Multi-Scale Adaptive Graph Neural Network for Multivariate Time Series Forecasting

Ling Chen, Donghui Chen, Zongjiang Shang, Youdong Zhang, Bo Wen, and Chenghu Yang

***Abstract*—Multivariate time series (MTS) forecasting plays an important role in the automation and optimization of intelligent applications. It is a challenging task, as we need to consider both complex intra-variable dependencies and inter-variable dependencies. Existing works only learn temporal patterns with the help of single inter-variable dependencies. However, there are multi-scale temporal patterns in many real-world MTS. Single inter-variable dependencies make the model prefer to learn one type of prominent and shared temporal patterns. In this paper, we propose a multi-scale adaptive graph neural network (MAGNN) to address the above issue. MAGNN exploits a multi-scale pyramid network to preserve the underlying temporal dependencies at different time scales. Since the inter-variable dependencies may be different under distinct time scales, an adaptive graph learning module is designed to infer the scale-specific inter-variable dependencies without pre-defined priors. Given the multi-scale feature representations and scale-specific inter-variable dependencies, a multi-scale temporal graph neural network is introduced to jointly model intra-variable dependencies and inter-variable dependencies. After that, we develop a scale-wise fusion module to effectively promote the collaboration across different time scales, and automatically capture the importance of contributed temporal patterns. Experiments on four real-world datasets demonstrate that MAGNN outperforms the state-of-the-art methods across various settings.**

***Index Terms*—Multivariate time series forecasting, multi-scale modeling, graph neural network, graph learning.**

I. Introduction

Multivariate time series (MTS) are ubiquitous in various real-world scenarios, e.g., the traffic flows in a city, the stock prices in a stock market and the household power consumption in a city block [1]. MTS forecasting, which aims at forecasting the future trends based on a group of historical observed time series, has been widely studied in recent years. It is of great importance in a wide range of applications, e.g., a better driving route can be planned in advance based on the forecasted traffic flows, and an investment strategy can be designed with the forecasting of the near-future stock market [2]–[5].

Making accurate MTS forecasting is a challenging task, as both intra-variable dependencies (i.e., the temporal dependencies within one time series) and inter-variable dependencies (i.e., the forecasting values of a single variable are affected by other variables) need to be considered jointly. To solve this problem, traditional methods [6]–[8], e.g., vector auto-regression (VAR), temporal regularized matrix factorization (TRMF), vector auto-regression moving average (VARMA), and gaussian process (GP), often rely on the strict stationary assumption and cannot capture the non-linear dependencies among variables. Deep neural networks have shown superiority on modeling non-stationary and non-linear dependencies. Particularly, two variants of recurrent neural network (RNNs) [9], namely the long-short term memory (LSTM) and the gated recurrent unit (GRU), and temporal convolutional networks (TCNs) [10] have significantly achieved impressive performance in time series modeling. To capture both long-term and short-term temporal dependencies, existing works [3], [11]–[14] introduce several strategies, e.g., skip-connection, attention mechanism, and memory-based network. These works focus on modeling temporal dependencies, and process the MTS input as vectors and assume that the forecasting values of a single variable are affected by all other variables, which is unreasonable and hard to meet in realistic applications. For example, the traffic flows of a street are largely affected by its neighboring streets, while the impact from distant streets is relatively small. Thus, it is crucial to model the pairwise inter-variable dependencies explicitly.

Graph is an abstract data type representing relations between nodes. Graph neural networks (GNNs) [15], [16], which can effectively capture nodes' high-level representations while exploiting pairwise dependencies, have been considered as a promising way to handle graph data. MTS forecasting can be considered from the perspective of graph modeling. The variables in MTS can be regarded as the nodes in a graph, while the pairwise inter-variable dependencies as edges. Recently, several works [17]–[19] exploit GNNs to model MTS taking advantage of the rich structural information (i.e., featured nodes and weighted edges) of a graph. These works stack GNN and temporal convolution modules to learn temporal patterns, and have achieved promising results. Nevertheless, there are still two important aspects neglected in above works.

First, existing works only consider temporal dependencies on a single time scale, which may not properly reflect the variations in many real-world scenarios. In fact, the temporal patterns hidden in real-world MTS are much more complicated, including daily, weekly, monthly, and other specific periodic patterns. For example, Fig. 1 shows the power consumptions of 4 households within two weeks. There exists a mixture of short-term and long-term repeating patterns (i.e., daily and weekly). These multi-scale temporal patterns provide

This work was supported by the National Key Research and Development Program of China under Grant 2018YFB0505000. (*Ling Chen and Donghui Chen are co-first authors.*) (*Corresponding author: Ling Chen.*)

Ling Chen, Donghui Chen, and Zongjiang Shang are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: {lingchen, chendonghui, zongjiangshang}@cs.zju.edu.cn).

Youdong Zhang, Bo Wen, and Chenghu Yang are with Alibaba Group, Hangzhou 311100, China (e-mail: {linqing.zyd, wenbo.wb}@alibaba-inc.com; yexiang.ych@taobao.com).

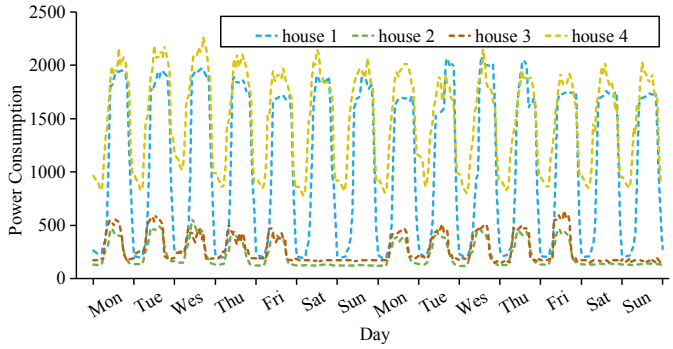


Fig. 1. The power consumptions of 4 households within two weeks (from Monday 00:00 to Sunday 24:00). Households 1 and 4 have both daily and weekly repeating patterns, while households 2 and 3 have weekly repeating patterns.

abundant information to model MTS. Furthermore, if the temporal patterns are learned from different time scale separately, and are then straightforwardly concatenated to obtain the final representation, the model is failed to capture cross-scale relationships and cannot focus on contributed temporal patterns. Thus, an accurate MTS forecasting model should learn a feature representation that can comprehensively reflect all kinds of multi-scale temporal patterns.

Second, existing works learn a shared adjacent matrix to represent the rich inter-variable dependencies, which makes the models be biased to learn one type of prominent and shared temporal patterns. In fact, different kinds of temporal patterns are often affected by different inter-variable dependencies, and we should distinguish the inter-variable dependencies when modeling distinct temporal patterns. For example, when modeling the short-term patterns of the power consumptions of a household, it might be essential to pay more attention to the power consumptions of its neighbors. Because the dynamics of short-term patterns are often affected by a common event, e.g., a transmission line fault decreases the power consumptions of a street block, and a sudden cold weather increases the power consumptions. When modeling the long-term patterns of the power consumptions of a household, it might be essential to pay more attention to the households that have similar living habits, e.g., working and sleeping hours, as these households would have similar daily and weekly temporal patterns. Therefore, the complicated inter-variable dependencies need to be fully considered when modeling these multi-scale temporal patterns.

In this paper, we propose a general framework termed Multi-scale Adaptive Graph Neural Network (MAGNN) for MTS forecasting to address above issues. Specifically, we introduce a multi-scale pyramid network to decompose the time series with different time scales in a hierarchical way. Then, an adaptive graph learning module is designed to automatically infer the scale-specific graph structures in the end-to-end framework, which can fully explore the abundant and implicit inter-variable dependencies under different time scales. After that, a multi-scale temporal graph neural network is incorporated into the framework to model intra-variable dependencies and inter-variable dependencies at each time scale. Finally, a scale-wise fusion module is designed to automatically consider the importance of scale-specific representations and capture the cross-scale correlations. In summary, our contributions are as follows:

- Propose MAGNN, which learns a temporal representation that can comprehensively reflect both multi-scale temporal patterns and the scale-specific inter-variable dependencies.
- Design an adaptive graph learning module to explore the abundant and implicit inter-variable dependencies under different time scales, and a scale-wise fusion module to promote the collaboration across these scale-specific temporal representations and automatically capture the importance of contributed temporal patterns.
- Conduct extensive experiments on four real-world MTS benchmark datasets. The experiment results demonstrate that the performance of our method is better than that of the state-of-the-art methods.

The remainder of this paper is organized as follows: Section II and Section III give a survey of related work and preliminaries. Section IV describes the proposed MAGNN method. Section V presents the experimental results and Section VI concludes the paper.

II. Related Work

We briefly review the related work from two aspects: the MTS forecasting and graph learning for MTS.

A. MTS Forecasting

The problem of time series forecasting has been studied for decades. One of the most prominent traditional methods used for time series forecasting is the auto-regressive integrated moving average (ARIMA) model, because of its statistical properties and the flexibility on integrating several linear models, including auto-regression (AR), moving average, and auto-regressive moving average. However, limited by the high computational complexity, ARIMA is infeasible to model MTS. Vector auto-regression (VAR) and vector auto-regression moving average (VARMA) are the extension of AR and ARIMA, respectively, that can model MTS. Gaussian process (GP) [6] is a Bayesian method to model distributions over a continuous domain of functions. GP can be used as a prior over the function space in Bayesian inference and has been applied to MTS forecasting. However, these works often rely on the strict stationary assumption and cannot capture the non-linear dependencies among variables.

Recently, deep learning-based methods have shown superior capability on capturing non-stationary and non-linear dependencies. Most of existing works rely on LSTM and GRU to capture temporal dependencies. Some efforts exploit TCNs and self-attention mechanism [20] to model long time series efficiently. To capture both long-term and short-term temporal dependencies, LSTNet [3] introduces the convolutional neural network to capture short-term temporal dependencies, and a recurrent-skip layer that can exploit the long-term periodic property hidden in time series. TPA-LSTM [13] utilizes an

attention mechanism, which enables the model to extract important temporal patterns and focus on different time steps for different variables. MTNet [12] exploits the memory component and attention mechanism to effectively capture long-term temporal dependencies and periodic patterns. However, these works assume that each variable affects all other variables equally, which is unreasonable and hard to meet in realistic applications.

B. Graph Learning for MTS

Graph neural networks (GNNs) [15], [16], which can model the interaction between nodes through weighted edges, have received increasing attention. Recently, there are many works using GNNs to capture inter-variable dependencies in the area of MTS modeling. One of the challenges of the GNNs-based MTS forecasting is to obtain a well-defined graph structure as the inter-variable dependencies. To solve this problem, existing methods can be roughly divided into three major categories: prior-knowledge-based, rule-based, and learning-based methods.

Prior-knowledge-based methods [21]–[24] often exploit the extra information (e.g., road networks, physical structures, and extra feature matrices) in their specific scenarios. For example, in traffic flow forecasting [21], [23], the graph structure can be constructed by the connections of road networks. If there is a connected road between two nodes, an edge is constructed in the graph structure, as the traffic flow at the upstream node will affect the traffic flow at the downstream node. In skeleton-based action recognition [24], the graph structure can be constructed by the physical structure of the human body, e.g., the multiple joints on the same arm are linked by the human skeleton, and edges can be constructed between these joints. In the ride-hailing demand forecasting [22], multiple different graph structures are constructed from different views: the proximity of spatial distance, the connection of urban road network, and the similarity of region functionality. However, these methods require domain knowledge to design a graph structure, which is difficult to transfer between different scenarios.

Rule-based methods [19], [25]–[27], as non-parametric methods, provide a data-driven manner to construct the graph structure. These methods usually include causal discovery (e.g., Granger causality and additive noise model) [25], [27], entropy-based methods (e.g., transfer entropy and relative entropy) [19], similarity-based methods (e.g., Pearson correlation, mutual information, DTW distance, and edit distance) [26]. For example, Huang et al. [27] used Granger causality to construct a causal graph. Xu et al. [19] calculated the pairwise transfer entropy between variables, which is regarded as the adjacency matrix of the graph structure. He et al. [26] exploited dynamic time warping (DTW) algorithm, which is competent to capture the pattern similarities between two time series. However, these methods are non-parameterized methods and have limited flexibility, which can only learn a kind of specific inter-variable dependency.

Learning-based methods [17], [18], [28]–[33] introduce a parameterized module to learn pairwise inter-variable dependencies automatically. Kipf et al. [30] first introduced a neural relational inference model, which uses the original time series as input and exploits the variational inference to learn a graph structure. Subsequently, Webb et al. [32] proposed a decomposition-based neural relational inference model to learn multiple types of graph structure. Graber et al. [29] proposed a neural relational inference model that achieves different graph structures at each time step. The attention-based learning methods use the attention mechanism to learn the pairwise inter-variable dependencies. For traffic flow forecasting, Tang et al. [31] used a graph attention module to learn graph structure. Zheng et al. [33] used spatial attention mechanism to learn the correlation of traffic flow at different nodes. In addition, several works achieve this more directly, i.e., randomly intializing the representation of each node, and calculating the pairwise similarity of these nodes. The representations of the nodes can be optimized to obtain the most suitable value for the current data distribution. For example, Wu et al. [18] exploited a graph learning module to learn inter-variable dependencies, and modelled MTS using the GNNs and dilated convolution networks. Bai et al. [17] introduced a data adaptive graph generation module to infer the inter-variable dependencies and a node adaptive parameter learning module to capture node-specific features. However, existing works only learn single inter-variable dependencies, making the models biased to learn one type of prominent and shared temporal patterns among MTS.

III. Preliminaries

A. Problem Formulation

Problem Statement. In this paper, we focus on MTS forecasting task. Formally, given a sequence of observed time series signals $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_t, ..., \boldsymbol{x}_T\}$, where $\boldsymbol{x}_t \in \mathbb{R}^N$ denotes the values at time step t, N is the variable dimension, and $\boldsymbol{x}_{t,i}$ denotes the value of the i^{th} variable at time step t, MTS forecasting task aims at forecasting the future values $\widehat{\boldsymbol{x}}_{T+h} \in \mathbb{R}^N$ at time step $T+h$, where h denotes the look-ahead horizon. The problem can be formulated as:

$$\widehat{\boldsymbol{x}}_{T+h} = \mathcal{F}\left(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T; \theta\right), \tag{1}$$

where $\mathcal{F}$ is the mapping function and θ denotes all learnable parameters.

Then, we give several definitions of MTS from the perspective of graph.

Definition 1. *MTS to Graph.* A graph is defined as $G = (V, E)$, where V denotes the node set and $|V| = N$. E is the edge set. Given the MTS $\boldsymbol{X} \in \mathbb{R}^{N \times T}$, the i^{th} variable is regarded as the i^{th} node $v_i \in V$, the values of $\{\boldsymbol{x}_{1,i}, \boldsymbol{x}_{2,i}, ..., \boldsymbol{x}_{T,i}\}$ are the features of v_i, and each edge $(v_i, v_j) \in E$ indicates there is a connection between v_i and v_j. Then, MTS forecasting is modified as:

$$\widehat{\boldsymbol{x}}_{T+h} = \mathcal{F}\left(\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_T; G; \theta\right). \tag{2}$$

Definition 2. *Weighted Adjacency Matrix.* The weighted adjacency matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$ of a graph is a type of mathematical representation to store the weights of the edges, where $\boldsymbol{A}_{i,j} > 0$, if $(v_i, v_j) \in E$, and $\boldsymbol{A}_{i,j} = 0$, if $(v_i, v_j) \notin E$.

We focus on modeling pure MTS data without any prior knowledge, and the weighted adjacency matrix of the graph will be learned by MAGNN.

B. Graph Neural Networks

Graph neural networks (GNNs) [15], [16] are a type of deep neural network applied to graphs. Graphs can be irregular, a graph may have a variable size of unordered nodes, and nodes from a graph may have a different numbers of neighbors. GNNs can be easy to compute in the graph domain, which can overcome the limitation of CNNs.

GNNs can be divided into two categories based on the implementation philosophy: spectral-based and spatial-based methods [15]. Spectral-based methods define the graph convolution by introducing a filter from the perspective of graph signal processing. The graph convolution operation can be interpreted as removing noise from the graph signal. Space-based methods define the graph convolution through information propagation, which aggregates the representation of a central node and the representations of its neighbors to get the updated representation for the node.

We briefly describe the graph convolution operation applied in our method, which can be defined as:

$$\boldsymbol{x} *_G \theta = \sigma(\theta(\widetilde{\boldsymbol{D}}^{-\frac{1}{2}}\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{D}}^{-\frac{1}{2}})\boldsymbol{x}), \tag{3}$$

where $G = (V, E, \boldsymbol{A})$ is a graph with a weighted adjacency matrix, $\boldsymbol{x}$ is the representations of nodes, σ is an activation function, θ is the learnable parameter matrix, $\widetilde{\boldsymbol{A}} = \boldsymbol{I}_n + \boldsymbol{A}$ is the adjacency matrix with self-connection, $\widetilde{\boldsymbol{D}}$ is the diagonal degree matrix of $\widetilde{\boldsymbol{A}}$, and $\widetilde{\boldsymbol{D}}_{ii} = \sum_j \widetilde{\boldsymbol{A}}_{ij}$. By stacking the graph convolution operation multiple layers, we can aggregate the information of multi-order neighbors.

IV. METHODOLOGY

A. Framework

Fig. 2 illustrates the framework of MAGNN, which consists of four main parts: a) a multi-scale pyramid network to preserve the underlying temporal hierarchy at different time scales; b) an adaptive graph learning module to automatically infer inter-variable dependencies; c) a multi-scale temporal graph neural network to capture all kinds of scale-specific temporal patterns; d) a scale-wise fusion module to effectively promote the collaboration across different time scales.

B. Multi-Scale Pyramid Network

A multi-scale pyramid network is designed to preserve the underlying temporal dependencies at different time scales. Following the pyramid structure, it applies multiple pyramid layers to hierarchically transform raw time series into feature representations from smaller scale to larger scale. Such multi-scale structure gives us the opportunity to observe raw time series in different time scales. Specifically, the smaller scale feature representations can retain more fine-grained details, while the larger scale feature representations can capture the slow-varying trends.

Multi-scale pyramid network generates multi-scale feature representations through pyramid layers. Each pyramid layer takes the outputs of a preceding pyramid layer as the input and generates the feature representations of a larger scale as the output. Specifically, given the input MTS $\boldsymbol{X} \in \mathbb{R}^{N \times T}$, the multi-scale pyramid network generates feature representations of K scales, and the k^{th} scale feature representation is denoted as $\boldsymbol{X}^k \in \mathbb{R}^{N \times \frac{T}{2^{k-1}} \times c^k}$, where N is the variable dimension, $\frac{T}{2^{k-1}}$ is the sequence length in the k^{th} scale, and c^k is the channel size of the k^{th} scale.

A pyramid layer takes convolutional neural networks to capture local patterns in the time dimension. Following the design philosophy of image processing, different pyramid layers employ different kernel sizes. The beginning convolution kernel has larger filter, and the size is slowly decreased at each pyramid layer, which can control the receptive field size and maintain the sequence characteristics of large scale time series. For example, the kernel sizes can be set as 1×7, 1×6, and 1×3 at each pyramid layer, and the stride size of convolution is set to 2 to increase the time scale. Formally,

$$\boldsymbol{X}^k_{\text{rec}} = ReLU(\boldsymbol{W}^k_{\text{rec}} \otimes \boldsymbol{X}^{k-1} + \boldsymbol{b}^k_{\text{rec}}), \tag{4}$$

where $\otimes$ denotes convolution operator. $\boldsymbol{W}^k_*$ and $\boldsymbol{b}^k_*$ denote the convolution kernel and bias vector in the k^{th} pyramid layer, respectively. However, different pyramid layers are expected to preserve the underlying temporal dependencies at different time scales. The flexibility of using only one convolutional neural network is limited, as the granularities of the temporal dependencies captured in the feature representations at two consecutive pyramid layers are highly sensitive to the hyper-parameter settings (i.e., kernel size and stride size). To alleviate this issue, following the existing works in image processing [34], [35], we introduce another convolutional neural network with kernel size 1×1 and a 1×2 pooling layer, which is a parallel structure with the original convolutional neural network, formally,

$$\boldsymbol{X}^k_{\text{norm}} = Pooling\left(ReLU(\boldsymbol{W}^k_{\text{norm}} \otimes \boldsymbol{X}^{k-1} + \boldsymbol{b}^k_{\text{norm}})\right). \tag{5}$$

Then, a point-wise addition is utilized to the outputs of these two convolutional neural networks at each scale:

$$\boldsymbol{X}^k = \boldsymbol{X}^k_{\text{rec}} + \boldsymbol{X}^k_{\text{norm}}. \tag{6}$$

After that, the learned multi-scale feature representations are flexible and comprehensive to preserve various kinds of temporal dependencies. During the process of feature representation learning, to avoid the interaction between the variables of MTS, the convolutional operations are performed on the time dimension, and the variable dimension is fixed, i.e., the kernels are shared between the variable dimension at each pyramid layer.

C. Adaptive Graph Learning

The adaptive graph learning module automatically generates adjacency matrices to represent the inter-variable dependencies among MTS. Existing learning-based methods [15], [16] only learn a shared adjacency matrix, which is useful to learn the most prominent inter-variable dependencies among MTS in

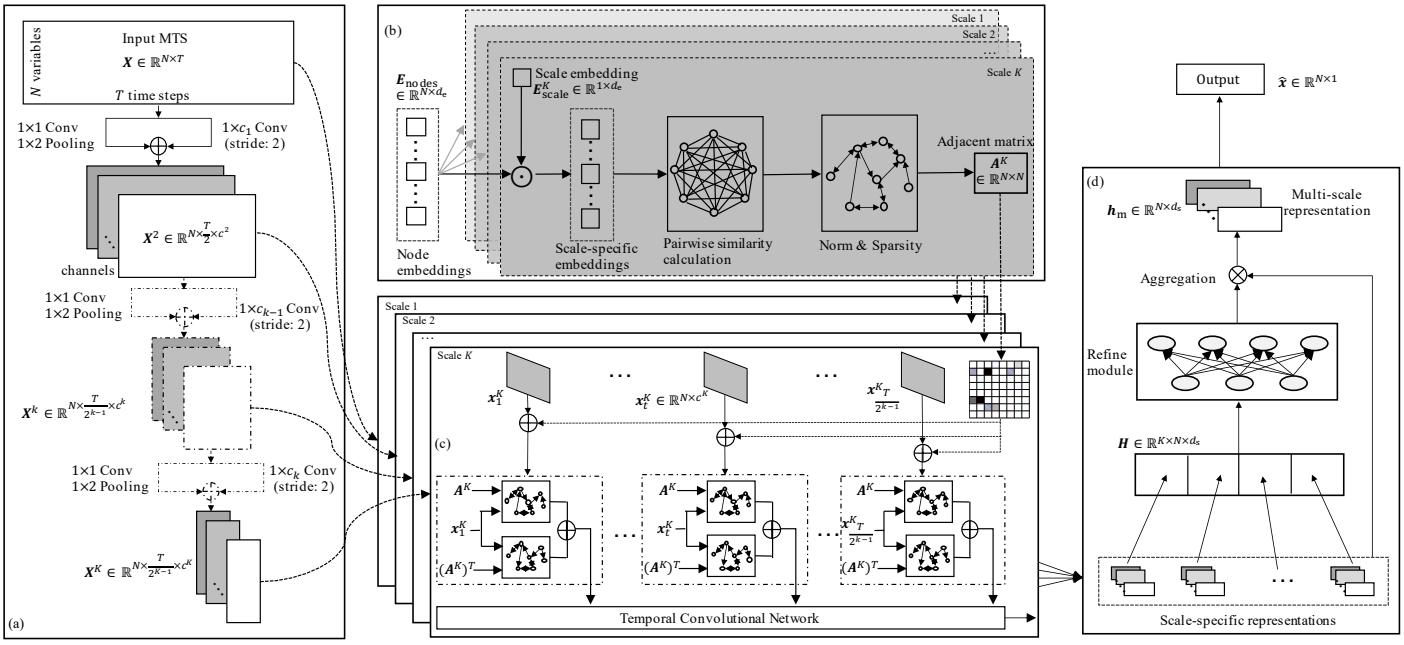


Fig. 2. The Multi-scale Adaptive Graph Neural Network (MAGNN) framework, which consists of four main parts: (a) The two parallel convolutional neural networks and point-wise additions at each layer transform feature representations from smaller scale to larger scale hierarchically. (b) An adaptive graph learning module takes node embeddings and scale embeddings as inputs and outputs the scale-specific adjacency matrices. (c) Each scale-specific feature representation and adjacency matrix are fed into a temporal GNN to obtain scale-specific representations. (d) Scale-specific representations are weighted fused to capture the contributed temporal patterns.

many problems, and can significantly reduce the number of parameters and avoid the overfitting problem. However, the inter-variable dependencies may be different under different time scales. The shared adjacency matrix makes the models biased to learn one type of prominent and shared temporal patterns. Therefore, it is essential to learn multiple scale-specific adjacency matrices.

However, directly learning a unique adjacent matrix for each scale will introduce too many parameters and make the model hard to train, especially when the number of nodes is large. To solve this problem, we propose an adaptive graph learning (AGL) module, which is inspired by the matrix factorization. AGL initializes two kinds of parameters: 1) shared node embeddings $\boldsymbol{E}_{\text{nodes}} \in \mathbb{R}^{N \times d_e}$ between all scales, where d_e is the embedding dimension and $d_e \ll N$; 2) scale embeddings $\boldsymbol{E}_{\text{scale}} \in \mathbb{R}^{K \times d_e}$. From the view of one scale, we can extract scale-specific intra-variable dependencies that capture both the common information and scale-specific information. Specifically, the AGL module includes shared node embeddings $\boldsymbol{E}_{\text{nodes}}$ and K scale-specific layers. First, the learnable $\boldsymbol{E}_{\text{nodes}}$ are randomly initialized and fed into the scale-specific layers. For the k^{th} scale-specific layer, the k^{th} scale embedding $\boldsymbol{E}^k_{\text{scale}} \in \mathbb{R}^{1 \times d_e}$ is randomly initialized to conduct point-wise multiplication with $\boldsymbol{E}_{\text{nodes}}$:

$$\boldsymbol{E}^k_{\text{spec}} = \boldsymbol{E}_{\text{nodes}} \odot \boldsymbol{E}^k_{\text{scale}}, \tag{7}$$

where $\boldsymbol{E}^k_{\text{spec}} \in \mathbb{R}^{N \times d_e}$ denote the scale-specific embedding in the k^{th} layer. $\boldsymbol{E}^k_{\text{spec}}$ contains both the shared node information and the scale-specific information. Then, similar as calculating the node proximities by a similarity function, we calculate pairwise node similarities as follows:

$$\begin{aligned} \boldsymbol{M}^k_1 &= [tanh(\boldsymbol{E}^k_{\text{spec}}\theta^k)]^T, \\ \boldsymbol{M}^k_2 &= tanh(\boldsymbol{E}^k_{\text{spec}}\varphi^k), \\ \boldsymbol{A}^k_{\text{full}} &= ReLU(\boldsymbol{M}^k_1\boldsymbol{M}^k_2 - (\boldsymbol{M}^k_2)^T(\boldsymbol{M}^k_1)^T), \end{aligned} \tag{8}$$

where $\theta^k \in \mathbb{R}^{1\times 1}$ and $\varphi^k \in \mathbb{R}^{1\times 1}$ are learnable parameters. The values of $\boldsymbol{A}^k_{\text{full}} \in \mathbb{R}^{N\times N}$ are then normalized to $0-1$, which is used as the soft edges among the nodes. To reduce the computation cost of the graph convolution, reduce the impact of noise, and make the model more robust, we introduce a strategy to make $\boldsymbol{A}^k_{\text{full}}$ sparse:

$$\boldsymbol{A}^k = Sparse(Softmax\left(\boldsymbol{A}^k_{\text{full}}\right)), \tag{9}$$

where $\boldsymbol{A}^k \in \mathbb{R}^{N\times N}$ is the final adjacent matrix of the k^{th} layer, $Softmax$ function is used to achieve normalization, and $Sparse$ function is defined as:

$$\boldsymbol{A}^k_{ij} = \begin{cases} \boldsymbol{A}^k_{ij}, & \boldsymbol{A}^k_{ij} \in TopK(\boldsymbol{A}^k_{i*}, \tau) \\ 0, & \boldsymbol{A}^k_{ij} \notin TopK(\boldsymbol{A}^k_{i*}, \tau) \end{cases}, \tag{10}$$

where τ is the threshold of $TopK$ function and denotes the max number of neighbors of a node. The overall architecture of the AGL module is shown in Fig. 3. Finally, we can obtain the scale-specific adjacent matrices $\{\boldsymbol{A}^1, \ldots, \boldsymbol{A}^k, \ldots, \boldsymbol{A}^K\}$.

D. Multi-Scale Temporal Graph Neural Network

Given the multi-scale feature representations $\{\boldsymbol{X}^1, \ldots, \boldsymbol{X}^k, \ldots, \boldsymbol{X}^K\}$ generated from the multi-scale pyramid network, and the scale-specific adjacent matrices $\{\boldsymbol{A}^1, \ldots, \boldsymbol{A}^k, \ldots, \boldsymbol{A}^K\}$ generated from the AGL module, a

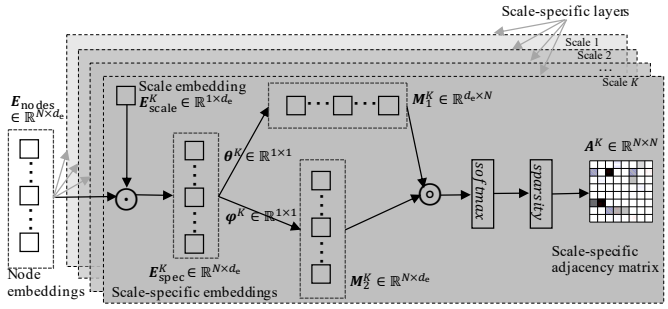


Fig. 3. The detailed architecture of the adaptive graph learning module.

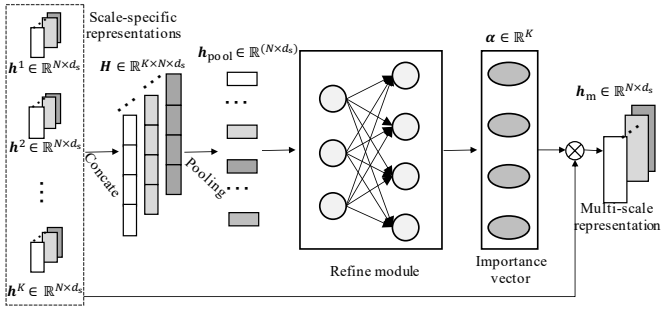


Fig. 4. The detailed architecture of the scale-wise fusion module.

multi-scale temporal graph neural network (MTG) is proposed to capture scale-specific temporal patterns across time steps and variables.

Existing works [17], [21] integrate the GRU and the GNN, which replaces the MLP in the GRU with the GNN to learn inter-variable dependencies. However, the RNN-based solutions often suffer from the gradient vanishing and exploding problems, and adopt the step-by-step strategy for recurrent layers, which makes the model training inefficient, especially when the time series is long enough [36]. Temporal convolutional networks (TCNs) have shown superiority on modeling temporal patterns. Thus, we propose a solution that combines the GNNs and temporal convolution layers, i.e., replacing the GRU with temporal convolution layers.

Specifically, the MTG consists of K temporal graph neural networks, each of which combines the TCNs and the GNN to capture scale-specific temporal patterns. For the k^{th} scale, we first split $\boldsymbol{X}^k$ at time dimension and obtain $\{\boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_t^k, \ldots, \boldsymbol{x}_{\frac{T}{2^{k-1}}}^k\}(\boldsymbol{x}_t^k \in \mathbb{R}^{N \times c^k})$. Similar with [18], [37], we introduce $\boldsymbol{A}^k$ and the transpose of $\boldsymbol{A}^k$ (i.e., $(\boldsymbol{A}^k)^T$), and exploit two GNNs to capture both incoming information and outgoing information. Then, the results of GNNs are added:

$$\widetilde{\boldsymbol{h}}_t^k = GNN_{\text{in}}^k(\boldsymbol{x}_t^k, \boldsymbol{A}^k, \boldsymbol{W}_{\text{in}}^k) + GNN_{\text{out}}^k(\boldsymbol{x}_t^k, (\boldsymbol{A}^k)^T, \boldsymbol{W}_{\text{out}}^k), \tag{11}$$

where $\boldsymbol{W}_*^k$ denotes the trainable parameters of GNNs in the k^{th} scale. Then, we can obtain all the outputs $\{\widetilde{\boldsymbol{h}}_1^k, \ldots, \widetilde{\boldsymbol{h}}_t^k, \ldots, \widetilde{\boldsymbol{h}}_{\frac{T}{2^k}}^k\}$, which are fed into a temporal convolution layer to obtain the scale-specific representations $\boldsymbol{h}^k$:

$$\boldsymbol{h}^k = TCN^k(\left[\widetilde{\boldsymbol{h}}_1^k, \ldots, \widetilde{\boldsymbol{h}}_t^k, \ldots, \widetilde{\boldsymbol{h}}_{\frac{T}{2^k}}^k\right], \boldsymbol{W}_{\text{tcn}}^k), \tag{12}$$

where $\boldsymbol{W}_{\text{tcn}}^k$ denotes the trainable parameters in the k^{th} temporal convolution layer.

We can see the advantages of exploiting MTG: 1) it can capture scale-specific temporal patterns across time steps and variables; 2) the graph convolution operator enables the model to explicitly consider the inter-variable dependencies.

E. Scale-Wise Fusion

All the scale-specific representations $\{\boldsymbol{h}^1, ..., \boldsymbol{h}^k, ..., \boldsymbol{h}^K\}$ can comprehensively reflect all kinds of temporal patterns, where $\boldsymbol{h}^k \in \mathbb{R}^{N \times d_s}$, and d_s denotes the output dimension of TCNs. To obtain the final multi-scale representation, the intuitive solution is to directly concatenate these scale-specific representations or aggregate these representations by a global pooling layer. However, this solution treats each scale-specific representation equally and ignores the difference in contribution to the final forecasting results. For example, the small scale representations are more important for short-term forecasting, while the large scale representations are more important for long-term forecasting. Thus, we propose a scale-wise fusion module to learn a robust multi-scale representation from these scale-specific representations, which can consider the importance of scale-specific temporal patterns and capture the cross-scale correlations.

Fig. 4 shows the overall architecture of the scale-wise fusion module. Given the scale-specific representations $\{\boldsymbol{h}^1, ..., \boldsymbol{h}^k, ..., \boldsymbol{h}^K\}$, we first concatenate these representations to obtain the multi-scale matrix $\boldsymbol{H} \in \mathbb{R}^{K \times N \times d_s}$:

$$\boldsymbol{H} = Concat(\boldsymbol{h}^1, \ldots, \boldsymbol{h}^k, \ldots, \boldsymbol{h}^K), \tag{13}$$

where $Concat$ denotes the concatenation operation. Then, we exploit an average pooling layer on the scale dimension to obtain the point-wise dependencies:

$$\boldsymbol{h}_{\text{pool}} = \frac{\sum_{k=1}^{K} \boldsymbol{H}^k}{K}, \tag{14}$$

where $\boldsymbol{h}_{\text{pool}} \in \mathbb{R}^{1 \times N \times d_s}$. Then, we flat $\boldsymbol{h}_{\text{pool}}$ to a 1-D vector $\boldsymbol{h}_{\text{pool}} \in \mathbb{R}^{N \times d_s}$, and fed it into a refining module that consists of two full connected layers to compact the fine-grained information across different time scales:

$$\begin{aligned} \boldsymbol{\alpha}_1 &= ReLU\left(\boldsymbol{W}_1 \boldsymbol{h}_{\text{pool}} + \boldsymbol{b}_1\right), \\ \boldsymbol{\alpha} &= Sigmoid\left(\boldsymbol{W}_2 \boldsymbol{\alpha}_1 + \boldsymbol{b}_2\right), \end{aligned} \tag{15}$$

where $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ are weight matrices. $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are bias vectors. The $sigmoid$ activation function is used in the second layer. $\boldsymbol{\alpha} \in \mathbb{R}^K$ is defined as the importance score vector that represents the importance of different scale-specific representations. Finally, an aggregation layer is exploited to weighted combine the scale-specific representations:

$$\boldsymbol{h}_{\text{m}} = ReLU(\sum_{k=1}^{K} \boldsymbol{\alpha}[k] \times \boldsymbol{h}^k), \tag{16}$$

where $\boldsymbol{h}_{\text{m}}$ is the final multi-scale representation.

F. Output Module & Objection Function

The output module includes a convolutional neural network with $1 \times d_{\mathrm{s}}$ kernel size to transform $\boldsymbol{h}_{\mathrm{m}} \in \mathbb{R}^{N \times d_{\mathrm{s}}}$ into the desired output dimension, and a followed convolutional neural network with 1×1 kernel size to obtain the predicted values $\widehat{\boldsymbol{x}} \in \mathbb{R}^{N \times 1}$.

We choose $\mathcal{L}_2$ loss as the objection function of the forecasting task, which can be formulated as:

$$\mathcal{L}_2 = \frac{1}{\mathcal{T}_{\text{train}}} \sum_{i=1}^{\mathcal{T}_{\text{train}}} \sum_{j=1}^{N} \left(\widehat{\boldsymbol{x}}_{i,j} - \boldsymbol{x}_{i,j}\right)^2, \tag{17}$$

where $\mathcal{T}_{\text{train}}$ is the number of training samples, and N is the number of variables. $\widehat{\boldsymbol{x}}_{i,j}$ and $\boldsymbol{x}_{i,j}$ are the predicted value and ground-truth of the j^{th} variable in the i^{th} sample, respectively.

G. Complexity Analysis

The time complexity of MAGNN consists of the main four modules. For the multi-scale pyramid network, the time complexity of the k^{th} scale is $\Theta\left(N \times \frac{T}{2^{k-1}} \times c_{\text{in}} \times c_{\text{out}}\right)$ and the overall time complexity is $\Theta\left(N \times T \times c_{\text{in}} \times c_{\text{out}}\right)$, where N is the variable dimension, T is the input sequence length, c_{in} and c_{out} are the numbers of input channels and output channels, respectively. Since c_{in} and c_{out} are regarded as constants, the time complexity of the multi-scale pyramid network is $\Theta\left(N \times T\right)$. For the AGL module, the time complexity is $\Theta\left(K \times N \times d_{\text{e}}^2 + K \times N^2 \times d_{\text{e}}\right)$, where K is the number of scales and d_{e} is the dimension of node or scale embedding. The first half part denotes the point-wise multiplication between node embeddings and scale embeddings. The latter part denotes the pairwise similarity calculation. Since d_{e} is regarded as a constant, the time complexity of the AGL module is $\Theta\left(K \times N^2\right)$. For the MTG module, the time complexity is $\Theta\left(K\left(m \times d_1 + N \times d_{\text{in}} \times d_{\text{s}}\right)\right)$, where m denotes the number of edges. d_{in} and d_{s} denote the input dimension and output dimension, respectively. This result comes from the message passing and information aggregation of GNN. Regarding d_1, d_{in}, and d_{s} as constants, the time complexity of the MTG module is $\Theta\left(K\left(m + N\right)\right)$. For the scale-wise fusion module, the time complexity is $\Theta\left(N \times d_{\text{s}} \times d_1 + d_1 \times K\right)$, where d_1 is the output dimension of the first full connected layer. Since d_{s} and d_1 are regarded as constants, the time complexity of the scale-wise fusion module is $\Theta\left(N + K\right)$.

V. EXPERIMENTS

A. Datasets and Settings

Datasets. To evaluate the performance of MAGNN, we conduct experiments on four public benchmark datasets[1]: Solar-Energy, Traffic, Electricity, and Exchange-Rate. Table I gives the summarized dataset statistics, and the details about the four public benchmark datasets are given as follows:

- Solar-Energy: This dataset contains the collected solar power from the National Renewable Energy Laboratory, which is sampled every 10 minutes from 137 PV plants in Alabama State in 2007.

[1]https://github.com/laiguokun/multivariate-time-series-data

TABLE I
DATASET STATISTICS.

Datasets	# Samples	# Variables	Sample rate
Solar-Energy	52560	137	10 minutes
Traffic	17544	862	1 hour
Electricity	26304	321	1 hour
Exchange-Rate	7588	8	1 day

- Traffic: This dataset contains the road occupancy rates (between 0 and 1) from the California Department of Transportation, which is hourly aggregated from 862 sensors in San Francisco Bay Area from 2015 to 2016.
- Electricity: This dataset contains the electricity consumption from the UCI Machine Learning Repository, which is hourly aggregated from 321 clients from 2012 to 2014.
- Exchange-Rate: This dataset contains the exchange rates of eight countries, which is sampled daily from 1990 to 2016.

Following existing works [3], [13], [18], the four datasets are split into the training set (60%), validation set (20%), and test set (20%) in chronological order.

Experimental settings. MAGNN is implemented in Python with PyTorch, and the source code is released on GitHub[2]. For experimental settings, unlike existing works that conduct grid search over all tunable hyper-parameters, we exploit Neural Network Intelligence (NNI)[3] toolkit to automatically search the best hyper-parameters, which can greatly reduce computation costs. The search space of hyper-parameters and the configures of NNI are given in Table II. Following existing works [3], [18], the input window size T is set to 168. The learning rate is set to 0.001. Adam optimizer is used and all trainable parameters can be optimized through back-propagation. We set horizon $h = \{3, 6, 12, 24\}$, respectively, which means the forecasting horizons are set from 30 to 240 minutes for the Solar-Energy dataset, from 3 to 24 hours for the Traffic and Electricity datasets, from 3 to 24 days for the Exchange-Rate dataset. The larger the forecasting horizons are, the harder the forecasting tasks are.

Evaluation metrics. Root Relative Squared Error (RSE) and Empirical Correlation Coefficient (CORR) are exploited as evaluation metrics, which are defined as:

$$\text{RSE} = \frac{\sqrt{\sum_{i=1}^{\mathcal{T}_{\text{test}}} \sum_{j=1}^{N} \left(\widehat{\boldsymbol{x}}_{i,j} - \boldsymbol{x}_{i,j}\right)^2}}{\sqrt{\sum_{i=1}^{\mathcal{T}_{\text{test}}} \sum_{j=1}^{N} \left(\boldsymbol{x}_{i,j} - \text{mean}(\boldsymbol{x})\right)^2}}, \tag{18}$$

$$\text{CORR} = \frac{1}{\mathcal{T}_{\text{test}}} \sum_{j=1}^{N} \frac{\sum_{i=1}^{\mathcal{T}_{\text{test}}} \left(\boldsymbol{x}_{i,j} - \text{mean}\left(\boldsymbol{x}_{*,j}\right)\right)\left(\widehat{\boldsymbol{x}}_{i,j} - \text{mean}\left(\widehat{\boldsymbol{x}}_{*,j}\right)\right)}{\sqrt{\sum_{i=1}^{\mathcal{T}_{\text{test}}} \left(\boldsymbol{x}_{i,j} - \text{mean}\left(\boldsymbol{x}_{*,j}\right)\right)^2 \left(\widehat{\boldsymbol{x}}_{i,j} - \text{mean}\left(\widehat{\boldsymbol{x}}_{*,j}\right)\right)^2}}, \tag{19}$$

where $\mathcal{T}_{\text{test}}$ is the total time steps used for test. For RSE, a lower value is better, while for CORR, a higher value is better.

B. Methods for Comparison

The methods in our comparative evaluation are as follows.

[2]https://github.com/shangzongjiang/MAGNN
[3]https://nni.readthedocs.io/en/latest/

TABLE II
SETTINGS OF NNI.

	Parameters	Choise
Search space	Channel size	$\{8, 16, 32, 64\}$
	Dropout rate	$\{0.1, 0.5\}$, uniform
	# neighbors (the first three datasets)	$\{20, 30, 40, 50\}$
	# neighbors (the last dataset)	$\{5, 6, 7, 8\}$
Configures	Max trial number	15
	Optimization algorithm	Tree-structured Parzen Estimator
	Early stopping strategy	Curvefitting

Conventional methods:

- AR: It stands for the auto-regressive model.
- TRMF [7]: It stands for the auto-regressive model using temporal regularized matrix factorization.
- GP [6]: It stands for the Gaussian process time series model.
- VAR-MLP [38]: It stands for a hybrid model that combines auto-regressive model (VAR) and multilayer perception (MLP).
- RNN-GRU [9]: It stands for the RNN using GRU cell for time series forecasting.

Attentive recurrent methods:

- LSTNet [3]: It introduces the CNNs to capture short-term temporal dependencies, and a recurrent-skip layer to capture long-term periodic patterns.
- MTNet [12]: It exploits the memory component and attention mechanism to capture long-term temporal dependencies and periodic patterns.
- TPA-LSTM [13]: It utilizes an attention mechanism to extract important temporal patterns from different time steps and different variables.

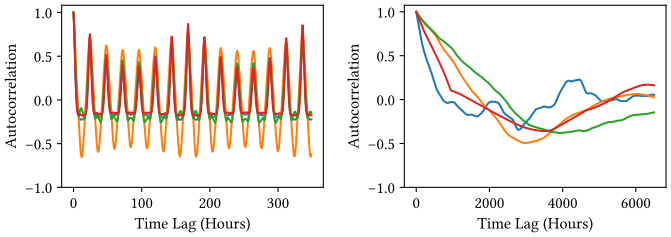
MTS modeling with graph learning:

- MTGNN [18]: It uses a graph learning module to learn inter-variable dependencies, and models MTS using GNN and dilated convolution.
- MAGNN: It is our proposed method.

C. Main Results

Table III reports the evaluation results of all the methods on the four datasets, and the following tendencies can be discerned:

1) Our method (MAGNN) achieves the state-of-the-art results on these datasets. Particularly, on Traffic dataset, MAGNN outperforms existing methods on all the horizons and all the metrics. The reason might be that the traffic data is very suitable for our assumption, as there are multi-scale temporal dependencies and complicated inter-variable dependencies. In addition, on Solar-Energy dataset, MAGNN obtains slightly worse performance than existing methods. To explore the reasons, Fig. 5 shows the autocorrelation graphs of sampled variables on Traffic and Solar-Energy datasets. For Traffic dataset, we can clearly observe the daily and weekly patterns. For Solar-Energy dataset, we can hardly see the multi-scale temporal dependencies. These observations provide empirical guidance for the success of using MAGNN in modeling MTS.



(a) Traffic dataset (b) Solar-Energy dataset

Fig. 5. The autocorrelation graphs of four sampled variables.

2) Traditional methods (AR, TRMF, and GP) get worse results than deep learning methods, as they cannot capture the non-stationary and non-linear dependencies.

3) Deep learning methods (VAR-MLP, RNN-GRU, LSTNet, MTNet, and TPA-LSTM) do not explicitly model the pairwise inter-variable dependencies. Thus, they get worse performance than MTGNN and MAGNN.

4) MTGNN is the state-of-the-art method that uses a graph learning module to learn inter-variable dependencies. However, MTGNN fails to consider multi-scale inter-variable dependencies and gets worse performance than MAGNN. In contrast, MAGNN learns a temporal representation that can comprehensively reflect both multi-scale temporal patterns and the scale-specific inter-variable dependencies.

D. Effect of Multi-Scale Modeling

To investigate the effect of multi-scale modeling, we evaluate the performance of MAGNN with different numbers of scales (i.e., 2 scales, 3 scales, 4 scales, and 5 scales). Fig. 6 shows the results of MAGNN under different numbers of scales on Traffic dataset. We can observe that when the number of scales increases from 2 to 4, the performance of MAGNN is significantly improved. This is because MAGNN can capture more diversified short-term and long-term patterns. When the number of scales increases up to 5, the performance of MAGNN has not improved, which might be because the number of scales is already meet the needs of the task, and excessive parameters are prone to overfitting.

E. Effect of Adaptive Graph Learning

To demonstrate the effect of adaptive graph learning, we conduct ablation study by carefully designing the following three variants.

TABLE III
RESULTS SUMMARY (IN TERMS OF RSE AND CORR) OF ALL METHODS ON FOUR DATASETS.

Methods		Solar-Energy				Traffic				Electricity				Exchange-Rate			
		3	6	12	24	3	6	12	24	3	6	12	24	3	6	12	24
AR	RSE	0.2435	0.379	0.5911	0.8699	0.5991	0.6218	0.6252	0.63	0.0995	0.1035	0.105	0.1054	0.0228	0.0279	0.0353	0.0445
	CORR	0.971	0.9263	0.8107	0.5314	0.7752	0.7568	0.7544	0.7519	0.8845	0.8632	0.8591	0.8595	0.9734	0.9656	0.9526	0.9357
TRMF	RSE	0.2473	0.347	0.5597	0.9005	0.6708	0.6261	0.5956	0.6442	0.1802	0.2039	0.2186	0.3656	0.0351	0.0875	0.0494	0.0563
	CORR	0.9703	0.9418	0.8475	0.5598	0.6964	0.743	0.7748	0.7278	0.8538	0.8424	0.8304	0.7471	0.9142	0.8123	0.8993	0.8678
VAR-MLP	RSE	0.1922	0.2679	0.4244	0.6841	0.5582	0.6579	0.6023	0.6146	0.1393	0.162	0.1557	0.1274	0.0265	0.0394	0.0407	0.0578
	CORR	0.9829	0.9655	0.9058	0.7149	0.8245	0.7695	0.7929	0.7891	0.8708	0.8389	0.8192	0.8679	0.8609	0.8725	0.828	0.7675
GP	RSE	0.2259	0.3286	0.52	0.7973	0.6082	0.6772	0.6406	0.5995	0.15	0.1907	0.1621	0.1273	0.0239	0.0272	0.0394	0.058
	CORR	0.9751	0.9448	0.8518	0.5971	0.7831	0.7406	0.7671	0.7909	0.867	0.8334	0.8394	0.8818	0.8713	0.8193	0.8484	0.8278
RNN-GRU	RSE	0.1932	0.2628	0.4163	0.4852	0.5358	0.5522	0.5562	0.5633	0.1102	0.1144	0.1183	0.1295	0.0192	0.0264	0.0408	0.0626
	CORR	0.9823	0.9675	0.915	0.8823	0.8511	0.8405	0.8345	0.83	0.8597	0.8623	0.8472	0.8651	0.9786	0.9712	0.9531	0.9223
LSTNet	RSE	0.1843	0.2559	0.3254	0.4643	0.4777	0.4893	0.495	0.4973	0.0864	0.0931	0.1007	0.1007	0.0226	0.028	0.0356	0.0449
	CORR	0.9843	0.969	0.9467	0.887	0.8721	0.869	0.8614	0.8588	0.9283	0.9135	0.9077	0.9119	0.9735	0.9658	0.9511	0.9354
MTNet	RSE	0.1847	0.2398	0.3251	0.4285	0.4764	0.4855	0.4877	0.5023	0.084	0.0901	0.0934	0.0969	0.0212	0.0258	0.0347	**0.0442**
	CORR	0.984	0.9723	0.9462	0.9013	0.8728	0.8681	0.8644	0.857	0.9319	0.9226	0.9165	0.9147	0.9767	0.9703	0.9561	**0.9388**
TPA-LSTM	RSE	0.1803	**0.2347**	0.3234	0.4389	0.4487	0.4658	0.4641	0.4765	0.0823	0.0916	0.0964	0.1006	**0.0174**	**0.0241**	**0.0341**	0.0444
	CORR	0.985	**0.9742**	0.9487	0.9081	0.8812	0.8717	0.8717	0.8629	0.9439	0.9337	0.925	0.9133	**0.979**	**0.9709**	**0.9564**	0.9381
MTGNN	RSE	0.1778	0.2348	0.3109	0.427	0.4162	0.4754	0.4461	0.4535	**0.0745**	0.0878	0.0916	**0.0953**	0.0194	0.0259	0.0349	0.0456
	CORR	0.9852	0.9726	0.9509	0.9031	0.8963	0.8667	0.8794	0.881	**0.9474**	0.9316	0.9278	**0.9234**	0.9786	0.9708	0.9551	0.9372
MAGNN	RSE	**0.1771**	0.2361	**0.3015**	**0.4108**	**0.4097**	**0.4555**	**0.4423**	**0.4434**	**0.0745**	**0.0876**	**0.0908**	0.0963	0.0183	0.0246	0.0343	0.0474
	CORR	**0.9853**	0.9724	**0.9539**	**0.9097**	**0.8992**	**0.8753**	**0.8815**	**0.8813**	**0.9476**	**0.9323**	**0.9282**	0.9217	0.9778	0.9712	0.9557	0.9339

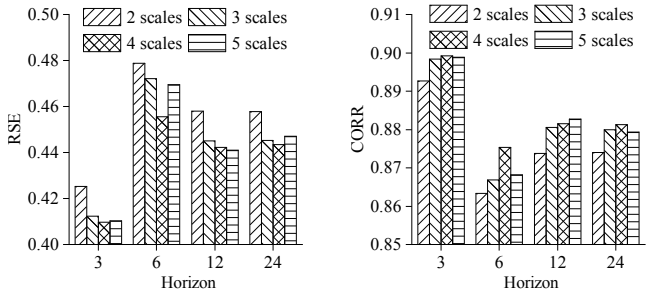


Fig. 6. The results of MAGNN under different numbers of scales.

- MAGNN-dy: For the j^{th} scale at time step t, the graph learning module takes the dynamic feature representation $\boldsymbol{x}_t^k$ as input rather than the static node embeddings. Thus, the graphs are different at different time steps.
- MAGNN-full: It removes the sparsity strategy and obtains the multi-scale full-connected adjacent matrices.
- MAGNN-one: It only learns one shared adjacent matrix to describe the inter-variable dependencies of multi-scale feature representations.

The results of these methods on Solar-Energy dataset are shown in Table IV. (Note: Traffic dataset is too large to run MAGNN-dy on our machine. Thus, this ablation study is conducted on Solar-Energy dataset). We can see that the full version of our MAGNN achieves the best performance, which implies: 1) Although MAGNN-dy can capture the dynamic adjacent matrices over the time, the dramatic fluctuation of adjacent matrices makes the model difficult to converge to the optimum result. 2) MAGNN-full exploits full-connected adjacent matrices, which makes the model fall into the over-smoothing problem. 3) MAGNN-one cannot learn scale-specific adjacent matrices and makes the model prefer to learn one type of prominent and shared temporal patterns.

F. Effect of Scale-Wise Fusion

To demonstrate the effect of scale-wise fusion, we conduct ablation study by carefully designing the following two variants.

- MAGNN-con: It removes the scale-wise fusion module and directly concatenates these scale-specific representations.
- MAGNN-pooling: It removes the scale-wise fusion module and aggregates these scale-specific representations by a global pooling layer.

The results of these methods on Traffic dataset are shown in Table V. Obviously, MAGNN achieves the best performance in all cases. The results imply that MAGNN learns a robust multi-scale representation from these scale-specific representations, as our scale-wise fusion can consider the importance of scale-specific temporal patterns and capture the cross-scale correlations.

G. Parameter Study

We study the two important parameters (i.e., convolutional channel size and the number of neighbors), which could influence the performance of MAGNN. Fig. 7(a) shows the results of MAGNN on Traffic dataset by varying convolutional channel size from 4 to 128. The best performance can be obtained when convolutional channel size is 16. It might be that a small convolutional channel size limits the expressive ability of MAGNN, and a large convolutional channel size would make the model hard to train. Fig. 7(b) shows the results of MAGNN on Traffic dataset by varying the number of neighbors from 20 to 200. The best performance can be obtained when the number of neighbors is 30. The reason may be that a small number of neighbors limits the ability to exploit inter-variable dependencies, and a large number of neighbors would introduce noises.

H. Computation Cost

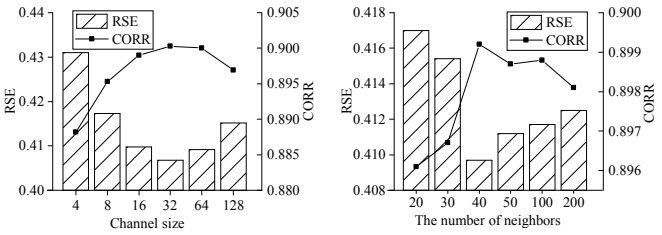
To evaluate the computation cost, we compare the parameter numbers and training time of MAGNN with LSTNet, TPA-LSTM, and MTGNN on Exchange-Rate dataset in Table

TABLE IV
THE RESULTS OF DIFFERENT GRAPH LEARNING METHODS.

Methods	3		6		12		24	
	RSE	CORR	RSE	CORR	RSE	CORR	RSE	CORR
MAGNN-dy	**0.1766**	**0.9854**	0.2372	0.9721	0.3138	0.9504	**0.4086**	**0.9109**
MAGNN-full	0.1772	0.9853	0.2398	0.9716	0.3068	0.9522	0.4246	0.9032
MAGNN-one	0.1769	0.9853	0.2377	0.972	0.3085	0.9516	0.4257	0.9021
MAGNN	0.1771	0.9853	**0.2361**	**0.9724**	**0.3015**	**0.9539**	0.4108	0.9097

TABLE V
THE RESULTS OF DIFFERENT FUSION METHODS.

Methods	3		6		12		24	
	RSE	CORR	RSE	CORR	RSE	CORR	RSE	CORR
MAGNN-con	0.4173	0.8961	0.4991	0.8569	**0.4422**	**0.8819**	0.4492	0.8789
MAGNN-pooling	0.4181	0.8957	0.4846	0.8597	0.4464	0.8801	0.4505	0.8775
MAGNN	**0.4097**	**0.8992**	**0.4555**	**0.8753**	0.4423	0.8815	**0.4434**	**0.8813**



(a) Traffic dataset (b) Solar-Energy dataset

Fig. 7. The autocorrelation graphs of four sampled variables.

TABLE VI
THE COMPUTATION COSTS OF DIFFERENT METHODS.

Methods	# parameters	Training Time (epoch)
LSTNet	19998	13.54s
TPA-LSTM	39634	90.16s
MTGNN	337345	66.18s
MAGNN	95037	43.29s

VI LSTNet has least parameter number and runs fastest in these methods. But it gets worst forecasting results. Although MAGNN has over two times more parameters than TPA-LSTM, MAGNN runs two times faster than TPA-LSTM, as TPA-LSTM requires more time to train the attention network. MTGNN is the state-of-the-art method and requires more parameters and training time than MAGNN. Overall, comprehensively considering the significant forecasting performance improvement and the computation cost, MAGNN demonstrates the superiority over existing methods.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a multi-scale adaptive graph neural network (MAGNN) for MTS forecasting. By exploiting a multi-scale pyramid network to model temporal hierarchy, an adaptive graph learning module to automatically infer inter-variable dependencies, a multi-scale temporal graph neural network to model intra-variable and inter-variable dependencies, and a scale-wise fusion module to promote the collaboration across different time scales, MAGNN outperforms the state-of-the-art methods on four datasets. With the theoretical analysis and experimental verification, we believe that MAGNN can capture multi-scale temporal patterns and complicated inter-variable dependencies for accurate MTS forecasting.

In the future, it is of interest to extend this work in the following two aspects: First, we will design a method to learn dynamic adjacency matrices at different time steps, and introduce a regularizer to constrain the dramatic fluctuation of adjacent matrices. Second, we will design a neural architecture search framework to automatically capture both inter-variable dependencies and intra-variable dependencies.

REFERENCES

[1] Ajoy K Palit and Dobrivoje Popovic. *Computational intelligence in time series forecasting: Theory and engineering applications*. Springer Science & Business Media, 2006.

[2] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Congrui Huang, Yunhai Tong, Bixiong Xu, Ying Bai, Jie Tong, et al. Spectral temporal graph neural network for multivariate time-series forecasting. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems*, pages 17766–17778, 2020.

[3] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104, 2018.

[4] Donghui Chen, Ling Chen, Youdong Zhang, Bo Wen, and Chenghu Yang. A multiscale interactive recurrent network for time-series forecasting. *IEEE Transactions on Cybernetics*, pages 1–11, 2021.

[5] Zheyi Pan, Wentao Zhang, Yuxuan Liang, Weinan Zhang, Yong Yu, Junbo Zhang, and Yu Zheng. Spatio-temporal meta learning for urban traffic prediction. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[6] Roger Frigola. *Bayesian time series learning with Gaussian processes*. PhD thesis, University of Cambridge, 2015.

[7] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In *Proceedings of the 29th Annual Conference on Neural Information Processing Systems*, pages 847–855, 2016.

[8] Bovas Abraham and Johannes Ledolter. *Statistical methods for forecasting*. John Wiley & Sons Inc, 2005.

[9] Rui Fu, Zuo Zhang, and Li Li. Using LSTM and GRU neural network methods for traffic flow prediction. In *Proceedings of the 31st Youth Academic Annual Conference of Chinese Association of Automation*, pages 324–328, 2016.

[10] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

[11] Weiqi Chen, Ling Chen, Yu Xie, Wei Cao, Yusong Gao, and Xiaojie Feng. Multi-range attentive bicomponent graph convolutional network for traffic forecasting. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 3529–3536, 2020.

[12] Yen-Yu Chang, Fan-Yun Sun, Yueh-Hua Wu, and Shou-De Lin. A memory-network based solution for multivariate time-series forecasting. *arXiv preprint arXiv:1809.02105*, 2018.

[13] Shun-Yao Shih, Fan-Keng Sun, and Hung-yi Lee. Temporal pattern attention for multivariate time series forecasting. *Machine Learning*, 108(8):1421–1441, 2019.

[14] Ling Chen, Weiqi Chen, Binqing Wu, Youdong Zhang, Bo Wen, and Chenghu Yang. Learning from multiple time series: A deep disentangled approach to diversified time series forecasting. *arXiv preprint arXiv:2111.04942*, 2021.

[15] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2020.

[16] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 32:57–81, 2020.

[17] Lei Bai, Lina Yao, Can Li, Xianzhi Wang, and Can Wang. Adaptive graph convolutional recurrent network for traffic forecasting. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems*, pages 17804–17815, 2020.

[18] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 753–763, 2020.

[19] Haoyan Xu, Yida Huang, Ziheng Duan, Jie Feng, and Pengyu Song. Multivariate time series forecasting based on causal inference with transfer entropy and graph neural network. *arXiv preprint arXiv:2005.01185*, 2020.

[20] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems*, pages 5243–5253, 2019.

[21] Cen Chen, Kenli Li, Sin G Teo, Xiaofeng Zou, Kang Wang, Jie Wang, and Zeng Zeng. Gated residual recurrent graph neural networks for traffic prediction. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 485–492, 2019.

[22] Xu Geng, Yaguang Li, Leye Wang, Lingyu Zhang, Qiang Yang, Jieping Ye, and Yan Liu. Spatiotemporal multi-graph convolution network for ride-hailing demand forecasting. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pages 3656–3663, 2019.

[23] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. Urban traffic prediction from spatio-temporal data using deep meta learning. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1720–1730, 2019.

[24] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In *Proceedings of the 14th IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12026–12035, 2019.

[25] Chainarong Amornbunchornvej, Elena Zheleva, and Tanya Berger-Wolf. Variable-lag granger causality and transfer entropy for time series analysis. *ACM Transactions on Knowledge Discovery from Data*, 15(4):1–30, 2021.

[26] Kaiwen He, Xu Chen, Qiong Wu, Shuai Yu, and Zhi Zhou. Graph attention spatial-temporal network with collaborative global-local learning for citywide mobile traffic prediction. *IEEE Transactions on Mobile Computing*, 2020.

[27] Hao Huang, Chenxiao Xu, and Shinjae Yoo. Bi-directional causal graph learning through weight-sharing and low-rank neural network. In *Proceedings of the 19th IEEE International Conference on Data Mining*, pages 319–328, 2019.

[28] Shengnan Guo, Youfang Lin, Huaiyu Wan, Xiucheng Li, and Gao Cong. Learning dynamics and heterogeneity of spatial-temporal graph data for traffic forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[29] Colin Graber and Ryan Loh. Unsupervised discovery of dynamic neural circuits. In *Proceedings of the 33rd Annual Conference on Neural Information Processing Systems*, 2019.

[30] Thomas N Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard S Zemel. Neural relational inference for interacting systems. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2688–2697, 2018.

[31] Cong Tang, Jingru Sun, Yichuang Sun, Mu Peng, and Nianfei Gan. A general traffic flow prediction approach based on spatial-temporal graph attention. *IEEE Access*, 8:153731–153741, 2020.

[32] Ezra Webb, Ben Day, Helena Andres-Terre, and Pietro Lió. Factorised neural relational inference for multi-interaction systems. *arXiv preprint arXiv:1905.08721*, 2019.

[33] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. GMAN: A graph multi-attention network for traffic prediction. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, pages 1234–1241, 2020.

[34] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-ResNet and the impact of residual connections on learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, pages 4278–4284, 2017.

[35] Pedro Savarese and Michael Maire. Learning implicitly recurrent CNNs through parameter sharing. In *Proceedings of the 25th International Conference on Learning Representations*, 2018.

[36] Antônio H Ribeiro, Koen Tiels, Luis A Aguirre, and Thomas Schön. Beyond exploding and vanishing gradients: Analysing RNN training using attractors and smoothness. In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, pages 2370–2380, 2020.

[37] Razvan-Gabriel Cirstea, Chenjuan Guo, and Bin Yang. Graph attention recurrent neural networks for correlated time series forecasting. *MileTS19@ KDD*, 2019.

[38] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003.