Artificial Intelligence 334 (2024) 104158

Contents lists available at ScienceDirect

Artificial Intelligence

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

Functional Relation Field: A Model-Agnostic Framework for Multivariate Time Series Forecasting

Ting Li [a], Bing Yu [b], Jianguo Li [a], Zhanxing Zhu [c,*]

[a] Ant Group, Beijing, China
[b] Peking University, China
[c] University of Southampton, UK

ARTICLE INFO

Keywords:
Multi-variate time series prediction
Constrained optimization
Graph neural network

ABSTRACT

In multivariate time series forecasting, the most popular strategy for modeling the relationship between multiple time series is the construction of graph, where each time series is represented as a node and related nodes are connected by edges. However, the relationship between multiple time series is typically complicated, e.g. the sum of outflows from upstream nodes may be equal to the inflows of downstream nodes. Such relations widely exist in many real-world scenarios for multivariate time series forecasting, yet are far from well studied. In these cases, graph might be insufficient for modeling the complex dependency between nodes. To this end, we explore a new framework to model the inter-node relationship in a more precise way based our proposed inductive bias, *Functional Relation Field*, where a group of functions parameterized by neural networks are learned to characterize the dependency between multiple time series. Essentially, these learned functions then form a "field", i.e. a particular set of constraints, to regularize the training loss of the backbone prediction network and enforce the inference process to satisfy these constraints. Since our framework introduces the relationship bias in a data-driven manner, it is flexible and model-agnostic such that it can be applied to any existing multivariate time series prediction networks for boosting performance. The experiment is conducted on one toy dataset to show our approach can well recover the true constraint relationship between nodes. And various real-world datasets are also considered with different backbone prediction networks. Results show that the prediction error can be reduced remarkably with the aid of the proposed framework.

1. Introduction

Multivariate time series forecasting has surged recently due to its strong expressiveness of the spatio-temporal dependence among the data and its enormous popularity in vast application areas, such as the prediction of urban traffic, computer network flow, cloud micro-services calling flow, and rigid body motion, to name a few [1–5]. Fig. 1 sketches the task of multivariate time series forecasting. The most popular and straightforward strategy for modeling the relationship between multiple time series is the introduction of graph, where each time series is represented as a node and related nodes are connected by edges. This particular inductive bias for

* Corresponding author.
E-mail addresses: liting6259@gmail.com (T. Li), yubing11@126.com (B. Yu), lijg.zero@antgroup.com (J. Li), z.zhu@soton.ac.uk (Z. Zhu).

https://doi.org/10.1016/j.artint.2024.104158
Received 25 June 2023; Received in revised form 6 February 2024; Accepted 16 May 2024

Available online 5 June 2024

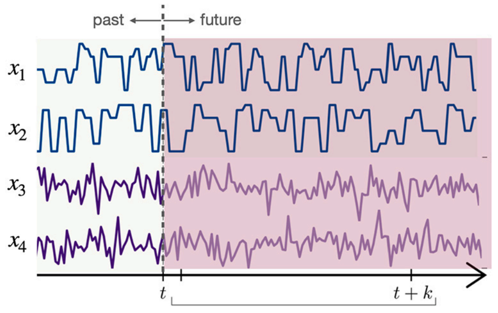

Fig. 1. Multivariate time series forecasting. We have t times steps of observed data for four time series. The task is to predict the future k steps for all the time series.

multivariate time series prediction results in the so called spatial-temporal graph neural networks [2]. The graph structure is either given *apriori* (e.g. in traffic flow prediction, each road as a node has connected roads forming the graph) or learned based the similarity between nodes [6,3,7]. However, in practice, the relationship between multiple time series is typically more complicated. For instance, there often exist *physical constraints* among the nodes, ranging from the equality between the inflow and the outflow for a node in a traffic network to the geometric constraints of the rigid body motion, even more complicated dependencies in video-related multi-modal learning [8]. Such relations widely exist in many real-world multivariate time series forecasting scenarios, yet are far from well studied. In these cases, graph might not be sufficient for characterizing the dependency between nodes.

As a remedy, in this work, we explore a new framework to model the inter-node relationship in a more precise manner than graph, **Functional Relation Field** (FRF), where a group of functions parameterized by neural networks are learned to characterize the dependency between multiple time series *explicitly*. These learned functions are versatile: first they can then be used to discover the underlying graph structure by identifying the most relevant neighbors of the target node; and on the other hand, the learned functions will form a "field" where the nodes in the backbone prediction networks are further enforced to satisfy the constraints defined by these functions during both training and inference processes. Different from the traditional graph neural networks assuming similar time series only have edge connections, our framework models the dependency between nodes through *an explicit functional relationship*, e.g. a linear form to enforce the constraints between the flows between target and dependent nodes.

In our framework, we mainly solve the following two issues: (i) *How to learn the functional field?* We need to select the dependent nodes that have a relationship with the target node, and express the constraint in a functional form; (ii) *How to guarantee the constraint satisfaction?* The (functional) constraints relationship should be maintained in the predicted output in both the training and inference process.

To address these issues, we propose a two-stage approach that can discover the functional relations (i.e. constraints) from data and further integrate the constraints seamlessly when forecasting the multivariate time series. Specifically, we first train a neural network with a selected target node as its output and all the other nodes as dependent variables (i.e. the input of this neural network), and identify the most relevant dependent nodes based on this trained network. We then re-train it to learn the relationship among the target and the discovered relevant nodes. Next, we incorporate these functional constraints into the network backbones by imposing them to the predicted output during both training and test process. More precisely, the output of the network could be guaranteed to satisfy the constraints by utilizing the constraint-satisfied transformation and loss minimization. We compare the proposed approach with SVM, fully connected networks, fully connected LSTM, and five backbone models (i.e., STGCN [2], AGCRN [3], Autoformer [9], FEDformer [10], SCINet [11]). Experimental results show that our approach significantly improves the performance over the original network backbones and other baseline models.

Organization. The paper is organized as follows. We first review the related works in the following and compare them with our approach. Sec. 2 describes the proposed functional relation field, from a motivated example to a more general take. The two-stage procedure is also elaborated in this part, Sec. 2.1 and, Sec. 2.2, respectively. Intensive experimental results on both synthetic and real-world tasks are presented in Sec. 3, including the experimental settings, prediction results, visualization, computational complexity analysis and ablation study. Finally, Sec. 4 concludes the paper.

Related work

Univariate time series forecasting. Recently, much research focuses on time series forecasting with deep learning models due to their powerful representational capability and prediction performance, including feed-forward neural network, RNN [12] and its variants LSTM [13] and GRU [14]. The transformer architecture and its variants [15–20,9,10] also made much progress on univariate time-series forecasting on learning long-range dependence. In order to model the trend and seasonality of time series in an interpretable way, N-beats [21] network that stacked very deep full-connection network based on backward and forward residual links has improved the multi-horizon prediction accuracy significantly. Moreover, DeepAR [22] and Deep State-Space Model (DSSM) [23] stack multi-layer LSTM network to generate parameters of one-step-ahead Gaussian predictive distributions for multi-horizon prediction.

Multivariate time series forecasting. Spatio-temporal graph neural networks [2,24–26] have been proposed to model the spatial correlation and temporal dependency in multivariate time series. Apart from capturing the temporal dependence, these methods further model the spatial dependence among all time series via graph neural networks, leveraging the information from the neighboring time series to help forecasting the target one. It is well known that an informative graph structure is important to the graph time series forecasting. Therefore, many algorithms [3,27,7] were proposed to discovery the underlying graph structure. AGCRN [3] assumed the graph structure is unknown and adopted an adaptive approach to learn the embedding vectors for all nodes, and then replaced the adjacency matrix in graph convolutions with a function of the node embeddings. However, the similarity graph calculated with the learned node embedding is a dense and continuous graph instead of a sparse and discrete graph. Therefore, GTS [7] formulated the graph structure learning problem as a probabilistic graph model to learn the discrete graph through optimizing the mean performance over the graph distribution.

However, these graph-based approaches ignore the underlying constraints among the nodes, e.g. the equality between the inflow and the outflow for a node in a traffic network. Such relations widely exist in many real-world multivariate time series forecasting scenarios, yet are far from well studied. In these cases, graph might not be sufficient for characterizing the dependency between nodes. Different from these approaches, we precisely and explicitly characterize the underlying constraints (expressed as functional relations) between the multiple time series. This new inductive bias can be applied to different backbone networks to regularize both training and test process.

2. Methodology: functional relation field

Multivariate time series forecasting. Suppose we have N time series $\{x_i\}_{i=1}^N$ with length T, written compactly as $X \in \mathbb{R}^{N\times T}$. Each time series can be denoted as a node, where $x_{i,t} \in \mathbb{R}$ for each node i and time step t. $x_t \in \mathbb{R}^N$ is the time slice of X at the t-th time step. The multi-step forecasting problem of a multivariate time series can be formulated as predicting the future M frames of the multivariates given the last H time slices:

$$\hat{y}_{t+1:t+M} = G_\Theta(x_{t-H+1:t}), \tag{1}$$

where $\hat{y}_{t+1:t+M}$ represent predicted values at the future time steps, M is the number of future steps, and we call $G(\cdot)$ *backbone prediction network*. Note that here we use y to denote the output so as to differentiate it from the input x. And the loss function for learning the parameters of $G(\cdot)$ is defined as

$$\min_\Theta \mathcal{L}(\Theta) = \frac{1}{M}\|y_{t+1:t+M} - G_\Theta(x_{t-H+1:t})\|_2^2 \tag{2}$$

Forecasting with functional relations. In many real-world scenarios, the relationship between multiple time series is typically complicated, graph might not be sufficient for modeling their dependency, particularly for the cases values of multivariate time series at each time step are subject to some *intrinsic constraints*. Existing methods have not incorporated these constraints into their models. In this work, we intend to show that models with the account of constraints (expressed with functional relationship) are superior to those without constraints in terms of prediction performance. As an example, suppose that the flow in a computer network satisfies the homogeneous linear constraints, at each time step t, the following linear constraints hold for slice x_t:

$$Ax_t = 0, \forall t, \tag{3}$$

where $A \in \mathbb{R}^{m\times N}$ is a matrix that is constant across time. In other more complex cases, the constraints can be non-homogeneous, non-linear, or even intertemporal. Here, we concentrate on time-invariant constraints that is not intertemporal. As such, the constraints can be described by a set of functions $\mathcal{F}$ with size m, i.e. functional relation field,

$$\mathcal{F} = \{f_1, f_2, ..., f_m\}. \quad f_i(x_t) = 0,\ \forall i,\ \forall t. \tag{4}$$

In the linear constraints case, each functional constraint $f_i(x_t) = 0$ corresponds to $A_{i,:}^T x_t = 0$, where $A_{i,:}$ denotes the i-th row of the matrix A.

Based on the constraints defined above, we consider the following constrained multivariate time series prediction problem,

$$\begin{aligned} &\min_\Theta \mathcal{L}(\Theta) = \frac{1}{M}\|y_{t+1:t+M} - \hat{y}_{t+1:t+M}\|_2^2, \\ &s.t.\ f_i(\hat{y}_{t+\tau}) = 0,\ 1 \le \tau \le M,\ 1 \le i \le m. \end{aligned} \tag{5}$$

However, in most real-world scenarios, neither the functional form f_i nor the specific weights variables involved in the constraints are given, and therefore one of our objectives is to extract such information from the data and solve the problem (5). We now elaborate the functional relation field for multivariate times series prediction in the following.

The schematic diagram of the proposed framework is depicted in Fig. 2, including two parts. The first part displayed in the upper panel of Fig. 2 shows how we learn the functional relations, i.e. the constraints between nodes. Assuming that the constraints are unknown, we aim to find the constrained nodes and the specific functional form for these constraints. The constraint function in this paper is approximated by a neural network, named as functional relation network or constraint network. After training the functional relation network, we can identify the most relevant nodes for the target node and produce a more informative graph structure. Then we can proceed to integrate the learned constraints into the backbone graph neural networks for multivariate time series prediction. We enforce these constraints to the output of spatio-temporal graph neural networks during both training and test phases, as shown in the bottom panel of Fig. 2. In the training phase, the learned functional relations (i.e. constraints) are used as

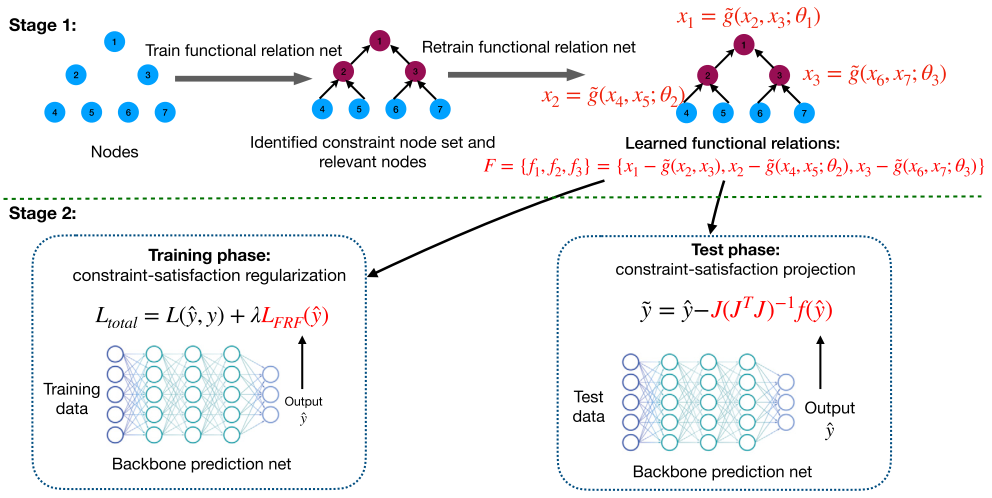
Stage 1:
Train functional relation net
Retrain functional relation net
Nodes
Identified constraint node set and relevant nodes
$x_1 = \tilde{g}(x_2, x_3; \theta_1)$
$x_2 = \tilde{g}(x_4, x_5; \theta_2)$
$x_3 = \tilde{g}(x_6, x_7; \theta_3)$
Learned functional relations:
$F = \{f_1, f_2, f_3\} = \{x_1 - \tilde{g}(x_2, x_3), x_2 - \tilde{g}(x_4, x_5; \theta_2), x_3 - \tilde{g}(x_6, x_7; \theta_3)\}$
Stage 2:
Training phase: constraint-satisfaction regularization
$L_{total} = L(\hat{y}, y) + \lambda L_{FRF}(\hat{y})$
Training data
Output $\hat{y}$
Backbone prediction net
Test phase: constraint-satisfaction projection
$\tilde{y} = \hat{y} - J(J^T J)^{-1} f(\hat{y})$
Test data
Output $\hat{y}$
Backbone prediction net

Fig. 2. The schematic diagram of functional relation field framework. The two subfigures denote the two stages. **Stage 1: upper panel.** For each target node, we train a functional relation network (i.e. constraint network) to first identify its dependent nodes and then obtain the explicit functional relationship. For instance, for node x_1, we first figure out its dependent nodes are $\{x_2, x_3\}$, and then obtain the constraint as $x_1 = \tilde{g}(x_2, x_3; \theta_1)$. **Stage 2: bottom panel.** The learned constraints are incorporated in the backbone prediction network (cf. Section 2.2) for both training and test phases to improve the forecasting performance. In the training phase, the learned functional relations (i.e. constraints) are used as a regularization term to achieve constraint satisfaction; while in test phase, iterative projections are implemented to enforce the output of the backbone to satisfy the constraints.

a regularization term to achieve constraint satisfaction; During the inference process we add a constraint-satisfaction transformation iteratively to the output of the backbone such that it satisfy the constraints. Altogether, we refer to the proposed framework as functional relation field-enhanced spatio-temporal graph networks (FRF-STG). It is model-agnostic and can be applied to different backbone graph networks. In the following, we will describe the two stages including learning functional relation network and how to apply the constraints induced by the functional relation between nodes in more details.

2.1. *Learning the functional relation network*

We start with discussing the first question: *how to learn the unknown constraints (i.e. the functional relations) from the multivariate time series data?* As demonstrated in Fig. 2(a), we assume that there exists a constraint for each node. We first discover the relevant nodes involved in these constraints and then express the constraint functions via neural networks.

Identifying constrained nodes and their relevant nodes. Here we consider a simplified case where the functional relation between nodes can be formulated as:

$$x_{t,i} = g_i(x_{t,\setminus i}), \forall t \tag{6}$$

i.e. for each target node i, we use a constraint network g_i to approximate the function relation taking all the remaining $(N-1)$ nodes as input. We then train the constraint network to predict the value of the i-th node with the loss function:

$$\mathcal{L}_{pred,(i)} = \|\hat{x}_{t,i} - x_{t,i}\|^2 \tag{7}$$

where $\hat{x}_{t,i}$ and $x_{t,i}$ represent the estimated and observed values of node i at time step t. Second, a threshold ϵ_{err} is set, and we treat x_i as a constrained node if both the training and validation error are smaller than ϵ_{err}. Otherwise, x_i is unpredictable with the other nodes, indicating it has weak dependency with other nodes. Then, to identify the most relevant nodes set $\mathcal{N}_i$ for target node i, we introduce the sensitivity of input change to the output for the trained constraint network, measured by the absolute value of the partial derivative:

$$\delta_{i,j} = \left|\frac{\partial g_i}{\partial x_{t,j}}\right|, j \neq i \tag{8}$$

We calculate the average gradients over the training and the validation set for node j. Then, we specify another threshold ϵ_{grad} here and consider the node j as the most relevant node of target i if $\delta_{i,j}$ is larger than ϵ_{grad}. Besides, if the cardinality of $\mathcal{N}_i$ is larger than the scale threshold S, we further shrink $\mathcal{N}_i$ by only keeping the top-S nodes with the largest $\delta_{i,j}$.

Retraining the functional relation network. Since we filter out the irrelevant nodes for the discovered constrained node x_i, it is necessary to re-train the constraint network using the relevant nodes in $\mathcal{N}_i$ as inputs, denoted as $x_{t,\mathcal{N}_i} = \{x_{t,ij} | j \in \mathcal{N}_i\}$,

$$\hat{x}_{t,i} = \tilde{g}_i(x_{t,\mathcal{N}_i}). \tag{9}$$

Regarding the architecture of the functional relation network $\tilde{g}_i$, we adopt a simple attention-based structure for each node i using weighted average of the relevant nodes to represent the target node, described as follows.

$$\alpha_{t,i} = Softmax(\mathbf{MLP}_i(x_{t,\mathcal{N}_i})), \quad \hat{x}_{t,i} = \alpha_{t,i}^T x_{t,\mathcal{N}_i}, \tag{10}$$

where $\alpha_{t,i}$ is the attention weight vector generated from the relevant nodes $x_{t,\mathcal{N}_i}$, and $\hat{x}_{t,i}$ is the reconstructed input with the constraint nodes. Other alternatives for designing the functional relation network are also possible.

2.2. Integrating the constraints into the backbone networks

The constraints learned by the functional relation network are versatile. A naive usage is to construct meaningful graph structure by drawing edges between the identified target and its dependent nodes. Secondly, we propose to incorporate the learned constraints into the backbone prediction network in both training and test process through *constraint-satisfaction loss minimization* and *constraint-satisfaction transformation*, respectively. Both of them are used to guarantee that the constraints are maintained in the outputs of the backbone network.

Constraint satisfaction in training phase. We expect the output of the backbone network, $\hat{y} = \{\hat{y}_{t+1}, \hat{y}_{t+2}..., \hat{y}_{t+M}\}$, to satisfy the learned constraints that could reveal the underlying structure of the multivariate time series. A straightforward yet effective way of implementing the constraint satisfaction is loss minimization over the functional relation network based on the output of the backbone prediction network,

$$\mathcal{L}_{FRF}(\hat{y}) = \sum_{i=1}^{N}\sum_{\tau=1}^{M} \|\hat{y}_{t+\tau,i} - \tilde{g}(\{\hat{y}_{t+\tau,j}\}, j \in \mathcal{N}_i)\|_2^2 \tag{11}$$

Therefore, the overall loss function for training the backbone prediction network includes two terms,

$$\mathcal{L}_{total} = \mathcal{L}(\hat{y}, y) + \lambda \mathcal{L}_{FRF}(\hat{y}), \tag{12}$$

where λ is a tradeoff coefficient for balancing the supervised term and constraint satisfaction.

Constraint satisfaction in testing phase. Furthermore, although the constraints are fully utilized during training, there is no guarantee that the constraints hold for the outputs during the inference process. Therefore, it is necessary to perform *constraint-satisfaction transformation* on outputs of the prediction networks.

Let us first consider the linear constraint $Ax_t = 0, \forall t$. Suppose that $\hat{y} = \{\hat{y}_{t+1}, \hat{y}_{t+2}..., \hat{y}_{t+M}\}$ and $y = \{y_{t+1}, y_{t+2}, ..., y_{t+M}\}$ denote the predicted output of the backbone network and the ground truth, respectively. To make the output $\hat{y}_{t+\tau}$ to satisfy the linear constraint, we can project the predicted output onto the hyperplane $Ax_t = 0$ as $\tilde{y}_{t+\tau}$ with a closed-form solution,

$$\tilde{y}_{t+\tau} = \hat{y}_{t+\tau} - A^T(AA^T)^{-1}A\hat{y}_{t+\tau}. \tag{13}$$

On the other hand, for non-linear constraint set $f(y) = (f_1(y), ..., f_m(y))^T = 0$, where each constraint $f_i(y) = 0$ represents $y_i - \tilde{g}_i(y_{t,\mathcal{N}_i}) = 0$, there are no analytical solutions, but we can solve an optimization problem with nonlinear equality constraints, i.e. finding the nearest projection point on the plane $f(y) = 0$ given the reference point $\hat{y}_{t+\tau}$ for $\tau = 1, \dots, m$

$$\min_{\tilde{y}_{t+\tau}} \|\tilde{y}_{t+\tau} - \hat{y}_{t+\tau}\|_2^2, \text{ s.t. } f(\tilde{y}_{t+\tau}) = 0. \tag{14}$$

A simple approximate method for solving this equality-constrained quadratic programming is to conduct iterative projections. Denote $\mathcal{J} = \frac{\partial f}{\partial x}$ as the Jacobian matrix. Assuming $\hat{y}_{t+\tau} \approx \tilde{y}_{t+\tau}$, closed to the surface $f(x) = 0$. We derive the first-order Taylor expansion of $f(x)$ at $\hat{y}_{t+\tau}$ as

$$f(x) \approx f(\hat{y}_{t+\tau}) + \mathcal{J}^T \cdot (x - \hat{y}_{t+\tau}). \tag{15}$$

Equating $f(x)$ to zero with $x = \tilde{y}_{t+\tau}$ yields

$$\tilde{y}_{t+\tau} = \hat{y}_{t+\tau} - \mathcal{J}(\mathcal{J}^T\mathcal{J})^{-1}f(\hat{y}_{t+\tau}). \tag{16}$$

Then we can repeat the above transformation several times (e.g. number of projections $K = 10$ times used in our experiments) until the constraints are well satisfied by evaluating whether $F(x) = \sum_{j=1}^{m} |f_j(x)|$ is small enough.

2.3. Functional relation field-enhanced spatio-temporal graph networks

In this part, we integrate the proposed functional relation field framework into five representative backbone models, STGCN [2], AGCRN [3], Autoformer [9], FEDformer [10] and SCINet [11] to boost their prediction performance, referred as FRF-STGCN, FRF-AGCRN, FRF-Autoformer, FRF-FEDformer and FRF-SCINet, respectively. In the first stage, we learn the functional relation network, based on which the most relevant nodes can be identified. And the resultant graph structure could be used for the five backbone networks. In the second stage, we enforce the learned constraints in the training and inference process, as described in Fig. 2.

Since different backbone networks have their own specific design, we need adapt FRF to these backbones. For the constraint satisfaction of output, in AGCRN and SCINet, the networks produce all the prediction results at multiple time steps in one batch, and therefore, the constraint-satisfied transformation is applied to the prediction at each time step respectively for K times as described in Eq. (16). For STGCN, we apply the above transformation sequentially to each future time step, obtain the transformed predictions, and then feed the predictions to STGCN to produce the predictions at the next time step. We repeat this procedure until we finish the multi-step forecasting task.

Algorithm 1 Training and inference of FRF.

```
1: Input: Trained function relation networks $f$, $\lambda$ and $K$.
2: Output: constraint-satisfied output $\tilde{y}_{t+\tau}$
   # Training Phase
3: repeat
4:     Forward on backbone network to obtain $\hat{y}_{t+\tau}$
5:     Back-propagate with the loss $\mathcal{L}_{total}$ in Eq. (12) and run Adam.
6: until stopping criteria is met
   # Inference Phase
7: Forward on the trained backbone network to obtain $\hat{y}_{t+\tau}$
8: for $k = 0$ to K do
9:     Calculate $\tilde{y}_{t+\tau}$ by Eq. (16)
10: end for
```

2.4. Limitation

Our model assumes static, time-invariant constraints between nodes. This might limit its applicability in the scenarios where the relationships or dependencies between time series evolve *frequently*. While it is true that dynamic relationship may be better than static constraints, there are two considerations why we chose to investigate the static relationship. On the one hand, in the considered applications, the relationship might be stationary in reality. For instance, the MiniApp calling relationship rarely changes over time in almost one or two years to maintain the stability of the whole online payment system, see Sec. 3.1 for more details. For traffic flow prediction, the road network and traffic condition is also unchanged given a fixed network topology. Therefore, we assume the stationary relationship is realistic and efficient for many real-world forecasting tasks, at least for those considered and similar ones in our paper. On the other hand, we adopt static relationship due to the computational advantage. Concretely, non-linear optimization for satisfying the constraints has no analytical solution and thus a simple approximation method for solving this problem is to conduct iterative projections, as shown in Eq. (16). It is extremely challenging to solve the non-linear programming problems with changing variables. The dynamic relationship modeling will be taken as a key point for future exploration.

3. Experiments

In this section, we conduct experiments on five datasets including one synthetic graph dataset, two real-word MiniApp calling flow datasets and two traffic flow datasets to demonstrate the effectiveness of FRF on learning the underlying relationship between nodes and boosting the prediction performance of the backbone networks. The code for reproducibility is online, https://github.com/zhanxingzhu/Functional_Relation_Field_Time_Series/.

The baseline models. We first compare our framework with two traditional forecasting models including Historical Average (HA) and Support Vector Regression (SVR). Then, we also implement experiments using two popular deep learning models, including Feed-Forward Neural Network (FNN) and Full-Connected LSTM (FC-LSTM [28]). We select the widely used graph time series model STGCN [2], AGCRN [3], and the univariate time series forecasting models based on transformer architectures Autoformer [9], FEDformer [10] and another state-of-the-art univariate prediction model SCINet [11]) as our backbone networks. To demonstrate that our proposed FRF can consistently improve the graph structure learning models, we also incorporate the FRF into two more approaches involved with graph structure learning GTS [7] and NRI [29]. The first one GTS learns the graph structures and performs forecasting simultaneously with a GNN. The second model neural relational inference (NRI) is an unsupervised one that learns to infer interactions and forecasting with an LSTM.

3.1. Datasets

Binary tree dataset. We first generate a synthetic graph time series dataset. The graph structure for this dataset is a complete binary tree with 255 nodes. For each leaf node i, its value is a noisy sinusoidal wave across time, $x_{i,t} = n_{i,t} A_i \sin(\frac{2\pi t}{T_i} + \phi)$, where $n_{i,t}$ follows a uniform distribution, i.e., $n_{i,t} \sim \mathcal{U}(0.95, 1.05)$. We sort all leaf nodes from left to right in an increasing order of their periods. For a non-leaf node p, we denote its left and right child as l and r. We further set the value of node p to be the geometric mean of its two children l and r, $x_{p,t} = \sqrt{x_{l,t} \cdot x_{r,t}}$. We sample one point every 5 minutes, so there are 288 points per day. We generate the data for 40 days, including 30 days for training (i.e., $30 \times 288 = 8640$ time points), 5 days for validation, and 5 days for testing. We intentionally design this dataset since the true graph structure is available for these different time series and the constraints between nodes are explicit, and thus it is a suitable testbed to compare the superiority of FRF over those models without FRF. In

Table 1
Model performance on BinaryTree and MiniApp datasets. "(T)" and "(L)" represent the models with true and learned constraints, respectively. Bold font is used to show the advantage over backbones. "-" represents that the ground truth of the functional relationship is not available.

Methods	Binary tree		MiniApp 1		MiniApp 2	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
HA	12.64	19.19	3.97	9.77	11.02	35.23
SVR	8.71	14.00	2.56	7.06	6.83	21.68
FNN	5.77	10.04	2.09	6.26	5.43	16.84
FC-LSTM	17.08	22.83	2.05	4.08	8.14	19.64
STGCN	2.65	5.82	1.90	5.26	4.50	14.14
FRF-STGCN (T)	**2.40**	**5.68**	-	-	-	-
FRF-STGCN (L)	2.50	5.71	**1.21**	**3.32**	**4.19**	**11.11**
AGCRN	2.56	5.77	0.41	1.17	1.43	3.79
FRF-AGCRN (T)	**2.30**	**5.54**	-	-	-	-
FRF-AGCRN (L)	2.37	5.57	**0.35**	**0.92**	**1.33**	**3.39**
Autoformer	8.54	13.16	1.03	2.79	2.69	6.85
FRF-Autoformer (T)	**8.34**	**12.79**	-	-	-	-
FRF-Autoformer (L)	**8.34**	12.83	**0.77**	**2.18**	**2.46**	**5.70**
FEDformer	8.54	13.24	0.60	1.80	2.08	5.13
FRF-FEDformer (T)	**8.10**	**12.80**	-	-	-	-
FRF-FEDformer (L)	8.29	12.99	**0.58**	**1.76**	**2.03**	**4.98**
SCINet	5.43	9.37	0.52	1.51	1.78	3.88
FRF-SCINet (T)	**5.36**	9.34	-	-	-	-
FRF-SCINet (L)	5.37	**9.27**	**0.47**	**1.34**	**1.71**	**3.65**

the experiments, for the backbone with FRF, we assume the constraints are unknown and learn them using the proposed method in Section 2.1.

MiniApp calling flow dataset 1 and 2. These two datasets are real-word flow data from two popular online payment MiniApps, attached in the web page https://drive.google.com/file/d/1R4-LeYv2GCF5zzyYeqymU534GXn8nDpV/view?usp=share_link. For the two MiniApps, there are $N = 30, 23$ filtered pages linking to each other in the calling process, which produces visiting request flow from one page to another, constituting a graph with $N = 30, 23$ nodes. We aggregate the flow with averaged value every 5 minutes for each node, so there are 288 points per day. For the first MiniApp, we collect 21 days of data, including 15 days for training, 3 days for validation, and 3 days for test. For the second one, 24 days of data are collected, including 18 days for training, 3 days for validation, and 3 days for testing.

PEMSD4 and PEMSD8 traffic datasets. This benchmark dataset is widely used for multi-variate time series prediction, describing the traffic speed in San Francisco Bay Area with 307 sensors on 29 roads (https://paperswithcode.com/dataset/pemsd4). The other one consists of 170 detectors on 8 roads in San Bernardino area (https://paperswithcode.com/dataset/pemsd8).

3.2. Results

Overall prediction performance Table 1, 2 and 3 summarizes the performance of all the compared models on the five datasets, including the proposed FRF approach coupled with STGCN, AGCRN, Autoformer, FEDformer and SCINet, denoted as FRF-STGCN and FRF-AGCRN, FRF-Autoformer, FRF-FEDformer and FRF-SCINet, respectively. To conduct a fair comparison with these backbone networks, we only tune the parameters of FRF while keeping the other hyper-parameters setting the same as the original backbone networks. We conduct experiments on Binary tree, MiniApp1 and MiniApp2 datasets using the opensourced code (https://github.com/chaoshangcs/GTS.git) for GTS and (https://github.com/ethanfetaya/NRI.git) for NRI shown in Table 3, demonstrating that FRF can also improve the forecasting performance on GTS. For the binary tree dataset, we predict the future 12 time steps and evaluate the performance in terms of three metrics (MAE, RMSE, MAPE). Since the underlying true constraints are known, we report the experimental results of our models with both true and learned constraints, denoted as "T" and "L". We can observe that deep learning-based models typically outperform the traditional ones, as expected. Furthermore, the proposed functional relation field can further improve the performance of the original backbone models. Regardless of the differences between the two backbone networks, FRF can consistently improve the prediction accuracy for both of the backbones, indicating that the FRF framework could be potentially applied to a wide variety of backbones.

For the two MiniApp datasets, we omit the metric MAPE since the scale of data changes dramatically across time such that MAPE fails to characterize the performance of different models. Due to the error accumulation problem for multi-step prediction in STGCN, the performance of this model pales in comparison with its non-iterative counterpart. As a result, we only report the results of the non-iterative version of STGCN. Since the underlying true constraint relationship between nodes is not available, we only report the FRF with learned constraints. We can easily observe that augmentation of the proposed FRF can consistently boost the performance

Table 2
Model performance on two traffic datasets, PeMSD4 and PeMSD8. "-" represents the value is too large and thus ignored.

Model Type	Methods	PEMSD4		PEMSD8	
		MAE	RMSE	MAE	RMSE
Multivariate	STGCN [2]	21.61	35.25	17.28	27.19
	FRF-STGCN	**20.70**	**33.90**	**16.46**	**26.05**
	AGCRN [3]	19.81	32.58	16.52	26.12
	FRF-AGCRN	**19.59**	**31.85**	**16.04**	**25.28**
Univariate	Autoformer	21.42	34.09	18.49	28.78
	FRF-Autoformer	**21.24**	**33.93**	**18.23**	**28.66**
	FEDFormer	21.59	34.23	18.52	29.23
	FRF-FEDFormer	**21.29**	**33.82**	**18.15**	**28.61**
	SCINet [11]	19.27	31.27	15.71	24.60
	FRF-SCINet	**19.15**	**31.09**	**15.67**	**24.57**

Table 3
Performance comparison of FRF enhanced GTS and NRI networks on BinaryTree and MiniApp datasets. NRI FRF-AGCRN is the state-of-the-art model, i.e. AGCRN after incorporating our proposed FRF.

Methods	Binary tree		MiniApp 1		MiniApp 2	
	MAE	RMSE	MAE	RMSE	MAE	RMSE
GTS	5.85	9.19	1.92	2.32	3.88	7.26
FRF-GTS (L)	5.70	8.33	**1.77**	**1.84**	**2.70**	**5.14**
FRF-GTS (T)	**5.67**	**8.21**	-	-	-	-
NRI	22.77	30.15	2.50	6.89	8.04	16.92
FRF-NRI (L)	21.63	28.70	**2.47**	**6.88**	**7.98**	**14.35**
FRF-NRI (T)	**19.61**	**25.69**	-	-	-	-
FRF-AGCRN (SOTA)	**2.30**	**5.54**	**0.35**	**0.92**	**1.33**	**3.39**

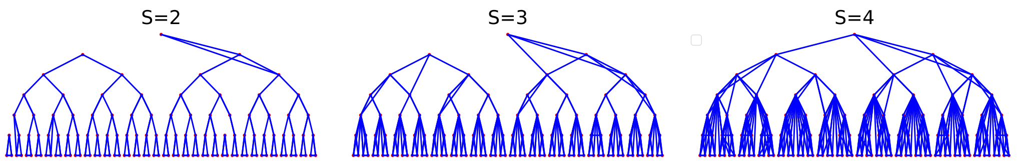

Fig. 3. The learned constraints of the Binary Tree dataset by connecting each constrained node with their most related nodes. We use $S = 2, 3, 4$ for every node to plot this figure, so there are lack of connections when $S = 2$ and some redundant connections when $S = 4$.

of the five backbone networks. Specifically, FRF improves STGCN by 36.3% and 6.9% on the two datasets, also improves AGCRN by 14.6% and 7.0%, respectively.

For traffic datasets PEMSD4 and PEMSD8, one particular reason we choose SCINet as the baseline is that the reported results can achieve state-of-the-art prediction performance on this task (Table 2). We can observe that even comparing such a strong baseline, FRF framework can still improve its performance of with a margin 0.6% and 0.3% on both datasets, respectively. For other backbones, we again see that FRF further improves the prediction performance, showing the effectiveness of FRF as a model-agnostic framework.

Table 3 shows the prediction results on Binary tree, MiniApp1 and MiniApp2 dataset using the FRF enhanced on GTS and NRI. We can easily observe that FRF can consistently improve the prediction performance of NRI for these datasets.

3.3. Learning the relationship between nodes

We further test whether FRF could discover the underlying true constraints between nodes. First, we investigate whether we can reliably estimate the target node given the values of constraint nodes. To be exact, we compute $\hat{x}_{t,i} = \tilde{g}(\{x_{t,\mathcal{N}_i}\})$ and compare $\hat{x}_{t,i}$ with $x_{t,i}$ in terms of Mean Absolute Percentage Errors (MAPE), where a large MAPE indicates the time-invariant constraint is weak. Through our experiments, the achieved MAPEs for the considered BinaryTree, MiniApp1, MiniApp2, PEMSD4, PEMSD8 datasets are small, i.e. 10%, 0.8%, 1%, 2%, 7%, respectively. Since a smaller MAPE means the strong constraint relationship between nodes, therefore the proposed FRF is applicable to the backbone network for boosting its prediction performance. On the other hand, we compare the prediction performance of the proposed algorithm when using the true and estimated constraints, showing the results in Table 1. We can observe that the performance based on both the true and estimated constraints is almost the same, indicating that the constraints are accurately learned. Therefore, using the learned constraints can well regularize the predictions given by the original

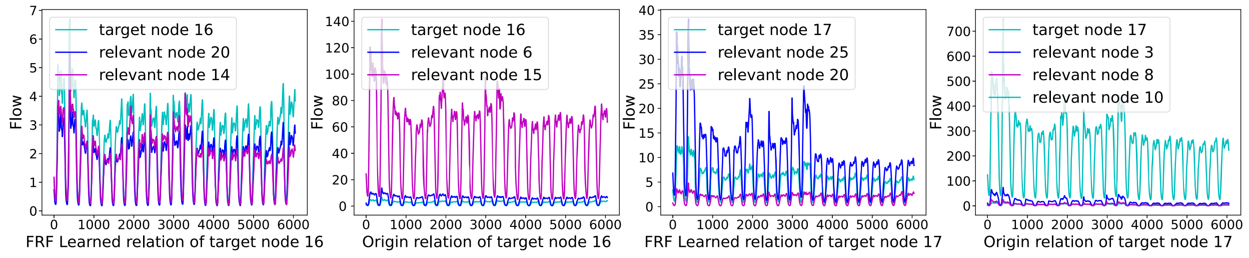

Fig. 4. Flow visualization of learned relation and origin one for MiniApp1 dataset. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

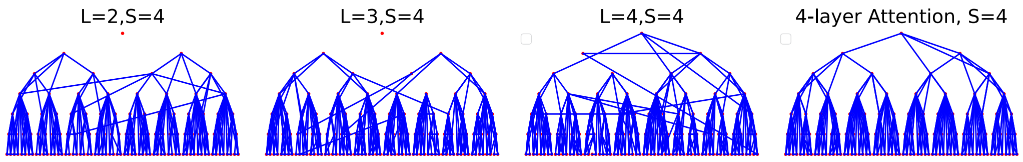

Fig. 5. The learned tree structure from the constraint networks with different number of layers. We use $S = 4$ for every node and the number of full-connected layers is $L = 2, 3, 4$ from the left to right panel as well as a four-layer attention network to demonstrate the sensitivity of constraint architectures.

network backbones as well as further improve the forecasting performance. Additionally, we visualize the learned constraints by connecting each constrained node with its most relevant neighbors as a graph, shown in Fig. 3. The structure of the binary tree is well recovered, although some extra edges are involved.

For realworld data, we compare the learned functional relation and origin one for MiniApp1 dataset. In this paper, the “origin graph” or “origin relation” indicates the relation extracted from prior knowledge which can be real road network in traffic graphs, or the calling relationship in cloud service. We now show that the learned relation is more informative than the original relation and could potentially help to improve the prediction performance. In Fig. 4, we present the flows of two target nodes and its relevant nodes learned from FRF or based on original relation. We can observe that the flow of the target node has the same pattern and scale as the relevant nodes on the learned function, while it has a significantly different pattern from the origin graph. The results demonstrate that the relation learned from FRF has captured more informative and effective flow relationship than the original relation relying on prior information.

3.4. Network architecture and hyperparameter configuration

The architecture of constraint network. We compare both fully-connected neural networks (including 2/3/4-layer) and a self-attention network introduced in Eq (10). The results on Binary Tree are shown in Fig. 5. We can observe that for fully-connected networks, the more layers we used, the more accurate the learned constraint relation is. As we further increase the number of layers of networks, the results saturate and are comparable with the attention network. This indicates the prediction results are stable and consistent across different network architectures.

In the following, we describe how we configure the hyper-parameters involving in the training of the constraint network. Generally we set the parameters according to some rules of thumb (described in the following) and finally tune them on the validation data to achieve the best performance. Through the experiments, the rules of thumb we have found could apply to all the tasks and datasets we have considered, which could be potentially generalizable to other tasks for users interested.

The cardinality of the neighborhood, S. We typically choose S as $10\% - 20\%$ of the total number of nodes to achieve a good coverage of the neighbors. For Binary Tree dataset, we set $S = 4$ to recover the functional relations shown in Fig. 3. We set $S = 6$ for two MiniApp flow calling datasets. For traffic dataset PEMSD4 with 307 nodes and PEMSD8 with 170 nodes, the best performance is obtained when $S = 30$.

Regularization coefficient λ and number of iterations K. In the training stage, we only tune the tradeoff coefficient λ and the number of iterations K while keeping all other parameters the same as SOTA settings in the benchmarks. The detailed settings are shown in Table 4.

Hyperparameter sensitivity of error threshold ϵ_{err}, λ and K. FRF enhanced models introduce additional three kinds of hyperparameters including validation error threshold ϵ_{err}, the loss tradeoff coefficient λ and the number of output transformation K. Therefore, we conduct hyper-parameters sensitivity experiments on binary tree dataset using backbone AGCRN as shown in Fig. 6. We can observe that the performance slightly improves when the ϵ_{err} increases due to more constraints are discovered, while the performance decreases with large ϵ_{err} because of the introduced noise. Even more, the FRF enhanced model performs worse than backbone network when $\epsilon_{err} = 5.0$. Consistently, FRF enhanced model performs better when $\lambda = 0.1$ and worse than backbone with large λ. For the number of iterations K, the larger K improves the backbone more significantly than smaller one since iterating more times makes the non-linear constraint optimization problem more accurate. We suggest users set $K = 10$ which typically could achieve satisfying prediction performance in various tasks.

Table 4
Detailed hyper-parameter settings of all graph time series and univariate backbone networks for the five datasets.

Methods	Binary tree		MiniApp 1		MiniApp 2		PEMSD4		PEMSD8	
	λ	K	λ	K	λ	K	λ	K	λ	K
FRF-STGCN	0.1	10	0.01	5	0.01	10	0.01	5	0.001	10
FRF-AGCRN	0.1	10	0.01	5	0.01	10	0.01	5	0.1	10
FRF-Autoformer	0.01	20	0.001	5	0.001	5	0.001	5	0.001	10
FRF-FEDformer	0.01	20	0.001	5	0.001	5	0.001	5	0.001	10
FRF-SCINet	0.01	10	0.001	5	0.001	5	0.0001	5	0.0001	5

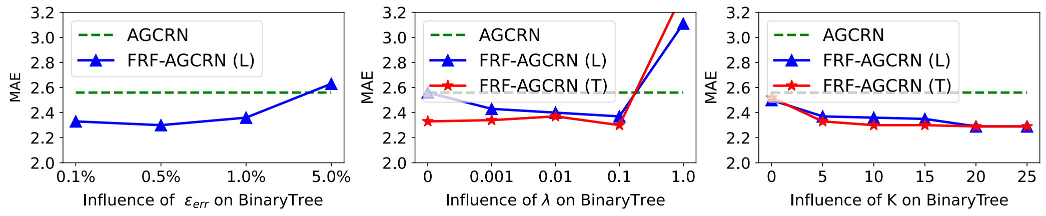

Fig. 6. Performance comparison of three kinds of hyper-parameters including ϵ_{err}, λ_R and K on Binary Tree dataset with the state-of-the-art backbone AGCRN.

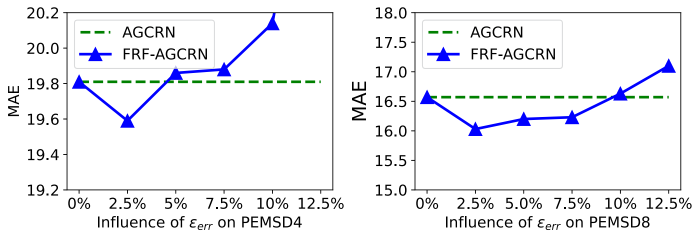

Fig. 7. Performance comparison of ϵ_{err} on PEMSD4 and PEMSD8 dataset for the backbone AGCRN.

Table 5
The computational time of relation network pre-training and re-training, training regularized backbone network, and inference time of FRF with different number of iterations K. The times is recorded every epoch in seconds.

Datasets	# Relation net (s)		# Train time (s)		# Inference time (s)			
	Pretrain	Retrain	w.o. λ	w. λ	$K = 0$	5	10	20
Binary tree	2384.25	63.96	10.14	10.65	0.64	1.54	2.47	4.23
MiniApp1	487.69	10.95	3.89	4.15	0.28	0.33	0.38	0.49
MiniApp2	388.41	11.78	4.57	4.89	0.06	0.33	0.40	0.52

For the binary tree dataset and MiniApp calling flow datasets involving strong constraint relationships, we set $\epsilon_{err} = 0.01$ to filter the constraint nodes. However, for traffic dataset PEMSD4 and PEMSD8 with relatively weak constraints, we set $\epsilon_{err} = 0.025$ to achieve the best performance. The hyper-parameters sensitivity experiments of ϵ_{err} on PEMSD4 and PEMSD8 datasets are shown in Fig. 7.

3.5. *Computational complexity analysis*

In this part, we perform experiments on Nividia A100-80G GPU then report the computational time of all the stages of our FRF enhanced models in practical settings, as shown in Table 5.

We can observe that the pretraining of the constraint network takes the longest time especially when the number of nodes is large, since the pretraining requires all the other nodes as input when identifying the target node and its related neighbors. Although learning the relation function costs much time, it is only calculated once in the whole process. The time for re-training is short and thus can be ignored. The computational time of Eq. (16) is also negligible. Though the inference time increases with the number of iterations K, it is only in the scale of seconds.

On one hand, the computational complexity increases in the forecasting network training caused by the K iterations of output constraint satisfaction. The K is usually set as a small number 5 or 10, which is computationally cheap. And the main time-consuming operations origin from forward and back propagation of backbones rather than the output constraint. On the other hand, we need to train the constraint network for all time series. Fortunately, the constraint network is a simple two-layer attention network, which

Table 6
Ablation study on the explicit graph, constraint graph learned from a constraint network, and constraint satisfaction components using STGCN as the backbone network.

#	Explicit Graph	Constraint Graph	Constraint Satisfaction	PeMSD4		PeMSD8	
				MAE	RMSE	MAE	RMSE
1	✓	×	×	21.61	35.25	17.28	27.19
2	×	✓	×	21.26	35.13	16.79	26.61
3	×	✓	✓	20.70	33.90	16.46	26.05

Table 7
Ablation study on constraint-satisfaction loss minimization and constraint-satisfaction transformation. The backbone AGCRN is used for Binary Tree, MiniApp2; and SCINet for PEMSD4.

#	FRF Training	FRF Inference	Binary Tree		MiniApp2		PEMSD4	
			MAE	RMSE	MAE	RMSE	MAE	RMSE
1	×	×	2.56	5.77	1.43	3.79	19.27	31.27
2	✓	×	2.51	5.68	1.38	3.66	19.20	31.16
3	×	✓	2.33	5.58	1.45	3.74	19.22	31.15
4	✓	✓	2.30	5.54	1.33	3.29	19.15	31.09

only has a small number of parameters but effective enough to capture the complex functional relations. For example, in MiniApp1 task, each constraint network only has around 3,000 parameters, the training time is in the scale of seconds. Thus, we believe training a constraint network is fast and does not require much computational resources. The small size of the constraint networks is amenable to large-scale multi-variate time series.

3.6. Ablation study

We first conduct an ablation study on the constraint graph learned from constraint network using the STGCN as backbone network in Table 6. We can observe that the constraint graph performs better than explicit graph extracted from prior knowledge on both traffic and MiniApp datasets. In addition, for backbone networks without explicit graph structure such as AGCRN and SCINet, we investigate the effectiveness of constraint-satisfaction loss minimization and constraint-satisfaction transformation as shown in Table 7, finding that both of the two components contribute to the forecasting performance. Specifically, for the backbone network AGCRN which achieves the state-of-the-art performance on binary tree dataset, FRF enhances the backbone by 1.95% in training phase and by 9.0% in inference phase, while the combination of two components improves the performance by 10.16% in total.

4. Conclusion

In this paper, we have proposed to enhance the multivariate time series forecasting with a new inductive bias, function relation field (FRF), which is model-agnostic. FRF can discover the intrinsic graph structure, as well as improve flow forecasting performance by applying constraint function relationship to the output in training and testing phases. The constraints learned by FRF can be incorporated into existing backbone networks, consistently improving the prediction performance. Experimental results show that the proposed FRF framework can reliably learn the constraints from the time-series data and restore the graph structure. Moreover, these constraints in turn help improve the prediction accuracy by a notable margin, regardless of the diversity of the network architecture in different backbone models. We expect that this FRF inductive bias could be potentially employed in other multivariate settings beyond times series scenarios. In the future, we will consider how to incorporate dynamic constraints to our framework such that it can be extended to more challenging tasks, for instance, modeling the functional relations with a slightly changing manner. It is worthy of considering to improve the efficiency and optimization of the neural network [30], which is important when deploying the proposed framework on real-world datasets.

CRediT authorship contribution statement

Ting Li: Data curation, Methodology, Software, Visualization, Writing – original draft. **Bing Yu:** Conceptualization, Methodology, Software, Writing – original draft. **Jianguo Li:** Conceptualization, Project administration, Supervision. **Zhanxing Zhu:** Conceptualization, Formal analysis, Investigation, Methodology, Project administration, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] Y. Li, R. Yu, C. Shahabi, Y. Liu, Diffusion convolutional recurrent neural network: data-driven traffic forecasting, in: International Conference on Learning Representations, 2018, https://openreview.net/forum?id=SJiHXGWAZ.

[2] B. Yu, H. Yin, Z. Zhu, Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting, in: IJCAI, 2018.

[3] L. Bai, L. Yao, C. Li, X. Wang, C. Wang, Adaptive graph convolutional recurrent network for traffic forecasting, in: Proceedings of 34th Conference on Neural Information Processing Systems, 2020.

[4] S. Yan, Y. Xiong, D. Lin, Spatial temporal graph convolutional networks for skeleton-based action recognition, in: AAAI, 2018.

[5] Z. Liu, H. Zhang, Z. Chen, Z. Wang, W. Ouyang, Disentangling and unifying graph convolutions for skeleton-based action recognition, in: CVPR, 2020.

[6] B. Yu, M. Li, J. Zhang, Z. Zhu, 3d Graph Convolutional Networks with Temporal Graphs: A Spatial Information Free Framework for Traffic Forecasting, 2019.

[7] C. Shang, J. Chen, J. Bi, Discrete graph structure learning for forecasting multiple time series, in: International Conference on Learning Representations, 2021, https://openreview.net/forum?id=WEHSlH5mOk.

[8] M. Li, P.-Y. Huang, X. Chang, J. Hu, Y. Yang, A. Hauptmann, Video pivoting unsupervised multi-modal machine translation, IEEE Trans. Pattern Anal. Mach. Intell. 45 (2022) 3918–3932.

[9] H. Wu, J. Xu, J. Wang, M. Long, Autoformer: decomposition transformers with auto-correlation for long-term series forecasting, in: Advances in Neural Information Processing Systems, 2021.

[10] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, R. Jin, FEDformer: frequency enhanced decomposed transformer for long-term series forecasting, in: Proc. 39th International Conference on Machine Learning (ICML 2022), 2022.

[11] M. Liu, A. Zeng, M. Chen, Z. Xu, Q. Lai, L. Ma, Q. Xu, Scinet: time series modeling and forecasting with sample convolution and interaction, in: Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS), 2022, p. 2022.

[12] D.E. Rumelhart, Learning Representations by Error Propagation, 1986.

[13] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (1997) 1735–1780.

[14] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using RNN encoder–decoder for statistical machine translation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Association for Computational Linguistics, Doha, Qatar, 2014, pp. 1724–1734.

[15] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L.u. Kaiser, I. Polosukhin, Attention is all you need, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 30, Curran Associates, Inc., 2017, https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[16] J. Simm, A. Arany, E.D. Brouwer, Y. Moreau, Expressive graph informer networks, arXiv:1907.11318, 2020.

[17] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, W. Zhang, Informer: beyond efficient transformer for long sequence time-series forecasting, in: The Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Virtual Conference, vol. 35, AAAI Press, 2021, pp. 11106–11115.

[18] R. Child, S. Gray, A. Radford, I. Sutskever, Generating long sequences with sparse transformers, arXiv:1904.10509, 2019.

[19] B. Lim, S.O. Arik, N. Loeff, T. Pfister, Temporal fusion transformers for interpretable multi-horizon time series forecasting, arXiv:1912.09363, 2020.

[20] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, X. Yan, Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019, https://proceedings.neurips.cc/paper/2019/file/6775a0635c302542da2c32aa19d86be0-Paper.pdf.

[21] B.N. Oreshkin, D. Carpov, N. Chapados, Y. Bengio, N-beats: neural basis expansion analysis for interpretable time series forecasting, in: International Conference on Learning Representations, 2020.

[22] D. Salinas, V. Flunkert, J. Gasthaus, T. Januschowski, Deepar: probabilistic forecasting with autoregressive recurrent networks, Int. J. Forecast. 36 (2020) 1181–1191, https://doi.org/10.1016/j.ijforecast.2019.07.001.

[23] S.S. Rangapuram, M.W. Seeger, J. Gasthaus, L. Stella, Y. Wang, T. Januschowski, Deep state space models for time series forecasting, in: S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (Eds.), Advances in Neural Information Processing Systems, vol. 31, Curran Associates, Inc., 2018, https://proceedings.neurips.cc/paper/2018/file/5cf68969fb67aa6082363a6d4e6468e2-Paper.pdf.

[24] C. Chen, K. Li, S.G. Teo, X. Zou, K. Wang, J. Wang, Z. Zeng, Gated residual recurrent graph neural networks for traffic prediction, Proc. AAAI Conf. Artif. Intell. 33 (2019) 485–492.

[25] Z. Pan, S. Ke, X. Yang, Y. Liang, Y. Yu, J. Zhang, Y. Zheng, Autostg: neural architecture search for predictions of spatio-temporal graph, in: Proceedings of the Web Conference 2021, WWW '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1846–1855.

[26] T. Li, J. Zhang, K. Bao, Y. Liang, Y. Li, Y. Zheng, Autost: efficient neural architecture search for spatio-temporal prediction, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 794–802.

[27] Y. Seo, M. Defferrard, P. Vandergheynst, X. Bresson, Structured Sequence Modeling with Graph Convolutional Recurrent Networks, 2016.

[28] I. Sutskever, O. Vinyals, Q.V. Le, Sequence to sequence learning with neural networks, in: Proceedings of 34th Conference on Neural Information Processing Systems, 2014.

[29] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, R. Zemel, Neural relational inference for interacting systems, arXiv preprint, arXiv:1802.04687, 2018.

[30] C. Li, G. Wang, B. Wang, X. Liang, Z. Li, X. Chang, Ds-net++: dynamic weight slicing for efficient inference in cnns and vision transformers, IEEE Trans. Pattern Anal. Mach. Intell. 45 (2022) 4430–4446.