BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 05/06 pp266–278
Volume 4, Number 4, December 2021
DOI: 10.26599/BDMA.2021.9020011

A Deep-Learning Prediction Model for Imbalanced Time Series Data Forecasting

Chenyu Hou, Jiawei Wu, Bin Cao, and Jing Fan*

Abstract: Time series forecasting has attracted wide attention in recent decades. However, some time series are imbalanced and show different patterns between special and normal periods, leading to the prediction accuracy degradation of special periods. In this paper, we aim to develop a unified model to alleviate the imbalance and thus improving the prediction accuracy for special periods. This task is challenging because of two reasons: (1) the temporal dependency of series, and (2) the tradeoff between mining similar patterns and distinguishing different distributions between different periods. To tackle these issues, we propose a self-attention-based time-varying prediction model with a two-stage training strategy. First, we use an encoder–decoder module with the multi-head self-attention mechanism to extract common patterns of time series. Then, we propose a time-varying optimization module to optimize the results of special periods and eliminate the imbalance. Moreover, we propose reverse distance attention in place of traditional dot attention to highlight the importance of similar historical values to forecast results. Finally, extensive experiments show that our model performs better than other baselines in terms of mean absolute error and mean absolute percentage error.

Key words: time series forecasting; imbalanced data; deep learning; prediction model

1 Introduction

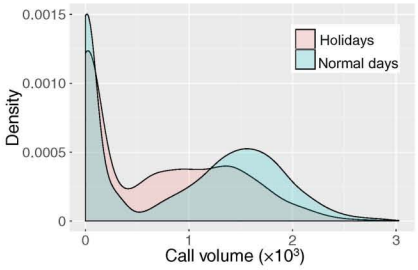
Time series forecasting has many real-world applications, such as call arrival forecasting[1], electricity power consumption forecasting[2], and air quality prediction[3]. In some scenarios, the time series may suffer from the imbalanced problem resulting from differences between normal and special periods. For example, the number of calls in call centers during holidays is significantly less than that on normal days. In these scenarios of imbalanced time series, existing methods, such as AutoRegressive Integrated Moving Average (ARIMA)[4–6], random forests[7], and Support Vector Machine (SVM)[8], bias toward normal periods (e.g., normal days), and thus suffering from accuracy degradation for special periods (e.g., holidays).

Take a true call traffic dataset as an example. Figure 1a represents the distributions of call volumes on holidays and normal days. The call volumes of holidays are generally lower. Then we conduct an empirical experiment to compare the prediction results of several methods for Spring Festival, as shown in Fig. 1b. Although these methods can roughly predict the trend of call arrivals, the predicted values are significantly higher than the true values because call arrivals decrease in holidays. However, these models fail to learn such a pattern because the size of holiday data only accounts for a very small portion of historical data.

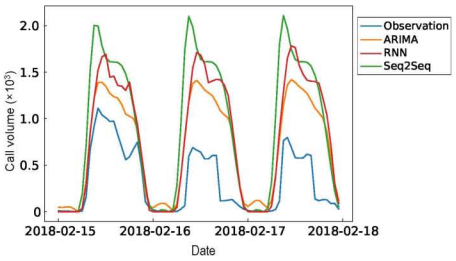
Motivated by this phenomenon, we aim to alleviate the imbalanced problem between normal periods and special periods and improve the prediction accuracy for special periods. This task is challenging because of the following two reasons:

• **Temporal dependency:** A crucial difference between time series forecasting and regression tasks is the temporal dependency. Time series forecasting

• Chenyu Hou, Jiawei Wu, Bin Cao, and Jing Fan are with the College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China. E-mail: {houcy, wujw, bincao, fanjing}@zjut.edu.cn.
* To whom correspondence should be addressed.
Manuscript received: 2021-05-21; accepted: 2021-06-10



(a) Distribution of call volumes

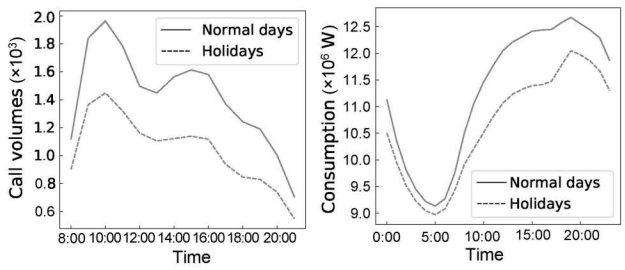


(b) Example of prediction

Fig. 1 An example of call arrival prediction.

models usually assume that a degree of correlation exists between successive values of the series. This assumption means that we cannot train an independent prediction model for special periods because it will break the temporal dependency. In addition, we cannot eliminate the influence of imbalanced data by simply oversampling the data of special periods because it will break the temporal dependency of time series.

• **Trade off between mining similar patterns and distinguishing different distributions:** Although time series on special periods show different distributions from those on normal periods, similar patterns exist between the two types of periods. For example, Fig. 2 shows that the trend of call traffics is similar on holidays and normal days. The same result can be observed in the trend of electricity consumption. Such patterns are hard to be learned merely based on data of special periods, whereas we can use the data of normal periods to facilitate the learning process for special periods.



(a) Call traffic (b) Electricity consumption

Fig. 2 Distribution differences throughout the day.

However, in this way, the model suffers from imbalanced data. How to balance the relationship between mining similar patterns and distinguishing different distributions on different types of periods is the second challenge.

Minimal attention has been paid to imbalanced time series forecasting[9]. Most methods are proposed to solve imbalanced classification problems, such as resampling[10,11] and cost-sensitive boosting[12,13]. Nevertheless, these techniques are infeasible for our problem because of the aforementioned challenges. A previous study focused on resampling methods for imbalanced time series forecasting[14]. However, this work treats rare values as the minority and focuses on the prediction accuracy of these rare cases, while we treat special periods as the minority and aim to improve the prediction accuracy of special periods.

In this paper, we propose a Self-attention based Time-Varying (STV) prediction model to overcome the aforementioned challenges and improve the prediction accuracy for special periods. First, we adopt an encoder–decoder module with a multi-head self-attention mechanism to capture the dependency of successive series and learn common patterns. Through the multi-head self-attention mechanism, the model can project the historical series into multiple subspaces and extract deep high-level features for prediction. Then, we adopt a decoder to predict successive multi-step results based on the encoding vector as primary results. By adopting this encoder–decoder module, the model can establish the relationship between input and prediction to extract common patterns of series between different periods. However, the outputs of the decoder cannot be regarded as the final forecasting results because they are biased toward the distribution of normal periods.

We propose a time-varying optimization module for the decoder results to distinguish different distributions of different periods. The time-varying optimization module takes the forecasting time as input. By extracting and embedding useful time information, this module outputs a scale factor $s \in [-1, 1]$ for each forecasting step to adjust the decoder results. In this way, the forecasting results of special periods can be adjusted to their own distributions rather than be dominated by data of normal periods. Finally, optimized results are regarded as the forecasting results.

In the encoder, we propose a novel attention mechanism, called "reverse distance attention". Instead of using dot attention as Transformer[15] did, we look for

moments with similar values for each time and assign higher self-attention scores to them. This mechanism enables the encoder to capture the periodicity of the time series and thus encoding the historical series better.

In addition, instead of training STV end-to-end directly, we propose a two-stage training strategy to improve the performance of STV. Following our proposed training mechanism, the encoder and the decoder are first trained to complete the forecasting tasks. Due to the imbalanced data, the first-stage forecasting results are more consistent with the distribution of normal periods. Second, we fix the weights of the encoder and the decoder and then train the time-varying optimization module individually to optimize the first-stage results.

Finally, we conduct extensive experiments based on a real call traffic dataset and an electricity consumption dataset. The experimental results prove the effectiveness of the proposed method.

The contribution of this paper can be summarized as follows:

- We find a novel problem that time series data are imbalanced between special and normal periods, leading to the performance degradation of existing models. As far as we know, this problem has not been addressed well in existing works.
- We propose an STV prediction model to deal with the imbalance of time series data. It not only mines common patterns between different periods, but also distinguishes different distributions of time series, thereby improving the prediction accuracy of special periods.
- To model the periodicity of time series, we design a "reverse distance attentio" mechanism. This attention mechanism can improve the prediction accuracy of STV. We also propose a simple yet effective two-stage training strategy to fine-tune the forecasting results.
- We conduct extensive experiments based on two real-world datasets and compare the performance of our model with five baselines. Experimental results demonstrate that STV outperforms baselines by an average of 18.64% and 20.87% on call traffic datasets, and 5.75% and 20% on the electricity consumption dataset, in terms of the Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) for special periods.

The rest of this paper is organized as follows. Section 2 outlines the preliminaries, including our problem definition and some important concepts. Section 3 describes our proposed model STV. Section 4 presents the evaluation results of our model on real-world datasets. Related works are presented in Section 5. Finally, we conclude our paper in Section 6.

2 Preliminary

In this section, we first formulate our multi-step time series forecasting problem and then present forecasting strategies for multi-step forecasting.

2.1 Problem formulation

Given the historical data of an observation variable $X = \{x_1, x_2, \ldots, x_t\}$, the univariate multi-step forecasting problem can be formulated as follows:

$$\{x_{t+1}, x_{t+2}, \ldots, x_{t+H}\} = F(x_1, x_2, \ldots, x_t) + \epsilon \quad (1)$$

where $F()$ is the prediction function approximated by the variable X. The function predicts the values of the series for the future time steps from $t+1$ to $t+H$, where H is the intended forecasting horizon. ϵ denotes the error associated with the function approximation. The target is finding an optimal function F, i.e., minimizing the error ϵ.

In model training, a lag parameter l is predefined to determine how much historical data are used to make forecasts. Only the latest l historical data are used in making forecasts. Considering lag parameters l, we can modify Eq.(1) as

$$\{x_{t+1}, x_{t+2}, \ldots, x_{t+H}\} = F(x_{t-l+1}, \ldots, x_{t-1}, x_t) + \epsilon \quad (2)$$

2.2 Forecasting strategies

Such H-step ($H > 1$) forecasts can be preduced in three ways.

Iterated forecasting: Similar to one-step time series forecasting problems, a "many-to-one" function $F()$ is trained,

$$\hat{y}_{t+1} = F(x_{t-l+1}, \ldots, x_{t-1}, x_t) \quad (3)$$

After training the forecasting model, the following forecasting results can be obtained by iteratively making one-step forecasts using previously forecasted values. It can be formulated as follows:

$$\begin{aligned} \hat{y}_{t+2} &= F(x_{t-l+2}, \ldots, x_t, \hat{y}_{t+1}), \\ &\vdots \\ \hat{y}_{t+H} &= F(x_{t+H-l+1}, \ldots, \hat{y}_{t+H-2}, \hat{y}_{t+H-1}) \end{aligned} \quad (4)$$

However, this forecasting strategy may suffer from high variance due to error accumulation in individual forecasts. This scenario means that low performance can be observed over longer time horizons H.

Multi-model forecasting: The second paradigm is multi-model forecasting, where we train H "many-to-one" functions, i.e., $F_1(), \ldots, F_H()$,

$$\begin{aligned}\hat{y}_{t+1} &= F_1(x_{t-l+1}, \ldots, x_{t-1}, x_t),\\ \hat{y}_{t+2} &= F_2(x_{t-l+1}, \ldots, x_{t-1}, x_t),\\ &\vdots\\ \hat{y}_{t+H} &= F_H(x_{t-l+1}, \ldots, x_{t-1}, x_t)\end{aligned} \tag{5}$$

Then, we can make H-step forecasts by using these functions. However, these models are trained independently, which cannot guarantee the relevance between predicted values. At the same time, the cost of training is high.

Multiple Input Multiple Output (MIMO) forecasting: In this paradigm, we train a "many-to-many" model,

$$\hat{y}_{t+1}, \ldots, \hat{y}_{t+H} = F(x_{t-l+1}, \ldots, x_{t-1}, x_t) \tag{6}$$

Then, given l-lag historical data, we can directly make H-step forecasts. Compared with the multi-model forecasting paradigm, only one model is required instead of H different models. Unlike the iterated forecasting paradigm, error accumulation does not occur in individual forecasts. Therefore, we build our proposed model based on the MIMO forecasting strategy.

3 Self-Attention Based Time-Varying Prediction Network

In this section, we first introduce the framework of our proposed model. Then, we introduce the modules of our model in detail, namely, the encoder–decoder module, the multi-head attention mechanism, and the time-varying optimization module.

3.1 Framework of STV

Figure 3 presents the framework of STV. STV adopts the encoder–decoder paradigm and appends a time-varying optimization module to optimize forecasting results. STV takes the observation values of the previous l timestamp as input and then passes it to 1×1 convolution layer for upsampling. The intuition behind this phenomenon is that time series can be decomposed into a trend, circle, and noise. We want the 1×1 convolution layer to model this decomposition. After up-sampling, an encoder consisting of multiple sub-encoder modules is used to extract high-level features of historical series by leveraging the self-attention mechanism and residual networks. On the basis of extracted features, the decoder outputs elementary forecasting results. So far, normal and special periods show no significant difference. Without additional actions being taken, this model will suffer from imbalanced data, causing poor performance on special periods. To solve this problem, we add a time-varying optimization module after the decoder. In the time-varying optimization module, we leverage embedding results of forecasting time and a dense network to extract timestamp-wise features. The timestamp-wise features enable the network to optimize the decoder results by leveraging a gate mechanism. Finally, forecasting results are generated by incorporating residuals and optimized results.

3.2 Encoder and decoder modules

Encoder: The encoder module aims to extract high-level features over the original input series. It is composed of a stack of s sub-encoder modules, where s is a hyperparameter. Each sub-encoder has two layers. The first one is a multi-head self-attention mechanism, whereas the second is a simple, fully connected feed-forward layer. In addition, a residual connection[16] is employed around each layer, followed

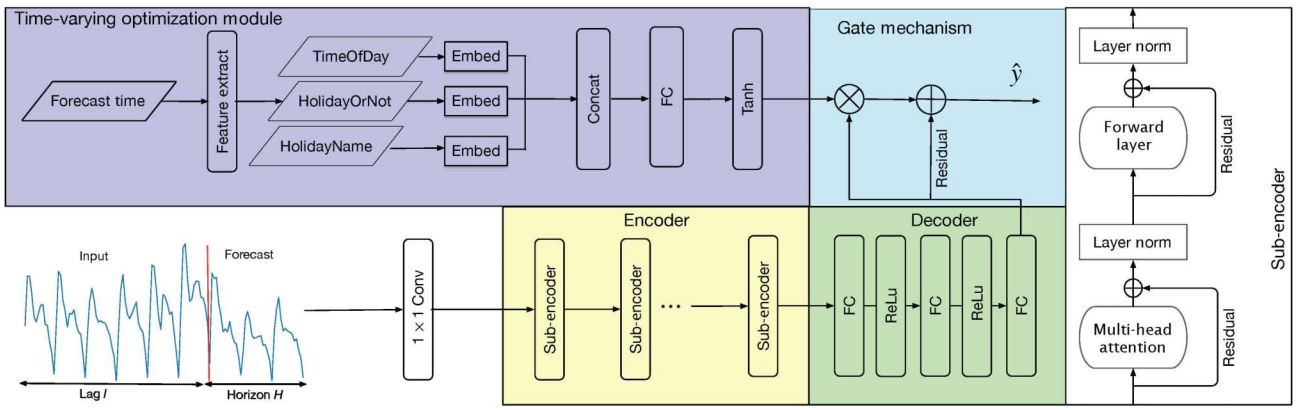


Fig. 3 Framework of STV.

by layer normalization[17]. Considering that the internal dimensions of each time series are related, while different samples in a batch have minimal relevance, we adopt layer normalization instead of batch normalization. To facilitate residual connections, we maintain the input and output dimensions of each sub-encoder equal to d_e. In our experiments, we set the dimension d_e to 8.

Suppose the input of a sub-encoder is $Z \in \mathbf{R}^{l\times d_e}$, then it can be formulated as follows:

$$Z^{(1)} = LayerNorm(Z + \text{Multi-head Self-Attn}(Z)) \tag{7}$$

$$Z^{(out)} = LayerNorm(Z^{(1)} + \text{FeedForward}(Z^{(1)})) \tag{8}$$

where $Z^{(out)}$ is the output of the sub-encoder.

Decoder: The decoder consists of three fully connected layers. Except for the last layer, a ReLu activation function[18] exists after each fully connected layer. We use fully connected layers instead of the recurrent neural network, because the outputs of each fully connected layer are independent, thereby avoiding error accumulation along with timelines. The decoder eventually outputs an H-dimension vector $d \in \mathbf{R}^H$. The output d can be regarded as the primary forecasting result. However, it is time independent and thus is more likely biased to the distribution of normal periods. To prevent this drawback, we further adopt a time-varying optimization module to modify the outputs, which will be discussed later.

Based on the context $c \in \mathbf{R}^{l\times d_e}$ produced by the encoder, the decoder can be formulated as follows:

$$z^{(1)} = ReLu(W_1^{\mathrm{T}} c + b_1) \tag{9}$$

$$z^{(2)} = ReLu(W_2^{\mathrm{T}} z^{(1)} + b_2) \tag{10}$$

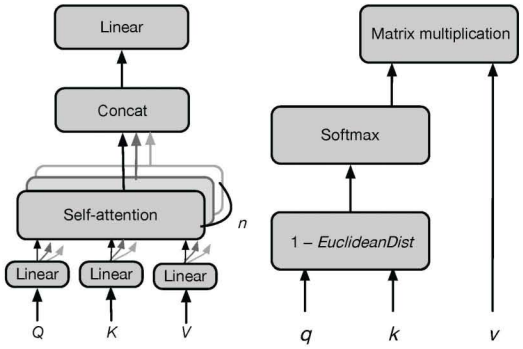
$$d = W_3^{\mathrm{T}} z^{(2)} + b_3 \tag{11}$$

where weight matrixes $W_1 \in \mathbf{R}^{(l\times d_e)\times(2l\times d_e)}$, $W_2 \in \mathbf{R}^{(2l\times d_e)\times(l\times d_e)}$, and $W_3 \in \mathbf{R}^{(l\times d_e)\times H}$, and bias vectors $b_1 \in \mathbf{R}^{2l\times d_e}$, $b_2 \in \mathbf{R}^{l\times d_e}$, and $b_3 \in \mathbf{R}^H$ are learnable parameters of the fully-connected layers.

3.3 Multi-head attention

We adopt multi-head self-attention architecture in the encoder to model the correlation from historical data. Self-attention is good at learning intrinsic relations from historical data, whereas the multi-head structure allows the model to learn useful information from different representation subspaces than single self-attention.

Multi-head attention: Figure 4a depicts the multi-head attention architecture. Instead of performing single self-attention, n self-attention modules are utilized. To learn comprehensive information from different



(a) Multi-head attention (b) Self-attention

Fig. 4 Attention architecture.

subspaces, the inputs Q, K, and V are first linearly projected to $n \times d_q$, $n \times d_k$, and $n \times d_v$ dimensions by passing them into different, learnable linear layers, where n is the number of self-attention modules. Then projection results are divided into n parts and then passed to multi-head self-attention modules to calculate the results in parallel. The outputs of each self-attention module are concatenated.

Two things need to be explained. First, Q, K, and V in Fig. 4a are copies of the outputs Z of last layers. Therefore, we actually calculate the attention between each time step of the input itself. This explains why we call this attention mechanism self-attention. Second, for easy interpretation, we maintain the dimension of q, k, and v in each single head equal to 1, i.e., $d_q = d_k = d_v = 1$. In this way, q, k, and v in every single head can be treated as decompositions of the original time series, and each head represents a decomposed space.

The multi-head attention mechanism can be formulated as below:

$$\text{MultiHead}(Q, K, V) = Concat(head_1, \ldots, head_n)W^L,$$
$$\text{s.t. } head_i = Attention(Q\times W_i^Q, K\times W_i^K, V\times W_i^V) \tag{12}$$

where $W_i^Q \in \mathbf{R}^{d_e\times n}$, $W_i^K \in \mathbf{R}^{d_e\times n}$, $W_i^V \in \mathbf{R}^{d_e\times n}$, and $W^L \in \mathbf{R}^{n\times d_e}$ are learnable parameters.

Self-attention: Figure 4b depicts the structure of self-attention. Self-attention can be treated as mapping a query and a set of key-value pairs to an output, where Q, K, and V are the same as inputs.

Most of the time series data are periodic. For example, the traffic volumes of each day display similar patterns. Therefore, similar values in the historical series are more valuable for forecasting. To capture the similarity between them, in this part, we propose a novel attention mechanism named "reverse distance attention". As mentioned before, the dimensions of Q_i, K_i, and V_i are equal to 1. Therefore, the Euclidean distance can

be used to measure the difference between queries and keys. We use $1 - EuclideanDist(\cdot,\cdot)$ to represent the similarity between two values. To ensure the sum of the weights equals 1, we should conduct a softmax operation to obtain the final attention scores,

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{l} \exp(e_{ik})} \tag{13}$$

where l is the length of time series, and

$$e_{ij} = 1 - |q_i - k_j| \tag{14}$$

this score semantically represents the importance of each historical value.

After obtaining attention scores, the final result can be calculated using the following equation:

$$o_i = \sum_{k=1}^{m} a_{ik} \times v_k, \quad i = 1, 2, \ldots, l \tag{15}$$

3.4 Time-varying optimization module

The time series of normal periods account for most of the historical data. Thus, if our model only receives the historical time series as inputs, then the outputs of the decoder will be more likely biased to the distribution of normal periods. To overcome this problem, we introduce a time-varying optimization module to optimize the forecasting results of the decoder. The input of this module is forecasting time, whereas the outputs are step-wise scale factors s ($\in [-1, 1]$) indicating how to adjust the primary results of the decoder. For example, if the output is –0.2 at time step t, then the final forecasting result should be 0.2 times smaller than the output of the decoder at time t. By further scaling the prediction results for each time step, we can adjust the prediction results of special periods to its own distribution rather than being dominated by the distribution of normal periods.

The structure of the time-varying optimization module is displayed in the upper part of Fig. 3. First, we extract three features from forecast time, namely, the time of the day, a holiday or not, and holiday name. All of them are categorical features. Therefore, we transform each feature into a low-dimensional vector by feeding them into different embedding layers separately, and then concatenate those embeddings to construct an embedding vector e. The embed dimensions are determined by

$$emb\_dim = \left\lfloor \frac{cat\_num + 1}{2} \right\rfloor \tag{16}$$

where *cat_num* is the number of categories. For example, if we want to embed "hour" information, the embedding dimension is 12.

Once we obtain the embedding vector, we feed it into a fully connected layer and activate the result using the Tanh function to determine the scale factor s. Finally, the Hadamard product of outputs of the decoder with scale factor s plus residual connections yields the eventual forecast results $\hat{y}$. A forecasting result $\hat{y}_t$ at a particular time step t can be calculated as follows:

$$\hat{y}_t = (1 + s_t) \times d_t \tag{17}$$

3.5 Model training and prediction

Instead of following the end-to-end manner to train our model, we propose a **two-stage training** manner that enables STV to overcome imbalanced data and thus improving forecasting accuracy. Figure 5 presents the main idea of our training strategy. Instead of training the whole model directly, we first train a sub-model that is composed of an encoder and a decoder (including 1×1 convolution layer). Then, we copy the updated weights to STV. We finally train the time-varying optimization module independently by fixing the weights of the encoder and the decoder in STV. Intuitively, we first use the encoder–decoder module to predict results that are biased toward normal periods. Then, we use a time-varying optimization module to adjust the prediction results by incorporating forecast time information and eliminate bias.

The proposed STV model is trained via backpropagation to minimize mean squared error between the forecasting results and the ground truth. The pseudo-code of the training and prediction process is presented in Algorithm 1. First, we obtain the training,

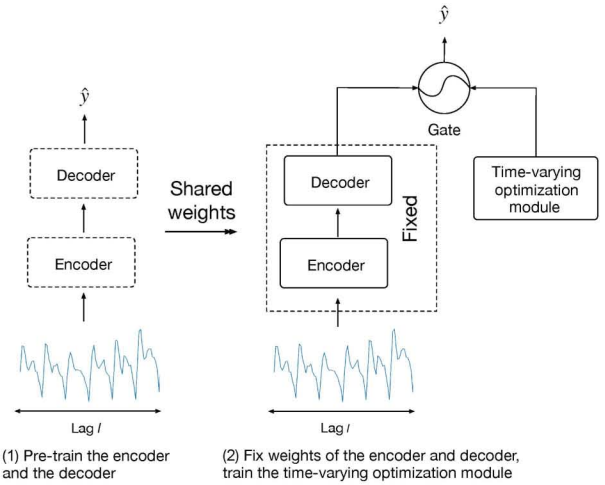


Fig. 5 Two-stage training strategy.

Algorithm 1 STV training and prediction

Input: Historical time series $X = x_1, \ldots, x_T$; Forecast horizon H; Lag l

Output: Predicted values of the future H steps

1 $D_{total} \leftarrow \varnothing$;
2 **for** each t in [l, $T - H$] **do**
3 Append $[x_{t-l+1}, \ldots, x_t; x_{t+1}, \ldots, x_{t+H}]$ to D_{total}
4 Divide D_{total} into D_{train}, D_{valid}, and D_{test};
5 sub-model $\leftarrow$ Copies of encoder and decoder of STV;
6 **for** i in [1, 1000] **do**
7 Update sAb-mAdel Ay Ainimizing Ahe Aoss D_{train};
8 $loss_{valid} \leftarrow$Aalculate Aoss D_{valid};
9 **if** $loss_{valid}$ does not decrease in consecutive 50 epochs **then**
10 Stop the first-stage training process;
11 Copy weights of sub-model to STV and fix them;
12 **for** i in [1, 1000] **do**
13 Update STV by minimizing the loss D_{train};
14 $loss_{valid} \leftarrow$calculate Aoss D_{valid};
15 **if** $loss_{valid}$ does not decrease in consecutive 50 epochs **then**
16 Stop the second-stage training process;
17 $\hat{y} \leftarrow$ STV$(x_{T-l+1}, \ldots, x_T)$;
18 **return** $\hat{y}$

validation, and test sets (Lines 1–4). Then, we pre-train the sub-model that excludes the time-varying module of STV (Lines 5–10). During training, we adopt the early stop strategy to prevent overfitting (Lines 9 and 10). Once the first-stage training is stopped, we share the weights of the sub-model with STV and fix them (Line 11). Then, the time-varying module is trained on the second stage, following the same training process (Lines 12–16). Finally, we input the latest l historical data to the STV model and predict the H-step results (Line 17).

4 Experiment

4.1 Experimental settings

Datasets: We collect the following four datasets from two domains to evaluate the effectiveness of the proposed model:

• **CallTraffic:** We collect three real-world call traffic datasets from China Telecom, including Hangzhou (HZ), Taizhou (TZ), and Lishui (LS) cities. These datasets contain call volume records at hourly granularity from January 2017 to May 2019. Considering that call volumes in the midnight and early morning are very low, which is not important for evaluation, we select call volumes from 7:00 to 21:00 every day for experiments. In addition, we divide periods into holidays and normal days according to the Chinese public holidays to study the performance of different methods.

• **Electricity Consumption (ElecCONS):** The electricity consumption dataset is collected from Commonwealth Edison company. This dataset records hourly power consumption data from 2014-01-01 to 2018-08-02. Considering that this dataset is collected in the United States, we divide the time according to American holidays. This dataset is available on https://www.kaggle.com/robikscube/hourly-energy-consumption.

Evaluation metrics: We use MAE and MAPE for evaluation. They are defined as follows:

$$\text{MAE} = \frac{1}{N}\sum_{t=1}^{N}|y_i - \hat{y}_i| \tag{18}$$

$$\text{MAPE} = \frac{1}{N}\sum_{t=1}^{N}\frac{|y_i - \hat{y}_i|}{y_i} \tag{19}$$

where y_i and $\hat{y}_i$ are the ground truth and the corresponding predicted value, and N is the total number of all available ground truth. MAE is more affected by large values, whereas MAPE receives more punishments from small values.

In our experiments, holidays are regarded as special periods while normal days are regarded as normal periods. To compare the performance of the model on normal and special periods, we calculate the MAE and MAPE from three perspectives: overall (denoted as MAE and MAPE), for normal days (denoted as MAE_N and MAPE_N), and for holidays (denoted as MAE_H and MAPE_H).

Baselines: The methods in our comparative evaluation are listed as follows.

• **ARIMA**[6]: It is a well-known statistic model for forecasting time series.

• **LSTM**[19]: It is a better RNN architecture that could alleviate the problem of gradient vanishing.

• **Seq2Seq**[20]: It uses an RNN to encode the input sequences into a feature representation and another RNN to make predictions iteratively.

• **LSTNet**[21]: The convolution neural network and the skip-recurrent neural network are used in LSTNet to extract short-term local dependency patterns. To model long-term patterns, LSTNet additionally exploits the traditional autoregressive model.

• **N-Beats**[22]: N-Beats is the state-of-the-art method for univariate time series forecasting problems. It is a deep neural architecture based on backward and forward residual links and a very deep stack of fully-connected

layers.

Model details: We use min-max normalization to normalize the target values into [0, 1], and use one-hot encoding to transform date-time features. In the evaluation, we rescale the predicted values back to normal values. The learning rate is 0.001, and the batch size is 1024. We leverage Adam[23] for stochastic gradient descent. We adopt the early stop strategy where training will be terminated when the validation loss does not decrease during consecutive 50 epochs. We conduct a grid search over all tuneable hyperparameters for the STV model. In specific, the stack number of encoders is chosen from {1, 2, 3}, the number of heads h is chosen from {2, 4, 6, 8}, the upsampling factor d_e is chosen from $\{2, 3, \ldots, 8\}$, and the hidden size of self-attention d_h is chosen from $\{2^5, 2^6, \ldots, 2^{10}\}$. For other baselines, we select the best setting by tuning their hidden size ranging from $\{2^5, 2^6, \ldots, 2^{10}\}$. The experiment platform is equipped with Intel Core i9-9940X CPU, 128 GB RAM, and Nvidia RTX 2080Ti GPU. We implement all neural network models with Pytorch in Ubuntu.

4.2 Results on CallTraffic

4.2.1 Model comparison

In this section, we compare the performance of our model against baselines. To compare our model comprehensively, we conduct experiments at different forecasting horizons, namely, $H_1 = 14$, $H_2 = 42$, and $H_3 = 98$. These three horizons represent "short", "middle", and "long" forecasting lengths, which can reflect the performance of our model in different scenarios.

Table 1 reports the overall MAP and MAPE of different methods in the three cities. We have the following observation: (1) When the forecasting horizon is 14 or 42, the STV model outperforms all baselines in the three cities. Compared with the runner-up method, STV achieves 5.02% and 7.21% improvement and in terms of MAE when $H = 14$ and $H = 42$, respectively. This result indicates that our proposed model is good at dealing with imbalanced time series for short or middle forecasting lengths. (2) When the forecasting length is 98, LSTNet achieves the best performance in HZ and TZ while STV outperforms other methods in LS. Compared with the runner-up method, their improvements are 0.09%, 3.55%, and 2.93%. This result implies that LSTNet and STV are competitive for long-term forecasting. (3) LSTM performs poorly in all scenarios. By diving into the training process, we find that the training is stopped immediately because of the early stop strategy. Thus, LSTM cannot converge, resulting in poor performance. This phenomenon demonstrates that LSTM is unsuitable for solving multi-step forecasting problems.

Table 1 Overall MAE and MAPE of different methods.

City	Method	H=14		H=42		H=98	
		MAE	MAPE	MAE	MAPE	MAE	MAPE
HZ	ARIMA	125.21	0.11	145.60	0.13	193.45	0.19
	LSTM	309.47	0.35	322.59	0.36	346.80	0.39
	Seq2Seq	110.69	0.10	153.42	0.14	206.46	0.19
	LSTNet	121.13	0.10	148.69	0.13	**166.48**	**0.15**
	NBeat	118.65	0.10	150.22	0.14	179.22	0.16
	STV	**104.81**	**0.09**	**131.53**	**0.12**	166.63	**0.15**
TZ	ARIMA	121.66	0.11	145.99	0.13	187.55	0.18
	LSTM	284.06	0.30	292.83	0.31	321.43	0.34
	Seq2Seq	105.89	0.09	132.02	0.11	148.46	0.13
	LSTNet	104.64	0.09	126.50	0.11	**131.48**	**0.11**
	NBeat	102.88	**0.08**	122.70	0.10	161.55	0.14
	STV	**99.47**	**0.08**	**111.50**	**0.09**	136.32	**0.11**
LS	ARIMA	100.90	0.11	122.27	0.14	158.94	0.19
	LSTM	229.29	0.31	242.09	0.33	261.26	0.36
	Seq2Seq	90.18	0.09	119.80	0.13	205.06	0.23
	LSTNet	93.99	0.10	93.98	0.10	116.69	0.13
	NBeat	83.35	0.09	98.39	0.11	123.09	0.14
	STV	**77.98**	**0.08**	**91.32**	**0.09**	**113.26**	**0.12**

To study the performance of STV on different types of periods, we compare the MAE and MAPE on normal days and holidays. The results are shown in Table 2. STV achieves the best performance on holidays regardless of the forecasting length, indicating that STV has a great advantage in the prediction for holidays. In detail, STV outperforms the runner-up method by an average of 15.50%, 19.18%, and 21.25% in terms of MAE in the three cities.

For normal days, STV is not dominant in all cases. STV achieves the best performance in terms of MAE_N only when the forecasting horizon is 14 or 42. In addition, the improvements are not as big as those of holidays (by an average of 7.07%, 4.97%, and 2.28% in the three cities). When the horizon is 98, LSTNet outperforms STV by 5.07%, 8.95%, and 1.78% in the three cities, explaining why the overall accuracy of LSTNet is higher than that of STV when $H = 98$ in HZ and TZ.

The accuracy of normal days and that of holidays significantly differ for each model, which also agrees with our empirical studies that models suffer from imbalanced data. However, the gap could be

Table 2 MAE and MAPE on normal days and holidays.

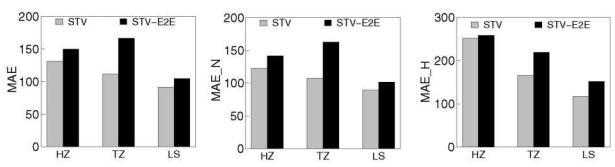
City	Method	$H = 14$				$H = 42$				$H = 98$			
		MAE_N	MAPE_N	MAE_H	MAPE_H	MAE_N	MAPE_N	MAE_H	MAPE_H	MAE_N	MAPE_N	MAE_H	MAPE_H
HZ	ARIMA	116.29	0.10	243.74	0.26	139.24	0.12	270.90	0.29	175.30	0.17	432.92	0.51
	LSTM	297.25	0.33	471.99	0.59	307.60	0.34	521.27	0.65	329.97	0.37	568.85	0.71
	Seq2Seq	103.62	0.09	204.65	0.21	138.13	0.12	356.15	0.38	186.98	0.17	463.31	0.50
	LSTNet	111.99	0.09	242.74	0.25	135.14	0.12	328.35	0.35	**152.81**	**0.13**	346.82	0.37
	NBeat	111.20	0.09	217.75	0.23	136.80	0.12	328.20	0.36	164.52	0.15	373.17	0.41
	STV	**98.73**	**0.08**	**185.76**	**0.19**	**122.41**	**0.11**	**252.49**	**0.27**	160.98	0.14	**241.05**	**0.25**
TZ	ARIMA	115.24	0.10	207.05	0.19	142.63	0.13	190.56	0.19	176.59	0.16	332.09	0.35
	LSTM	275.62	0.29	396.20	0.44	282.74	0.30	426.68	0.48	310.11	0.33	470.81	0.53
	Seq2Seq	101.19	0.08	168.38	0.15	126.10	0.10	210.63	0.19	138.56	0.12	279.04	0.27
	LSTNet	100.73	0.08	156.59	0.14	117.73	0.10	242.76	0.23	**124.06**	**0.10**	229.41	0.21
	NBeat	98.63	0.08	159.40	0.14	115.43	0.10	219.16	0.21	152.60	0.13	279.58	0.27
	STV	**95.72**	**0.08**	**149.29**	**0.13**	**107.37**	**0.09**	**166.30**	**0.15**	136.24	0.11	**137.33**	**0.12**
LS	ARIMA	97.96	0.11	139.87	0.17	120.65	0.13	143.70	0.19	151.71	0.18	254.33	0.34
	LSTM	224.61	0.30	291.57	0.42	235.78	0.32	325.88	0.46	254.05	0.35	356.33	0.50
	Seq2Seq	87.26	0.09	128.94	0.15	112.36	0.12	218.41	0.26	195.79	0.21	327.27	0.40
	LSTNet	89.66	0.09	151.60	0.17	89.43	0.10	154.33	0.18	**109.42**	**0.12**	212.61	0.26
	NBeat	80.15	0.08	125.90	0.15	93.55	0.10	162.55	0.20	118.99	0.13	177.16	0.21
	STV	**76.57**	**0.08**	**96.75**	**0.11**	**89.35**	**0.09**	**117.40**	**0.14**	111.41	0.12	**137.67**	**0.17**

narrowed using the STV model. This phenomenon also demonstrates that STV can effectively alleviate the problem that the forecasting results of holidays are biased toward the distribution of normal days.

4.2.2 Training strategy

In this section, we conduct experiments to demonstrate the effect of the proposed two-stage strategy for STV. We compare the performance of STV with the variant STV-E2E, which is trained following the end-to-end manner. The results of MAE of three aspects can be found in Fig. 6.

STV outperforms STV-E2E from three perspectives, indicating that the proposed two-stage training strategy can improve the performance of STV significantly. Compared with that of STV-E2E, the MAE of STV decreases by an average of 19.5%, 19.8%, and 16.5% from three perspectives. The improvement of MAE_N is greater than that of MAE_H. This result implies that STV-E2E can improve the prediction accuracy of holidays to a certain extent, but can reduce the prediction accuracy of normal days.



(a) Overall MAE (b) MAE on normal days (c) MAE on holidays

Fig. 6 Performance of different training mechanisms.

We analyze the training process to study the importance of fixed weights of different modules in STV. We remove different components of STV from the weight sharing step to study their performance. The results of $H = 42$ are shown in Table 3. All of the variants perform worse than STV in the three cities, indicating that removing any module from the weight sharing step leads to performance degradation.

4.2.3 Key components

In this section, we study the effects of each component of our method. To this end, we compare STV with its different variants:

- **STV-NConv:** The 1×1 convolution layer is excluded from our model, which can help reveal the significance of the convolution layer.
- **STV-NTV:** We simply remove the time-varying optimization module from our model to study the effect of this module.
- **STV-Dot:** We replace the attention mechanism with dot product attention to demonstrate the effect of

Table 3 Inside the two-stage training.

Removed module	HZ		TZ		LS	
	MAE	MAPE	MAE	MAPE	MAE	MAPE
Conv	253.76	0.21	113.15	0.09	223.07	0.23
Encoder	139.22	0.13	128.41	0.11	99.49	0.11
Decoder	133.72	0.12	138.05	0.12	105.22	0.11
No module removed	**131.53**	**0.12**	**111.50**	**0.09**	**91.32**	**0.09**

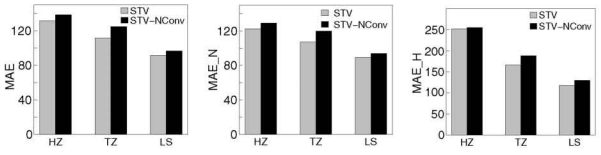
the "reverse distance attention" mechanism.

Take the forecasting horizon $H = 98$ as an example; we study the MAE of these variants in the three cities. The MAPE shows the same trend as the MAE. Thus, we omit the analysis of MAPE.

1×1 convolution layer. Figure 7 plots the MAE of STV and STV-NConv. STV outperforms STV-NConv by 6.9%, 7.0%, and 7.5% in terms of MAE, MAE_N, and MAE_H, respectively. This result reveals that the 1×1 convolution layer can improve the prediction accuracy of STV.

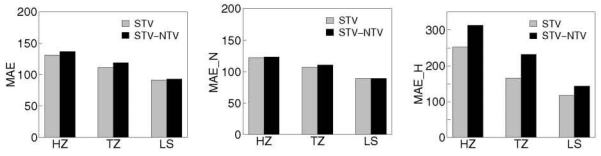
Time-varying optimization module. Figure 8 plots the MAE of STV and STV-NTV. First, MAE decreases in the three cities when the model is equipped with the time-varying optimization module. In specific, MAE drops by 4.0%, 1.2%, and 21.9% from three aspects. This result indicates that the time-varying optimization module can improve the forecasting accuracy of STV. Second, compared with MAE on normal days, the MAE on holidays decreases more significantly. This result demonstrates that the time-varying optimization module can optimize the forecasting results of holidays effectively. Without the time-varying optimization module, the prediction results of the model will be biased toward the data distribution of normal days. Therefore, the prediction results of the holidays can still be optimized.

Reverse distance attention. As shown in Fig. 9, STV outperforms STV-Dot in terms of MAE, MAE_N, and MAE_H (improve by 5.6%, 4.0%, and 17.1% on average, respectively). The improvement of STV demonstrates the effectiveness of the proposed attention mechanism.



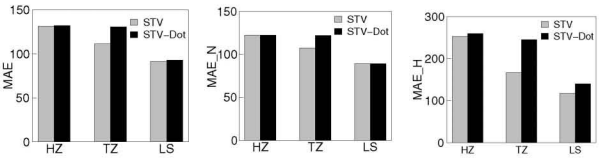
(a) Overall MAE (b) MAE on normal days (c) MAE on holidays

Fig. 7 MAE comparison on the convolution layer.



(a) Overall MAE (b) MAE on normal days (c) MAE on holidays

Fig. 8 MAE comparison of STV and STV-NTV.



(a) Overall MAE (b) MAE on normal days (c) MAE on holidays

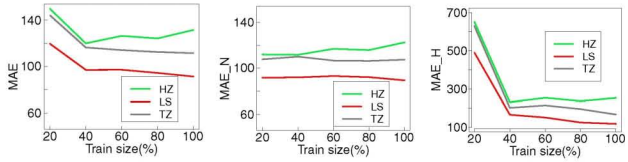
Fig. 9 MAE comparison of STV and STV-Dot.

4.2.4 Training size

In this section, we study the effectiveness of training data volume by selecting the latest training data from the original training set according to different ratios. Figure 10 plots the MAE results for $H = 42$. As the training data volume increases, the performance of STV initially improves and then fluctuates when the data volume exceeds 40%. This observation reveals that increasing time series data can properly improve prediction accuracy, but excessive outdated data may degrade the performance of the model. Excessive outdate historical data may affect the model to learn recent data distribution and thus decreasing the performance of the model. Increasing training data volume exerts minimal influence on the prediction accuracy of normal days while improves the accuracy of holidays significantly from 20% to 40%. This result can be ascribed to the fact that the historical data of holidays only account for a small amount of the whole data. Therefore, increasing training data volume can add more holiday data to learn holiday patterns. However, data of normal days are sufficient for model training even if only 20% of historical data are used.

4.2.5 Training efficiency

Figure 11 presents the training efficiency of different models, where bars represent training time and lines represent numbers of model parameters. Although STV contains many parameters, it is easy to train. The training time is about 200 s, which is acceptable in real-life deployments.



(a) Overall MAE (b) MAE on normal days (c) MAE on holidays

Fig. 10 MAE comparison of different training data volumes.

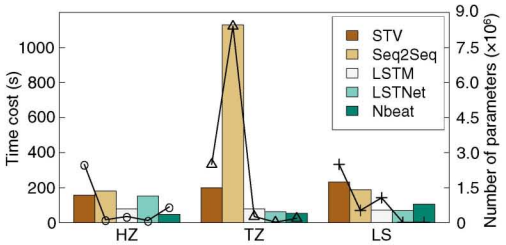


Fig. 11 Training efficiency of different models.

4.3 Results on ElecCONS

Table 4 shows model performances using the electricity consumption dataset. In this experiment, we set the forecasting horizon to 24. One important trait of the ElecCONS dataset is that the electricity consumption in the early morning of holidays is similar to that on normal days, which results in less difference between holidays and normal days versus that in the call traffic prediction task. Nonetheless, STV outperforms the best baseline by 6.99%, 6.91%, and 5.74% in terms of MAE, MAE_N, and MAE_H, respectively. Thus, STV not only works on the large-difference scenario, but is also adaptable to time series whose difference between holidays and normal days is smaller. However, the improvement of the holidays only slightly influences the overall performance, because American holidays only account for a very small portion of a year.

5 Related Work

In this section, we first investigate the related work about univariant time series forecasting problems from the following three groups: including statistical methods, machine learning methods, and neural network based methods. Then, because our model alleviates the impact of imbalanced data, we discuss some works in the field of imbalanced learning.

5.1 Univariant time series forecasting

Statistical methods establish statistical models according to the historical data, where the most famous method is ARIMA. Bianchi et al.[24] used ARIMA models with intervention analysis to forecast telemarketing call arrivals. Contreras et al.[25] used ARIMA to predict the next-day electricity price. However, ARIMA performs poorly when the forecasting time bucket becomes long[26].

Many machine learning models are applied by treating time series forecasting problems as regression problems to find an appropriate mapping between input features and prediction values. Lu et al.[27] proposed to use the support vector machine to forecast financial time series. Dudek[7] proposed to use random forests to predict the short-term electricity load.

Table 4 Model comparison on the ElecCONS dataset.

Method	MAE	MAPE	MAE_N	MAPE_N	MAE_H	MAPE_H
ARIMA	570.29	0.05	564.95	0.05	675.91	0.07
LSTM	1281.17	0.12	1273.13	0.12	1614.56	0.13
Seq2Seq	527.10	0.05	522.29	0.05	726.28	0.06
LSTNet	499.01	0.05	496.03	0.05	622.51	0.05
NBeat	515.10	0.05	513.14	0.05	596.65	0.05
STV	**464.09**	**0.04**	**461.72**	**0.04**	**562.40**	**0.04**

Recently, neural network based methods have been developed to solve this problem. Kong et al.[28] used an LSTM network to predict residential load. Rangapuram et al.[29] proposed a deep-state space model for time series prediction. Yi et al.[3] proposed a distributed fusion network to predict air quality.

Neither of these works considers that the time series patterns are different in different periods, which may influence their prediction accuracy. In our work, we divide the historical data into normal and special periods and learn them, thereby improving the prediction accuracy significantly.

5.2 Imbalanced learning

Existing imbalanced learning methods can be divided into three groups: sampling-based methods, cost-sensitive-based methods, and neural network based methods.

A conventional way to solve the imbalanced problem is through resampling. Resampling-based methods balance the data by increasing or decreasing the number of labels in the class. In specific, Chawla et al.[11] proposed the synthetic minority oversampling technique to increase the data of the class containing a small number of labels. Liu et al.[10] proposed BalanceCascade to under-sample the data of the class, which prevents information loss due to random undersampling methods. Batista et al.[30] proposed an oversampling method combined with data cleaning techniques to overcome category reordering.

For the second group, cost-sensitive-based methods aim to adjust the weights of samples in different classes. In this way, existing machine learning algorithms can still work well even during data imbalance. Sun et al.[12] proposed three cost-sensitive boosting methods for imbalanced learning, which introduce cost items into the weight updating strategy of Adaboost. Elkan et al.[31] used the Laplace smoothing method of the probability estimate and the Laplace pruning technique to improve the cost insensitivity in the decision tree.

For the last group, Wang et al.[32] used the mean false error loss function as they experimented with classifying imbalanced data with deep MLPs. Lin et al.[33] proposed a neural network that can overcome the extreme class imbalance in object detection problems. Wang et al.[34] used a cost-sensitive deep neural network method to

detect hospital readmissions.

However, all of the above methods aim to solve the imbalanced data in classification problems. Our work focuses on the imbalanced data in the regression problem. The closest work to us is the Ref. [14], which studied the application of resampling strategies with imbalanced time series data. However, they only focused on rare occasions. They claimed that these rare occasions are important to end-users. Different from it, our work focused on the imbalanced data in different types of periods rather than rare values.

6 Conclusion

We aim to solve the time series forecasting problem with the consideration of data imbalance between special and normal periods. To this end, we propose a novel STV model. First, we leverage the encoder–decoder module to mine the common patterns of time series between different periods. To address the imbalanced problem, we propose a time-varying module that takes the temporal features of forecasting timestamps as inputs and outputs scale factors to optimize the decoder results. Finally, we use two datasets to verify the effectiveness of our proposed method. Our approach advances baselines by an average of 18.64% and 20.87% on the CallTraffic dataset and 5.74% and 20% on the ElecCONS dataset in terms of MAE and MAPE for special periods (i.e., holidays). In the future, we will explore improving the model structure and accuracy for long-term forecasting.

Acknowledgment

This research was partially sponsored by the National Key R&D Program of China (No. 2018YFB1402800) and the Fundamental Research Funds for the Provincial Universities of Zhejiang (No. RF-A2020007).

References

[1] B. Cao, J. W. Wu, L. C. Cao, Y. S. Xu, and J. Fan, Long-term and multi-step ahead call traffic forecasting with temporal features mining, *Mobile Netw. Appl.*, vol. 25, no. 2, pp. 701–712, 2020.

[2] X. R. Shao, C. S. Kim, and P. Sontakke, Accurate deep model for electricity consumption forecasting using multi-channel and multi-scale feature fusion CNN–LSTM, *Energies*, vol. 13, no. 8, p. 1881, 2020.

[3] X. W. Yi, J. B. Zhang, Z. Y. Wang, T. R. Li, and Y. Zheng, Deep distributed fusion network for air quality prediction, in *Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, London, UK, 2018, pp. 965–973.

[4] S. L. Ho and M. Xie, The use of ARIMA models for reliability forecasting and analysis, *Computers & Industrial Engineering*, vol. 35, nos. 1&2, pp. 213–216, 1998.

[5] C. H. Liu, S. C. Hoi, P. L. Zhao, and J. L. Sun, Online ARIMA algorithms for time series prediction, in *Proc. 30th AAAI Conf. Artificial Intelligence*, Phoenix, AR, USA, 2016, pp. 1867–1873.

[6] G. E. P. Box and D. A. Pierce, Distribution of residual autocorrelations in autoregressive-integrated moving average time series models, *J. Am. Stat. Assoc.*, vol. 65, no. 332, pp. 1509–1526, 1970.

[7] G. Dudek, Short-term load forecasting using random forests, in *Intelligent Systems'2014*, D. Filev, J. Jablkowski, J. Kacprzyk, M. Krawczak, I. Popchev, L. Rutkowski, V. Sgurev, E. Sotirova, P. Szynkarczyk, and S. Zadrozny, eds. Cham, Germany: Springer, 2015, pp. 821–828.

[8] N. I. Sapankevych and R. Sankar, Time series prediction using support vector machines: A survey, *IEEE Comput. Intell. Mag.*, vol. 4, no. 2, pp. 24–38, 2009.

[9] B. Krawczyk, Learning from imbalanced data: Open challenges and future directions, *Prog. Artif. Intell.*, vol. 5, no. 4, pp. 221–232, 2016.

[10] X. Y. Liu, J. X. Wu, and Z. H. Zhou, Exploratory undersampling for class-imbalance learning, *IEEE Trans. Syst., Man, Cybern., Part B (Cybern.)*, vol. 39, no. 2, pp. 539–550, 2009.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, SMOTE: Synthetic minority over-sampling technique, *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[12] Y. M. Sun, M. S. Kamel, A. K. C. Wong, and Y. Wang, Cost-sensitive boosting for classification of imbalanced data, *Pattern Recognit.*, vol. 40, no. 12, pp. 3358–3378, 2007.

[13] S. H. Khan, M. Hayat, F. Sohel, and R. Togneri. Cost-sensitive learning of deep feature representations from imbalanced data, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 8, pp. 3573–3587, 2017.

[14] N. Moniz, P. Branco, and L. Torgo, Resampling strategies for imbalanced time series forecasting, *Int. J. Data Sci. Anal.*, vol. 3, no. 3, pp. 161–181, 2017.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need, in *Proc. 31st Int. Conf. Neural Information Processing Systems*, Long Beach, CA, USA, 2017, pp. 6000–6010.

[16] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, Deep residual learning for image recognition, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016, pp. 770–778.

[17] J. L. Ba, J. R. Kiros, and G. E. Hinton, Layer normalization, arXiv preprint arXiv: 1607.06450, 2016.

[18] V. Nair and G. E. Hinton, Rectified linear units improve restricted Boltzmann machines, in *Proc. 27th Int. Conf. Machine Learning*, Haifa, Israel, 2010, pp. 807–814.

[19] S. Hochreiter and J. Schmidhuber, Long short-term memory, *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] I. Sutskever, O. Vinyals, and Q. V. Le, Sequence to sequence learning with neural networks, in *Proc. 27th Int. Conf. Neural Information Processing Systems*, Montreal, Canada, 2014, pp. 3104–3112.

[21] G. K. Lai, W. C. Chang, Y. M. Yang, and H. X. Liu, Modeling long- and short-term temporal patterns with deep neural networks, in *Proc. 41st Int. ACM SIGIR Conf.*

Research & Development in Information Retrieval, Ann Arbor, MI, USA, 2018, pp. 95–104.
[22] B. N. Oreshkin, D. Carpov, N. Chapados, and Y. Bengio, N-BEATS: Neural basis expansion analysis for interpretable time series forecasting, arXiv preprint arXiv:1905.10437, 2019.
[23] D. P. Kingma and J. L. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv: 1412.6980, 2014.
[24] L. Bianchi, J. Jarrett, and R. C. Hanumara, Improving forecasting for telemarketing centers by ARIMA modeling with intervention, *Int. J. Forecasting*, vol. 14, no. 4, pp. 497–504, 1998.
[25] J. Contreras, R. Espinola, F. J. Nogales, and A. J. Conejo, ARIMA models to predict next-day electricity prices, *IEEE Trans. Power Syst.*, vol. 18, no. 3, pp. 1014–1020, 2003.
[26] E. D. Feigelson, G. J. Babu, and G. A. Caceres, Autoregressive times series methods for time domain astronomy, *Front. Phys.*, vol. 6, p. 80, 2018.
[27] C. J. Lu, T. S. Lee, and C. C. Chiu, Financial time series forecasting using independent component analysis and support vector regression, *Decis. Support Syst.*, vol. 47, no. 2, pp. 115–125, 2009.
[28] W. C. Kong, Z. Y. Dong, Y. W. Jia, D. J. Hill, Y. Xu, and Y. Zhang, Short-term residential load forecasting based on LSTM recurrent neural network, *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 841–851, 2019.
[29] S. S. Rangapuram, M. Seeger, J. Gasthaus, L. Stella, Y. Y. Wang, and T. Januschowski, Deep state space models for time series forecasting, in *Proc. 32nd Int. Conf. Neural Information Processing Systems*, Montréal, Canada, 2018, pp. 7796–7805.
[30] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, A study of the behavior of several methods for balancing machine learning training data, *ACM SIGKDD Explorat. Newsl.*, vol. 6, no. 1, pp. 20–29, 2004.
[31] C. Elkan, The foundations of cost-sensitive learning, in *Proc. 17th Int. Joint Conf. Artificial Intelligence*, Seattle, WA, USA, 2001, pp. 973–978.
[32] S. J. Wang, W. Liu, J. Wu, L. B. Cao, Q. X. Meng, and P. J. Kennedy, Training deep neural networks on imbalanced data sets, in *Proc. Int. Joint Conf. Neural Networks*, Vancouver, Canada, 2016, pp. 4368–4374.
[33] T. Y. Lin, P. Goyal, R. Girshick, K. M. He, and P. Dollár, Focal loss for dense object detection, in *Proc. IEEE Int. Conf. Computer Vision*, Venice, Italy, 2017, pp. 2999–3007.
[34] H. S. Wang, Z. C. Cui, Y. X. Chen, M. Avidan, A. B. Abdallah, and A. Kronzer, Predicting hospital readmission via cost-sensitive deep learning, *IEEE/ACM Trans. Comput. Biol. Bioinform.*, vol. 15, no. 6, pp. 1968–1978, 2018.

Chenyu Hou received the BEng degree from Zhejiang University of Technology, Hangzhou, China in 2016. He is now a PhD student at Zhejiang University of Technology. He has published many papers on international journals and conferences, like *TKDE*, *CIKM*, and *WWWJ*, etc. His research interests are database and data mining.

Jiawei Wu received the BEng degree from Zhejiang University of Technology, Hangzhou, China in 2017. He is now a PhD student at Zhejiang University of Technology. He has published several papers on international journals and conferences, like *TOIT*, *MONET*, etc. His research interest is natural language processing.

Bin Cao received the PhD degree in computer science from Zhejiang University, China in 2013. He then worked as a research associate at Hongkong University of Science and Technology and Noah's Ark Lab, Huawei. He joined Zhejiang University of Technology, Hangzhou, China in 2014, and is now an associate professor at the College of Computer Science and Technology, Zhejiang University of Technology. He has published more than 30 papers on many international authoritative journals and conferences, including *TKDE*, *TSC*, and *CIKM*. His research interests include data mining and natural language processing.

Jing Fan received the BEng, MEng, and PhD degrees in computer science from Zhejiang University, China in 1990, 1993 and 2003, respectively. She is now a professor at the College of Computer Science and Technology, Zhejiang University of Technology, China. She is a director of China Computer Federation (CCF). She has published more than 100 papers on many international journals and conferences, including *TSC*, *TKDE*, and *IEEE Virtual Reality*. Her current research interests include service computing, virtual reality, and intelligent interaction.