Structure Meets Sequences: Predicting Network of Co-evolving Sequences

Yaojing Wang[†], Yuan Yao[†], Feng Xu[†], Yada Zhu[§], Hanghang Tong[‡]
[†] State Key Laboratory for Novel Software Technology, Nanjing University, China
[§]IBM Research, USA
[‡] University of Illinois at Urbana-Champaign, USA
wyj@smail.nju.edu.cn,{y.yao,xf}@nju.edu.cn,yzhu@us.ibm.com,htong@illinois.edu

ABSTRACT

Co-evolving sequences are ubiquitous in a variety of applications, where different sequences are often inherently inter-connected with each other. We refer to such sequences, together with their inherent connections modeled as a structured network, as *network of co-evolving sequences (NoCES)*. Typical NoCES applications include road traffic monitoring, company revenue prediction, motion capture, etc. To date, it remains a daunting challenge to accurately model NoCES due to the coupling between network structure and sequences. In this paper, we propose to modeling NoCES with the aim of simultaneously capturing both the dynamics and the interplay between network structure and sequences. Specifically, we propose a joint learning framework to alternatively update the network representations and sequence representations as the sequences evolve over time. A unique feature of our framework lies in that it can deal with the case when there are co-evolving sequences on both network nodes and edges. Experimental evaluations on four real datasets demonstrate that the proposed approach (1) outperforms the existing competitors in terms of prediction accuracy, and (2) scales linearly w.r.t. the sequence length and the network size.

CCS CONCEPTS

• **Networks** → **Network dynamics**; • **Computing methodologies** → **Neural networks**.

KEYWORDS

Co-evolving sequences, sequence prediction, network structure

ACM Reference Format:
Yaojing Wang[†], Yuan Yao[†], Feng Xu[†], Yada Zhu[§], Hanghang Tong[‡]. 2022. Structure Meets Sequences: Predicting Network of Co-evolving Sequences. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3488560.3498411

WSDM '22, February 21–25, 2022, Tempe, AZ, USA

ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3498411

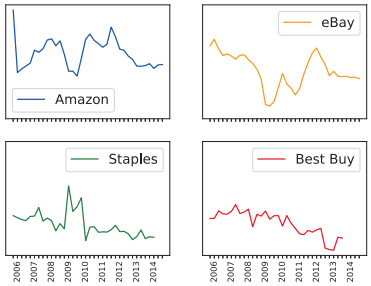


Figure 1: The quarterly revenues of four well-known companies from 2006 to 2014.

1 INTRODUCTION

Sequential data (e.g., time series) has been widely studied to recognize human actions [1], understand company revenues [28], predict traffic conditions [31], estimate article citations [46], etc. In these applications, a common character is that different sequences are often influenced by or correlated with other sequences. For example, company revenues are correlated since there exist competition and cooperation relationships among them; the congestion conditions in a local area are correlated due to the simple fact that vehicles have to drive on connected roads. To capture the correlations between sequences, one natural choice is to model their inherent connections as a structured network, whose nodes (e.g., sensor readings) and edges (e.g., road congestions) may have co-evolving sequences. In this work, we refer to these sequences together with their underlying network as a *network of co-evolving sequences (NoCES)*.

Fig. 1 shows an example of NoCES. This example contains the quarterly revenues from four well-known companies in United States, i.e., Amazon, eBay, Staples, and Best Buy. The revenues are from 2006 to 2014. We can see from the figures that there may exist correlations among some of the revenue sequences. For example, both Amazon and eBay run into a decreasing of revenue around 2009, while the revenue increases for Staples in the same time period; in contrast, Best Buy is not significantly affected by the other companies at that time. The possible reasons include: both Amazon and eBay focus on the e-commerce whereas Staples offers solutions with direct sales, and Best Buy mainly sells electronic products whereas the other three companies offer a variety of other products.

Despite its widespread existence and applicability, it largely remains a challenging problem to accurately model NoCES due to the following reasons:

- (A) The *interplay and dynamics* between network structure and sequences. On one hand, the sequences could be influenced by each other via the underlying network; on the other hand, the node/edge representations could change dynamically as the sequences evolve over time.
- (B) The *generality* of the model. To the best of our knowledge, none of the existing work can deal with the case when there are co-evolving sequences on both nodes and edges in a network.

NoCES modeling is closely related to two lines of existing work. The first line is multivariate time series recovering or forecasting [4, 9, 23, 42]. However, these work does not utilize the explicit network structure in the learning process, and thus becomes suboptimal when the network structure is available. The second line is traffic monitoring [14, 16, 24, 45, 48]. These work takes into account the localness and diffusion nature of traffic to improve prediction accuracy. However, such methods are specially designed for traffic prediction and may become less effective in other NoCES scenarios when the localness and diffusion nature does not hold. Additionally, none of the above work considers the case when there are both node sequences and edge sequences.

In this paper, we propose a joint learning framework, SeeS (Structure Meets Sequences), to model NoCES. SeeS learns the network representations and sequence representations in a mutually beneficial way so as to capture the interplay and dynamics between them. Specifically, in each time step, SeeS first treats the current sequence representations as node/edge attributes, and simultaneously learns the representations of both nodes and edges in the network; then, SeeS updates the sequence representations using the learned node/edge representations; such an alternative learning process repeats as new sequence data keeps arriving. A unique advantage of SeeS is that it can handle the case when there are co-evolving sequences on both nodes and edges. To evaluate the effectiveness and efficiency of the proposed approach, we conduct extensive experiments on four real NoCES scenarios from different domains (e.g., company revenues, body motions, and traffic controls), with two prediction tasks (i.e., missing value recovery and future value prediction). The results demonstrate that the proposed approach outperforms existing competitors in terms of prediction accuracy, and scales linearly w.r.t. the sequence length and network size.

In summary, the main contributions of this paper include:

- The general learning framework SeeS for NoCES, which considers the bidirectional and dynamical effect between sequences and network structure.
- Extensive experimental evaluations, which demonstrate that SeeS outperforms the existing approaches on various NoCES scenarios. For example, SeeS achieves up to 51.4% relative improvement over the best competitor in terms of recovering the missing values.

The rest of the paper is organized as follows. Section 2 states the problem definition. Section 3 describes the proposed approach, and Section 4 presents the experimental results. Section 5 reviews the related work, and Section 6 concludes.

2 PROBLEM DEFINITION

We use $S = S_v \cup S_e$ to denote the set of sequences in a NoCES, where S_v is the set of node sequences and S_e is the set of edge sequences. We allow either S_v or S_e to be empty. For simplicity, we suppose each sequence is of length T, and use superscript to indicate the time step unless otherwise stated. We use $G = (V, E)$ to denote the underlying network between the sequences, where V is the node set ($|V| = N$) and E is the edge set ($|E| = M$). For this network, we use A to denote its adjacency matrix, and use $Z_v^t \in \mathbb{R}^{N\times d}$ and $Z_e^t \in \mathbb{R}^{M\times d}$ to denote its *node representations* and *edge representations* at time step t, respectively. We also use $Y_v^t \in \mathbb{R}^{N\times d}$ and $Y_e^t \in \mathbb{R}^{M\times d}$ to denote the *sequence representations* on nodes and edges at time step t, respectively. For simplicity, we assume all the representations share the same dimension size d. With the above notations, we formally define NoCES as follows.

Definition 1. *Network of Co-evolving Sequences (NoCES)*
A network of co-evolving sequences is defined as the triple $\langle S, G, \pi\rangle$, where $S = S_v \cup S_e$ is a set containing $(N + M)$ sequences for each node/edge, $G = (V, E)$ is the underlying network of these sequences, and π is a bijection function that maps each sequence to a corresponding node/edge in the network.

To evaluate the modeling power of different approaches, we evaluate two widely-studied tasks, i.e., missing value recovery and future value prediction. For convenience, we rewrite the sequence set S as a matrix $X \in \mathbb{R}^{(N+M)\times T}$ containing column vectors, i.e., $X = [x^1, x^2, ..., x^T]$ where x^t is the column vector containing all the node/edge sequence values at time step t. In the following, we interchangeably use S and X to indicate the sequences. In practice, there could be many missing values in the sequences. We use an indicator matrix $U \in \mathbb{R}^{(N+M)\times T}$ to indicate whether the values in X are observed or missing (i.e., $U_{ij} = 1$ means that the corresponding value is missing). Then, the above two tasks are defined as follows.

Problem 1. *Missing Value Recovery Problem in NoCES*
Given: *a network of co-evolving sequences $\langle S, G, \pi\rangle$, and the indicator matrix U for missing values;*
Find: *the missing values in S indicated by U.*

Problem 2. *Future Value Prediction Problem in NoCES*
Given: *a network of co-evolving sequences $\langle S, G, \pi\rangle$;*
Find: *the future sequence values $[x^{T+1}, x^{T+2}, ..., x^{T+T_p}]$ in T_p future steps.*

3 THE PROPOSED APPROACH

3.1 Overview

The overview of the SeeS framework is depicted in Fig. 2, where we show the process of one time step for a given target node for brevity. To capture the interplay between network structure and node sequences, we alternatively feed network structure and sequences into the modeling of each other at each time step. To capture the dynamics, we repeat the alternative modeling over each time step.

More specifically, at time step t, we first learn the network representations of both nodes and edges based on the network structure and the current sequence representations (i.e., *Structure Encoder*). Then, we update the sequence representations using x^t (i.e., *Sequence Encoder*), and aggregate the sequences of neighboring

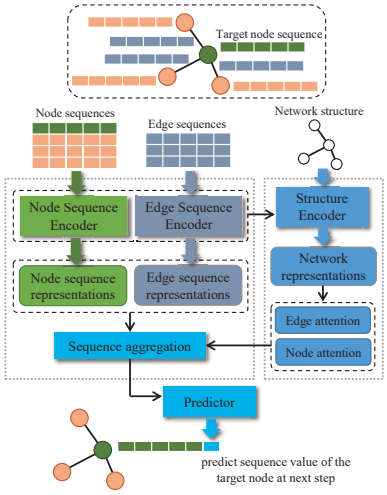


Figure 2: The learning framework of SEES at each time step.

nodes/edges for the target node (i.e., *Sequence Aggregation*). During the aggregation, we use the learned node/edge attention from network representations to capture the correlations between sequences. For training purpose, the aggregated sequence representation is used to predict the future value of the target node at time step $t+1$.

3.2 Structure Encoder

We next explain SEES in details by starting with the structure encoder. In the following, we ignore the time step in the superscript when it is clear from the context. Here, the current sequence representations are treated as node/edge attributes. Consequently, evolving sequences indicate changing attributes, and thus we need to update the node/edge representations accordingly. To simultaneously learn the representations of both nodes and edges, we propose a network embedding method named *Dual-GCN*, i.e.,

$$(Z_v^{(L)}, Z_e^{(L)}) = \textit{Dual-GCN}(A, Z_v^{(0)}, Z_e^{(0)}), \tag{1}$$

where A is the adjacency matrix of the network, $Z_v^{(0)} = Y_v^t$ and $Z_e^{(0)} = Y_e^t$ are the current sequence representations of nodes and edges, and L is the number of layers in Dual-GCN.

Dual-GCN is built upon GCN [19], which is a powerful neural network architecture for learning network's node representations. We extends GCN to further learn edge representations. Before presenting the details of Dual-GCN, we first define several matrices. In order to exchange information between edges and nodes, we define two incidence matrices $P \in \mathbb{R}^{N\times M}$ and $Q \in \mathbb{R}^{M\times N}$ as follows,

$$\begin{aligned} P_{ik} &= \begin{cases} 1, & \exists j \in V, k = (i,j) \in E \\ 0, & o.w. \end{cases}, \\ Q_{kj} &= \begin{cases} 1, & \exists i \in V, k = (i,j) \in E \\ 0, & o.w. \end{cases}, \end{aligned} \tag{2}$$

where $i, j \in V$ are nodes, and $k \in E$ is an edge. Note that our method applies to both directed and undirected networks; when the network is undirected, we have $P = Q'$ where Q' stands for the transpose of Q. Based on P and Q, we further define an adjacency matrix $\Delta \in \mathbb{R}^{M\times M}$ for edges as,

$$\Delta_{kf} = \begin{cases} 1, & \exists i \in V, Q_{ki} = 1, P_{if} = 1 \\ 0, & o.w. \end{cases}, \tag{3}$$

where $k, f \in E$ are edges and $i \in V$ is a node.

With the above matrices, we define the following equations for each Dual-GCN layer to simultaneously learn the representations of both nodes and edges,

$$\begin{aligned} Z_v^{(l+1)} &= \hat{D}_v^{-\frac{1}{2}} \hat{A} \hat{D}_v^{-\frac{1}{2}} (Z_v^{(l)} W_{v,v}^{(l)} + P Z_e^{(l)} W_{v,e}^{(l)}), \\ Z_e^{(l+1)} &= \hat{D}_e^{-\frac{1}{2}} \hat{\Delta} \hat{D}_e^{-\frac{1}{2}} (Z_e^{(l)} W_{e,e}^{(l)} + Q Z_v^{(l)} W_{e,v}^{(l)}), \end{aligned} \tag{4}$$

where $W_{(\cdot)}^{(l)} \in \mathbb{R}^{d\times d}$ stands for the parameters. For A matrix, $\hat{A} = A + I$ where I is the identity matrix, and $\hat{D}_v$ is the diagonal node degree matrix of $\hat{A}$. The corresponding $\hat{\Delta}$ matrix is similarly defined.

In Eq. (4), Dual-GCN considers two types of aggregations, i.e., *self-aggregation* and *cross-aggregation* to aggregate the neighborhood information. Take the node representations $Z_v^{(l+1)}$ as an example. For self-aggregation, we aggregate the information from the direct neighbors for each node (as indicated by the $\hat{D}_v^{-\frac{1}{2}} \hat{A} \hat{D}_v^{-\frac{1}{2}} Z_v^{(l)}$ term). For cross-aggregation, we aggregate the information from connected edges for each node (as indicated by the $\hat{D}_v^{-\frac{1}{2}} \hat{A} \hat{D}_v^{-\frac{1}{2}} P Z_e^{(l)}$ term). Here, the P matrix helps transfer the edge representations for learning node representations. Note that we use the simplified GCN [40] in Eq. (4) as it is observed to have little negative impact on accuracy while runs much faster. Moreover, we can sample a subset of entries in the P, Q, Δ matrices to save time especially when these matrices are dense.

Attention Computation. With the learned node/edge representations, we calculate the attention between these sequences as,

$$\begin{aligned} \eta_{ij} &= ReLU([Z_i^{(L)}, Z_j^{(L)}] W_a), \\ \alpha_{ij} &= \text{softmax}(\eta_{ij}) = \frac{\exp(\eta_{ij})}{\sum_{k\in C_i} \exp(\eta_{ik})}, \end{aligned} \tag{5}$$

where $W_a \in \mathrm{R}^{2d\times 1}$ is the parameter, $ReLU(\cdot)$ is the activation function, and C_i indicates the set of i's neighbors. C_i contains both nodes and edges that directly connect to node i, and we can also randomly sample a subset of C_i for efficiency concerns.

3.3 Sequence Encoder

For sequence encoder, we separately update each sequence representation based on the newly arrived sequence data. A straightforward way is to apply a recurrent neural network structure for both node sequences and edge sequences as

$$H^{t+1} = tanh(Y^t W_h + x^{t+1} W_x), \tag{6}$$

where H^{t+1} stands for the updated sequence representations, Y^t is the current sequence representations, x^t contains the sequence data newly arrived at time t, and W_h and W_x are parameters. However, such an RNN structure easily forgets long-term dependencies, making it less effective in terms of memorizing the tendency of the sequences. Therefore, we adopt the self-attention mechanism [38] to encode the sequences when the sequence tendency matters.

More specifically, we first expand the dimension of the newly arrived data, compute the self-attention, and obtain the updated

sequence representations as follows,

$$\begin{aligned} E^{t+1} &= x^t W_s + b_s, \\ Q,\ K,\ V &= Y^t W_q,\ Y^t W_k,\ E^{t+1} W_v, \\ H^{t+1} &= \text{softmax}(QK')V, \end{aligned} \tag{7}$$

where $W_s \in \mathbb{R}^{1\times d}, b_s \in \mathbb{R}^{(N+M)\times d}$, $W_q, W_k, W_v \in \mathbb{R}^{d\times d}$ are parameters, and $H^{t+1} \in \mathbb{R}^{(N+M)\times d}$ contains the updated sequence representations. In practice, we further feed H^{t+1} into two fully connected layers following the Transformer structure.

3.4 Sequence Aggregation

To feed network into sequences, we next aggregate the sequence representations based on the learned network representations. In the following, we take the updating of node i as an example, and the updating of edges can be analogously obtained.

Using the correlations/attention α_i (computed as Eq. (5)) between node i and its related nodes/edges, we first aggregate the neighboring sequence representations for node i as,

$$F_{v,i}^{t+1} = \alpha_i [H_v^{t+1}; H_e^{t+1}], \tag{8}$$

where H_v^{t+1} and H_e^{t+1} are the output from the sequence encoder for nodes and edges, respectively, and $F_{v,i}^{t+1} \in \mathbb{R}^{1\times d}$ is the aggregated results from i's neighbors. Here, $F_{v,i}^{t+1}$ stands for the overall influence of i's neighboring nodes and edges.

Then, we combine $F_{v,i}^{t+1}$ and the sequence representation $H_{v,i}^{t+1}$ for node i to obtain the aggregated sequence representation,

$$Y_{v,i}^{t+1} = [F_{v,i}^{t+1}, H_{v,i}^{t+1}] W_y + b_y, \tag{9}$$

where $W_y \in \mathbb{R}^{2d\times d}$ and $b_y \in \mathbb{R}^{1\times d}$ are parameters. The aggregated sequence representations Y^{t+1} will be fed into the next time step, and serve to predict the next sequence value.

3.5 Training

Next, we describe how to train SeeS, which is non-trivial due to the bidirectional modeling of network structure and sequences. We still take time step t as an example and present the training objective function as follows,

$$\begin{aligned} &\arg\min \delta(x^{t+1}, \hat{x}^{t+1}), \\ &s.t.\ \hat{x}_v^{t+1} = Y_v^{t+1} \cdot W_{p,v} + b_{p,v}, \\ &\quad \hat{x}_e^{t+1} = Y_e^{t+1} \cdot W_{p,e} + b_{p,e}, \\ &\quad (Y_v^{t+1}, Y_e^{t+1}) = \textit{Seq-Updater}(x^t, Y_v^t, Y_e^t, \alpha), \\ &\quad \alpha = \textit{Attention}(Z_v^t, Z_e^t), \\ &\quad (Z_v^t, Z_e^t) = \textit{Dual-GCN}(A, Y_v^t, Y_e^t), \end{aligned} \tag{10}$$

where $\delta(\cdot)$ is the loss function decided by the specific prediction tasks. In this work, we use mean absolute error for $\delta(\cdot)$ as our sequence values are numerical. *Attention* stands for the attention computation in Eq. (5), and *Seq-Updater* stands for the sequence encoder and the sequence aggregation. $W_{(\cdot)}$ and $b_{(\cdot)}$ are the parameters.

We can either train the above loss on time t or over each time step before $t+1$ when predicting the values x^{t+1}. However, in practice, it would be less effective if we train only on time t, and time-consuming if we train over all the time steps especially when the sequence is long. As a result, we add a sliding-window of size c, and train only c steps before t to make the predictions.

Algorithm 1 The SeeS Algorithm.

Input: The adjacency matrix A of the network, the sequences X, the window size c, the Dual-GCN layer number L
Output: The sequence representations Y_v^T and Y_e^T

```
1:  Initialize Y_v^0 and Y_e^0;
2:  Compute P, Q, Δ according to Eq. (2) - (3);
3:  for iter = 1 : r do
4:    for t = 1 : T − 1 do
5:      for τ = t − c : t do
6:        Z_v^(0) ← Y_v^τ, Z_e^(0) ← Y_e^τ;
7:        (Z_v^(L), Z_e^(L)) = Dual-GCN(A, Z_v^(0), Z_e^(0));
8:        Calculate α by Eq. (5);
9:        Calculate H^{τ+1} by Eq. (7);
10:       F^{τ+1} = α[H_v^{τ+1}; H_e^{τ+1}];
11:       Update Y^{τ+1} by Eq. (9);
12:     end for
13:     x̂_v^{t+1} = Y_v^{t+1} · W_{p,v} + b_{p,v};
14:     x̂_e^{t+1} = Y_e^{t+1} · W_{p,e} + b_{p,e};
15:     Update model parameters by Eq. (10);
16:   end for
17:   if The parameters converge then
18:     Break;
19:   end if
20: end for
21: return Y_v^T and Y_e^T as well as other model parameters;
```

3.6 Algorithm and Analysis

Alg. 1 summarizes the proposed SeeS algorithm. After initializations (Lines 1-2), Line 3 starts the iterations where r is the maximum iteration number. Line 4 iterates over each time step, and Line 5 iterates over the time window with size c. Lines 6-8 learn the node/edge representations and compute the attention, and Lines 9-11 update and aggregate the sequence representations. Lines 13-15 predict the next sequence values, and update the model parameters accordingly.

The time complexity of the proposed SeeS is $O((N+M)d^3rTc)$, where N is the node number, M is the edge number, d is the representation dimension, r is a maximum iteration number, T is the sequence length, and c is the training window size. By assuming r, c, and d to be small constants, the time complexity can be rewritten as $O((N+M)T)$, meaning that SeeS scales linear w.r.t. the sequence length T and the network size $M+N$.

4 EXPERIMENTAL EVALUATIONS

4.1 Datasets and Scenarios

The proposed SeeS can be used in a variety of NoCES applications. In this work, we evaluate it in the following four datasets/scenarios. Table 1 summarizes the statistics of the datasets. The sequence length varies in the Motion dataset as there are different body motions.[1]

[1]The code of the proposed algorithms is publicly available at: https://github.com/SoftWiser-group/SeeS

Company Quarterly Revenue (Revenue). This dataset records the quarterly revenues of 170 companies from 2006 to 2017. The network is defined by the co-search relationships, which is built from SEC[2] fillings. That is, we extract the co-search pairs from the web logs of searches on the SEC's EDGAR website following Lee et al. [21]. For example, if a user searches Intel and Microsoft in the same session, we consider these two companies having a co-search relationship. In summary, we found 1379 co-search relationships among the 170 companies. Then, the adjacency matrix of the co-search relationships is defined as follows,

$$A_{ij} = \frac{CoSearch(i,j)}{\sum_{i'}\sum_{j'} CoSearch(i',j')},$$

where $CoSearch(i,j)$ denotes the number of co-searches between companies i and j in the web logs.

Body Motion Capture (Motion). This dataset contains a set of sensor reading sequences recorded from several body motions. To capture the body motions, a skeleton hierarchy of sensors is defined to describe the body. The skeleton defines a 'root' segment at first, and each segment is defined under the 'root' segment as a tree. For example, 'root' is the father of 'hips', and 'hips' is the father of 'hips1', 'hips2', and 'hips3'. Then the adjacency matrix of this dataset is defined by the skeleton hierarchy as follows,

$$A_{ij} = \begin{cases} \frac{1}{l_{ij}}, \ Father(j) = i, \\ 0, \ o.w. \end{cases},$$

where l_{ij} is the distance between the father segment i and son segment j in three-dimensional Euclidean space, and $Father(j)$ denotes the father of j in the skeleton hierarchy. Each segment records three-dimensional tensors (i.e., $[x, y, z]$) as sequences over time.

Traffic Speed (Traffic). This dataset records the real-time traffic speed on the roads in five boroughs of the New York City. Each road has some sensors recording the average speed of passing vehicles in every five minutes. We construct the network relying on the map.[3] That is, each road is marked as an edge on the map by its sensors' coordinates. Then, we add nodes at the two terminals of each road. Note that the constructed network is a directed graph, according to the vehicles' driving directions. The first two datasets contain only node sequences, and this dataset contains only edge sequences.

City Bike Trip (Bike). This dataset records the start and end stations of each city bike ride in the New York City. We extract the data of 204 bike stations in May 2019, and treat each day as a time step. For the network, we use bike stations as nodes, and the bike rides between them as edges. This network has both node sequences and edge sequences: node sequences are the number of incoming bikes minus the number of outgoing bikes in each day, and edge sequences are the normalized number of trips between the stations in each day.

4.2 Experimental Setup

Comparison Methods. We compare our proposed approach SeeS with the following methods:

[2]U.S. Securities and Exchange Commission
[3]http://nyctmc.org/

Table 1: Statistics of the datasets.

Dataset	sequence length	# nodes	# edges
Revenue	52	170	1379
Motion	299−1472	29	28
Traffic	3000	113	83
Bike	31	204	25016

- *m-LSTM, m-RNN.* These two competitors treat the sequences in NoCES as a multivariate sequence and model them with LSTM and RNN, respectively.
- *PMF* [30]. PMF is a matrix factorization algorithm for recommender systems. We apply it on the X matrix to recover missing values.
- *SoRec* [26]. SoRec is a social collaborative filtering algorithm for recommender systems. Compared to PMF, it further models the network structure.
- *DynaMMo* [23]. DynaMMo is proposed to recover the missing values in co-evolving sequences. It applies Kalman filter and smoother to adjust the recovered values.
- *DCMF* [2]. This is a dynamic contextual matrix factorization algorithm for a network of co-evolving sequences. Compared to DynaMMo, it further considers network structure.
- *MTGNN* [42]. MTGNN learns the network structure for multivariate time series forecasting when the explicit network structure is unavailable.
- *StemGNN* [4]. StemGNN is also a multivariate time series forecasting method that captures the interplay between sequences and temporal dependencies in the spectral domain.
- *DCRNN* [24]. DCRNN encodes the diffusion process into a convolutional recurrent neural network. It is designed to predict traffic transportation flows.
- *Graph-WaveNet* [43]. Graph-WaveNet is also designed for traffic prediction. It learns an adaptive adjacency matrix and encodes the diffusion process by a convolution layer.

Evaluation Protocol. We formulate two prediction tasks. The first one is missing value recovery. In the experiments, we randomly hide some values in the sequences as missing values and use the rest to recover. For SeeS, m-LSTM, and m-RNN, we use the existing sequences before t to recover the missing values at time step t. For the other methods, we use all the observed values. In this task, we compare SeeS with all the competitors listed above.

The second task is future value prediction. For this task, we hide all the sequence values after a specific time step t and then predict the future sequences in T_p steps. Intuitively, this task is more difficult as the errors may accumulate over time. In the competitors, PMF, SoRec, DynaMMo, and DCMF are not designed for predicting the future values. For DCMF, since it is built upon the linear dynamic system, we adapt it for this task. Therefore, we compare SeeS with m-RNN, m-LSTM, DCMF, MTGNN, StemGNN, DCRNN and Graph-WaveNet in this task. For evaluation metrics of both tasks, we adopt the root mean squared error (RMSE) between the predicted values and the real values.

Parameters and Implementations. For the proposed SeeS, we search the parameters d and c using cross validation. We search d within $[8, 16, 32, 64]$ and search c within $[4-8]$. The resulting

Table 2: Effectiveness results of missing value recovery. SeeS generally outperforms the competitors in all the datasets. Graph-WaveNet is better than SeeS on the Traffic data since it is specially designed for traffic prediction.

	Revenue	Motion					Traffic	Bike	
		Wave Hello	Jump	Boxing	Football Throw	Mawashi Geri		Edge Seq.	Node Seq.
T	52	299	547	773	1091	1472	3000	31	31
m-RNN	0.6563	5.56	11.5	5.44	13.2	13.7	19.68	2.68	7.919
m-LSTM	0.6347	8.62	12.7	4.73	11.1	13.6	18.01	2.68	8.027
PMF	4.119	12.2	18.1	17.4	31.0	26.2	42.03	1.76	8.810
SoRec	6.840	16.7	48.3	16.8	36.6	34.8	15.31	8.77	8.485
DynaMMo	0.2639	19.7	26.9	21.9	40.9	33.3	22.62	0.609	7.447
DCMF	0.2613	6.49	13.1	12.1	31.8	22.1	17.96	–	8.306
MTGNN	0.2219	3.15	4.92	15.2	4.60	9.78	7.290	–	8.564
StemGNN	0.3127	13.6	10.1	22.9	3.03	6.93	11.04	–	8.067
DCRNN	0.3373	1.49	2.18	8.15	0.740	1.10	19.81	–	7.653
Graph-WaveNet	0.2345	0.598	1.12	6.32	0.741	0.718	**3.286**	–	7.946
SeeS	**0.1337**	**0.582**	**0.796**	**3.07**	**0.668**	**0.526**	6.656	**0.601**	**7.249**

parameters are as follows. For the revenue data, we set $d = 8, c = 5$; for the motion data, we set $d = 16, c = 5$ for all the motions for simplicity; for the traffic data, we set $d = 16, c = 5$; for the bike data, we set $d = 16, c = 5$. The number of Transformer layers in SeeS is set to 2; we also use multi-head attention and set the head number to 2. Following GCN, we set $L = 2$ in our Dual-GCN. For the maximum iteration number in Alg. 1, we set $r = 100$. For the bike data, since its network is dense, we constrain the context size C_i in the Δ matrix to be 10. For the compared methods DynaMMo, DCMF, MTGNN, StemGNN, DCRNN, and Graph-WaveNet, we directly use the code provided by the authors and use their default parameter settings. For SoRec and PMF, we use the publicly available library (i.e., LibRec), and follow the default setting given by the library. For m-LSTM and m-RNN, we implement them ourselves using the same parameter setting with SeeS including the window size c and representation size d. We also use early stop to prevent the models from over-fitting.

The experiments on Revenue, Motion, and Traffic data run on the same machine with 64G memory, 6 CPU cores (at 3.4GHz using 12 threads), and an Nvidia GeForce RTX 1080 graphics card. For the Bike dataset, we run the experiments on a server with 256G memory, 48 CPU cores (at 2.3GHz) since DynaMMo and DCMF easily run out of memory on this dataset using the 64G memory machine.

4.3 Effectiveness Results

(A) Missing Value Recovery. We first present the effectiveness results in the missing value recovery task. For all the datasets, we randomly hide 10% of the values in sequences as test set. For the Motion data, we select five body motions including waving hello, jump, boxing, catch and football throw, and mawashi geri in karate. Existing competitors consider only node sequences. For the Traffic data and Bike data where the sequences exist on edges, we apply DCMF on the edge adjacency matrix Δ. For the Bike data, we run the compared methods separately on node sequences and edge sequences. The RMSE results of recovering the hidden values in all the datasets are shown in Table 2, where '–' indicates the corresponding method is computationally prohibitive (i.e., cannot return results within 48 hours or run out of memory) on the corresponding dataset.

We can observe from the table that the proposed SeeS generally outperforms all the competitors in all the datasets. For example, on the Revenue data, SeeS achieves 39.7% improvement over its best competitor (i.e., MTGNN). PMF and SoRec perform relatively poor in this dataset, indicating the importance of temporal smoothness for revenue prediction. On the five body motions, Graph-WaveNet performs relatively well among the competitors, and SeeS improves it by 2.6%, 28.9%, 51.4%, 9.7%, and 26.7% for the five body motions, respectively. On the Traffic data, Graph-WaveNet achieves the best result among all the methods. This is due to the fact that Graph-WaveNet is specially designed for traffic prediction. However, in other NoCES scenarios, SeeS is much better than Graph-WaveNet. Except for Graph-WaveNet, SeeS still performs best among the other compared methods. On the Bike data, the relative improvement over the best competitor (i.e., DynaMMo) is 1.3% and 2.7% for edge sequences and node sequences, respectively. SeeS's improvement in this dataset is not as significant as that in the other datasets. The possible reason is that the network is very dense and we randomly sample only 10 neighbors for each node/edge. In other words, we trade effectiveness for efficiency (note that DCMF, MTGNN, StemGNN, DCRNN, and Graph-WaveNet are computationally prohibitive on the edge sequences).

Overall, for the competitors, m-LSTM and m-RNN do not consider the network structure; PMF and SoRec do not consider the temporal order of sequences. MTGNN and StemGNN learn the network structure from sequences, and perform less effective in NoCES when the underlying network is available. Although DynaMMo, DCMF, DCRNN, and Graph-WaveNet consider both network structure and sequences, they either treat the network in a static way or are specially designed for traffic prediction only. Therefore, the above result indicates the effectiveness of the bidirectional and dynamic modeling between network structure and sequences for the missing value recovery task in NoCES.

(B) Future Value Prediction. Next, we present the results of the future value prediction task. As mentioned in Section 2, we need to predict T_p future steps based on the current sequences. We cut the NoCES data into small time slices with size $t + T_p$. The goal is to predict the sequence values from $t + 1$ to $t + T_p$ in each slice. We use

Table 3: Effectiveness results of future value prediction. SeeS generally outperforms all the competitors. Graph-WaveNet is better than SeeS on the Traffic data since it is specially designed for traffic prediction.

	Revenue						Traffic					
T_p	1	2	3	4	5	Average	1	2	3	4	5	Average
m-RNN	0.557	0.338	0.390	0.386	0.667	0.468	22.15	21.84	22.74	24.11	25.45	23.25
m-LSTM	0.416	0.252	0.267	0.321	0.857	0.423	19.94	18.88	20.37	22.75	24.43	21.27
DCMF	0.167	0.211	0.241	0.234	0.479	0.266	17.04	16.78	19.22	18.83	21.56	18.68
MTGNN	0.221	0.415	0.380	0.371	0.368	0.335	7.71	7.78	14.25	14.25	14.24	11.64
StemGNN	0.386	0.393	0.398	0.405	0.389	0.394	12.8	15.1	17.5	18.9	20.5	17.0
DCRNN	0.337	0.324	0.341	0.364	0.449	0.363	19.81	19.83	19.93	20.01	20.07	19.93
Graph-WaveNet	0.234	0.273	0.318	0.346	0.342	0.302	3.28	4.23	4.81	5.29	5.66	**4.65**
SeeS	0.137	0.164	0.187	0.200	0.195	**0.176**	6.61	7.56	8.23	8.73	9.05	8.04
	Jump Motion						Mawashi Geri Motion					
T_p	1	2	3	4	5	Average	1	2	3	4	5	Average
m-RNN	9.91	10.31	12.31	11.27	16.78	12.12	14.91	16.45	16.68	15.46	17.93	16.28
m-LSTM	12.27	13.56	14.21	14.29	15.95	14.06	16.81	19.32	21.96	26.07	22.87	21.41
DCMF	10.78	10.44	11.20	12.67	15.89	12.19	11.90	16.59	17.96	24.34	24.13	18.98
MTGNN	5.47	23.62	23.67	23.73	23.78	20.05	9.59	49.33	36.65	34.91	41.25	34.35
StemGNN	9.58	9.61	9.78	9.91	10.3	9.83	5.29	5.40	5.60	6.24	6.45	5.79
DCRNN	2.18	3.09	4.10	5.18	6.26	4.16	1.10	1.44	1.86	2.31	2.77	1.90
Graph-WaveNet	1.12	1.01	1.40	1.90	2.51	1.59	0.71	0.67	1.02	1.39	1.76	1.11
SeeS	0.83	0.96	1.12	1.31	1.55	**1.15**	0.66	0.80	0.97	1.18	1.40	**1.00**

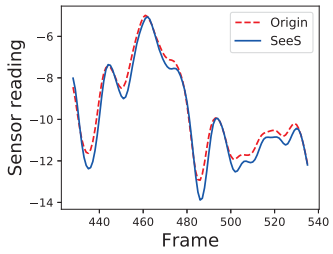
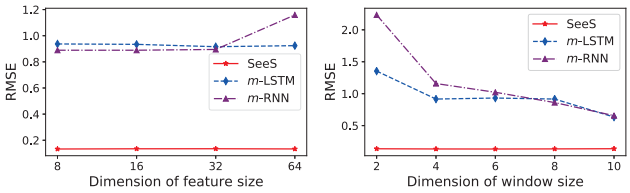


Figure 3: The prediction results of jump motion.



(a) Representation dimension d. (b) Training window size c.

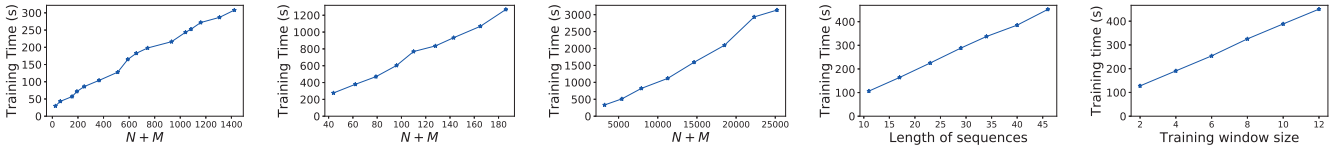
Figure 4: Parameter sensitivity results. SeeS is robust w.r.t. the two parameters in a relatively wide range.

the first 70% time slices for training, the middle 10% for validating and the last 20% for testing. In this experiment, we set T_p to 5 and compute the RMSE results on the Revenue data, Traffic data, and two Motion data as shown in Table 3. Similar results are observed on the other datasets and thus are omitted for brevity.

We can first observe from Table 3 that, SeeS generally outperforms all the competitors in terms of accurately predicting the future sequence values. For example, averaging over the five future time steps, SeeS improves its best competitor (DCMF) by 33.9% on the Revenue data; on Jump Motion and Mawashi Geri Motion, the relative improvements over the best competitor (Graph-WaveNet) are 27.6% and 9.9%, respectively. On the traffic data, similar to the results in Table 2, Graph-WaveNet performs the best among all the methods as it is specially designed for traffic prediction. However, SeeS achieves 30.9% improvement over the second best competitor (MTGNN). Second, as mentioned above, the prediction errors may accumulate over time. We do observe such trends for all the methods. For example, the RMSE sharply increases from $T_p = 2$ for MTGNN on the motion data, which is probably because MTGNN learns the network structure in each step and leads to severe error accumulations. However, we also observe that the improvements of SeeS become larger when T_p becomes larger. For example, on the two motions, the relative improvements over the best competitors are 25.8%, and 7.0% when $T_p = 1$, and become 38.2% and 20.5% when $T_p = 5$.

For better visualization of the results, we also present the prediction results of one specific body skeleton in the jump motion in Fig. 3, where the x-axis is the frame number of the last 20% test set, and y-axis is one of the corresponding sensor readings. Each frame is predicted with the previous t frames as input. Fig. 3 shows that the predicted frames suffer little loss compared to the original frames.

(C) Parameter Sensitivity. Next, we evaluate the sensitivity of the two parameters in SeeS, i.e., the dimension d of network/sequence representations and the training window size c. We take the Revenue dataset as an example, and the results are shown in Fig. 4. We also plot the results of two competitors (m-LSTM and m-RNN) in the figure for comparison. As we can see, for both parameters, there is a relatively wide range where SeeS performs relatively stable and significantly outperforms the competitors.



(a) Scalability against network size $N + M$ on Revenue data.

(b) Scalability against network size $N + M$ on Traffic data.

(c) Scalability against network size $N + M$ on Bike data.

(d) Scalability against sequence length T on Revenue data.

(e) Scalability against training window size c on Revenue data.

Figure 5: Scalability results. SeeS scales linearly w.r.t. the sequence length T, the window size c, and the network size $N + M$.

4.4 Efficiency Results

Finally, we evaluate the scalability of the proposed SeeS. First, we present the wall-clock time of SeeS against the network size $N + M$. Here, we randomly select subsets of the network, and apply SeeS on the resulting datasets. We fix the other parameters (e.g., sequence length and training window size) in this experiment, and the results on the Revenue, Traffic, and Bike datasets are shown in Fig. 5(a), Fig. 5(b), and Fig. 5(c), respectively. As we can see, the proposed SeeS scales linearly w.r.t. the network size. Next, we test the scalability against the sequence length T and the window size c. We report the results in Fig. 5(d) and Fig. 5(e), the proposed SeeS also scales linearly w.r.t. both parameters.

5 RELATED WORK

In this section, we briefly review the related work.

Time Series Modeling. The first branch of related work is from time series modeling [17]. Typical examples include time series representation [35], classification [12], group detection [6], outlier detection [50], missing value recovery [41], etc. For example, Contreras et al. [8] predict the next-day electricity prices by ARIMA. Raza et al. [34] propose a derivative-based linear method to predict the trend of data measured by sensors. Kim et al. [18] propose a decision tree framework for spatio-temporal sequence prediction.However, these methods are suboptimal for modeling NoCES as they ignore the network structure/correlations between multiple sequences.

Multivariate Time Series Modeling. To better model the correlations between sequences, multivariate time series recovering and forecasting have been widely studied [4, 9, 23, 29, 42]. For example, Li et al. [23] implicitly consider the network structure of multiple sequences by using their correlations, and propose to recover the missing values based on the correlations. Matsubara et al. [29] also capture the implicit connections between sequences, and propose to model the non-linear evolution of sequences. Wu et al. [42] propose to learn the correlations between sequences and make use of the learned correlations in the sequence prediction. Different from these proposals, we consider the NoCES problem setting where there exists explicit network structure underlying the sequences.

Traffic Prediction. Another branch of related work is from traffic prediction [14, 16, 24, 25, 43, 45, 48]. Existing methods are mainly built upon the localness and diffusion nature of traffic. For example, Li et al. [24] model the diffusion phenomenon of traffic flows on the network and incorporate it into sequence modeling. Yao et al. [45] use a local CNN to handle spatial dependencies between traffic flows. Since they are specially designed for traffic data, these methods could be less effective for other NoCES domains when the localness and diffusion nature does not hold.

NoCES Modeling. We are not the first to study the general NoCES problem. In literature, Cai et al. explicitly incorporate the network structure to smooth the sequences [2], and further extend their model to handle high-order time series data [3]. Different from the above work which feeds network features into sequences, we further model the bidirectional and dynamical effect between network structure and sequences.

Static Network Embedding. Our work is also related to network embedding. Network embedding methods typically take the network structure as input and output the representations for each node. In literature, several researchers propose to apply the skip-gram model to learn the node representations [13, 32, 36]. Some others treat the problem as a matrix factorization problem and propose to factorize the adjacency matrix to obtain the network representations [5, 22, 33, 44]. Recently, graph neural networks [7, 10, 15, 19, 39, 40, 47] have received increasing attention. In this work, we adapt and extend the simplified GCN model [40] for simultaneously learning the representations of both nodes and edges in SeeS.

Dynamic Network Embedding. Another line of related work is on embedding dynamic networks [11, 20, 27, 37, 49, 51]. These methods learn the network embeddings while the network keeps evolving. For example, Du et al. [11] update the a subset of influenced nodes for a structure change in the network; Ma et al. [27] encode the high-order proximity as latent states of nodes so as to learn the embeddings for new nodes. These existing work is orthogonal to our work. That is, they are handling the case when the network structure evolves over time; in contrast, the network structure in NoCES is assumed to be static, while the nodes and edges have co-evolving sequences (i.e., dynamic attributes).

6 CONCLUSIONS

In this paper, we propose a joint learning framework for network of co-evolving sequences. The proposed framework considers the bidirectional dynamics and interplay between network structure and sequences, and also the mutual influence between sequences on nodes and edges. We evaluate the proposed approach in two prediction tasks, i.e., missing value recovery and future value prediction. The results on four different scenarios show that the proposed approach outperforms its competitors in terms of prediction accuracy.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China (No. 61690204) and the Collaborative Innovation Center of Novel Software Technology and Industrialization. Hanghang Tong is partially supported by NSF (1947135, 2134079 and 1939725). Yuan Yao is the corresponding author.

REFERENCES

[1] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. 2016. Social lstm: Human trajectory prediction in crowded spaces. In *CVPR*. 961–971.

[2] Yongjie Cai, Hanghang Tong, Wei Fan, and Ping Ji. 2015. Fast mining of a network of coevolving time series. In *SDM*. 298–306.

[3] Yongjie Cai, Hanghang Tong, Wei Fan, Ping Ji, and Qing He. 2015. Facets: Fast comprehensive mining of coevolving high-order time series. In *KDD*. 79–88.

[4] Defu Cao, Yujing Wang, Juanyong Duan, Ce Zhang, Xia Zhu, Conguri Huang, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, et al. 2020. Spectral temporal graph neural network for multivariate time-series forecasting. In *NeurIPS*.

[5] Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *CIKM*. ACM, 891–900.

[6] Georgios Chatzigeorgakidis, Dimitrios Skoutas, Kostas Patroumpas, Themis Palpanas, Spiros Athanasiou, and Spiros Skiadopoulos. 2019. Local pair and bundle discovery over co-evolving time series. In *SSTD*. 160–169.

[7] Yu Chen, Lingfei Wu, and Mohammed J. Zaki. 2020. Iterative Deep Graph Learning for Graph Neural Networks: Better and Robust Node Embeddings. In *NeurIPS*.

[8] Javier Contreras, Rosario Espinola, Francisco J Nogales, and Antonio J Conejo. 2003. ARIMA models to predict next-day electricity prices. *IEEE transactions on power systems* 18, 3 (2003), 1014–1020.

[9] Emmanuel de Bézenac, Syama Sundar Rangapuram, Konstantinos Benidis, Michael Bohlke-Schneider, Richard Kurle, Lorenzo Stella, Hilaf Hasson, Patrick Gallinari, and Tim Januschowski. 2020. Normalizing Kalman Filters for Multivariate Time Series Analysis. In *NeurIPS*.

[10] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. 2020. GraphZoom: A Multi-level Spectral Approach for Accurate and Scalable Graph Embedding. In *ICLR*.

[11] Lun Du, Yun Wang, Guojie Song, Zhicong Lu, and Junshan Wang. 2018. Dynamic Network Embedding: An Extended Approach for Skip-gram based Network Embedding. In *IJCAI*. 2086–2092.

[12] Jing Gao, Bolin Ding, Wei Fan, Jiawei Han, and S Yu Philip. 2008. Classifying Data Streams with Skewed Class Distributions and Concept Drifts. *IEEE Internet Computing* 12, 6 (2008), 37–49.

[13] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *KDD*. ACM, 855–864.

[14] Shengnan Guo, Youfang Lin, Ning Feng, Chao Song, and Huaiyu Wan. 2019. Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *AAAI*, Vol. 33. 922–929.

[15] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1024–1034.

[16] Rongzhou Huang, Chuyin Huang, Yubao Liu, Genan Dai, and Weiyang Kong. 2020. LSGCN: Long Short-Term Traffic Prediction with Graph Convolutional Networks. In *IJCAI*. 2355–2361.

[17] Eamonn Keogh. 2011. Tutorial: Machine learning in time series databases (and everything is a time series!). AAAI.

[18] Taehwan Kim, Yisong Yue, Sarah Taylor, and Iain Matthews. 2015. A decision tree framework for spatiotemporal sequence prediction. In *KDD*. ACM, 577–586.

[19] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

[20] Srijan Kumar, Xikun Zhang, and Jure Leskovec. 2019. Predicting dynamic embedding trajectory in temporal interaction networks. In *KDD*. 1269–1278.

[21] Charles MC Lee, Paul Ma, and Charles CY Wang. 2015. Search-based peer firms: Aggregating investor perceptions through internet co-searches. *Journal of Financial Economics* 116, 2 (2015), 410–431.

[22] Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. In *NIPS*. 2177–2185.

[23] Lei Li, James McCann, Nancy S Pollard, and Christos Faloutsos. 2009. Dynammo: Mining and summarization of coevolving sequences with missing values. In *KDD*. ACM, 507–516.

[24] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. 2018. Diffusion Convolutional Recurrent Neural Network: Data-Driven Traffic Forecasting. In *ICLR*.

[25] Bin Lu, Xiaoying Gan, Haiming Jin, Luoyi Fu, and Haisong Zhang. 2020. Spatiotemporal adaptive gated graph convolution network for urban traffic flow forecasting. In *CIKM*. 1025–1034.

[26] Hao Ma, Haixuan Yang, Michael R Lyu, and Irwin King. 2008. Sorec: social recommendation using probabilistic matrix factorization. In *CIKM*. ACM, 931–940.

[27] Jianxin Ma, Peng Cui, and Wenwu Zhu. 2018. Depthlgp: Learning embeddings of out-of-sample nodes in dynamic networks. In *AAAI*.

[28] Zhongming Ma, Olivia RL Sheng, and Gautam Pant. 2009. Discovering company revenue relations from news: A network approach. *Decision Support Systems* 47, 4 (2009), 408–414.

[29] Yasuko Matsubara, Yasushi Sakurai, and Christos Faloutsos. 2015. The Web as a Jungle: Non-Linear Dynamical Systems for Co-evolving Online Activities. In *WWW*. 721–731.

[30] Andriy Mnih and Ruslan R Salakhutdinov. 2008. Probabilistic matrix factorization. In *NIPS*. 1257–1264.

[31] Zheyi Pan, Yuxuan Liang, Weifeng Wang, Yong Yu, Yu Zheng, and Junbo Zhang. 2019. Urban traffic prediction from spatio-temporal data using deep meta learning. In *KDD*.

[32] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *KDD*. ACM, 701–710.

[33] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec. In *WSDM*. ACM, 459–467.

[34] Usman Raza, Alessandro Camerra, Amy L Murphy, Themis Palpanas, and Gian Pietro Picco. 2015. Practical data prediction for real-world wireless sensor networks. *TKDE* 27, 8 (2015), 2231–2244.

[35] Jin Shieh and Eamonn Keogh. 2008. iSAX: indexing and mining terabyte sized time series. In *KDD*. ACM, 623–631.

[36] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *WWW*. 1067–1077.

[37] Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR*.

[38] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NeurIPS*. 5998–6008.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2018. Graph attention networks. In *ICLR*.

[40] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying Graph Convolutional Networks. In *ICML*.

[41] Simeng Wu, Liang Wang, Tianheng Wu, Xianping Tao, and Jian Lu. 2019. Hankel Matrix Factorization for Tagged Time Series to Recover Missing Values During Blackouts. In *ICDE*. 1654–1657.

[42] Zonghan Wu, Shirui Pan, Guodong Long, Jing Jiang, Xiaojun Chang, and Chengqi Zhang. 2020. Connecting the dots: Multivariate time series forecasting with graph neural networks. In *KDD*. 753–763.

[43] Z Wu, S Pan, G Long, J Jiang, and C Zhang. 2019. Graph WaveNet for Deep Spatial-Temporal Graph Modeling. In *IJCAI*.

[44] Cheng Yang, Zhiyuan Liu, Deli Zhao, Maosong Sun, and Edward Chang. 2015. Network representation learning with rich text information. In *IJCAI*.

[45] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. 2019. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *AAAI*, Vol. 33. 5668–5675.

[46] Xiao Yu, Quanquan Gu, Mianwei Zhou, and Jiawei Han. 2012. Citation prediction in heterogeneous bibliographic networks. In *SDM*. 1119–1130.

[47] Chuxu Zhang, Dongjin Song, Chao Huang, Ananthram Swami, and Nitesh V Chawla. 2019. Heterogeneous graph neural network. In *KDD*. 793–803.

[48] Chuanpan Zheng, Xiaoliang Fan, Cheng Wang, and Jianzhong Qi. 2020. Gman: A graph multi-attention network for traffic prediction. In *AAAI*, Vol. 34. 1234–1241.

[49] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. 2018. Dynamic network embedding by modeling triadic closure process. In *AAAI*.

[50] Yuxun Zhou, Han Zou, Reza Arghandeh, Weixi Gu, and Costas J Spanos. 2018. Non-parametric outliers detection in multiple time series a case study: Power grid data analysis. In *AAAI*.

[51] Yuan Zuo, Guannan Liu, Hao Lin, Jia Guo, Xiaoqian Hu, and Junjie Wu. 2018. Embedding temporal network via neighborhood formation. In *KDD*. ACM, 2857–2866.