Multivariate Time Series Forecasting Using Multiscale Recurrent Networks With Scale Attention and Cross-Scale Guidance

Qiang Guo, *Member, IEEE*, Lexin Fang, Ren Wang, and Caiming Zhang

***Abstract*— Multivariate time series (MTS) forecasting is considered as a challenging task due to complex and nonlinear interdependencies between time steps and series. With the advance of deep learning, significant efforts have been made to model long-term and short-term temporal patterns hidden in historical information by recurrent neural networks (RNNs) with a temporal attention mechanism. Although various forecasting models have been developed, most of them are single-scale oriented, resulting in scale information loss. In this article, we seamlessly integrate multiscale analysis into deep learning frameworks to build scale-aware recurrent networks and propose two multiscale recurrent network (MRN) models for MTS forecasting. The first model called MRN-SA adopts a scale attention mechanism to dynamically select the most relevant information from different scales and simultaneously employs input attention and temporal attention to make predictions. The second one named as MRN-CSG introduces a novel cross-scale guidance mechanism to exploit the information from coarse scale to guide the decoding process at fine scale, which results in a lightweight and more easily trained model without obvious loss of accuracy. Extensive experimental results demonstrate that both MRN-SA and MRN-CSG can achieve state-of-the-art performance on five typical MTS datasets in different domains. The source codes will be publicly available at https://github.com/qguo2010/MRN.**

***Index Terms*— Attention mechanism, cross-scale guidance, multiscale decomposition, recurrent neural networks (RNNs), time series forecasting.**

Manuscript received 25 November 2021; revised 2 December 2022 and 12 March 2023; accepted 15 October 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 61873145 and Grant 61802229, in part by the NSFC Joint Fund with Zhejiang under Key Project under Grant U1609218, in part by the Natural Science Foundation of Shandong Province for Excellent Young Scholars under Grant ZR2017JL029, in part by the Special Funds of Taishan Scholars Project of Shandong Province under Grant tstp20221137, and in part by the Science and Technology Innovation Program for Distinguished Young Scholars of Shandong Province Higher Education Institutions under Grant 2019KJN045. *(Corresponding author: Qiang Guo.)*

Qiang Guo is with the School of Computer Science and Technology and the Shandong Provincial Key Laboratory of Digital Media Technology, Shandong University of Finance and Economics, Jinan 250014, China (e-mail: guoqiang@sdufe.edu.cn).

Lexin Fang and Ren Wang are with the School of Software, Shandong University, Jinan 250100, China (e-mail: fanglexin@mail.sdu.edu.cn; rwang@mail.sdu.edu.cn).

Caiming Zhang is with the School of Software, Shandong University, Jinan 250100, China, and also with the Shandong Provincial Laboratory of Future Intelligence and Financial Engineering, Yantai 264005, China (e-mail: czhang@sdu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2023.3326140.

Digital Object Identifier 10.1109/TNNLS.2023.3326140

I. Introduction

TIME series data, as a sequence of observations taken sequentially in time order [1], are ubiquitous in a wide range of real-world applications such as economic and financial analysis, supply chain forecasting, electric power consumption study, audio and video processing, traffic flow prediction, and other similar areas. In most cases, the collected data are multivariate [2]. For instance, electricity consumption forecasting involves the times series of consumption records collected from multiple clients. These consumption data are correlated due to the shared transmission infrastructure and weather. Multivariate time series (MTS) forecasting aims to accurately predict future value by jointly considering intra- and inter-series correlations among historical data. However, due to complex underlying generation process, the observed data are nonstationary and nonlinear, which makes MTS forecasting very challenging. Especially, complex dynamic interdependencies among multiple variables further complicate the forecasting task.

Many methods have been developed to address the above challenges in the past, e.g., statistical model-based methods [3], matrix factorization-based methods [4], machine learning-based methods [5], deep learning-based methods [6], [7], and hybrid methods [8]. Among these methods, vector autoregression (VAR) is one of the most popular model for MTS forecasting, which generalizes the classical autoregressive (AR) statistical model by allowing for more than one evolving variable. Although VAR shows significant effectiveness in modeling the linear dependencies, it fails to capture the high-order dependencies of MTS data [9]. To address this issue, Gultekin and Paisley [10] treat MTS as a matrix, and predict future values by solving a low-rank matrix factorization problem. Indeed, tensor factorization provides an effective technique for extracting valuable structure information from tensorial data [11]. In [9], tensor factorization was integrated with AR model to obtain two different multilinear AR forecasting models. Despite their success, such models are known to suffer from poor computational efficiency.

Recently, deep neural networks have received a substantial amount of attention from the research community. In particular, the recurrent neural networks (RNNs), e.g., long short-term memory (LSTM) [12] and gated recurrent unit (GRU) [13], have shown promising performance in MTS forecasting due to their ability to model nonlinear interdependencies.

By integrating both the input attention and temporal attention mechanisms into RNN, Qin et al. [14] proposed a dual-stage attention-based RNN (DA-RNN), which can adaptively select the most relevant deriving variables to make more accurate predictions. After that, various temporal attention-based forecasting models including long- and short-term time-series network (LSTNet) [8], temporal pattern attention based LSTM (TPA-LSTM) [2], and dual self-attention network (DSANet) [15] have been proposed to improve the prediction accuracy. However, most prior works pay little attention to capture the scale dependencies among multiscale features of MTS.

In this article, we incorporate the attention mechanism into an LSTM by two different schemes to capture the scale dependencies and adaptively fuse the features from different scales. We also fine-tune the wavelet filter banks used in multiscale decomposition to fit the data by end-to-end training. These changes allow us to significantly improve the prediction performance. In summary, the main contributions of this article lie in the following four aspects.

1) We seamlessly integrate multiscale analysis into deep learning frameworks to build scale-aware recurrent networks, and explore the scale attention and cross-scale guidance mechanisms. To the best of our knowledge, this is the first work to adopt multiscale recurrent networks (MRNs) with these two mechanisms for time series forecasting.
2) We propose an MRN to predict future samples of time series, which simultaneously employs input attention, temporal attention, and scale attention. Due to the explicit usage of these attentions, this model obtains more accurate prediction.
3) We propose a novel MRN with cross-scale guidance mechanism, which exploits the information from coarse scale to guide the decoding process at fine scale. This design brings us a lightweight and more easily trained model without loss of accuracy.
4) We conduct extensive experiments on five benchmark datasets for MTS forecasting. The experimental results and comparisons demonstrate the superior performance and efficiency of our proposed models.

The remainder of this article is organized as follows. In Section II, we first discuss the related studies. Then, Section III briefly describes the problem of MTS forecasting and reviews the stationary wavelet transform (SWT) used in multiscale time series decomposition. In Section IV, we present the proposed *MRN with scale attention* (MRN-SA) and *MRN with cross-scale guidance* (MRN-CSG) models that are equipped with the scale attention module and the cross-scale guidance module, respectively. Section V reports our experimental results that confirm the significant advantage of our models. Finally, we conclude this work in Section VI.

II. Related Work

A large number of MTS forecasting methods have been proposed in the literature. These methods can be roughly classified into two broad categories: traditional methods and deep learning-based methods. We briefly review some representative works in this section. For more details on this topic, we refer the reader to the recent comprehensive survey [16] and the references therein.

A. Traditional Methods

Statistics-based methods are classical time series forecasting models. They assume that the time series are generated by a linear aggregation of random shocks. A representative method is the AR moving average (ARMA) model, i.e., a combination of the AR and moving average (MA) models. Its extension to nonstationary time series forecasting, known as AR integrated MA (ARIMA) that incorporates a differencing technique to eliminate the influences of trend components of data, is one of the most popular linear models due to its great flexibility. Another one is exponential smoothing (also known as ETS) that relies on the weighted average of past observations [17]. In fact, ARIMA and ETS can be represented in state-space models (SSMs) that provide a unified and powerful framework for time series modeling [18]. Various SSMs have been presented to deal with more complex tasks [19]. However, this type of methods relies heavily on parametric settings and some assumptions of the underlying data distribution [20], which make such models unsuitable for many practical usages. In addition, they are originally limited to linear univariate time series and do not scale well to the multivariate setting. To cope with MTS forecasting, VAR was proposed, which generalizes the univariate AR-based model by allowing for more than one evolving variable. But it fails to capture the nonlinearity of MTS data and also suffers from poor computational efficiency.

Unlike the above models that require certain assumptions on time series data, singular spectrum analysis (SSA) is a useful nonparametric technique of time series analysis and forecasting [21]. It embeds the original time series that approximately satisfy some linear recurrent formula (LRF) into a Hankel matrix (also called trajectory matrix), and produces a prediction of the future values of time series via the LRF coefficients that are calculated from a spectral decomposition of the Hankel matrix. To simultaneously handle multiple time series, Patterson et al. [22] extended the SSA to a multivariate SSA, in which the joint trajectory matrix of MTS is block-Hankel rather than simple Hankel. However, these SSA-based methods usually require access to noise-free and fully observed data. As a result, their performance is highly sensitive to data contamination such as corruptions, outliers, and missing values. To address this limitation, Gillard and Usevich [23] formulated the problem of forecasting a given time series as the structured low-rank matrix completion of a Hankel matrix. A prediction of the future values of time series can be obtained by minimizing the nuclear norm of this Hankel matrix. In [4], a temporal regularized matrix factorization framework was developed for MTS forecasting, which uses a novel AR regularizer on the temporal factor matrix to model temporal dependency among time series data. Jing et al. [9] integrated tensor factorization with AR model to derive two different multilinear AR forecasting models. Despite the success of factorization based models, it is ineffective in modeling highly nonlinear time series, and also

very computationally expensive for applications that require real-time performance [24].

B. Deep Learning-Based Methods

Driven by the success of deep learning in various tasks, many recent works leverage deep neural networks to address the aforementioned limitations. In particular, RNNs have achieved remarkable performance due to their flexibility in capturing nonlinear relationship. As pointed out in [25], for reconstructing state-space trajectories of dynamic systems, RNNs are universal approximators in theory, thus making them suitable candidates for modeling nonlinear time series. Thus, they have been successfully employed for various prediction tasks [26]. However, the standard RNN suffers from the vanishing gradient and exploding gradient problems [27], which limits its capability in capturing temporal long-term dependencies in sequential data.

To mitigate this problem, LSTM and GRU have been designed with gating units. Meanwhile, incorporating attention mechanisms into RNNs also has achieved great success in many natural language processing tasks [28], [29]. The attention mechanism allows the network to dynamically select the relevant parts of the input. Hence, it is natural to consider the gated RNNs and attention mechanisms for time series prediction. In [14], by equipping an LSTM with the temporal attention and input attention layers, a DA-RNN was developed to select the relevant driving series for making more accurate predictions. Unlike DA-RNN, DSANet [15] employs two parallel convolution modules, i.e., global temporal convolution and local temporal convolution, to, respectively, capture global and local temporal patterns. It then applies a self-attention module to model the dependencies among variables. Cinar et al. [30] presented a position-based attention model to capture patterns of (pseudo-) periods in time series. Besides, Lai et al. [8] proposed an LSTNet for MTS forecasting, which utilizes the convolutional layer to capture the local dependencies among input variables and the recurrent layer to model complex long-term dependencies. It also introduces a recurrent-skip structure based on temporal attention to discover very long-term dependency patterns. Several variants of this framework, such as [2], [31], and [32], have been proposed.

Besides, as shown in [33], integrating wavelet transform into RNNs enables us to boost the prediction performance. Based on the transformed data, RNNs can effectively capture multiscale patterns of time series [34], [35]. However, most of the existing works decompose a time series into multiple sequences with different scales by using the wavelet transform with fixed filter banks, and indiscriminately combine the features learned by RNNs to produce a prediction of the time series. To the best of our knowledge, there is no work focusing on the scale dependencies among multiscale features of MTS. To fill this gap, we propose two network models by integrating scale attention and cross-scale guidance mechanisms with an LSTM, and fine-tune the wavelet filter banks to fit the data by end-to-end training. As a result, our models can capture the scale dependencies and adaptively fuse the features from different scales, achieving highly competitive performance.

III. Preliminaries

In this section, we first introduce the notations used in this article, and then formulate the problem of MTS forecasting. This is followed by giving a brief review of the SWT, which motivates us to decompose a time series into multiscale components.

A. Problem Formulation

In this work, we are interested in the task of MTS forecasting. Given a series of fully observed time series signals, we represent them in a matrix form as $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T) \in \mathbb{R}^{n\times T}$, where T is the length of the input window size, and $\mathbf{x}_t = (x_t^1, x_t^2, \ldots, x_t^n)^\top \in \mathbb{R}^n$ denotes the time series at time t, where n is the number of driving series. Assuming that historical observations of all series are available, MTS forecasting is to estimate a series of future signals from the known $\mathbf{X}$, which can be formulated as learning a function that maps $\mathbf{X}$ to the value of the next time stamp $T+\tau$, i.e.,

$$\mathbf{y}_{T+\tau} = f(\mathbf{X}) \tag{1}$$

where $f(\cdot)$ is typically a nonlinear mapping function that we aim to learn, and τ is the desirable horizon ahead of the current time stamp. $\mathbf{y}_{T+\tau}$ may either be the driving series $\mathbf{x}_{T+\tau}$ or be any one of the variables $x_{T+\tau}^i (i = 1, \ldots, n)$. In this article, we only focus on the latter case, i.e., $\mathbf{y}_{T+\tau} = x_{T+\tau}^i$. In fact, for the former, we can predict $\mathbf{x}_{T+\tau}$ by repeatedly learning n individual mapping functions.

B. Stationary Wavelet Transform

The wavelet transform is a powerful tool for signal processing, which can provide analysis of a signal in both time and frequency domains. Thus the discrete wavelet transform (DWT), a discrete version of the wavelet transform, has been widely used to extract multiscale time-frequency features from a time series. However, the decimation step in the standard DWT results in different lengths of wavelet coefficient sequences at different scales, making it difficult to relate information at a given time point at the different scales [36]. To overcome this problem, it is common to resort to the SWT, also known as *algorithme à trous* [37], in which the length of wavelet coefficients computed at each scale equals the length of the input time series. Furthermore, SWT has a valuable role in the exploration and spectral analysis of nonstationary time series [38]. These properties enable SWT to model a nonstationary time series and predict its future value.

The basic idea of the SWT is to modify the wavelet filters at each scale by inserting a zero between every adjacent pair of elements of the filter [37]. This leads to an overdetermined representation of the original sequence. More specifically, for the low-pass filter (LPF) $l[n]$ used in the DWT, we denote as $l_j[n]$ the filter obtained by inserting $2^j - 1$ zeros between each sample of $l[n]$. Similarly, for the high-pass filter (HPF) $g[n]$ of the DWT, $g_j[n]$ can be derived by the same way. Given the lowest scale J, the SWT decomposition of the input signal a_0

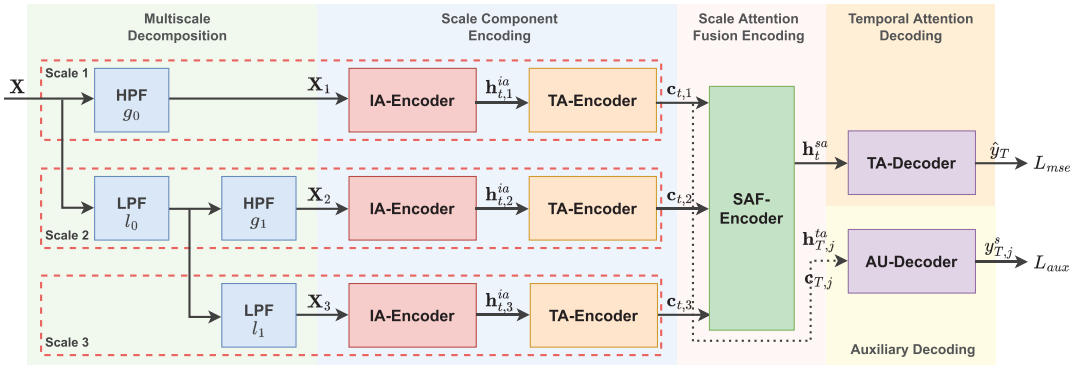

Fig. 1. Architecture of the proposed MRN-SA. HPF and LPF denote the high-pass filter and low-pass filter, respectively, which are initialized by *db2* wavelet filters and then fine-tuned on the training data.

is defined as [39]

$$a_{j+1}[n] = \big(\bar{l}_j * a_j\big)[n] = \sum_k l_j[k] a_j\big[n + 2^j k\big]$$
$$d_{j+1}[n] = \big(\bar{g}_j * a_j\big)[n] = \sum_k g_j[k] a_j\big[n + 2^j k\big] \quad (2)$$

where $*$ is the convolution operator, $\bar{l}_j[n] = l_j[-n]$, $\bar{g}_j[n] = g_j[-n]$, and $j = 0, \ldots, J-1$. The signal can be reconstructed by the following synthesis operation [39]:

$$a_j[n] = \frac{1}{2}\big[\big(\tilde{l}_j * a_{j+1}\big)[n] + \big(\tilde{g}_j * d_{j+1}\big)[n]\big]$$

where $\tilde{l}_j$ and $\tilde{g}_j$ are the corresponding synthesis filters of the analysis filters l_j and g_j, respectively.

For an input sequence, the low-frequency components a_j preserve the information of its long-term temporal patterns, and the high-frequency components d_j preserve the short-term temporal patterns. Multiple frequency components achieved by the SWT are helpful for capturing inherent different temporal patterns that can improve the prediction performance. Inspired by this, we employ multiple RNNs to simultaneously extract features from different scales, and then fuse them to generate a prediction, achieving significant performance improvement.

IV. Proposed Methods

In this section, we propose two novel MRN architectures that perform time series prediction by exploiting multi-scale hidden features extracted from multiple attention-based LSTMs. The first one is called MRN-SA, which adopts a scale attention mechanism to adaptively fuse hidden states across all scales. While the second one is referred as MRN-CSG, which gradually fuses the hidden features from coarse scale to fine scale. Extensive experiments show that both models obtain high accuracy for MTS forecasting on several public datasets.

A. MRN With Scale Attention

Fig. 1 provides an architecture overview of the proposed MRN-SA with three scale components, which works in an end-to-end manner. The MRN-SA consists of four main parts: the *multiscale decomposition* that hierarchically decomposes an MTS into several subseries with different frequencies, the *scale component encoding* that extracts the hidden state of each subseries by successively applying an input attention encoder (IA-Encoder) and a temporal attention encoder (TA-Encoder), the *scale attention fusion encoding* that explicitly fuses the hidden states extracted from different scales by a scale attention fusion encoder (SAF-Encoder), and the *temporal attention decoding* that generates a prediction by a temporal attention decoder (TA-Decoder). In the following, we present some details of the network components used in the MRN-SA.

1) Multiscale Decomposition: As discussed before, the short-term prediction of time series depends more on its high-frequency pattern while the long-term prediction tends to focus more on the low-frequency pattern. Therefore, we use the SWT to decompose MTS into a group of subseries with different scales, and learn multiple frequency patterns from these subseries for prediction.

Specifically, we perform an HPF g and an LPF l on each time series $\mathbf{x}^k$ of $\mathbf{X}$ along time axis to obtain a fine-scale component $\mathbf{X}_{\text{fs}}$ and a coarse-scale component $\mathbf{X}_{\text{cs}}$, respectively. This can be formulated as

$$\mathbf{X}_{\text{fs}}(k,:) = \bar{g} * \mathbf{X}(k,:)$$
$$\mathbf{X}_{\text{cs}}(k,:) = \bar{l} * \mathbf{X}(k,:) \quad (3)$$

where $\mathbf{X}(k,:) = \mathbf{x}^k = (x_1^k, \ldots, x_T^k)$ represents the kth row of $\mathbf{X}$, and $k = 1, \ldots, n$. The filtering process can be carried out recursively on the previous coarse-scale component. For ease of description, we refer to the components at different scales as $\mathbf{X}_j$, as shown in Fig. 1.

Note that different from existing wavelet-based approaches for time series forecasting, which use the HPF and LPF with fixed filter coefficients, our methods first apply the commonly used *db2* wavelet to initialize these filter coefficients, and then fine-tune them on the training data in an end-to-end manner.

2) Scale Component Encoding: The attention mechanism is significantly important to select relevant information for the task of MTS forecasting. The input attention is able to adaptively select the relevant driving series at each time step, while the temporal attention is suitable to select the relevant hidden states across all the time steps. Therefore, we adopt a two-stage encoding strategy to extract the most relevant information from each scale component for prediction, which includes an IA-Encoder for selecting the input sequences, and a TA-Encoder for capturing the long-term dependencies.

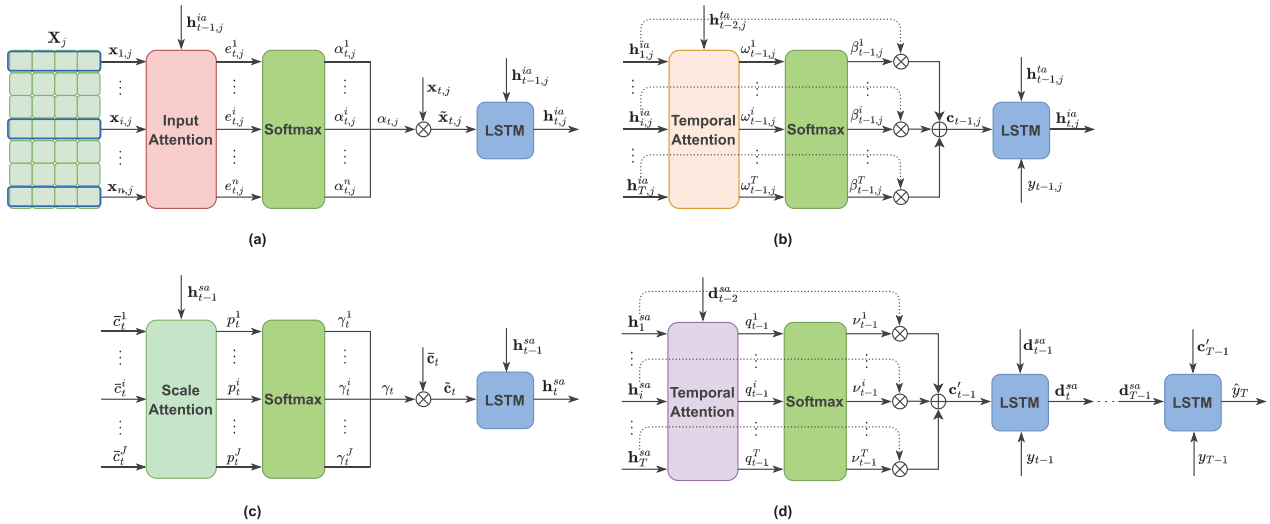

Fig. 2. Graphical illustration of various attention-based encoders and decoders used in MRN-SA. (a) IA-Encoder. (b) TA-Encoder. (c) SAF-Encoder. (d) TA-Decoder.

a) Input attention encoder: The encoder and the decoder, commonly used for time series prediction, are essentially an RNN (e.g., LSTM and GRU) that encodes the input sequences in the hidden state of an RNN in such a way that a decoder could reconstruct them. Fig. 2(a) shows a detailed graphical illustration of IA-Encoder. Given the jth scale component of MTS data $\mathbf{X}_j = (\mathbf{x}_{1,j}, \mathbf{x}_{2,j}, \ldots, \mathbf{x}_{T,j})$, where $\mathbf{x}_{t,j} \in \mathbb{R}^n$ denotes the time series at time t and scale j, the IA-Encoder uses an LSTM unit to learn a mapping from $\mathbf{x}_{t,j}$ to latent representation $\mathbf{h}^{\text{ia}}_{t,j} \in \mathbb{R}^m$ at time step t. Formally, we denote the memory cell state of LSTM as $\mathbf{s}^{\text{ia}}_{t,j}$, the internal sigmoid gates as $\mathbf{f}^{\text{ia}}_{t,j}$, $\mathbf{i}^{\text{ia}}_{t,j}$, and $\mathbf{o}^{\text{ia}}_{t,j}$. The update of an LSTM unit can be formulated as follows:

$$\begin{aligned}
\mathbf{f}^{\text{ia}}_{t,j} &= \text{sigmoid}\big(\mathbf{W}^{\text{ia}}_{f,j}\big[\mathbf{h}^{\text{ia}}_{t-1,j}; \mathbf{x}_{t,j}\big] + \mathbf{b}^{\text{ia}}_{f,j}\big) \\
\mathbf{i}^{\text{ia}}_{t,j} &= \text{sigmoid}\big(\mathbf{W}^{\text{ia}}_{i,j}\big[\mathbf{h}^{\text{ia}}_{t-1,j}; \mathbf{x}_{t,j}\big] + \mathbf{b}^{\text{ia}}_{i,j}\big) \\
\mathbf{o}^{\text{ia}}_{t,j} &= \text{sigmoid}\big(\mathbf{W}^{\text{ia}}_{o,j}\big[\mathbf{h}^{\text{ia}}_{t-1,j}; \mathbf{x}_{t,j}\big] + \mathbf{b}^{\text{ia}}_{o,j}\big) \\
\mathbf{s}^{\text{ia}}_{t,j} &= \mathbf{f}^{\text{ia}}_{t,j} \odot \mathbf{s}^{\text{ia}}_{t-1,j} + \mathbf{i}^{\text{ia}}_{t,j} \odot \tanh\big(\mathbf{W}^{\text{ia}}_{s,j}\big[\mathbf{h}^{\text{ia}}_{t-1,j}; \mathbf{x}_{t,j}\big] + \mathbf{b}^{\text{ia}}_{s,j}\big) \\
\mathbf{h}^{\text{ia}}_{t,j} &= \mathbf{o}^{\text{ia}}_{t,j} \odot \tanh\big(\mathbf{s}^{\text{ia}}_{t,j}\big)
\end{aligned} \tag{4}$$

where $[\cdot;\cdot]$ represents the concatenation operator, $\odot$ means the elementwise multiplication, $\mathbf{W}^{\text{ia}}_{k,j} \in \mathbb{R}^{m\times(m+n)}$ and $\mathbf{b}^{\text{ia}}_{k,j} \in \mathbb{R}^m (k = f, i, o, s)$ are learnable parameters.

In order to enable the encoder to selectively focus on the relevant input driving series, we integrate an input attention mechanism into the encoder. Currently, there exist various forms of attention weights, among which we use the most widely used one [14], [29]. Given the kth driving series at scale j, $\mathbf{x}^k_j = (x^k_{1,j}, x^k_{2,j}, \cdots, x^k_{T,j})^\top \in \mathbb{R}^T$, the previous hidden state $\mathbf{h}^{\text{ia}}_{t-1,j}$, and the internal state $\mathbf{s}^{\text{ia}}_{t-1,j}$, we feed them into a single neural network layer to output an relevant scores $e^k_{t,j}$. Then, a softmax function is applied to generate the attention weight $\alpha^k_{t,j}$. This can be summarized as follows:

$$\begin{aligned}
e^k_{t,j} &= \mathbf{v}^\top_{e,j} \tanh\big(\mathbf{W}_{e,j}\big[\mathbf{h}^{\text{ia}}_{t-1,j}; \mathbf{s}^{\text{ia}}_{t-1,j}\big] + \mathbf{U}_{e,j}\mathbf{x}^k_j + \mathbf{b}_{e,j}\big) \\
\alpha^k_{t,j} &= \text{softmax}\big(e^k_{t,j}\big), \qquad \text{for} \quad k = 1, \ldots, n
\end{aligned} \tag{5}$$

where $\mathbf{v}_{e,j}$, $\mathbf{W}_{e,j}$, $\mathbf{U}_{e,j}$, and $\mathbf{b}_{e,j}$ are learnable parameters. The attention weight quantifies the importance of each driving series for prediction. Taking these weights as the coefficients, we can update the input and the hidden state at time t as

$$\begin{aligned}
\tilde{\mathbf{x}}_{t,j} &= \big(\alpha^1_{t,j}x^1_{t,j}, \alpha^2_{t,j}x^2_{t,j}, \ldots, \alpha^n_{t,j}x^n_{t,j}\big)^\top \\
\mathbf{h}^{\text{ia}}_{t,j} &= f_1\big(\mathbf{h}^{\text{ia}}_{t-1,j}, \tilde{\mathbf{x}}_{t,j}\big)
\end{aligned} \tag{6}$$

where f_1 is an LSTM unit as defined in (4) with $\mathbf{x}_{t,j}$ replaced by $\tilde{\mathbf{x}}_{t,j}$.

b) Temporal attention encoder: To capture the long-term dependencies in time series, we utilize the TA-Encoder shown in Fig. 2(b) to further learn temporal patterns useful for better forecasting, in which the temporal attention mechanism is used to adaptively weight the IA-Encoder hidden states across all time steps. The attention weights $\beta^i_{t,j}$ of each IA-Encoder hidden state $\mathbf{h}^{\text{ia}}_{t,j}$ are computed as

$$\begin{aligned}
\omega^i_{t,j} &= \mathbf{v}^\top_{\omega,j} \tanh\big(\mathbf{W}_{\omega,j}\big[\mathbf{h}^{\text{ta}}_{t-1,j}; \mathbf{s}^{\text{ta}}_{t-1,j}\big] + \mathbf{U}_{\omega,j}\mathbf{h}^{\text{ia}}_{i,j} + \mathbf{b}_{\omega,j}\big) \\
\beta^i_{t,j} &= \text{softmax}\big(\omega^i_{t,j}\big), \quad \text{for } i = 1, \ldots, T
\end{aligned} \tag{7}$$

where $\mathbf{h}^{\text{ta}}_{t-1,j}$ and $\mathbf{s}^{\text{ta}}_{t-1,j}$ are, respectively, the hidden state and the cell state of the LSTM unit f_2 that will be defined in the following. Similar to (5), $\mathbf{v}_{\omega,j}$, $\mathbf{W}_{\omega,j}$, $\mathbf{U}_{\omega,j}$, and $\mathbf{b}_{\omega,j}$ are learnable parameters.

After obtaining the temporal attention weight $\beta^i_{t,j}$ that indicates the importance of the ith IA-Encoder hidden state $\mathbf{h}^{\text{ia}}_{i,j}$, we can easily calculate the context vector $\mathbf{c}_{t,j}$ at time t and scale j as

$$\mathbf{c}_{t,j} = \sum_{i=1}^{T} \beta^i_{t,j}\mathbf{h}^{\text{ia}}_{i,j}. \tag{8}$$

In essence, the context vector $\mathbf{c}_{t,j}$ is a weighted fusion of all the IA-Encoder hidden states, which further encodes the latent representations of the input data adaptively.

Intuitively, we can fuse the context vectors of different scales and use them to produce a prediction. In practice, however, such an MRN is hard to train due to a large number

of parameters that would need to be learned. To solve this problem, inspired by the deep supervision mechanism recently introduced in [40], [41], and [42], we deeply supervise the learning process of each IA-Encoder and TA-Encoder at different scales by using the given target series $(y_1, y_2, \ldots, y_{T-1})$, which is beneficial to reduce gradient vanishing and accelerate training convergence [43]. Specifically, we first fuse the context vector and the decomposed target series as follows:

$$\tilde{y}_{t-1,j} = \tilde{\mathbf{w}}_j^\top \left[y_{t-1,j}; \mathbf{c}_{t-1,j}\right] + \tilde{b}_j \tag{9}$$

and then use $\tilde{y}_{t-1,j}$ to learn an LSTM encoder f_2 as

$$\mathbf{h}_{t,j}^{\text{ta}} = f_2\left(\mathbf{h}_{t-1,j}^{\text{ta}}, \tilde{y}_{t-1,j}\right) \tag{10}$$

where f_2 is defined by the following equations with the learnable weight matrices $\mathbf{W}_{f,j}^{\text{ta}}, \mathbf{W}_{i,j}^{\text{ta}}, \mathbf{W}_{o,j}^{\text{ta}}, \mathbf{W}_{s,j}^{\text{ta}}$, and the bias vectors $\mathbf{b}_{f,j}^{\text{ta}}, \mathbf{b}_{i,j}^{\text{ta}}, \mathbf{b}_{o,j}^{\text{ta}}, \mathbf{b}_{s,j}^{\text{ta}}$:

$$\begin{aligned}
\mathbf{f}_{t,j}^{\text{ta}} &= \text{sigmoid}\left(\mathbf{W}_{f,j}^{\text{ta}}\left[\mathbf{h}_{t-1,j}^{\text{ta}}; \tilde{y}_{t-1,j}\right] + \mathbf{b}_{f,j}^{\text{ta}}\right) \\
\mathbf{i}_{t,j}^{\text{ta}} &= \text{sigmoid}\left(\mathbf{W}_{i,j}^{\text{ta}}\left[\mathbf{h}_{t-1,j}^{\text{ta}}; \tilde{y}_{t-1,j}\right] + \mathbf{b}_{i,j}^{\text{ta}}\right) \\
\mathbf{o}_{t,j}^{\text{ta}} &= \text{sigmoid}\left(\mathbf{W}_{o,j}^{\text{ta}}\left[\mathbf{h}_{t-1,j}^{\text{ta}}; \tilde{y}_{t-1,j}\right] + \mathbf{b}_{o,j}^{\text{ta}}\right) \\
\mathbf{s}_{t,j}^{\text{ta}} &= \mathbf{f}_{t,j}^{\text{ta}} \odot \mathbf{s}_{t-1,j}^{\text{ta}} + \mathbf{i}_{t,j}^{\text{ta}} \\
&\quad \odot \tanh\left(\mathbf{W}_{s,j}^{\text{ta}}\left[\mathbf{h}_{t-1,j}^{\text{ta}}; \tilde{y}_{t-1,j}\right] + \mathbf{b}_{s,j}^{\text{ta}}\right) \\
\mathbf{h}_{t,j}^{\text{ta}} &= \mathbf{o}_{t,j}^{\text{ta}} \odot \tanh\left(\mathbf{s}_{t,j}^{\text{ta}}\right).
\end{aligned} \tag{11}$$

Next, we obtain a prediction of the series at current time T by an auxiliary decoder (AU-Decoder) with a simple structure. Specifically, AU-Decoder maps a concatenation of the hidden state vector $\mathbf{h}_{T,j}^{\text{ta}}$ and the context vector $\mathbf{c}_{T,j}$ to the prediction result by a linear function, i.e.,

$$y_{T,j}^s = \mathbf{v}_{s,j}^\top\left(\mathbf{W}_{s,j}\left[\mathbf{h}_{T,j}^{\text{ta}}; \mathbf{c}_{T,j}\right] + \mathbf{b}_{s,j}\right) + b_{v,j}. \tag{12}$$

Here again, $\mathbf{v}_{s,j}$, $\mathbf{W}_{s,j}$, $\mathbf{b}_{s,j}$, and $b_{v,j}$ are parameters to learn. In order to make the network training converge to a stable solution, we introduce an auxiliary prediction loss

$$L_{\text{aux}} = \frac{1}{NJ}\sum_{i=1}^{N}\sum_{j=1}^{J}\left(y_T^i - y_{T,j}^{i,s}\right)^2 \tag{13}$$

where $y_{T,j}^{i,s}$ is the prediction value at scale j of the ith sample produced by AU-Decoder, and N is the number of training samples. The L_{aux} loss is used as an auxiliary supervision signal to train the MRN-SA network.

3) Scale Attention Fusion Encoding: SAF-Encoder aims to fuse the information from different scale components, which is illustrated in Fig. 2(c). Once we obtain the context vectors series of each scale $\mathbf{C}_j = (\mathbf{c}_{1,j}, \mathbf{c}_{2,j}, \ldots, \mathbf{c}_{T,j}) \in \mathbb{R}^{m\times T}$, we perform an 1-D convolutional filter on each context vector $\mathbf{c}_{i,j}$ for dimension reduction

$$\begin{aligned}
\bar{C}_j &= \left(\bar{c}_{1,j}, \bar{c}_{2,j}, \ldots, \bar{c}_{T,j}\right) \\
&= \mathbf{v}_{c,j}^\top \mathbf{C}_j, \quad \text{for } j = 1, \ldots, J
\end{aligned} \tag{14}$$

where $\mathbf{v}_{c,j}$ are parameters to learn. Then, we concatenate all the reduced context vectors from different scales together into a matrix $\bar{\mathbf{C}} = (\bar{\mathbf{c}}_1, \bar{\mathbf{c}}_2, \ldots, \bar{\mathbf{c}}_T)$, where $\bar{\mathbf{c}}_t \in \mathbb{R}^J$ and J is the number of scales. Next, we can further fuse them into a hidden state vector $\mathbf{h}_t^{\text{sa}}$ by

$$\mathbf{h}_t^{\text{sa}} = f_3\left(\mathbf{h}_{t-1}^{\text{sa}}, \bar{\mathbf{c}}_t\right). \tag{15}$$

Here, f_3 is an LSTM unit that is defined as

$$\begin{aligned}
\mathbf{f}_t^{\text{sa}} &= \text{sigmoid}\left(\mathbf{W}_f^{\text{sa}}\left[\mathbf{h}_{t-1}^{\text{sa}}; \bar{\mathbf{c}}_t\right] + \mathbf{b}_f^{\text{sa}}\right) \\
\mathbf{i}_t^{\text{sa}} &= \text{sigmoid}\left(\mathbf{W}_i^{\text{sa}}\left[\mathbf{h}_{t-1}^{\text{sa}}; \bar{\mathbf{c}}_t\right] + \mathbf{b}_i^{\text{sa}}\right) \\
\mathbf{o}_t^{\text{sa}} &= \text{sigmoid}\left(\mathbf{W}_o^{\text{sa}}\left[\mathbf{h}_{t-1}^{\text{sa}}; \bar{\mathbf{c}}_t\right] + \mathbf{b}_o^{\text{sa}}\right) \\
\mathbf{s}_t^{\text{sa}} &= \mathbf{f}_t^{\text{sa}} \odot \mathbf{s}_{t-1}^{\text{sa}} + \mathbf{i}_t^{\text{sa}} \odot \tanh\left(\mathbf{W}_s^{\text{sa}}\left[\mathbf{h}_{t-1}^{\text{sa}}; \bar{\mathbf{c}}_t\right] + \mathbf{b}_s^{\text{sa}}\right) \\
\mathbf{h}_t^{\text{sa}} &= \mathbf{o}_t^{\text{sa}} \odot \tanh\left(\mathbf{s}_t^{\text{sa}}\right).
\end{aligned} \tag{16}$$

However, since the importance of context vectors from different scales is distinct, we introduce a scale attention weight γ_t^j to measure the importance of the jth scale feature at time t, which is calculated by

$$\begin{aligned}
p_t^j &= \mathbf{v}_p^\top \tanh\left(\mathbf{W}_p\left[\mathbf{h}_{t-1}^{\text{sa}}; \mathbf{s}_{t-1}^{\text{sa}}\right] + \mathbf{U}_p\bar{\mathbf{c}}^j\right) \\
\gamma_t^j &= \text{softmax}\left(p_t^j\right), \quad \text{for } j = 1, \ldots, J
\end{aligned} \tag{17}$$

where $\bar{\mathbf{c}}^j = (\mathbf{c}_1^j, \mathbf{c}_2^j, \ldots, \mathbf{c}_T^j)^\top \in \mathbb{R}^T$. Using the scale attention weights, we can adaptively select the context vectors via

$$\tilde{\mathbf{c}}_t = \left(\gamma_t^1\bar{c}_t^1, \gamma_t^2\bar{c}_t^2, \cdots, \gamma_t^J\bar{c}_t^J\right)^\top. \tag{18}$$

And then we update the hidden states by

$$\mathbf{h}_t^{\text{sa}} = f_3\left(\mathbf{h}_{t-1}^{\text{sa}}, \tilde{\mathbf{c}}_t\right) \tag{19}$$

where f_3 is the LSTM defined in (16) with $\bar{\mathbf{c}}_t$ replaced by the weighted $\tilde{\mathbf{c}}_t$.

4) Temporal Attention Decoding: TA-Decoder, shown in Fig. 2(d), is used to generate a prediction output by decoding the fused information. To yield the prediction result, we again resort to an LSTM unit like the one used in TA-Encoder to decode the fused context vector with the given target series $(y_1, y_2, \ldots, y_{T-1})$. Following the encoder with the attention mechanism, a temporal attention is to guide the decoder to select the relevant context vectors across all time steps. Specifically, the temporal attention weight ν_t^i, representing the importance of the ith hidden state for the prediction, is calculated as

$$\begin{aligned}
q_t^i &= \mathbf{v}_q^\top \tanh\left(\mathbf{W}_q\left[\mathbf{d}_{t-1}; \mathbf{s}_{t-1}\right] + \mathbf{U}_q\mathbf{h}_i^{\text{sa}}\right) \\
\nu_t^i &= \text{softmax}\left(q_t^i\right), \quad \text{for } i = 1, \ldots, T.
\end{aligned} \tag{20}$$

Then, all the encoder hidden states $\mathbf{h}_i^{\text{sa}}$ are weighted by ν_t^i to obtain the context vector $\mathbf{c}_t'$, i.e.,

$$\mathbf{c}_t' = \sum_{i=1}^{T}\nu_t^i\mathbf{h}_i^{\text{sa}}. \tag{21}$$

After that, we integrate $\mathbf{c}_{t-1}'$ with the target series y_{t-1} as

$$\tilde{y}_{t-1} = \tilde{\mathbf{w}}^\top\left[y_{t-1}; \mathbf{c}_{t-1}'\right] + \tilde{b}. \tag{22}$$

Then, $\tilde{y}_{t-1}$ is used to update the hidden state of the decoder as follows:

$$\mathbf{d}_t = f_4\left(\mathbf{d}_{t-1}, \tilde{y}_{t-1}\right) \tag{23}$$

where f_4 is an LSTM unit that is formulated as

$$\begin{aligned}
\mathbf{f}_t &= \text{sigmoid}\left(\mathbf{W}_f\left[\mathbf{d}_{t-1}; \tilde{y}_{t-1}\right] + \mathbf{b}_f\right) \\
\mathbf{i}_t &= \text{sigmoid}\left(\mathbf{W}_i\left[\mathbf{d}_{t-1}; \tilde{y}_{t-1}\right] + \mathbf{b}_i\right) \\
\mathbf{o}_t &= \text{sigmoid}\left(\mathbf{W}_o\left[\mathbf{d}_{t-1}; \tilde{y}_{t-1}\right] + \mathbf{b}_o\right) \\
\mathbf{s}_t &= \mathbf{f}_t \odot \mathbf{s}_{t-1} + \mathbf{i}_t \odot \tanh\left(\mathbf{W}_s\left[\mathbf{d}_{t-1}; \tilde{y}_{t-1}\right] + \mathbf{b}_s\right) \\
\mathbf{d}_t &= \mathbf{o}_t \odot \tanh(\mathbf{s}_t).
\end{aligned} \tag{24}$$

Algorithm 1 Training for MRN-SA

Input: MTS X, number of training iterations K
Output: Model parameter set Θ

Initialize Θ;
for *epoch* $= 1, 2, \ldots, K$ **do**
  // Stage 1: Multiscale Decomposition
  $\{X_i\}_{i=1}^{J} \leftarrow$ Apply SWT to decompose X into a group of sub-series with different scales, as described in (3);
  // Stage 2: Scale Component Encoding
  for $i = 1, 2, \ldots, J$ **do**
    $\alpha_{t,i} \leftarrow$ Compute input attention weights by (5);
    $\mathbf{h}_{t,i}^{ia} \leftarrow$ Forward propagation of IA-Encoder with sub-series X_i by (6);
    $\beta_{t,i} \leftarrow$ Compute temporal attention weights by (7);
    $\mathbf{c}_{t,i} \leftarrow$ Compute the context vector at time t and scale i by (8);
    $\tilde{y}_{t,i} \leftarrow$ Fuse the context vector and the decomposed target series by (9);
    $\mathbf{h}_{t,i}^{ta} \leftarrow$ Forward propagation of TA-Encoder with $\tilde{y}_{t-1,i}$ by (10);
    $y_{T,i}^{s} \leftarrow$ Obtain the prediction values of the sub-series with different scales by (12);
  end for
  // Stage 3: Scale Attention Fusion Encoding
  $\mathbf{C} \leftarrow$ Compute the context matrix by (14);
  $\gamma_t \leftarrow$ Compute scale attention weights by (17);
  $\mathbf{h}_t^{sa} \leftarrow$ Fuse the context vector into the hidden state vector by (18) and (19);
  // Stage 4: Temporal Attention Decoding
  $\nu_t \leftarrow$ Compute the temporal attention weights used in TA-Decoder by (20);
  $\mathbf{c}_t' \leftarrow$ Compute the context vector by (21);
  $\mathbf{d}_t \leftarrow$ Update the hidden state of TA-Decoder by (22) and (23);
  $\hat{y}_T \leftarrow$ Produce the final prediction by (25);
  $L_{aux} \leftarrow$ Compute the auxiliary loss by (13);
  $L_{mse} \leftarrow$ Compute the final output loss by (26);
  $\Theta \leftarrow$ Update the model parameters by minimizing $L_{\text{aux}} + L_{\text{mse}}$.
end for

At last, we use the following linear function to produce the final prediction output:

$$\hat{y}_T = \mathbf{v}_y^{\top}\left(\mathbf{W}_y\left[\mathbf{d}_T; \mathbf{c}_T'\right] + \mathbf{b}_w\right) + b_v \tag{25}$$

where $\mathbf{W}_y$, $\mathbf{v}_y$, $\mathbf{b}_w$, and b_v are learnable parameters.

In order to learn the network parameters, the commonly used mean square error (mse) is adopted as the final output loss function, which can be formulated as

$$L_{\text{mse}} = \frac{1}{N}\sum_{i=1}^{N}\left(y_T^i - \hat{y}_T^i\right)^2 \tag{26}$$

where N is the number of training samples. Algorithm 1 summarizes the training procedure of the resulting MRN-SA.

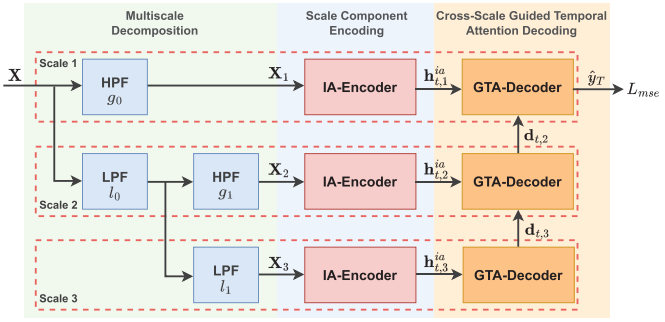

Fig. 3. Architecture of the proposed MRN-CSG.

B. MRN With Cross-Scale Guidance

As described above, our MRN-SA model directly utilizes three kinds of attentions (i.e., input attention, temporal attention, and scale attention) to select the appropriate driving series, hidden state vectors, and context vectors, respectively. Despite it is intuitive to understand and effective to adaptively fuse various features for accurate prediction, MRN-SA is very complex and computationally expensive. As an alternative to MRN-SA, we further propose a simple and novel MRN model called MRN-CSG for MTS forecasting.

Fig. 3 illustrates the architecture of the proposed MRN-CSG with three scale components, which is composed of three parts: *multiscale decomposition*, *scale component encoding*, and *cross-scale guided temporal attention decoding*. However, unlike the MRN-SA that employs two encoders to extract the hidden states of each scale, the MRN-CSG only uses one encoder (i.e., IA-Encoder) in the stage of scale component encoding. Moreover, instead of explicitly fusing hidden states by an SAF-Encoder and then generating a prediction by a TA-Decoder in the stage of decoding, a novel cross-scale guided temporal attention decoder (GTA-Decoder) is introduced to directly decode the hidden states back a prediction, which implicitly incorporates multiscale information in the decoder and exploits the hidden states from coarse scale to guide the decoding process at fine scale. This design brings us a lightweight and easily trained model. As a result, the training procedure of the MRN-CSG model does not need the deep supervision. Compared with MRN-SA, several modifications have been made in the architecture of MRN-CSG.

Modification 1 (Scale Component Encoding via Simpler Encoder): Instead of using two encoders as in MRN-SA, MRN-CSG removes the TA-Encoder from the stage of scale component encoding and only relies on the IA-Encoder to learn latent representations of time series data, which can use the input attention to adaptively select the relevant input driving series. Here the formulation of IA-Encoder is as same as the one used in MRN-SA. This means that the jth scale component $\mathbf{X}_j$ of MTS data can be encoded as $\mathbf{h}_{t,j}^{\text{ia}}$ (see Section IV-A2a).

Modification 2 (Temporal Attention Decoding via Cross-Scale Guidance): Instead of generating a prediction by a TA-Decoder, MRN-CSG adopts a novel cross-scale GTA-Decoder to decode the hidden states back a prediction, in which the multiscale information is used in an implicit manner. Fig. 4 illustrates the GTA-Decoder. It is quite similar to

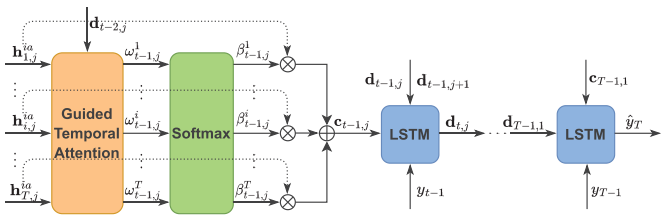

Fig. 4. Graphical illustration of the GTA-Decoder used in MRN-CSG.

TA-Decoder used in the MRN-SA model. The only difference is that GTA-Decoder exploits the hidden states from coarse scale to guide the decoding process at fine scale. Specifically, the decoder hidden state $\mathbf{d}_{t,j}$ at time t and scale j is calculated as follows:

$$\mathbf{d}_{t,j} = f_5\big(\mathbf{d}_{t-1,j} \odot \tanh\big(\mathbf{d}_{t-1,j+1}\big), \tilde{y}_{t-1}\big) \tag{27}$$

where $\tilde{y}_{t-1}$ is the updated target series by

$$\tilde{y}_{t-1} = \tilde{\mathbf{w}}^\top \big[y_{t-1}; \mathbf{c}_{t-1,j}\big] + \tilde{b}. \tag{28}$$

$\mathbf{c}_{t-1,j}$ can be obtained by (8), and f_5 is an LSTM unit defined as

$$\begin{aligned}
\mathbf{f}_{t,j} &= \text{sigmoid}\big(\mathbf{W}_{f,j}\big[\mathbf{d}_{t-1,j} \odot \mathbf{d}_{t-1,j+1}; \tilde{y}_{t-1}\big] + \mathbf{b}_{f,j}\big) \\
\mathbf{i}_{t,j} &= \text{sigmoid}\big(\mathbf{W}_{i,j}\big[\mathbf{d}_{t-1,j} \odot \mathbf{d}_{t-1,j+1}; \tilde{y}_{t-1}\big] + \mathbf{b}_{i,j}\big) \\
\mathbf{o}_{t,j} &= \text{sigmoid}\big(\mathbf{W}_{o,j}\big[\mathbf{d}_{t-1,j} \odot \mathbf{d}_{t-1,j+1}; \tilde{y}_{t-1}\big] + \mathbf{b}_{o,j}\big) \\
\mathbf{s}_{t,j} &= \mathbf{f}_{t,j} \odot \mathbf{s}_{t-1,j} + \mathbf{i}_{t,j} \\
&\quad \odot \tanh\big(\mathbf{W}_s\big[\mathbf{d}_{t-1,j} \odot \mathbf{d}_{t-1,j+1}; \tilde{y}_{t-1}\big] + \mathbf{b}_{s,j}\big) \\
\mathbf{d}_{t,j} &= \mathbf{o}_{t,j} \odot \tanh\big(\mathbf{s}_{t,j}\big).
\end{aligned} \tag{29}$$

Note that in the above definition the coarse-scale hidden state vector $\mathbf{d}_{t-1,j+1}$ is used to decode the fine-scale hidden state vector $\mathbf{d}_{t,j}$. Similar to (25), the final prediction output can be obtained by

$$\hat{y}_T = \mathbf{v}_y^\top \big(\mathbf{W}_y\big[\mathbf{d}_T; \mathbf{c}_{T,1}\big] + \mathbf{b}_w\big) + b_v \tag{30}$$

where $\mathbf{W}_y$, $\mathbf{v}_y$, $\mathbf{b}_w$, and b_v are learnable parameters.

Modification 3 (Simple Supervision): We remove the deep supervision used in MRN-SA, which avoids redundant computation and makes the MRN-CSG more simple. That is, unlike the MRN-SA model that is trained by simultaneously minimizing the auxiliary prediction loss L_{aux} defined in (13) and the final output loss L_{mse} defined in (26), we only use the loss L_{mse} to train MRN-CSG. Such a model is appealing due to its simplicity and powerfulness in capturing different types of dependencies.

The aforementioned modifications result in a lightweight model without decreasing prediction accuracy, as verified in our experiments in Section V-C. Algorithm 2 describes the training procedure of this lightweight model.

V. Experiments

In this section, we compare the proposed two models with several recent state-of-the-art methods on five benchmark datasets for MTS forecasting tasks. We first give some details of the datasets used in our experiments, and then describe the experimental settings. Finally, the comparison results are reported to justify the effectiveness of our models.

Algorithm 2 Training for MRN-CSG

Input: MTS X, number of training iterations K
Output: Model parameter set Θ

```
Initialize Θ;
for epoch = 1, 2, ..., K do
    // Stage 1: Multiscale Decomposition
    {X_i}_{i=1}^J ← Apply SWT to decompose X into a group of sub-series with different scales, as described in (3);
    for i = 1, 2, ..., J do
        // Stage 2: Scale Component Encoding
        α_{t,i} ← Compute input attention weights by (5);
        h^{ia}_{t,i} ← Forward propagation of IA-Encoder with sub-series X_i by (6);
        β_{t,i} ← Compute temporal attention weights by (7);
        c_{t,i} ← Compute the context vector at time t and scale i by (8);
        // Stage 3: Cross-Scale Guided Temporal Attention Decoding
        d_{t,J−i+1} ← Compute the hidden state of GTA-Decoder by (27);
        ŷ_T ← Produce the final prediction by (30);
        L_mse ← Compute the final output loss by (26);
        Θ ← Update the model parameters by minimizing L_mse.
    end for
end for
```

A. Datasets

We conduct experiments on the following five datasets.

1) *NASDAQ100:* A collection of the stock prices of 81 major corporations under the NASDAQ 100 index and the index value of the NASDAQ 100 from July 26, 2016 to December 22, 2016. The data exhibiting nonseasonality were sampled every minute.
2) *Exchange-Rate:* The daily exchange rates of eight countries (i.e., Australia, British, Canada, China, Japan, New Zealand, Singapore, and Switzerland) were collected from 1990 to 2016, which are nonseasonal time series.
3) *Traffic:* A collection of the road occupancy rates from 2015 to 2016, which was hourly measured by different sensors on San Francisco Bay area freeways and displays complex multiple seasonality.
4) *Solar-Energy:* The solar power production data with a clear seasonal pattern were recorded every 10 min from 137 solar photovoltaic power plants in 2006.
5) *Electricity:* This dataset contains electricity consumption records of 370 clients. The data that exhibit multiple seasonal patterns were recorded every 15 min from 2012 to 2014.

The first dataset is collected by Qin et al. [14], and the last four ones are published by Lai et al. [8]. These datasets are widely used for MTS forecasting evaluation.

B. Experimental Setup

1) Experimental Setting: For the assessment of the prediction accuracy, we compare the two proposed models with five

recent competitors, including four deep learning-based models (i.e., DA-RNN [14],[1] LSTNet [8],[2] TPA-LSTM [2],[3] DSANet [15][4]), and one low-rank tensor model (i.e., BHT-ARIMA [44] that employs a block Hankel tensor ARIMA to produce a prediction[5]). For these competing methods, we use the publicly available codes (or the codes shared by the authors) with the hyperparameters suggested by the authors to run the experiments. For our proposed models, we initialize the network parameters by random initialization, and employ the Adam optimizer [45] with the two momentum values set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The models are trained for 50k iterations with a mini-batch size of 128. The initial learning rate is set to 0.0005, and then multiplied by 0.1 after every 10k iterations. All experiments are conducted on a workstation equipped with dual Intel Xeon E5-2643 3.4-GHz CPUs, 256 GB of memory, and two NVIDIA GeForce GTX 1080 GPUs. The proposed models are implemented in Python under the Tensorflow framework [46].

In order to train all the aforementioned deep learning-based models, we split each dataset into three disjoint sets in chronological order, 60% for training, 20% for validation, and the rest is for testing, following [8] and [14]. For the BHT-ARIMA model, we use 80% for training and 20% for testing. In the multistep prediction experiments, we consider three horizon values, i.e., $\tau \in \{1, 3, 5\}$, which means the horizon values were set from 1 to 5 min for the prediction evaluation on the NASDAQ100 dataset, from 10 to 50 min on the Solar-Energy dataset, from 1 to 5 h on the Electricity and Traffic datasets, and from 1 to 5 days on the Exchange-Rate dataset. Besides, for all deep learning-based models, we train them ten times and report the average prediction performance and standard deviations over ten runs.

2) Evaluation Metrics: To measure the prediction quality of the competing methods, we use five typical evaluation metrics, including root mean squared error (RMSE), root relative squared error (RRSE), mean absolute error (MAE), symmetric mean absolute percentage error (sMAPE), and empirical correlation coefficient (CORR), each of which captures a different aspect of the quality of the prediction results.

1) *RMSE:* It describes the sample standard deviation of the differences between the predicted and ground-truth values, and is defined as

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(y_T^i - \hat{y}_T^i\right)^2} \tag{31}$$

where y_T^i and $\hat{y}_T^i$ are the ground-truth value and the predicted one, respectively.

2) *RRSE:* It is a normalized version of the RMSE for making more readable evaluation, regardless of the data scale, i.e.,

$$\text{RRSE} = \frac{\sqrt{\sum_{i=1}^{N}\left(y_T^i - \hat{y}_T^i\right)^2}}{\sqrt{\sum_{i=1}^{N}\left(y_T^i - \bar{y}_T\right)^2}} \tag{32}$$

where $\bar{y}_T$ represents the mean of the test target series.

3) *MAE:* Its definition is given as

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}\left|y_T^i - \hat{y}_T^i\right|. \tag{33}$$

4) *sMAPE:* It is a metric based on percentage errors and widely used in many forecasting competitions

$$\text{sMAPE} = \frac{1}{N}\sum_{i=1}^{N}\frac{\left|y_T^i - \hat{y}_T^i\right|}{\left|y_T^i\right| + \left|\hat{y}_T^i\right| + \epsilon} \tag{34}$$

where ϵ is a small constant used to avoid division by zero.

5) *CORR:* It indicates the degree of association between two variables

$$\text{CORR} = \frac{1}{N}\sum_{i=1}^{N}\frac{\sum_T\left(y_T^i - \bar{y}^i\right)\left(\hat{y}_T^i - \bar{\hat{y}}^i\right)}{\sqrt{\sum_T\left(y_T^i - \bar{y}^i\right)^2\sum_T\left(\hat{y}_T^i - \bar{\hat{y}}^i\right)^2}} \tag{35}$$

where $\bar{y}$ represents the mean of set y.

For the CORR metric, a higher value is better, whereas for the other metrics, lower values means better prediction performance.

C. Comparison With the State-of-the-Art

The averaged results and standard deviations over ten runs for one-step ahead prediction (i.e., $horizon = 1$), obtained by all the competing methods on five MTS datasets, are tabulated in Table I, which provide comparative analysis in terms of all the above metrics.[6] From Table I, we have several observations. First, the deep learning-based forecasting models outperform the low-rank statistical model except on the NASDAQ100 dataset. This well demonstrates that, by mapping the input time series to a hidden variable sequence, deep recurrent networks can learn complex temporal patterns much better than the statistical model. Second, MRN-CSG and MRN-SA achieve very competitive results for all the metrics on almost all datasets, which consistently demonstrate the superiority of the proposed models. Comparing with TPA-LSTM and LSTNet, two state-of-the-art forecasting models, our models deliver superior performance on the NASDQ100 and Exchange-Rate datasets. Third, although TPA-LSTM can obtain slightly better prediction accuracy than ours on Traffic and Electricity datasets, it fails to keep its performance on other datasets. Fourth, the proposed two models exhibit relative small values of standard deviations on all datasets, which indicates that our models are more stable than others. Finally, all the models report larger RMSE and

[1]The authors share their source codes with us.
[2]Code is available at https://github.com/laiguokun/LSTNet.
[3]Code is available at https://github.com/gantheory/TPA-LSTM.
[4]Code is available at https://github.com/bighuang624/DSANet.
[5]Code is available at https://github.com/yokotatsuya/BHT-ARIMA.

[6]Note that for each MTS dataset we use its last series as the target series and all the others as the driving series, following [14].

TABLE I
ONE-STEP AHEAD FORECASTING PERFORMANCE COMPARISON OF DIFFERENT METHODS ON FIVE DATASETS (MEAN ± STANDARD DEVIATION)

Datasets	Metrics	Forecasting Methods ($Horizon = 1$)						
		BHT-ARIMA	LSTNet	TPA-LSTM	DSANet	DA-RNN	MRN-SA	MRN-CSG
NASDAQ100	RMSE	2.2649	6.6860±1.3983	1.5525±0.0874	4.5357±0.0853	0.3813±0.0510	0.3235±0.0390	**0.3057**±0.0195
	RRSE	0.0335	0.0992±0.0207	0.0229±0.0013	0.0673±0.0033	0.0055±0.0041	0.0047±0.0021	**0.0045**±0.0003
	MAE	1.4946	5.1696±1.5703	1.0184±0.1095	2.9727±0.1453	0.2913±0.1215	0.2425±0.0568	**0.2269**±0.0219
	sMAPE	0.0154	0.0531±0.0161	0.0105±0.0011	0.0300±0.0015	0.0036±0.0013	0.0028±0.0011	**0.0023**±0.0012
	CORR	0.9994	0.9970±0.0001	0.9998±0.0001	0.9845±0.0003	0.9997±0.0001	0.9998±0.0001	**1.0000**±0.0000
Exchange-Rate	RMSE	0.0035	0.0039±0.0010	0.0055±0.0011	0.0070±0.0016	0.0027±0.0018	0.0025±0.0011	**0.0022**±0.0003
	RRSE	0.0906	0.1011±0.0249	0.1409±0.0192	0.1819±0.0422	0.0693±0.0470	0.0646±0.0278	**0.0567**±0.0068
	MAE	0.0025	0.0030±0.0008	0.0044±0.0010	0.0075±0.0015	0.0020±0.0011	0.0018±0.0007	**0.0016**±0.0002
	sMAPE	0.1682	0.1969±0.0503	0.2854±0.0293	0.4116±0.0934	0.1348±0.0817	0.1226±0.0490	**0.1086**±0.0157
	CORR	0.9959	0.9950±0.0025	0.9900±0.0020	0.9869±0.0056	0.9974±0.0030	0.9979±0.0013	**0.9987**±0.0001
Traffic	RMSE	0.0127	0.0053±0.0001	**0.0051**±0.0001	0.0077±0.0009	0.0068±0.0005	0.0052±0.0002	**0.0051**±0.0001
	RRSE	0.5363	0.2235±0.0058	**0.2145**±0.0029	0.3841±0.0439	0.3008±0.0168	0.2201±0.0070	0.2159±0.0045
	MAE	0.0086	0.0031±0.0001	**0.0029**±0.0001	0.0061±0.0008	0.0055±0.0004	0.0030±0.0001	**0.0029**±0.0001
	sMAPE	17.745	5.1475±0.2553	**4.5291**±0.1425	14.797±1.6964	8.5295±0.6823	4.9679±0.3150	4.9538±0.2110
	CORR	0.8777	0.9795±0.0007	0.9803±0.0003	0.9585±0.0042	0.9722±0.0018	**0.9805**±0.0005	0.9802±0.0004
Solar-Energy	RMSE	2.7523	1.4103±0.1100	1.4154±0.0028	2.1700±0.0921	1.1846±0.0653	1.1604±0.0427	**1.1561**±0.0366
	RRSE	0.2467	0.1581±0.0112	0.1269±0.0003	0.1900±0.0080	0.1061±0.0471	0.1040±0.0382	**0.1036**±0.0069
	MAE	1.0849	0.5627±0.0231	0.5681±0.0067	0.9930±0.0165	0.6220±0.0226	0.5976±0.0152	**0.5564**±0.0138
	sMAPE	74.364	71.924±9.6757	66.304±0.0742	66.543±1.5619	66.182±1.2728	65.791±0.7731	**65.412**±0.3052
	CORR	0.9705	0.9921±0.0001	0.9919±0.0001	0.9819±0.0002	0.9937±0.0061	0.9941±0.0047	**0.9950**±0.0003
Electricity	RMSE	274.53	206.24±8.6182	**161.31**±8.4053	210.41±25.258	190.79±24.679	183.06±13.352	164.93±4.4737
	RRSE	0.4845	0.3641±0.0152	**0.2748**±0.0197	0.4239±0.0451	0.3667±0.0434	0.3316±0.0417	0.3214±0.0229
	MAE	189.63	149.26±5.9772	109.24±6.5116	164.65±22.649	151.68±18.873	129.26±14.096	**107.17**±8.0426
	sMAPE	2.7553	2.1870±0.0872	**1.5980**±0.1261	2.4473±0.3415	2.2618±0.2722	2.0759±0.2256	1.7910±0.1188
	CORR	0.8964	0.9317±0.0062	**0.9606**±0.0037	0.9170±0.0070	0.9441±0.0153	0.9525±0.0067	0.9579±0.0056

The best and second best results are highlighted in bold and underline, respectively.

MAE on Electricity and Traffic than the other three datasets, while the RRSE and sMAPE values are small. It means that there exist larger data variations on these two datasets. Despite this, both MRN-CSG and MRN-SA still perform well on them. To explicitly illustrate the performance, we show the one-step prediction results of different methods over the NASDAQ100 dataset in Fig. 5(a) and also plot their error curves in Fig. 5(b). We can observe that our models as well as DA-RNN fit the ground truth significantly better than others, while LSTNet and DSANet produce quite large errors in change points.

To further analyze the performance, we compare the multistep forecasting ability of all competing methods under two different horizon values (3 and 5). Considering that BHT-ARIMA is originally designed for one-step forecasting, we thus perform the recursive prediction strategy on it to achieve the multistep forecasting. For deep learning-based methods, we train their direct multistep forecasting versions. Tables II and III report the averaged results and standard deviations over ten runs of different methods for multistep forecasting. As shown in both tables, our two models are also superior to other methods on most evaluation metrics. The visual comparisons between the five-step ahead prediction results (i.e., horizon $=$ 5) and true values of various forecasting methods on the Solar-Energy dataset are shown in Fig. 5(c), and the corresponding error curves are plotted in Fig. 5(d). According to the results shown in these two subfigures, the following conclusions can be drawn. First, our two models achieve significantly better results than the competing methods, which indicates the effectiveness of our scale attention and cross-scale guidance schemes for learning the periodic temporal patterns of time series data. Second, for forecasting the nonperiodic data, e.g., NASDAQ100 and Exchange-Rate, BHT-ARIMA can work well and even outperform DSANet that is among the state-of-the-art deep forecasting models. However, the experimental results shown in Fig. 5(c) and (d) clearly indicate that BHT-ARIMA is incompetent for the periodic forecasting task because it does not capture long-term temporal patterns.

Additionally, to determine whether the performance of the proposed models over the state-of-the-art models is statistically significant, we apply the paired t-test with significance level 0.05 to assess the significance of our models against the competing deep forecasting models. The p-values for the RMSE metric are shown in Table IV. We can see that all the p-values of the t-tests except MRN-SA are notably small, which reveals that MRN-CSG statistically outperforms the competing models. It also shows that there is no significant difference between MRN-CSG and MRN-SA.

D. Effect of Parameters

There are three key parameters in the proposed models, i.e., the number of scales J, the input window size T, and the dimension of latent representation vector $\mathbf{h}$. In this section, we evaluate the sensitivity of our models with respect to these parameters by conducting some experiments on NASDAQ100. It is necessary to point out that, although the following evaluation results are obtained on the NASDAQ100 dataset, similar observations can be made on other datasets.

1) Effect of the Number of Scales: We analyze the behavior of the proposed multiscale forecasting models for varying number of decomposition scales. Intuitively, the prediction performance would improve if we increase the number of

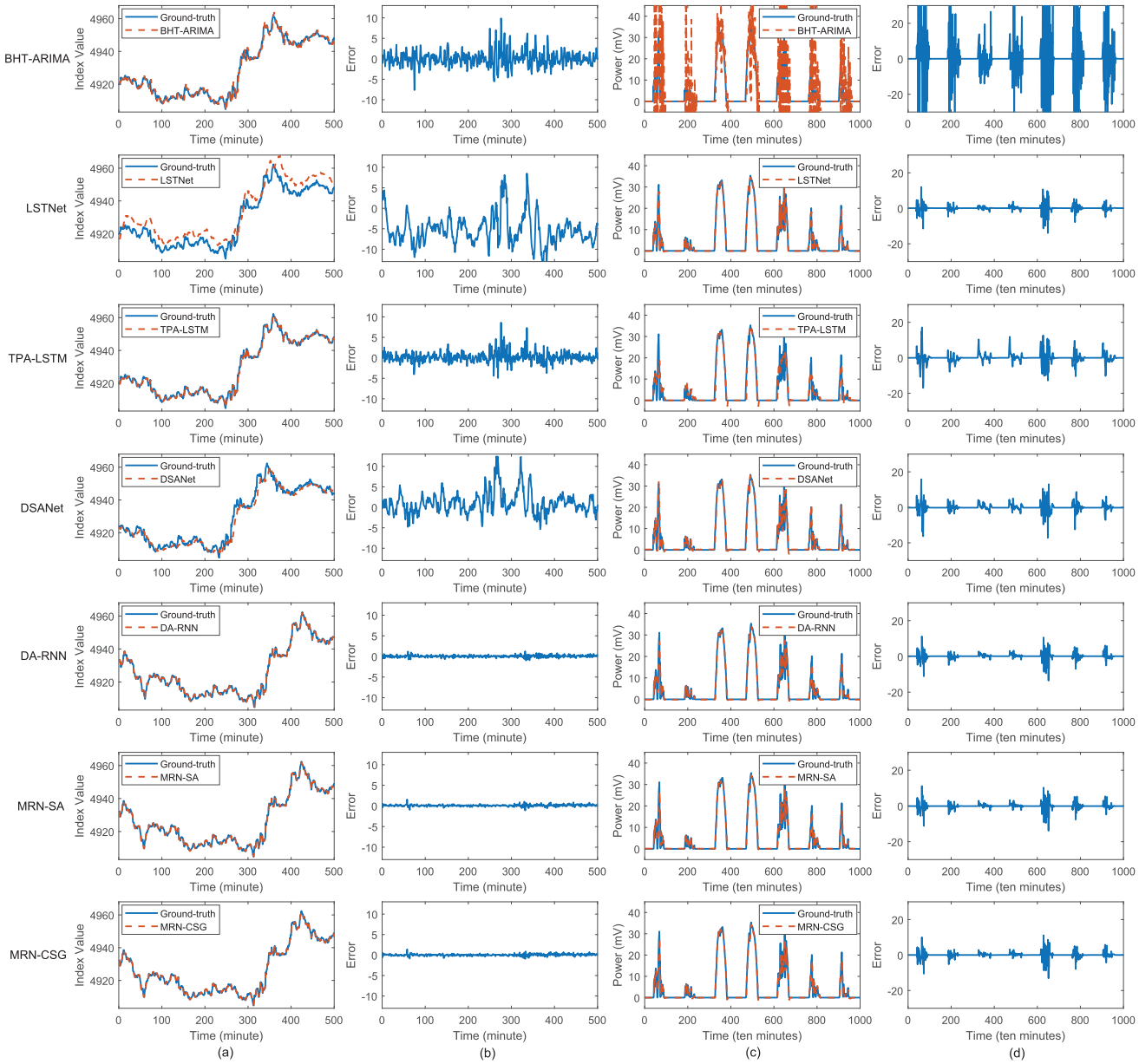

Fig. 5. Prediction results and prediction errors of different methods. From top row to bottom row, they are BHT-ARIMA, LSTNet, TPA-LSTM, DSANet, DA-RNN, MRN-SA, and MRN-CSG, respectively. (a) One-step prediction results of different methods on the NASDAQ100 dataset. (b) One-step prediction errors of different methods on the NASDAQ100 dataset. (c) Five-step prediction results of different methods on the Solar-Energy dataset. (d) Five-step prediction errors of different methods on the Solar-Energy dataset.

scales at the cost of running time. To explore the relationship between the prediction performance and the number of scales, we train our models on the NASDAQ100 dataset with five different scales. The quantitative results obtained by MRN-CSG with different scales and horizons are shown in the top row of Fig. 6. It is clear that the prediction performance can be improved by increasing the decomposition scale, which justifies the intuition of a direct correlation between performance and scales. The best prediction performance is achieved when the number of scales is 2. When it increases more than 2, the performance begins to drop slightly. The same trend can also be observed for MRN-SA. Furthermore, when using more scales, the computational complexity will significantly increase. Therefore, in the following experiments, we set the number of scales to 2.

2) Effect of the Input Window Size: We also study the effect of the input window size T on the forecasting performance by varying T. In the second row of Fig. 6, we plot the quantitative results for different sizes of the input window. It is easily observed that too small size of T results in a worse performance. These results also show that our models are insensitive to T when $T > 10$. That is, increasing the size does not have further performance gains. Thus, we empirically set the input window size to 10, which provides an appropriate trade-off between prediction accuracy and inference speed.

3) Effect of the Dimension of Latent Vector: In our models the dimension of latent vector $\mathbf{h}$ indicates the number of features expected to learn from time series data. Generally, the more expected features the model learns, the more complex the model is. To investigate its effect on the performance,

TABLE II

THREE-STEP AHEAD FORECASTING PERFORMANCE COMPARISON OF DIFFERENT METHODS ON FIVE DATASETS (MEAN ± STANDARD DEVIATION)

Datasets	Metrics	Forecasting Methods ($Horizon = 3$)						
		BHT-ARIMA	LSTNet	TPA-LSTM	DSANet	DA-RNN	MRN-SA	MRN-CSG
NASDAQ100	RMSE	4.0521	6.5779±2.3637	2.7601±0.0177	6.8551±0.4905	1.9146±0.4979	1.5992±0.4167	**1.4332**±0.0354
	RRSE	0.0598	0.0976±0.0351	0.0408±0.0003	0.1012±0.0028	0.0284±0.0074	0.0237±0.0062	**0.0213**±0.0005
	MAE	2.7252	5.2538±0.3165	1.8269±0.0204	4.9068±0.2916	1.4279±0.4707	1.1610±0.1119	**0.9932**±0.0422
	sMAPE	0.0280	0.0539±0.0236	**0.0188**±0.0002	0.0992±0.0059	0.0294±0.0096	0.0238±0.0084	0.0204±0.0009
	CORR	0.9982	0.9977±0.0001	0.9992±0.0000	0.9485±0.0017	**0.9998**±0.0000	**0.9998**±0.0000	**0.9998**±0.0000
Exchange-Rate	RMSE	0.0090	0.0050±0.0011	0.0055±0.0002	0.0099±0.0002	0.0049±0.0012	0.0043±0.0001	**0.0032**±0.0003
	RRSE	0.2337	0.1303±0.0285	0.1429±0.0049	0.2065±0.0051	0.1272±0.0327	0.1118±0.0024	**0.0845**±0.0015
	MAE	0.0065	0.0039±0.0009	0.0044±0.0002	0.0071±0.0002	0.0033±0.0011	0.0032±0.0001	**0.0026**±0.0005
	sMAPE	0.4280	0.2582±0.0623	0.2897±0.0142	0.4800±0.0162	0.2151±0.0766	0.2139±0.0045	**0.1718**±0.0049
	CORR	0.9737	0.9928±0.0016	0.9913±0.0009	0.9795±0.0000	0.9929±0.0053	0.9941±0.0003	**0.9968**±0.0004
Traffic	RMSE	0.0328	0.0080±0.0002	0.0085±0.0005	0.0152±0.0015	0.0101±0.0006	0.0080±0.0002	**0.0079**±0.0003
	RRSE	1.3907	0.3388±0.0069	0.3592±0.0208	0.7542±0.0767	0.4148±0.0238	0.3405±0.0080	**0.3349**±0.0068
	MAE	0.0209	0.0053±0.0001	0.0057±0.0005	0.0123±0.0012	0.0067±0.0005	0.0053±0.0002	**0.0051**±0.0002
	sMAPE	40.378	8.3671±0.6360	10.019±1.1332	23.633±1.2120	9.3038±0.9222	**8.3664**±0.3327	8.4070±0.9261
	CORR	0.6201	0.9686±0.0016	0.9666±0.0009	0.8664±0.0081	0.9630±0.0042	**0.9690**±0.0013	0.9658±0.0013
Solar-Energy	RMSE	7.5542	2.2504±0.0353	2.1902±0.0099	3.4557±0.0216	1.4361±0.0625	**1.3180**±0.0110	1.4289±0.0577
	RRSE	0.6771	0.2522±0.0040	0.1963±0.0009	0.3025±0.0019	0.1286±0.0560	**0.1181**±0.0038	0.1280±0.0051
	MAE	2.8343	1.0946±0.0500	0.9687±0.0177	1.8858±0.0271	0.7292±0.0329	0.6747±0.0126	**0.6690**±0.0180
	sMAPE	80.441	68.936±0.3271	69.054±0.1088	69.535±4.0962	66.609±1.8369	66.171±0.3348	**66.158**±0.2491
	CORR	0.8290	0.9801±0.0003	0.9807±0.0002	0.9538±0.0006	0.9906±0.0074	**0.9932**±0.0004	0.9921±0.0005
Electricity	RMSE	782.24	299.49±11.371	225.38±8.3800	243.76±5.2972	217.77±12.558	214.65±7.8848	**206.91**±7.7887
	RRSE	1.3806	0.5288±0.0201	0.3974±0.0158	0.6264±0.0136	0.3779±0.0362	0.3759±0.0130	**0.3579**±0.0143
	MAE	522.10	218.12±8.3566	173.33±7.9000	181.21±5.3464	169.05±9.4850	**163.21**±7.1091	164.80±6.6677
	sMAPE	7.7023	3.1997±0.1263	2.6171±0.2740	2.9939±0.0811	2.4950±0.2811	**2.4231**±0.1018	2.4559±0.2480
	CORR	0.6023	0.8523±0.0109	0.9200±0.0002	0.8027±0.0042	0.9423±0.0161	0.9437±0.0068	**0.9525**±0.0052

The best and second best results are highlighted in bold and underline, respectively.

TABLE III

FIVE-STEP AHEAD FORECASTING PERFORMANCE COMPARISON OF DIFFERENT METHODS ON FIVE DATASETS (MEAN ± STANDARD DEVIATION)

Datasets	Metrics	Forecasting Methods ($Horizon = 5$)						
		BHT-ARIMA	LSTNet	TPA-LSTM	DSANet	DA-RNN	MRN-SA	MRN-CSG
NASDAQ100	RMSE	6.1073	7.6190±3.3888	3.7453±0.1134	8.7219±0.5346	2.3920±0.5526	1.7813±0.1390	**1.6181**±0.1181
	RRSE	0.0902	0.1125±0.0500	0.0557±0.0016	0.1288±0.0079	0.0355±0.0082	0.0273±0.0021	**0.0240**±0.0018
	MAE	4.0878	5.9652±3.3009	2.5605±0.1750	6.1727±0.8224	1.9214±0.6112	1.2913±0.1521	**1.1583**±0.1330
	sMAPE	0.0420	0.0617±0.0336	0.0268±0.0004	0.1248±0.0166	0.0395±0.0125	0.0266±0.0031	**0.0238**±0.0027
	CORR	0.9960	0.9967±0.0007	0.9983±0.0003	0.4373±0.0294	0.9997±0.0000	0.9997±0.0000	**0.9998**±0.0000
Exchange-Rate	RMSE	0.0189	0.0052±0.0017	0.0065±0.0006	0.0101±0.0012	0.0058±0.0025	0.0046±0.0016	**0.0030**±0.0011
	RRSE	0.4916	0.1356±0.0176	0.1691±0.0155	0.3004±0.0251	0.1331±0.0717	**0.0926**±0.0560	0.1033±0.0275
	MAE	0.0133	0.0040±0.0006	0.0055±0.0008	0.0099±0.0008	0.0038±0.0017	0.0035±0.0015	**0.0032**±0.0010
	sMAPE	0.8829	0.2676±0.0405	0.3467±0.0386	0.5166±0.0509	0.4061±0.0247	0.3159±0.0141	**0.2152**±0.0167
	CORR	0.8971	0.9908±0.0022	0.9882±0.0012	0.9814±0.0032	0.9915±0.0060	0.9916±0.0016	**0.9974**±0.0004
Traffic	RMSE	0.0703	0.0086±0.0003	0.0089±0.0005	0.0201±0.0014	0.0105±0.0007	0.0090±0.0004	**0.0084**±0.0003
	RRSE	2.9795	0.3652±0.0121	0.3782±0.0199	0.9998±0.0709	0.4291±0.0276	0.3824±0.0161	**0.3578**±0.0106
	MAE	0.0424	0.0058±0.0002	0.0062±0.0004	0.0161±0.0013	0.0069±0.0003	0.0062±0.0003	**0.0057**±0.0002
	sMAPE	54.162	9.7516±0.8883	10.949±0.4699	27.115±1.1236	**9.5136**±1.3046	10.328±0.7756	9.7186±0.6875
	CORR	0.3494	0.9612±0.0028	0.9583±0.0021	0.8563±0.0040	0.9608±0.0063	**0.9618**±0.0013	0.9615±0.0024
Solar-Energy	RMSE	16.299	2.7839±0.0309	2.6969±0.0097	4.3901±0.0015	1.7632±1.0524	1.4844±0.0107	**1.4348**±0.0089
	RRSE	1.4609	0.3121±0.0035	0.2450±0.0076	0.3843±0.0001	0.1564±0.0929	0.1330±0.0157	**0.1285**±0.0053
	MAE	5.9897	1.4944±0.0644	1.2535±0.0835	2.8184±0.0096	0.9635±0.6801	0.7192±0.1204	**0.6740**±0.0748
	sMAPE	65.114	71.669±0.3780	69.795±0.2360	69.286±0.4474	68.300±2.8717	67.816±0.2218	**67.013**±0.1825
	CORR	0.5703	0.9706±0.0009	0.9882±0.0012	0.9240±0.0003	0.9848±0.0166	0.9913±0.0019	**0.9920**±0.0002
Electricity	RMSE	1710.9	337.84±6.8437	259.45±7.1752	296.58±6.3044	231.24±17.605	230.83±8.939	**229.70**±6.1539
	RRSE	3.0197	0.5965±0.0121	0.4609±0.0104	0.7621±0.0162	0.4180±0.0314	0.4154±0.0327	**0.4009**±0.0102
	MAE	1128.5	246.37±4.3594	198.23±6.9350	224.51±6.4814	183.95±13.3150	183.07±6.9030	**174.99**±5.5122
	sMAPE	17.565	3.6160±0.0634	2.9647±0.1134	3.7147±0.0967	2.5633±0.2028	2.5082±0.2436	**2.4974**±0.0791
	CORR	0.3359	0.8099±0.0089	0.8994±0.0107	0.7088±0.0075	0.9197±0.0123	0.9230±0.0137	**0.9275**±0.0074

The best and second best results are highlighted in bold and underline, respectively.

we conduct some experiments with various dimensions. The results of our MRN-CSG model are illustrated in the bottom row of Fig. 6. It can be seen that the model achieves the best performance with 128-D latent vector. When increasing the dimension of latent vector, the overfitting is observed. Additionally, we also find that the CORR is not a very effective metric to measure the prediction performance due to its limited discrimination ability, as shown in the last column of Fig. 6.

TABLE IV

P-VALUES OF THE t-TESTS BETWEEN MRN-CSG AND OTHER MODELS ON FIVE DATASETS

Horizons	Datasets	LSTNet	TPA-LSTM	DSANet	DA-RNN	MRN-SA
$Horizon = 1$	NASDAQ100	8.1×10^{-8}	5.7×10^{-7}	1.4×10^{-16}	4.9×10^{-11}	1.0×10^{-1}
	Exchange-Rate	1.4×10^{-5}	1.1×10^{-5}	4.1×10^{-6}	3.6×10^{-5}	9.1×10^{-2}
	Traffic	1.0×10^{-3}	5.9×10^{-2}	5.0×10^{-6}	1.8×10^{-10}	3.3×10^{-1}
	Solar-Energy	3.2×10^{-12}	6.2×10^{-10}	3.3×10^{-12}	4.4×10^{-6}	4.9×10^{-1}
	Electricity	4.2×10^{-4}	6.1×10^{-2}	2.2×10^{-4}	1.8×10^{-4}	4.0×10^{-2}
$Horizon = 3$	NASDAQ100	3.7×10^{-5}	4.7×10^{-4}	8.0×10^{-15}	5.9×10^{-3}	1.1×10^{-1}
	Exchange-Rate	6.2×10^{-4}	3.9×10^{-4}	2.8×10^{-11}	8.5×10^{-3}	1.2×10^{-1}
	Traffic	3.6×10^{-3}	1.3×10^{-5}	8.8×10^{-8}	4.2×10^{-6}	1.1×10^{-1}
	Solar-Energy	3.1×10^{-11}	8.0×10^{-4}	9.7×10^{-16}	4.9×10^{-3}	6.3×10^{-2}
	Electricity	5.7×10^{-3}	4.5×10^{-4}	1.3×10^{-4}	8.0×10^{-4}	1.5×10^{-1}
$Horizon = 5$	NASDAQ100	1.4×10^{-4}	1.2×10^{-4}	2.4×10^{-12}	3.9×10^{-4}	9.6×10^{-2}
	Exchange-Rate	4.3×10^{-3}	2.7×10^{-3}	6.9×10^{-7}	3.5×10^{-3}	1.1×10^{-1}
	Traffic	7.4×10^{-3}	6.3×10^{-4}	3.5×10^{-10}	9.0×10^{-6}	5.7×10^{-2}
	Solar-Energy	2.4×10^{-10}	8.5×10^{-5}	4.1×10^{-17}	1.7×10^{-3}	2.0×10^{-1}
	Electricity	2.8×10^{-3}	3.6×10^{-3}	6.0×10^{-10}	4.1×10^{-3}	1.6×10^{-1}

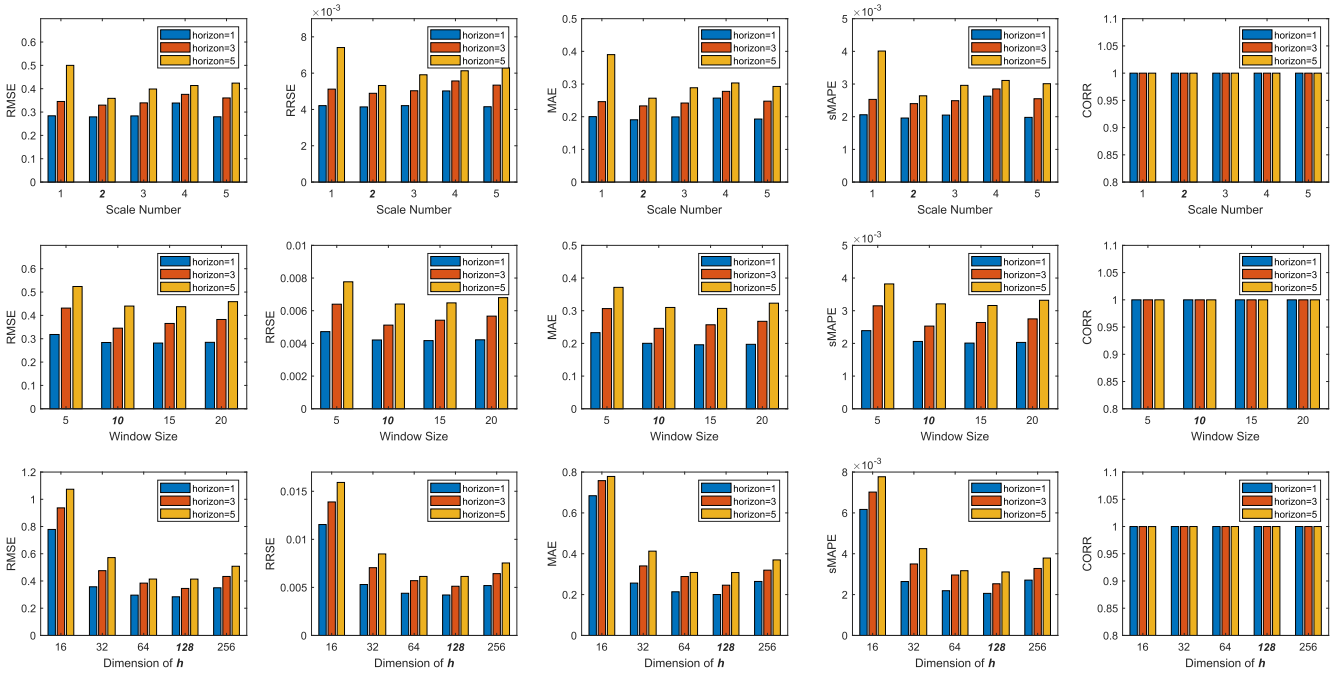

Fig. 6. Effect of different parameters of our models on the prediction performance on the NASDAQ100 dataset. Similar observations can be made on other datasets. The first, second, and third rows, respectively, show the performance for varying number of decomposition scales, different window sizes, and various dimensions of latent vector.

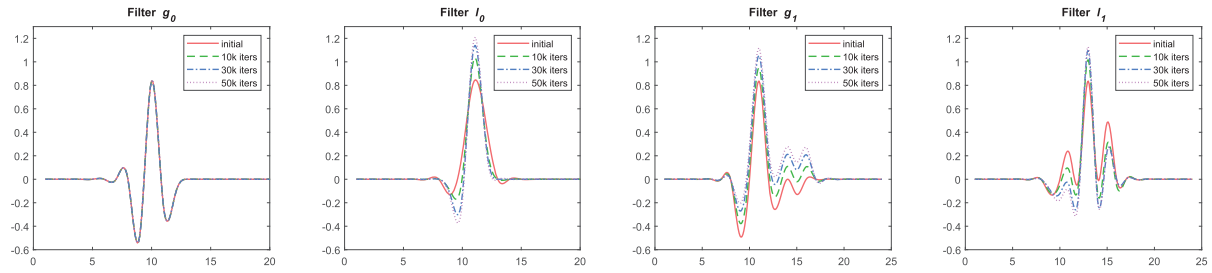

Fig. 7. Visual comparisons between the HPF and LPF of the fixed wavelet (*db*2) and ones of our learned wavelet. The initial *db*2 filters are shown in red solid line, while the learned filters over training iterations are shown in dashed lines with different colors. Note that, for better visualization, we interpolate the filter coefficients using the spline interpolation method.

E. Learned Wavelet Filter Versus Fixed Wavelet Filter

In our forecasting models, the wavelet filter is used to decompose time series into multiple components with different scales, which is beneficial for discovering the underlying patterns from data. Fig. 7 shows visual comparisons between the HPF and LPF of the fixed wavelet (*db*2) and the ones

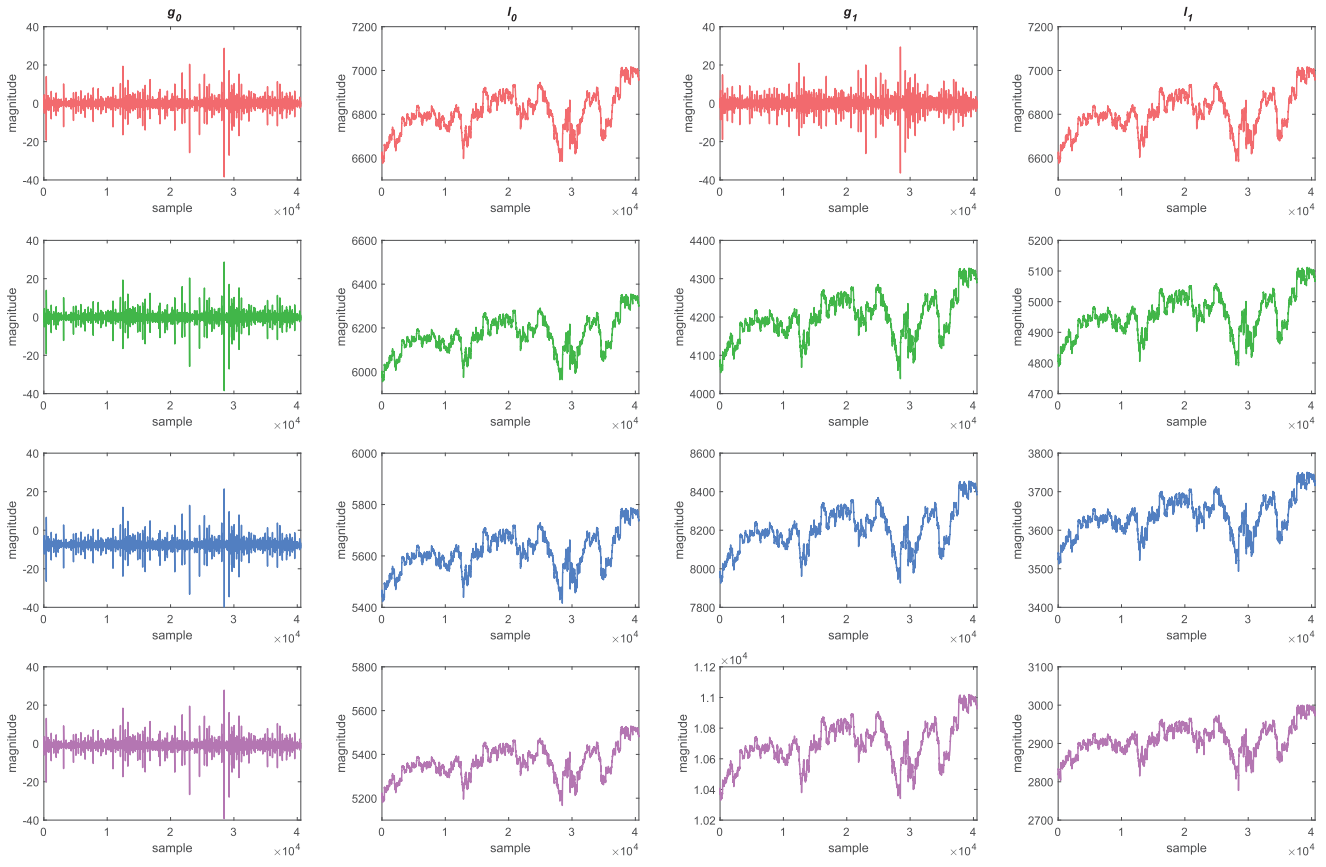

Fig. 8. Visual comparisons between different components obtained by fixed wavelets (*db*2) and ones obtained by learned wavelets on NASDAQ100. Top row (from left to right): Outputs of the fixed *db*2 filters g_0, l_0, g_1, and l_1, respectively. Second row: Outputs of the learned filters with 10k iterations (from left to right are the filters g_0, l_0, g_1, l_1). Third row: Outputs of the learned filters with 30k iterations. Bottom row: Outputs of the learned filters with 50k iterations.

learned from the training data. It can be seen that the learned filters are highly structured, like the fixed wavelet filters. For the filter g_0, the shape of its learned versions from the training data is almost entirely the same to that of the initial *db*2 filters, while for the filter g_1, the learned versions appear to shift noticeably from its initial position along the vertical direction. Different from g_0 and g_1, the learned filters l_0 and l_1 show a consistent change, having a wider dynamic range than their initial versions.

To investigate how the learned wavelet filters capture the temporal patterns in data, we analyze the multiscale components obtained by the learned wavelets and the fixed *db*2 wavelets, respectively. Specifically, given a time series, we examine the structure information localized in its different components. Fig. 8 shows the different components produced by the fixed and learned wavelet filters on the NASDAQ100 dataset. In Fig. 8, the four columns from left to right relate to the outputs of the filters g_0, l_0, g_1, l_1, respectively. From top to bottom, the four rows show the results produced by the fixed *db*2 filters, the learned filters with 10k, 30k, 50k iterations, respectively. From Fig. 8, we make the following important observations. The *db*2 wavelet filters produce an energetic hierarchy of structures localized in its different components due to the orthogonality of the wavelet bases. However, the learned filters do not have this property. It is because we do not impose an orthogonality penalty on the learned filters, which makes them work in a different manner. The component obtained by the fixed *db*2 filter g_1 looks significantly different from those by the learned filters g_1 (see the third column of Fig. 8). The former captures the high-frequency components, while the latter exhibits a pattern-preserving behavior with a higher volatility that is more beneficial for modeling inherent temporal patterns of time series by the subsequent encoders.

VI. Conclusion

In this article, we aims at fully exploring the potential of multiscale information within time series data and building scale-aware deep recurrent networks for MTS forecasting. To this end, we propose two MRN models, i.e., MRN-SA and MRN-CSG. The former applies two encoders to extract the hidden states of each scale, adaptively fuses them by a scale attention mechanism, and then generate a prediction by a decoder. The latter introduces a cross-scale guidance mechanism to directly decode the hidden states back a prediction, in which the hidden states from coarse scale is used to guide the decoding process at fine scale. Our experimental results on real-world datasets show that the proposed models can obtain significant performance improvements over several state-of-the-art forecasting methods, which reveals the effectiveness of the scale attention and cross-scale guidance mechanisms.

ACKNOWLEDGMENT

The authors would like to thank Prof. Jiaye Wang with Shandong University and the anonymous reviewers for their insightful comments and suggestions which helped them improve the quality of this article significantly.

REFERENCES

[1] S. Liu, H. Ji, and M. C. Wang, "Nonpooling convolutional neural network forecasting for seasonal time series with trends," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 8, pp. 2879–2888, Aug. 2020.

[2] S.-Y. Shih, F.-K. Sun, and H.-Y. Lee, "Temporal pattern attention for multivariate time series forecasting," *Mach. Learn.*, vol. 108, nos. 8–9, pp. 1421–1441, Sep. 2019.

[3] G. E. P. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th ed. Hoboken, NJ, USA: Wiley, 2016.

[4] H.-F. Yu, N. Rao, and I. S. Dhillon, "Temporal regularization matrix factorization for high-dimensional time series prediction," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 847–855.

[5] L. J. Cao and F. E. H. Tay, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1506–1518, Nov. 2003.

[6] N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis, "Deep adaptive input normalization for time series forecasting," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 9, pp. 3760–3765, Sep. 2020.

[7] K. Bandara, C. Bergmeir, and H. Hewamalage, "LSTM-MSNet: Leveraging forecasts on sets of related time series with multiple seasonal patterns," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 4, pp. 1586–1599, Apr. 2021.

[8] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long- and short-term temporal patterns with deep neural networks," in *Proc. 41st Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jun. 2018, pp. 95–104.

[9] P. Jing, Y. Su, X. Jin, and C. Zhang, "High-order temporal correlation model learning for time-series prediction," *IEEE Trans. Cybern.*, vol. 49, no. 6, pp. 2385–2397, Jun. 2019.

[10] S. Gultekin and J. Paisley, "Online forecasting matrix factorization," *IEEE Trans. Signal Process.*, vol. 67, no. 5, pp. 1223–1236, Mar. 2019.

[11] A. Cichocki, N. Lee, I. Oseledets, A.-H. Phan, Q. Zhao, and D. P. Mandic, "Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions," *Found. Trends Mach. Learn.*, vol. 9, nos. 4–5, pp. 249–429, 2016.

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

[13] K. Cho et al., "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.

[14] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. W. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," in *Proc. 26th Int. Joint Conf. Artif. Intell.*, Aug. 2017, pp. 2627–2633.

[15] S. Huang, D. Wang, X. Wu, and A. Tang, "DSANet: Dual self-attention network for multivariate time series forecasting," in *Proc. 28th ACM Int. Conf. Inf. Knowl. Manage.*, Nov. 2019, pp. 2129–2132.

[16] F. Petropoulos et al., "Forecasting: Theory and practice," *Int. J. Forecasting*, vol. 38, no. 3, pp. 705–871, Jul. 2022.

[17] E. S. Gardner, "Exponential smoothing: The state of the art—Part II," *Int. J. Forecasting*, vol. 22, no. 4, pp. 637–666, Oct. 2006.

[18] R. J. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting with Exponential Smoothing—The State Space Approach*. Berlin, Germany: Springer-Verlag, 2008.

[19] J. Durbin and S. J. Koopman, *Time Series Analysis by State Space Methods*, 2nd ed. London, U.K.: Oxford Univ. Press, 2012.

[20] D. T. Tran, A. Iosifidis, J. Kanniainen, and M. Gabbouj, "Temporal attention-augmented bilinear network for financial time-series data analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 5, pp. 1407–1418, May 2019.

[21] H. Hassani and D. Thomakos, "A review on singular spectrum analysis for economic and financial time series," *Statist. Interface*, vol. 3, no. 3, pp. 377–397, 2010.

[22] K. Patterson, H. Hassani, S. Heravi, and A. Zhigljavsky, "Multivariate singular spectrum analysis for forecasting revisions to real-time data," *J. Appl. Statist.*, vol. 38, no. 10, pp. 2183–2211, Oct. 2011.

[23] J. Gillard and K. Usevich, "Structured low-rank matrix completion for forecasting in time series analysis," *Int. J. Forecasting*, vol. 34, no. 4, pp. 582–597, Oct. 2018.

[24] Q. Guo, Y. Zhang, S. Qiu, and C. Zhang, "Accelerating patch-based low-rank image restoration using Kd-forest and Lanczos approximation," *Inf. Sci.*, vol. 556, pp. 177–193, May 2021.

[25] L. Jin, P. N. Nikiforuk, and M. M. Gupta, "Approximation of discrete-time state-space trajectories using dynamic recurrent neural networks," *IEEE Trans. Autom. Control*, vol. 40, no. 7, pp. 1266–1270, Jul. 1995.

[26] H. Hewamalage, C. Bergmeir, and K. Bandara, "Recurrent neural networks for time series forecasting: Current status and future directions," *Int. J. Forecasting*, vol. 37, no. 1, pp. 388–427, Jan. 2021.

[27] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.

[29] B. Zhang, D. Xiong, and J. Su, "Neural machine translation with deep attention," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 1, pp. 154–163, Jan. 2020.

[30] Y. G. Cinar et al., "Position-based content attention for time series forecasting with sequence-to-sequence RNNs," in *Proc. Int. Conf. Neural Inf. Process.*, 2017, pp. 533–544.

[31] C. Fan et al., "Multi-horizon time series forecasting with temporal attention learning," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 2527–2535.

[32] Y. Li, K. Li, C. Chen, X. Zhou, Z. Zeng, and K. Li, "Modeling temporal patterns with dilated convolutions for time-series forecasting," *ACM Trans. Knowl. Discovery from Data*, vol. 16, no. 1, pp. 1–22, Feb. 2022.

[33] J. Wang, Z. Wang, J. Li, and J. Wu, "Multilevel wavelet decomposition network for interpretable time series analysis," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2018, pp. 2437–2446.

[34] Y. Zhao, Y. Shen, Y. Zhu, and J. Yao, "Forecasting wavelet transformed time series with attentive neural network," in *Proc. 18th IEEE Int. Conf. Data Mining*, 2018, pp. 1452–1457.

[35] G. Liu et al., "Multi-scale two-way deep neural network for stock trend prediction," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 4555–4561.

[36] O. Renaud, J.-L. Starck, and F. Murtagh, "Prediction based on a multiscale decomposition," *Int. J. Wavelets, Multiresolution Inf. Process.*, vol. 1, no. 2, pp. 217–232, Jun. 2003.

[37] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed. New York, NY, USA: Academic, 2008.

[38] G. P. Nason and B. W. Silverman, "The stationary wavelet transform and some statistical applications," *Wavelets and Statistics* (Lecture Notes in Statistics), vol. 103. New York, NY, USA: Springer, 1995, pp. 281–299.

[39] J.-L. Starck, J. Fadili, and F. Murtagh, "The undecimated wavelet decomposition and its reconstruction," *IEEE Trans. Image Process.*, vol. 16, no. 2, pp. 297–309, Feb. 2007.

[40] C.-Y. Lee., S. Xie, P. W. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proc. 18th Int. Conf. Artif. Intell. Statist.*, 2015, pp. 562–570.

[41] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue, "Object detection from scratch with deep supervision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 398–412, Feb. 2020.

[42] A. Mustafa, S. H. Khan, M. Hayat, R. Goecke, J. Shen, and L. Shao, "Deeply supervised discriminative learning for adversarial defense," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 9, pp. 3154–3166, Sep. 2021.

[43] C. Li, M. Z. Zia, Q.-H. Tran, X. Yu, G. D. Hager, and M. Chandraker, "Deep supervision with intermediate concepts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 8, pp. 1828–1843, Aug. 2019.

[44] Q. Shi et al., "Block Hankel tensor ARIMA for multiple short time series forecasting," in *Proc. AAAI Conf. Artif. Intell.*, Apr. 2020, pp. 5758–5766.

[45] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.

[46] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implement.*, 2016, pp. 265–283.