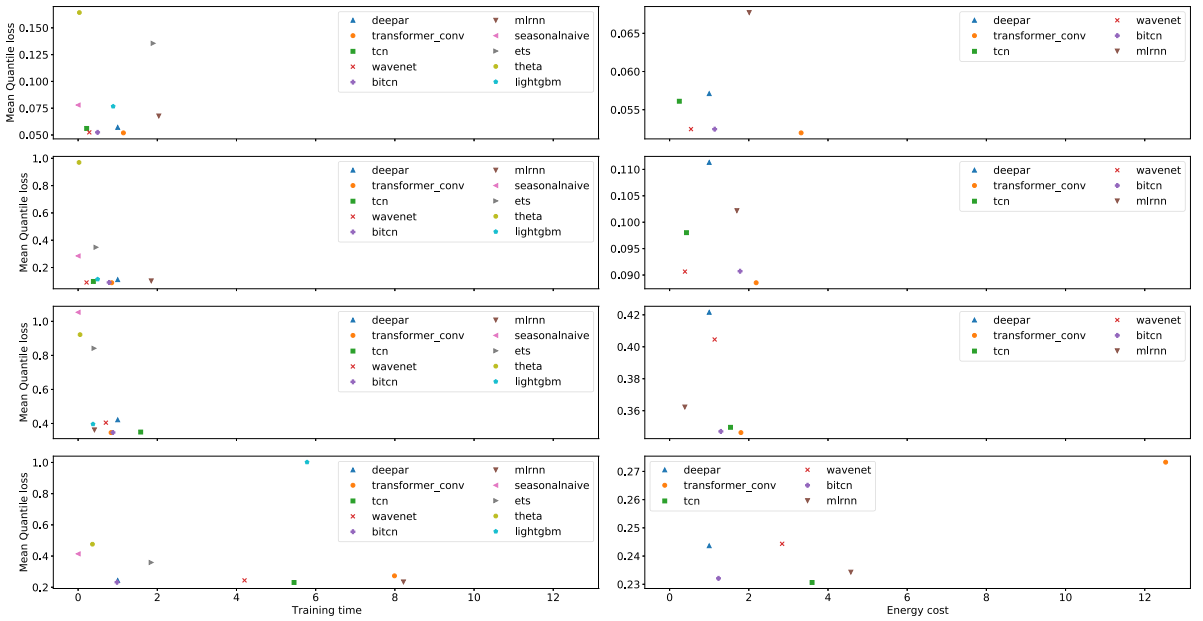

Fig. 4. Running time (left) and energy cost (right) compared to the mean quantile loss on the test set for the `Electricity` (top), `Traffic`, `Favorita`, and `Webtraffic` (bottom) datasets, where DeepAR is the baseline..

Table 3
GPU memory consumption in gigabytes during training of BiTCN compared to TransformerConv.

Batch size/Dataset	BiTCN	TransformerConv	Ratio (x)
128			
Electricity	1.54	3.43	2.23
Traffic	1.52	3.37	2.21
Favorita	1.34	2.15	1.60
WebTraffic	1.36	2.20	1.61
256			
Electricity	1.91	6.61	3.46
Traffic	1.90	6.55	3.45
Favorita	1.49	3.47	2.33
WebTraffic	1.51	3.46	2.30
512			
Electricity	2.62	10.20	3.89
Traffic	2.60	10.18	3.91
Favorita	1.77	5.95	3.36
WebTraffic	1.78	5.95	3.33

`Traffic` datasets for BiTCN using a parameterized Gaussian output distribution. We kept the same training settings as before, but we note that the Gaussian distribution in our forecasting setting requires clipping gradients to achieve a stable training regime. This requires tuning yet another hyperparameter—the maximum gradient norm. Why is this clipping necessary? It is a direct consequence of the thin-tailedness of the probability density function of the Gaussian, which is illustrated in Fig. 5. The thin tail of the Gaussian causes the log-probability during training to exert numerical instability. On the contrary, a fat-tailed distribution such as the Student's t(3)-distribution enables a more stable training regime. Aside from this practical disadvantage, a thin-tailed distribution is expected to perform worse on forecasting for processes that do not follow a normal distribution in their output data. We confirm this expectation by observing the results in Table 4, where we compare the forecasting performance with a Gaussian as output distribution vis-a-vis the Student's t(3)-distribution. On both the `Electricity` and `Traffic` datasets, we see performance differences of 5%–15% when using a Gaussian output distribution instead of a Student's t(3)-distribution. For the `Traffic` dataset, the differences are generally larger, which is expected, as this dataset is relatively skewed towards zero and hence benefits more from using an output distribution that has a heavier tail, such as the Student's t(3)-distribution. Finally, we observe lower variance in our test scores for the Student's t(3) distribution, which indicates a more stable training regime for this loss function. Our conclusion from this experiment is that the Student's t(3)-distribution improves forecasting performance whilst removing a hyperparameter from the optimization problem and enabling a more stable training regime.

5.4. Effect of forward-looking module

To study the effect of our forward-looking module, we re-ran the experiments from Section 5.1 on every dataset where we disabled the forward-looking module. We report the results in Table 5. We observe a positive impact of about 0%–2% from the forward module in all but one dataset, and especially in the `Traffic` and `Favorita` datasets. For the `Favorita` dataset this is expected, as this dataset provides very informative future covariates (e.g. whether there is a holiday on a particular day). On the contrary, the other datasets mostly contain covariates related to the day of the week or day of the month, which are less informative.

Table 4
Forecasting results on various point and range accuracy metrics comparing Student's t(3)-distribution with the Gaussian distribution as the loss function. Lower is better, and bold indicates the best loss function for the method. We report mean metrics over five different seeds per method, with the standard deviation shown in brackets.

Dataset/Method	Point metrics		Range metrics	
	sMAPE	NRMSE	Q(0.5)	mQ
`Electricity`				
BiTCN *Gaussian*	0.117 (0.0054)	0.713 (0.0280)	0.076 (0.0036)	0.062 (0.0023)
`Traffic`				
BiTCN *Gaussian*	0.175 (0.0089)	0.404 (0.0763)	0.141 (0.0016)	0.115 (0.0012)
BiTCN *Student's t(3)*	**0.127 (0.0005)**	**0.372 (0.0142)**	**0.108 (0.0005)**	**0.091 (0.0004)**

Table 5
Forecasting results on point and range accuracy metrics, ablating for our forward-looking module. We report mean metrics over five different seeds of parameter initializations per method, with the standard deviation of the metric shown in brackets. Lower is better, and bold indicates best method for the metric.

Dataset/Method	No. parameters	Point metrics		Range metrics	
		sMAPE	NRMSE	Q(0.5)	mQ
`Electricity`					
BiTCN	49k	**0.089 (0.0009)**	0.648 (0.0090)	**0.063 (0.0003)**	**0.052 (0.0003)**
BiTCN (w/o forward)	40k	0.089 (0.0012)	**0.647 (0.0050)**	0.064 (0.0005)	0.053 (0.0004)
`Traffic`					
BiTCN	61k	**0.127 (0.0005)**	**0.372 (0.0142)**	**0.108 (0.0005)**	**0.091 (0.0004)**
BiTCN (w/o forward)	52k	0.128 (0.0007)	0.546 (0.3427)	0.112 (0.0062)	0.094 (0.0047)
`Favorita`					
BiTCN	66k	**0.674 (0.0015)**	**1.317 (0.0179)**	**0.432 (0.0049)**	**0.347 (0.0033)**
BiTCN (w/o forward)	58k	0.683 (0.0024)	1.390 (0.0306)	0.460 (0.0084)	0.366 (0.0058)
`WebTraffic`					
BiTCN	242k	0.254 (0.0062)	**3.988 (0.2177)**	0.267 (0.0057)	0.232 (0.0041)
BiTCN (w/o forward)	233k	**0.252 (0.0055)**	4.004 (0.1903)	**0.265 (0.0046)**	**0.232 (0.0032)**

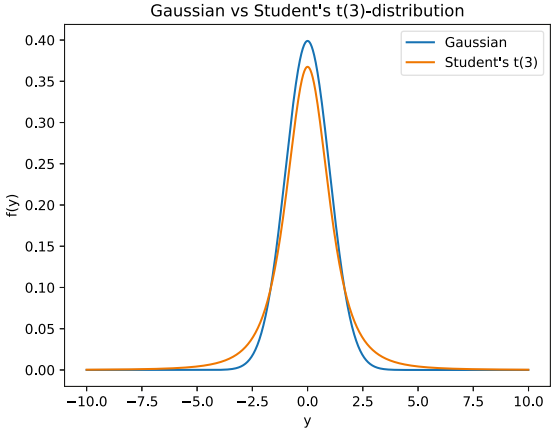

Fig. 5. Probability density function of the normal distribution (Gaussian with (loc, scale) = (0,1)) compared to the Student's t(3)-distribution with (loc, scale) = (0,1). The normal distribution has a thin tail compared to the Student's t(3)-distribution.

5.5. Effect of hyperparameters

Finally, we briefly study the impact of the choice of key hyperparameters of BiTCN. We re-ran a set of experiments for several choices of hyperparameters on the `Electricity` and `Traffic` datasets for which we display the results in Table 6. We highlight a number of interesting observations:

- The probabilistic forecasting performance of BiTCN seems relatively robust against a wide set of hyperparameter choices, as we commonly observe differences of 0%–5% in mean quantile loss when varying the hyperparameters. BiTCN would rank as a top performer among the competing methods in Table 2 for nearly all of the various hyperparameter settings.
- BiTCN's performance improves when the hidden dimension d_h (and thus the number of parameters) is increased. Conversely, the performance significantly degrades when the hidden dimension is reduced. However, an increased hidden size can result in both higher (`Electricity`) and lower (`Traffic`) training time and energy cost, which is due to the experiment requiring more (`Electricity`) and less (`Traffic`) epochs when increasing the hidden dimension. Also, the increased running time and energy cost are still less than those of TransformerConv (see Fig. 4), further demonstrating that our architecture achieves the same performance but does so more efficiently.
- The dropout rate p_d seems very important, as excluding it by setting it to zero results in a large performance hit for both datasets.
- It seems beneficial to increase the kernel size k of the convolutions, as performance generally increases when k is higher.

Table 6
Sensitivity of sMAPE, mean quantile loss, training time, and energy cost of BiTCN when varying key hyperparameters: the batch size b_s, the hidden size d_h, the kernel size k, the number of layers N, and the dropout rate p_d. For each row, only the value listed is changed with respect to the base case.

	No. parameters	b_s	d_h	k	N	p_d	sMAPE	mQ	Training time	Energy cost
`Electricity`										
Base case	49k	512	12	9	5	0.1	0.089	0.052	1.00	1.00
		256					0.091 (2.4%)	0.054 (2.7%)	0.82	0.75
		128					0.091 (2.1%)	0.053 (1.2%)	0.94	0.69
	166k		24				0.083 (−6.0%)	0.051 (−1.9%)	1.54	1.82
	19k		6				0.096 (8.1%)	0.054 (2.4%)	0.77	0.68
	39k			3	7		0.095 (7.4%)	0.055 (4.6%)	0.48	0.46
	42k			5	6		0.090 (1.6%)	0.053 (1.5%)	0.76	0.73
	50k			7	6		0.089 (0.3%)	0.053 (0.9%)	0.78	0.76
	55k			11	5		0.087 (−2.3%)	0.053 (0.2%)	1.00	1.02
						0.0	0.101 (13.2%)	0.054 (2.1%)	0.62	0.60
						0.2	0.093 (5.3%)	0.054 (2.2%)	0.71	0.71
						0.3	0.094 (5.6%)	0.053 (1.8%)	1.14	1.09
`Traffic`										
Base case	61k	512	12	9	5	0.1	0.127	0.091	1.00	1.00
		256					0.128 (0.6%)	0.090 (−1.0%)	1.14	1.05
		128					0.126 (−0.6%)	0.090 (−1.2%)	1.56	1.14
	178k		24				0.126 (−1.2%)	0.088 (−2.7%)	0.88	1.02
	31k		6				0.138 (8.3%)	0.099 (9.2%)	0.95	0.79
	51k			3	7		0.128 (1.0%)	0.092 (1.2%)	1.34	1.25
	54k			5	6		0.127 (−0.2%)	0.09 (−0.8%)	1.34	1.24
	62k			7	6		0.126 (−1.2%)	0.089 (−2.1%)	1.22	1.23
	67k			11	5		0.126 (−0.5%)	0.093 (3.0%)	0.91	0.95
						0.0	0.213 (67.7%)	0.143 (57.9%)	0.53	0.55
						0.2	0.13 (1.9%)	0.093 (2.5%)	1.05	1.06
						0.3	0.134 (5.8%)	0.096 (6.2%)	0.78	0.77

6. Conclusion and future work

In this work, we set out to find more parameter-efficient methods of probabilistic forecasting. We hypothesized that by (1) smartly leveraging future covariate information often available in real-world settings, (2) using a simple convolutional architecture, and (3) employing a Student's t(3)-distribution, it is possible to achieve state-of-the-art probabilistic forecasting performance compared to existing Transformer-based methods whilst requiring significantly fewer parameters. We found that our method, the bidirectional temporal convolutional network (BiTCN), confirmed these expectations, as we observed state-of-the-art forecasting effectiveness on a set of real-world benchmarks, even though BiTCN (i) uses an order of magnitude fewer parameters than the second-best Transformer-based method, (ii) requires at least 20% less energy, and (iii) requires about a quarter of the amount of memory to train the model on a GPU.

We believe that these findings qualify BiTCN as a generic probabilistic forecasting method among practitioners, due to its simplicity and computational efficiency.

Even though we observed the benefit of encoding future information to condition the current forecast on, the effect was relatively limited and even absent in some scenarios. For future work, we would therefore like to further investigate the benefit of this part of BiTCN by applying BiTCN in an industrial retail environment with thousands of products and stores, where a large set of historical data and future covariate information is available to condition a forecast on. An example of such a setting is the M5 forecasting dataset (Makridakis et al., 2020b). Secondly, we aim to investigate how our method scales to very long sequences (e.g., speech generation problems or high-frequency trading problems), where we expect to see more benefits of the forward-looking module. Finally, we intend to investigate creating a richer set of output distributions, in line with the recent work by Gasthaus, Benidis, Wang, Rangapuram, Salinas, Flunkert, and Januschowski (2019), which would further generalize our method by removing the choice of an output distribution.

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.

Acknowledgments

We thank the reviewers for their constructive feedback and help on improving our work.

This research was (partially) funded by the Hybrid Intelligence Center, a 10-year program funded by the Dutch Ministry of Education, Culture and Science through the Netherlands Organisation for Scientific Research.[2]

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.ijforecast.2021.11.011.

[2] https://www.hybrid-intelligence-centre.nl/

The code to reproduce the results of the experiments in this article can be found at https://github.com/elephaint/pedpf.

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., & Koyama, M. (2019). Optuna: a next-generation hyperparameter optimization framework. In *KDD '19, Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining* (pp. 2623–2631). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/3292500.3330701.

Alexandrov, A., Benidis, K., Bohlke-Schneider, M., Flunkert, V., Gasthaus, J., Januschowski, T., et al. (2020). GluonTS: probabilistic and neural time series modeling in python. *Journal of Machine Learning Research*, *21*(116), 1–6.

Assimakopoulos, V., & Nikolopoulos, K. (2000). The theta model: A decomposition approach to forecasting. In *The M3- Competition*: *International Journal of Forecasting*, In *The M3- Competition*: *16*(4), 521–530.http://dx.doi.org/10.1016/S0169-2070(00)00066-2,

Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. 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/1409.0473.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv:1803.01271 [cs].

Böse, J.-H., Flunkert, V., Gasthaus, J., Januschowski, T., Lange, D., Salinas, D., et al. (2017). Probabilistic demand forecasting at scale. *Proceedings of the VLDB Endowment*, *10*(12), 1694–1705. http://dx.doi.org/10.14778/3137765.3137775.

Box, G. E. P., & Pierce, D. A. (1970). Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American Statistical Association*, *65*(332), 1509–1526. http://dx.doi.org/10.2307/2284333.

Chen, Y., Kang, Y., Chen, Y., & Wang, Z. (2019). Probabilistic forecasting with temporal convolutional neural network. arXiv:1906.04397 [cs, stat].

Chen, T., Yin, H., Chen, H., Wu, L., Wang, H., Zhou, X., et al. (2018). TADA: trend alignment with dual-attention multi-task recurrent neural networks for sales prediction. In *2018 IEEE international conference on data mining* ICDM, (pp. 49–58). Singapore: IEEE, http://dx.doi.org/10.1109/ICDM.2018.00020.

Corporación Favorita Grocery Sales Forecasting (2018). https://kaggle.com/c/favorita-grocery-sales-forecasting.

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669. http://dx.doi.org/10.1016/j.ejor.2017.11.054.

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

Hendrycks, D., & Gimpel, K. (2018). Gaussian error linear units (GELUs). arXiv:1606.08415 [cs].

Hewamalage, H., Bergmeir, C., & Bandara, K. (2021). Recurrent neural networks for time series forecasting: current status and future directions. *International Journal of Forecasting*, *37*(1), 388–427. http://dx.doi.org/10.1016/j.ijforecast.2020.06.008.

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.

Holt, C. C. (2004). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, *20*(1), 5–10. http://dx.doi.org/10.1016/j.ijforecast.2003.09.015.

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

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

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

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., et al. (2017). LightGBM: a highly efficient gradient boosting decision tree. In *Advances in neural information processing systems. Vol. 30* (pp. 3146–3154). Curran Associates, Inc.

Kechyn, G., Yu, L., Zang, Y., & Kechyn, S. (2018). Sales forecasting using WaveNet within the framework of the kaggle competition. arXiv:1803.04037 [cs].

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International conference on learning representations ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference track proceedings*.

Lai, G., Chang, W.-C., Yang, Y., & Liu, H. (2018). Modeling long- and short-term temporal patterns with deep neural networks. In *SIGIR '18, The 41st International ACM SIGIR conference on research & development in information retrieval. Ann Arbor, MI, USA* (pp. 95–104). http://dx.doi.org/10.1145/3209978.3210006.

Laptev, N., Yosinski, J., Li, E. L., & Smyl, S. (2017). Time-series extreme event forecasting with neural networks at uber. In *ICML 2017 Time series workshop*.

Li, S., Jin, X., Xuan, Y., Zhou, X., Chen, W., Wang, Y.-X., et al. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Advances in neural information processing systems. Vol. 32* (pp. 5244–5254). Curran Associates, Inc.

Li, Y., Yu, R., Shahabi, C., & Liu, Y. (2018). Diffusion convolutional recurrent neural network: data-driven traffic forecasting. In *6th International conference on learning representations, ICLR 2018, vancouver, BC, Canada, April 30 - May 3, 2018, Conference track proceedings*. OpenReview.net.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020a). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, *36*(1), 54–74. http://dx.doi.org/10.1016/j.ijforecast.2019.04.014.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020b). The M5 accuracy competition: results, findings and conclusions.

Mariet, Z., & Kuznetsov, V. (2019). Foundations of sequence-to-sequence modeling for time series. In *The 22nd international conference on artificial intelligence and statistics* (pp. 408–417).

Montero-Manso, P., & Hyndman, R. J. (2021). Principles and algorithms for forecasting groups of time series: locality and globality. *International Journal of Forecasting*, http://dx.doi.org/10.1016/j.ijforecast.2021.03.004.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). PyTorch: an imperative style, high-performance deep learning library. In *Advances in neural information processing systems. Vol. 32* (pp. 8024–8035). Curran Associates, Inc.

Salimans, T., & Kingma, D. P. (2016). Weight normalization: a simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems. Vol. 29* (pp. 901–909). Curran Associates, Inc.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2019). DeepAR: probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, http://dx.doi.org/10.1016/j.ijforecast.2019.07.001.

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 *Advances in neural information processing systems. Vol. 32* (pp. 4838–4847). Curran Associates, Inc.

Strubell, E., Ganesh, A., & McCallum, A. (2020). Energy and policy considerations for modern deep learning research. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(09), 13693–13696. http://dx.doi.org/10.1609/aaai.v34i09.7123.

Taylor, S. J., & Letham, B. (2018). Forecasting at scale. *The American Statistician*, *72*(1), 37–45. http://dx.doi.org/10.1080/00031305.2017.1380080.

UCI Machine Learning Repository: ElectricityLoadDiagrams20112014 Data Set (2015). https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014.

UCI Machine Learning Repository: PEMS-SF Data Set (2011). https://archive.ics.uci.edu/ml/datasets/PEMS-SF.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., et al. (2016). WaveNet: a generative model for raw audio. In *The 9th ISCA speech synthesis workshop, sunnyvale, CA, USA, 13-15 September 2016* (p. 125). ISCA.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In *Advances in neural information processing systems. Vol. 30* (pp. 5998–6008). Curran Associates, Inc.

Web Traffic Time Series Forecasting (2017). https://kaggle.com/c/web-traffic-time-series-forecasting.

Wen, R., Torkkola, K., Narayanaswamy, B., & Madeka, D. (2018). A multi-horizon quantile recurrent forecaster. In *31st Conference on neural information processing systems*. Time series workshop. Long Beach, CA, USA.