SOLVING PREDICTION PROBLEMS FROM TEMPORAL EVENT DATA ON NETWORKS

by

Hao Sha

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy

Department of Computer and Information Science

Indianapolis, Indiana

August 2021

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. George Mohler, Chair

Department of Computer and Information Science

Dr. Mohammad Hasan

Department of Computer and Information Science

Dr. Murat Dundar

Department of Computer and Information Science

Dr. Snehasis Mukhopadhyay

Department of Computer and Information Science

Approved by:

Dr. Shiaofen Fang

To my wife and parents

ACKNOWLEDGMENTS

I want to thank Dr. George Mohler and Dr. Mohammad Hasan for letting me join their group. I appreciate their contributions of time, ideas, and resources to make my PhD career productive. I also want to thank Dr. Murat Dundar and Dr. Snehasis Mukhopadhyay for the guidance they provided as part of my dissertation committee. Last but not least, I want to thank my wife and parents for their love and support.

Contents

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Many complex processes can be viewed as sequential events on a network. In this thesis, we study the interplay between a network and the event sequences on it. We first focus on predicting events on a known network. Examples of such include: modeling retweet cascades, forecasting earthquakes, and tracing the source of a pandemic. In specific, given the network structure, we solve two types of problems - (1) forecasting future events based on the historical events, and (2) identifying the initial event(s) based on some later observations of the dynamics. The inverse problem of inferring the unknown network topology or links, based on the events, is also of great important. Examples along this line include: constructing influence networks among Twitter users from their tweets, soliciting new members to join an event based on their participation history, and recommending positions for job seekers according to their work experience. Following this direction, we study two types of problems - (1) recovering influence networks, and (2) predicting links between a node and a group of nodes, from event sequences.

1. INTRODUCTION

An event sequence is a series of timestamp and mark pairs organized in time ascending order [45, 106]. The timestamps denote the time when events occur; while the marks are features indicating event types, locations, participants, and so on. For instance, an event sequence could be a series of time and user ID pairs indicating when and who posts a tweet [173, 98]. It may also be a sequence of time and longitude/latitude coordinates indicating when and where earthquakes occur [108]. Modeling event sequences plays an important role in many areas, such as earthquake forecasting [108], crime prevention [99, 133], and user-behavior study in social networks [173, 98].

Networks are a kind of data structure that models a set of objects (nodes) and their relationships (edges) [103]. Network data arises in many real-world applications. The users of a social media platform can be viewed as nodes and their relationships as edges in a social network [80, 173]. Sensor stations deployed across a city form a sensor network, and their readings are time-dependent signals on the nodes [170]. In genetics, gene expression data can be represented as signals defined on the regulatory network [23, 47]. In neuroscience, brain networks can reveal structural and functional characteristics of the human brain [167].

Many real-world phenomena can be modeled by the interplay between event sequences and networks. In one particular kind of scenarios, sequential events occur at the nodes of a network whose structure is known. For instance, earthquake cascades can be mapped to the nodes of a fault-line network [15]. Contagion process (e.g. virus, rumor, idea) propagates through a contact network or a social network from one node to another [67, 126]. In these cases, we are interested in leveraging the known network topology for learning the sequences. In an opposite kind of scenarios, only the sequential events are observed, while the underlying network structure is unknown. For example, on Twitter, we only observe the times when a user tweets about something, but we do not know from whom she gets the idea, unless she cites the source. In cases like this, we are interested in inferring the network structure or links between particular nodes, based on the observed events. The applications of such include: identifying Twitter influencers by constructing the influence networks of users from their tweets [161]; recommending collaborations by inferring co-authorship networks based

on past publications [102, 149]. In this thesis, we discuss four prediction problems related to event sequences and networks: (1) given networks, predict future events; (2) given networks, predict sources of events; (3) given events, infer networks; and (4) given events, predict group links.

1.1 Given networks, predict future events

In this setting, we consider the problem of modeling sequential events occurring on the nodes of a static network whose topology is known. In specific, we study the forward prediction problem of inferring the nodes where future events are likely to occur. Sequential events are well-known to exhibit mutual correlations. For instance, the arrival of an event may increase the rate of observing a future event. This is called the self-exciting effect, which can be seen in earthquake aftershocks or preferential attachments in social media. Traditionally, various point processes are developed for modeling event correlation [106, 45, 60]. Particularly, Hawkes processes [60] introduce a time-varying intensity conditioned on all the past events on top of a Poisson noise of constant intensity. Hawkes process models can effectively capture the aforementioned self-exciting effect, and are applied broadly in seismicity [108], crime forecasting [99, 133], and social media analysis [98].

Although point processes can model the space-time coupling of event sequence, they rely on predefined kernel functions and assumptions of the generation process, thus limiting their capacity of modeling sequences of arbitrary distributions. [33, 157, 95] propose using Recurrent Neural Net (RNN) [121], and in particular, Long Short-term Memory (LSTM) [61, 50] to model arbitrary sequences. However, they do not consider the locations of events. Another potential approach is to pose event prediction as a node classification problem using network embeddings [171]. However, this approach ignores the temporal dependency of successive events, thus leading to poor performance. Although dynamic embedding methods [181, 105, 178] exist, they are not appropriate for our task, because they assume that the networks are changing, whereas in our case the networks are fixed. In Chapter 3, we propose a LSTM based approach that is capable of modeling arbitrarily distributed event sequences and incorporating both temporal and spatial dependencies.

1.2 Given networks, predict sources of events

In this section, we consider the inverse problem - source detection. In specific, given a network and some snapshots of the propagation, we want to locate the sources that induce the propagation. Here a snapshot contains the states of all the nodes of the network. For example, a snapshot of an epidemic spread contains the infection status (susceptible, infectious, recovered) of each individual in the population under consideration. Solving the source detection problem has broad applications, for example, identifying rumor centers in social networks [27], detecting sources of computer viruses [153], isolating initial failures that lead to rolling blackouts in power grids [166], and finding patient-zero of infectious diseases in human contact networks [126].

Various methods have been proposed to conduct source detection [67]. Early approaches resort to centrality measures, such as rumor center [127, 128, 29, 92], eigenvector cetner [116, 117, 44], and Jordan center [180]. However, these methods are heuristic and only give sub-optimal solutions. In contrast, Dynamic Message Passing (DMP) [89] provides near-optimal solutions. However, DMP is computationally expensive and requires the time of the snapshots, which are normally unavailable [67, 126]. [126] proposes a graph neural network [175] based model that takes advantage of the power of neural networks. However, it utilizes only one snapshot, whereas there can be multiple snapshots available in practise. Furthermore, it has been shown that using multiple independent snapshots together can enhance detectability [154]. So an ideal model should incorporate both the spatial information in the network and the temporal information in the multiple snapshots. In Chapter 4, we adopt a spatial temporal graph convolutional network (STGCN) architecture [170] for solving the source detection problem on networks and show that utilizing multi-snapshot leads to significant accuracy improvements.

1.3 Given events, infer networks

In many real-world cases, only time series of the node states are observed, while the network structure is not known. Inferring networks from time series data can provide insight into the internal structure of a system and help explain the dynamics. For example,

inferring genetic networks from expression data can help identifying molecular targets of pharmacological compounds [47]; constructing atmospheric teleconnection networks from extreme-rainfall events can help revealing the mechanisms of climate change [9]; recovering the network that hosts epidemic spreading or information propagation can help controlling infectious diseases or rumors [129].

Existing methods for inferring network from time series data fall into two categories. The first category includes methods like Granger causality [161, 38] and correlation measurements [137, 6], which discover functional connections. The other category includes methods like driving response [146], and compressed sensing [152, 129], which focus on structural connections. In Chapter 5, we construct influence networks among a group of U.S. politicians from their tweets related to COVID-19. In specific, we estimate Granger causality using a Hawkes Binomial Topic Model (HBTM) [100]. With the influence network, we can find out how decision makers have influenced each other, and whom among the decision makers have emerged as leaders.

1.4 Given events, predict group links

Events can be mapped to not only nodes, but also edges. In many real networks, edges between nodes are established by sequential events [62]. For example, in networks of communication via email, edges represent sequences of instantaneous contacts; in social networks such as Twitter, edges represent sequences of following events. An essential problem in modeling edge formation is predicting new edges. In network theory, the problem of predicting new edges is called link prediction [87, 59]. Link prediction is well-studied and has applications in areas like social media [87], e-commerce [14], and molecular biology [84]. Link prediction can be solved using heuristic measures (e.g. Common Neighbor, Jaccard Index, and Adamic/Adar) [87] or methods based on node embeddings [59]. Node embeddings are low-dimensional representations that preserve the network structure [171]. Traditional node embedding methods, like DeepWalk [112] and Node2Vec [54], are static methods based on the latest snapshot of the network. To account for the historical snapshots, various dynamic embedding methods, such as TNE [182], DynamicTriad [179], CTDNE [105] and HTNE

[183], are proposed. Moreover, as many real-world applications, such as recommender systems in e-commerce, involve nodes of different types (e.g. users and items), methods like metapath2vec [30] and HERec [132] are developed for learning embeddings for heterogeneous networks [131]. Recently, a method named change2vec [8] is proposed to learn embeddings for dynamic heterogeneous networks.

Patterns of link formation are not exclusively limited to two nodes. For instance, an author can collaborate with different groups of co-authors in different articles. Likewise, a Facebook user can join various Facebook Groups consisted of other users who share common interests. In these examples, links are formed between an individual and a group of individuals. We refer to this kind of link prediction problem as "group link prediction". Many recommendation problems can be modeled by group link prediction. For instance, Facebook makes suggestions for potential users to join certain Facebook Groups; LinkedIn recommends potential employers to a job-seeker or vice versa; Meetup.com recommends a user for an event, based on who else are participating in that event. To solve group link prediction, we can pose it as a traditional link prediction problem in a heterogeneous network where nodes are individuals and groups. However, in the cases where a group's identity is determined by its participants, we will end up constructing a very large network. For example, to model co-authorship involving N authors, we will need to consider 2^N groups, leading to a heterogeneous network of $\sim O(2^N)$ nodes. In Chapter 6, we propose a Conditional Variational Auto-encoder (**CVAE**) [136] based model that solves group link prediction without constructing a heterogeneous network. In addition, we provide a variant of the CVAE model - Conditional Variational Auto-encoder with History (**CVAEH**) to incorporate the temporal characteristics, where the historical links are considered.

1.5 Contribution

We summarize the major contributions of this dissertation:

- **Given networks, predict future events:** We combine two LSTMs to model both the slowly varying base intensity and the fast varying conditional intensity of an event sequence on a network. We introduce a new loss function using the network distance

distribution of consecutive events. We compare our model with various baselines on both synthetic and real-world datasets to show its superiority for event prediction and sequence generation.

- **Given networks, predict sources of events:** We propose a spatial temporal graph convolution based model - SD-STGCN for source detection on networks using multiple snapshots. We show that SD-STGCN outperforms state-of-the-art models over realistic transmissions, including non-Markovian epidemic simulations with delay, and realistic network topology, including both synthetic and empirical contact networks. Additionally, we apply our model to real COVID-19 case data. The experiments demonstrate the superior performance of SD-STGCN.

- **Given events, infer networks:** We analyze COVID-19 related tweets by prominent US politicians during early 2020. We use a Hawkes binomial topic model (HBTM) to track evolving topics and construct influence networks amongst these politicians based on Granger causality.

- **Given events, predict group links:** We reframe a special case of link prediction on heterogeneous networks that considers the links between an individual and a group, which we call "group link prediction". We propose a CVAE-based model to solve group link prediction. We also introduce a second CVAE model (named CVAEH) that considers the temporal effect by incorporating the historical links. We examine the group link prediction problem in five real-world datasets and show the superiority of our CVAE/CVAEH models in comparison with various competing methods.

1.6 Organization

The dissertation is organized into two tracks - (i) given networks predict events, and (ii) given events predict networks. On the first track, in Chapter 3, we perform the forward task of predicting future events, using a hierarchical LSTM with second-order statistic regularization; in Chapter 4, we perform the backward task of finding the source(s) of a sequence of events, using a spatial temporal graph convolution model (STGCN). On the second track,

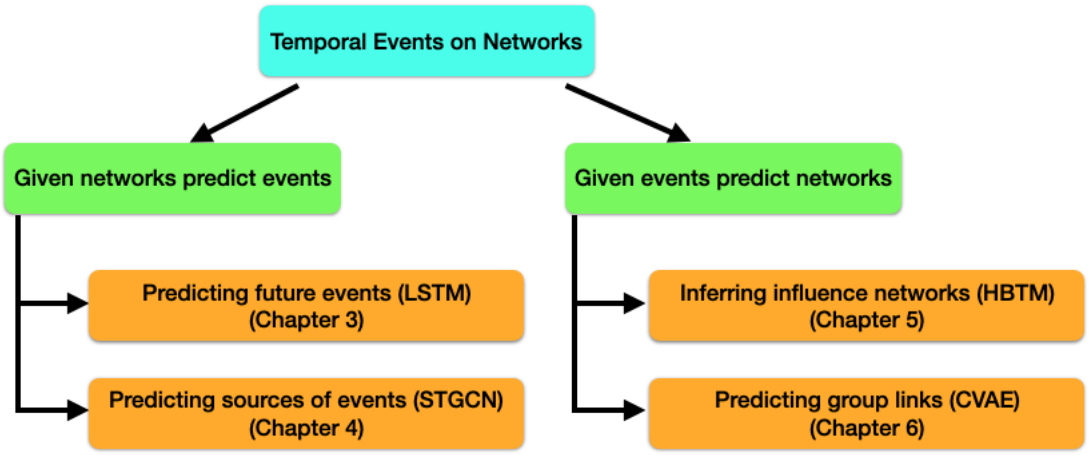


Figure1.1. Visual depiction of the thesis organization.

in Chapter 5, we construct influence networks of twitter users based on their tweets, using a Hawkes process binomial topic model (HBTM); in Chapter 6, we learn the link formation mechanism between an individual and a group of individuals, which we call "group link prediction", using Conditional Variational Auto-Encoder (CVAE). In Fig. 1.1, we illustrate this organization.

2. BACKGROUND

In this chapter, we discuss background material: networks, Hawkes process, recurrent neural networks, (conditional) variational autoencoder, and graph convolutional neural networks.

2.1 Networks

Network [103] is a natural data structure representing relationships among entities. A network contains a set of nodes connected by edges. A node represents an entity which can have attributes (such as text or metadata). An edge represents a relationship between two nodes, and it can have attributes as well. A particular kind of edge attribute is edge weight which measures the strength of the relationship. An edge can have direction(s), i.e. going from one node to another or going both ways. A network whose edges are all directed is called a directed network. In contrast, a network with only undirected edges is called a undirected network. Formally, we denote a network $G = (V, E)$, where V is the set of nodes (also called vertices) and E is the set of edges. The connectivity of a network can be summarized by an adjacency matrix $A \in \mathcal{R}^{|V|\times|V|}$, with $A_{uv} = 1$ if $u \in V$ and $v \in V$ are connected, and $A_{uv} = 0$ otherwise. If the edges are weighted, then $A_{uv} = w_{uv}$ with w_{uv} representing the weight of edge $\mathrm{e}_{uv} \in E$. We further denote the nodal attributes by a matrix $X \in \mathcal{R}^{|V|\times d}$, where d is the dimension of the attribute (feature) space.

Network goes hand in hand with event data. Firstly, we can model event propagating on nodes of networks. Examples along this line include: retweet cascades on the Twitter network [173]; viral marketing on social networks [13]; infectious diseases spreading in human contact networks [126]. In Chapter 3, we follow this approach and study the forward problem of forecasting future events given the historical events; in Chapter 4, we study the backward problem of finding the very first event that triggers the rest based on some later snapshots of the propagation. Another way of modeling events using networks is directly mapping events to nodes and their correlations to edges. For instance, in a Hawkes process [101] events trigger one another following certain underlining branching structures which are essentially networks with edge weight $w_{uv} = P_{uv}, u, v \in V$ representing the probability of v being triggered by u. Moreover, a Granger causality network [38] can be constructed among

different types of events, to discover, for instance, the mutual influence among Twitter users. In Chapter 5, we recover the branching structure of tweets posted by prominent U.S. politicians about COVID-19 and construct an influence network among these politicians. Last but not least, we can model event participants as nodes in a network and frame the event recommendation problem as a link prediction problem [87, 59, 14]. In specific, we want to recommend a new participant to an event or alternatively recommend an event for an individual to join. Traditionally, link prediction can be solved by ranking some heuristic measures between two nodes, including neighbor-based measures like Common Neighbor, Jaccard Index, Adamic/Adar and Preferential Attachment, and path-based measures like Graph Distance, Katz$_\beta$, and hitting time. Recent approaches focus on learning nodal representations (i.e. embeddings). Embeddings can be obtained through matrix factorization [79] or random walk [112, 54]. In Chapter 6, we propose solving the event participants prediction problem using a novel conditional variational auto-encoder (CVAE) approach.

2.2 Hawkes Processes

Univariate Hawkes Process A temporal point process [106] is an ordered set of event times $\{t_i\}_{i=0}^{N}$. We typically describe a point process by its conditional intensity function $\lambda(t|\mathcal{H}_{t^-})$ which can be interpreted as the instantaneous probability of an event occurring at time t given the history $\mathcal{H}_{t^-}$ consisting of all the events before t. This can be written as

$$\lambda(t|\mathcal{H}_{t^-}) := \lim_{\Delta t \to 0^+} \frac{P(N[t, t+\Delta t)|\mathcal{H}_{t^-})}{\Delta t} \tag{2.1}$$

where $N[t, t+\Delta t)$ is the number of events in $[t, t+\Delta t)$. Point processes can be specified by choosing a functional form of the intensity. In particular, a Hawkes process [60] has the following intensity:

$$\lambda(t|\mathcal{H}_{t^-}) = \mu(t) + \sum_{t_i \in \mathcal{H}_{t^-}} g(t - t_i) \tag{2.2}$$

where $\mu(t)$ is the base intensity and $g(t - t_i)$ is the triggering kernel that describes the self-exciting effect by a historical event at t_i. The log-likelihood of observing N events in $[0, T)$ is:

$$\begin{aligned} LL(\mu, g) &= \sum_{i=1}^{N} ln(\lambda(t_i|\mathcal{H}_{t_i^-})) - \int_0^T \lambda(t|\mathcal{H}_{t^-})dt \\ &= \sum_{i=1}^{N} ln(\mu(t_i) + \sum_{t_j<t_i} g(t_i - t_j)) - \int_0^T \mu(t)dt - \sum_{i=1}^{N} \int_0^{T-t_i} g(s)ds. \end{aligned} \tag{2.3}$$

Given sequences of observations, we can fit the intensity in Eq. 2.2 by maximizing Eq. 2.3, i.e. to obtain the maximum likelihood estimation (MLE) of μ and g - μ_{MLE} and g_{MLE}. However, an analytical solution of Eq. 2.3 is intractable, as $LL(\mu, g)$ involves a sum of logarithms of conditional intensities, which themselves involve sums over previous points. Moreover, [148] shows that the use of numerical methods like the Newton-type to maximize Eq. 2.3 can be problematic in cases where the log-likelihood is extremely flat. To overcome this problem, expectation maximization (EM) based algorithms [148, 101] are proposed. By introducing latent variables to describe the branching structure, these methods can significantly simplify the log-likelihood and find the MLE much more efficiently. With the estimated intensity, we can infer the arriving time of the next event or simulate a sequence of events through methods like thinning [107].

Multivariate Hawkes process In many real-life scenarios such as earthquake after-shocks [148], civilian death in conflicts [101] and user behaviors in social networks [98], event sequences involve events of different types and exhibit self-exciting and mutually-exciting properties - an event not only can be triggered by a previous event of the same type, but also can be triggered by a previous event of another type. Multivariate Hawkes process [177, 176] is thus introduced to capture such effects. Formally, let $\{(u_i, t_i)\}_{i=1}^N$ be a sequence of N events in a time interval $[0, T)$, with $u_i \in \mathcal{U}$ and $0 \leq t_i < T$ denoting the event type and occurrence time of event i, respectively. $\mathcal{U}$ is a finite set of event types. The intensity for type u is:

$$\lambda_u(t|\mathcal{H}_{t^-}) = \mu_u(t) + \sum_{(u_i,t_i)\in\mathcal{H}_{t^-}} g_{uu_i}(t - t_i), \tag{2.4}$$

where $\mathcal{H}_{t^-} = \{(u_\mathrm{i}, t_\mathrm{i}) | t_\mathrm{i} < t, u_\mathrm{i} \in \mathcal{U}\}$ is the history consisting of all the events before t. $\mu_u(t) \geq 0$ is the base intensity for type u. The kernel $g_{uu'}(t) \geq 0$ captures the mutually-exciting property between type u and u'. Suppose we have m samples $\{c_\mathrm{i}\}_{\mathrm{i}=1}^{m}$, with each sample c containing a sequence of observed events $\{(u_\mathrm{i}^c, t_\mathrm{i}^c)\}_{\mathrm{i}=1}^{N_c}$ in time intervals $[0, T_c)$, the log-likelihood of the multivariate Hawkes process is:

$$\begin{aligned} LL(\mu, g) &= \sum_c \left(\sum_{\mathrm{i}=1}^{N_c} \lambda_{u_\mathrm{i}^c}(t_\mathrm{i}^c) - \sum_{u \in \mathcal{U}} \int_0^{T_c} \lambda_u(t) dt \right) \\ &= \sum_c \left(\sum_{\mathrm{i}=1}^{N_c} ln\Big(\mu_{u_\mathrm{i}^c}(t_\mathrm{i}^c) + \sum_{t_\mathrm{j}^c < t_\mathrm{i}^c} g_{u_\mathrm{i}^c u_\mathrm{j}^c}(t_\mathrm{i}^c - t_\mathrm{j}^c)\Big) \right. \\ &\quad \left. - \sum_{u \in \mathcal{U}} \int_0^{T} \mu_u(t) dt - \sum_{u \in \mathcal{U}} \sum_{\mathrm{i}=1}^{N_c} \int_0^{T_c - t_\mathrm{i}} g_{u u_\mathrm{i}^c}(s) ds \right), \end{aligned} \tag{2.5}$$

where for simplicity, we use $\lambda_u(t)$ to denote $\lambda_u(t|\mathcal{H}_{t^-})$. The MLE of μ and g can be obtained by EM alogirthms [177, 176].

Assuming that the kernels in Eq. 2.4 share the same parametric form g, we have:

$$\lambda_u(t|\mathcal{H}_{t^-}) = \mu_u(t) + \sum_{(u_\mathrm{i}, t_\mathrm{i}) \in \mathcal{H}_{t^-}} a_{u u_\mathrm{i}} g(t - t_\mathrm{i}), \tag{2.6}$$

where the coefficient $a_{uu'} \geq 0$ captures the mutually-exciting effect between type u and $u' \in \mathcal{U}$. We collect all the pair-wise coefficients into a matrix, $\mathbf{A} = (a_{uu'})$, called the infectivity matrix. Note that an event can be triggered by a previous event of the same type, so the diagonal elements of $\mathbf{A}$ are not necessarily zero. In a scenario where u represents a node in a network allowing self-connections, $\mathbf{A}$ is effectively the adjacency matrix. Specifically, given a network $G = \{V, E\}$, we have a sequence of events $\{(u_\mathrm{i}, t_\mathrm{i})\}_{\mathrm{i}=1}^{N}$ occurring on the nodes, i.e. $u_\mathrm{i} \in V$. The cross-excitation effects of the events are modulated by the network topology, where an event on node u can be triggered by a previous event on either a neighboring node v or u itself (i.e. $a_{uv} > 0$ or $a_{uu} > 0$). The self-excitation on the other hand is governed by the background intensity term μ_u, indicating that an event can also occur spontaneously on any of the nodes. In Chapter 3, we study the event prediction problem under this multivariate/network Hawkes process framework.

Multivariate point process is a special case of the marked point process [60]. In general, point processes may be marked if features of events beyond their time are also observed. For example, we can model earthquakes using their magnitudes as marks. Formally, a marked point process is a point process of events $\{(u_{\mathrm{i}}, t_{\mathrm{i}})\}$, where $t_{\mathrm{i}} \in [0, T)$, and $u_{\mathrm{i}} \in \mathcal{U}$, with $\mathcal{U}$ the mark space. If the mark space $\mathcal{U}$ is a finite set, the marked point process becomes a multivariate point process. Let $\vec{m}$ denote the feature vector of an event, the intensity of a marked Hawkes process can be written as:

$$\lambda(t, \vec{m}) = \mu(t, \vec{m}) + \sum_{t_{\mathrm{i}} < t} g(t - t_{\mathrm{i}}, \vec{m}, \vec{m}_{\mathrm{i}}), \tag{2.7}$$

where we drop $\mathcal{H}_{t^-}$ for simplicity. In Chapter 5, we use $\vec{m}$ to represent the content of a tweet, and apply a Hawkes Binomial Topic Model (HBTM) to study how the Twitter narratives around COVID-19 evolve among prominent U.S. politicians.

2.3 Recurrent Neural Networks

Sequential data exhibits temporal correlation, so a well-suited model for modeling sequential data should be able to capture this property. Feedforward neural networks [63] are proved to be universal approximators, however they assume that the data points are independent. In these networks, the entire state is lost after each data point is processed. So feedforward neural networks are not ideal for modeling sequential data. In contrast, recurrent neural networks (RNN) [121] can process the data points successively and keep the network state up-to-date. Specifically, a typical RNN takes in the input at the current step and the state from the previous step, and outputs the prediction for the current step or the next step. RNNs are thus appropriate choices for modeling sequences of data points that are not independent.

Moreover, it has been proved that a finite-sized recurrent neural network with sigmoidal activation functions are Turing complete [134], which indicates that RNN is capable of performing nearly arbitrary computations. There are two reasons making RNN stand out from other universal Turing machines: (1) RNN is differentiable and thus can be learned via gradient descent; (2) RNN can be regularized via methods like dropout and weight decaying,

to prevent overfitting. Without the means of regularization, there exist countless programs that can generate desired outputs for training data, but failing to generalize to test data.

For an input sequence $\{x_t\}_{t=1}^T$ and a target sequence $\{y_t\}_{t=1}^T$, a traditional RNN is often specified by the following equations:

$$\begin{aligned} h_t &= \sigma(W^{hx}x_t + W^{hh}h_{t-1} + b_h) \\ \hat{y}_t &= softmax(W^{yh}h_t + b_y) \end{aligned} \tag{2.8}$$

where the first equation describes the latent model and the second equation describes the output model. h_t is the latent state at time t, and $\hat{y}_t$ is the predicted output at time t. W^{hx}, W^{hh}, and W^{yh} are the weight matrices mapping inputs to latent states, latent states from time step $t-1$ to t, and latent states to outputs, respectively. b_h and b_y are bias terms for the latent state and the output. σ is an activation function such as sigmoid, tanh, and rectified linear unit (ReLU). Note that we assume a multi-class classification, so a softmax activation is adopted in the output model. A linear activation should be used for regression or a sigmoid activation should be used for binary classification (logistic regression). RNNs can be learned through backpropagation through time (BPTT) [156]. For instance, for multi-class classification, we minimize the following cross-entropy loss:

$$loss = -\sum_{t=1}^{T}\sum_{k=1}^{m} y_t^k log(\hat{y}_t^k) \tag{2.9}$$

where m is the dimension of the output (observation) space ($m = |y|$). Note this is essentially MLE through gradient ascent, so the loss function is the negative log-likelihood for the task - for example mean squared error (MSE) should be used for regression.

The form of RNNs described in Eq. 2.8 is notoriously difficult to train due to vanishing (exploding) gradient, especially for modeling long-range dependencies [7, 111]. The gradient over T time steps is:

$$\frac{\partial loss}{\partial \theta} = \sum_{t=1}^{T} \frac{\partial loss_t}{\partial \theta}, \tag{2.10}$$

where θ denotes the set of parameters. The gradient at t is:

$$\begin{aligned}\frac{\partial loss_t}{\partial \theta} &= \frac{\partial loss_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\prod_{\mathrm{i}=2}^{t}\frac{\partial h_{\mathrm{i}}}{\partial h_{\mathrm{i}-1}}\right)\frac{\partial h_1}{\partial \theta} \\ &= \frac{\partial loss_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial h_t}\left(\prod_{\mathrm{i}=2}^{t}\sigma'(W^{hx}x_{\mathrm{i}} + W^{hh}h_{\mathrm{i}-1})W^{hh}\right)\frac{\partial h_1}{\partial \theta}\end{aligned} \tag{2.11}$$

where σ' is the derivative of the activation. For tanh and sigmoid, $\sigma'(x) \leq 1$; for ReLU, $\sigma'(x) = 1$ when $x > 0$. We can see that if $\sigma' W^{hh} < 1$, the product vanishes when $t \to \infty$; if $\sigma' W^{hh} > 1$, the product explodes when $t \to \infty$. So the signals far from the beginning rarely have any contributions to the total gradient. Furthermore, the latent state h_t for large t could easily vanish or explode, if the weight matrices were not initialized properly.

Various modern RNN models [17, 61] have been proposed to remedy the vanishing gradient problem. Long Short-term Memory (LSTM) [61] is one of the most popular. It introduces a cell state and uses gates to modulate the states. Specifically, LSTM can be described by the following equations:

$$\begin{aligned}&\mathrm{i}_t = \sigma(W_{\mathrm{i}}[h_{t-1}, x_t] + b_{\mathrm{i}}), \qquad f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \qquad o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \\ &\tilde{c}_t = tanh(W_c[h_{t-1}, x_t] + b_c), \qquad c_t = f_t \odot c_{t-1} + \mathrm{i}_t \odot \tilde{c}_t, \qquad h_t = o_t \odot tanh(c_t),\end{aligned} \tag{2.12}$$

where σ is the sigmoid function and $\odot$ is the Hadamard product. i_t, f_t, and o_t are input, forget, and output gates. x_t, h_t, and c_t are input, hidden, and cell states. W_*, U_*, V_*, and b_* are trainable parameters. Similar to Eq. 2.11, the gradient at t contains the product $\prod_{\mathrm{i}=2}^{t}\frac{\partial c_{\mathrm{i}}}{\partial c_{\mathrm{i}-1}}$, and it can be shown that

$$\begin{aligned}\frac{\partial c_t}{\partial c_{t-1}} &= \sigma'(W_f[h_{t-1}, x_t]) \cdot W_f \cdot o_{t-1} \odot tanh'(c_{t-1}) \cdot c_{t-1} + f_t \\ &+ \sigma'(W_{\mathrm{i}}[h_{t-1}, x_t]) \cdot W_{\mathrm{i}} \cdot o_{t-1} \odot tanh'(c_{t-1}) \cdot \tilde{c}_t \\ &+ \sigma'(W_c[h_{t-1}, x_t]) \cdot W_c \cdot o_{t-1} \odot tanh'(c_{t-1}) \cdot \mathrm{i}_t\end{aligned} \tag{2.13}$$

as a sum of four terms. In particular, the f_t term makes the gradient difficult to vanish. In addition, one can further constrain the weights to prevent the gradient from blowing out.

LSTM has been successfully applied to various sequence related learning problems, such as, speech recognition [51], language translation [140], handwriting synthesis [50], and image generation [52]. In Chapter 3, we combine two LSTMs for modeling event sequences on networks.

2.4 Variational Autoencoder

Variational autoencoder (VAE) [28, 73] is a generative model consist of an encoder and a decoder that are learned jointly through gradient descent. Let x be a random variable representing a data point. The data generating process can be modeled by:

$$p_\theta(x) = \int p_\theta(x, z)dz, \tag{2.14}$$

where θ denotes the set of parameters and z denotes a vector of latent variables. Learning $p_\theta(x)$ via maximum likelihood is generally difficult due to the integral in Eq. 2.14. Learning $p_\theta(z|x)$ is equally difficult because $p_\theta(z|x) = p(x, z)/p_\theta(x)$. So we approximate $p_\theta(z|x)$ by a parametric model $q_\phi(z|x)$, where ϕ denotes the set of parameters of the encoder. So the log-likelihood of the data can be written as:

$$\begin{aligned} log p_\theta(x) &= \mathbb{E}_{q_\phi(z|x)}[log p_\theta(x)] = \mathbb{E}_{q_\phi(z|x)}\left[log \frac{p_\theta(x, z)}{p_\theta(z|x)}\right] = \mathbb{E}_{q_\phi(z|x)}\left[log \frac{p_\theta(x, z)q_\phi(z|x)}{q_\phi(z|x)p_\theta(z|x)}\right] \\ &= \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[log \frac{p_\theta(x, z)}{q_\phi(z|x)}\right]}_{\mathcal{L}_{\theta,\phi}(x)} + \underbrace{\mathbb{E}_{q_\phi(z|x)}\left[log \frac{q_\phi(z|x)}{p_\theta(z|x)}\right]}_{D_{KL}(q_\phi(z|x)||p_\theta(z|x))}. \end{aligned} \tag{2.15}$$

where the log-likelihood is decomposed into two terms - $\mathcal{L}_{\theta,\phi}(x)$ and $D_{KL}(q_\phi(z|x)||p_\theta(z|x))$. The second term is the Kullback-Leibler (KL) divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$, which

is non-negative. So the first term $\mathcal{L}_{\theta,\phi}(x)$ is a lower bound on the log-likelihood, called the evidence lower bound (ELBO). We can further write:

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x,z))] - \mathbb{E}_{q_\phi(z|x)}[q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x|z))] + \mathbb{E}_{q_\phi(z|x)}[logp(z)] - \mathbb{E}_{q_\phi(z|x)}[q_\phi(z|x)] \\ &= \mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x|z))] - D_{KL}(q_\phi(z|x)||p(z)) \\ &\approx log(p_\theta(x|z)) - D_{KL}(q_\phi(z|x)||p(z))\end{aligned} \tag{2.16}$$

where $q_\phi(z|x)$ is the encoder and $p_\theta(x|z)$ is the decoder. Note that in the last line of Eq. 2.16, we take one sample $z \sim q_\phi(z|x)$ and approximate $\mathbb{E}_{q_\phi(z|x)}[log(p_\theta(x|z))]$ by $logp_\theta(x|z)$ at that z. The usual choice of $q_\phi(z|x)$ is a multivariate Gaussian - $q_\phi(z|x) = \mathcal{N}(z|\mu(x,\theta), \Sigma(x,\theta))$. We further assume that $p(z)$ is a standard Gaussian $p(z) = \mathcal{N}(0, I)$. So the KL-divergence in Eq. 2.16 becomes:

$$D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\Big(tr(\Sigma(x,\phi)) + \mu(x,\phi)^T\mu(x,\phi) - k - logdet(\Sigma(x,\phi))\Big) \tag{2.17}$$

where k is the dimensionality of the distribution. However, we cannot directly sample z from $\mathcal{N}(z|\mu(x,\theta), \Sigma(x,\theta))$, because the back-propagation would not go through. So [73] proposed a method called the "reparameterization trick", to first sample $\epsilon \sim \mathcal{N}(0, I)$, then compute $z = \mu(x,\phi) + \Sigma^{1/2}(x,\phi) \cdot \epsilon$. ELBO is now:

$$\begin{aligned}\mathcal{L}_{\theta,\phi}(x) &= log(p_\theta(x|z = \mu(x,\phi) + \Sigma^{1/2}(x,\phi) \cdot \epsilon)) \\ &\quad - \frac{1}{2}\Big(tr(\Sigma(x,\phi)) + \mu(x,\phi)^T\mu(x,\phi) - k - logdet(\Sigma(x,\phi))\Big),\end{aligned} \tag{2.18}$$

where $\mu(x,\phi)$ and $\Sigma(x,\phi)$ are the outputs of the encoder. The gradients of Eq. 2.18 w.r.t. θ and ϕ can be efficiently computed using packages such as Tensorflow.

In some cases, we not only have data but also observe some additional information. For example, in image inpaintings, we want to restore the missing pixels in an image given the observed ones. [65] proposed to solving such problems using an extended VAE model that is able to incorporate conditions, called conditional variational autoencoder (CVAE). Let y

denote the conditions. The ELBO of CVAE is similar to Eq. 2.16 except that x is conditioned on y:

$$\mathcal{L}_{\theta,\phi}(x|y) = log(p_\theta(x|y,z)) - D_{KL}(q_\phi(z|x,y)||p(z|y)). \tag{2.19}$$

Note that $p(z|y)$ can still be $\mathcal{N}(0,I)$, because z is sampled independently of y at test time. To implement $q_\phi(z|x,y)$ and $p_\theta(x|y,z)$, we can, for example, concatenate y with x and z at the encoder and the decoder, respectively. In Chapter 6, we develop a CVAE-based model for predicting the event participants.

2.5 Graph Convolutional Networks

Graph Convolutional Networks (GCNs) are neural networks that can perform convolution operations on graph signals [175]. Prior to GCNs, Convolutional Neural Networks (CNNs) [83] achieved great success in extracting multi-scale localized features from structured data like images. As graphs are also locally connected structures, it is natural to think of generalizing CNNs to graphs. However, it is hard to directly transform the localized convolutional filters from images to graphs, as the nodes in the later are not ordered.

One way to solve the problem is to define the filters in the Fourier domain. Given a undirected and connected graph $G = (V,E)$ with N nodes, let $A \in \mathbb{R}^{N\times N}$ denote the weighted adjacency matrix. The normalized graph Laplacian is $L = I - D^{-\frac{1}{2}}AD^{-\frac{1}{2}} \in \mathbb{R}^{N\times N}$ where $D \in \mathbb{R}^{N\times N}$ is the diagonal degree matrix with $D_{\mathrm{ii}} = \sum_{\mathrm{j}} A_{\mathrm{ij}}$. Let $U = [u_0,\ldots,u_{N-1}] \in \mathbb{R}^{N\times N}$ and $\{\lambda_{\mathrm{i}}\}_{\mathrm{i}=0}^{N-1}$ denote the eigenvectors and eigenvalues of L, then $L = U\Lambda U^T$ where $\Lambda = diag([\lambda_0,\ldots,\lambda_{N-1}]) \in \mathbb{R}^{N\times N}$. The graph Fourier transform of a signal $x \in \mathbb{R}^N$ is then defined as $\hat{x} = U^T x$ and its inverse as $x = U\hat{x}$. The convolution on graph in the Fourier domain is defined as $x * y = U((U^T x) \odot (U^T y))$, where $\odot$ is the Hadamard product, based on the convolution theorem - the Fourier transform of a convolution of two functions is the pointwise product of their Fourier transforms. Replacing $(U^T x)\odot$ by a matrix multiplication with a filter $g_\theta = diag(\theta)$ paremeterized by $\theta \in \mathbb{R}^N$, we arrive at the following definition of the convolution on $x \in \mathbb{R}^N$ [11]:

$$g_\theta * x = U g_\theta U^T x. \tag{2.20}$$

However, this form of convolution is expensive and non-localized. So [58] proposes approximating g_θ by a truncated expansion of Chebyshev polynomials $T_k(x)$ up to K'th order:

$$g_\theta * x \approx \sum_{k=0}^{K} \theta_k T_k(\tilde{L})x, \tag{2.21}$$

where $\tilde{L} = \frac{2}{\lambda_{max}}L - I$ with λ_{max} being the largest eigenvalue of L. $\theta \in \mathbb{R}^K$ is a vector of Chebyshev coefficients. The Chebyshev polynomials are $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. Eq. 2.21 is K-localized and needs less parameters $K < N$. Also, there is no need to calculate the eigenvectors of L.

Alternatively, [74] suggests stacking multiple layers of $K = 1$ convolution with $\lambda_{max} \approx 2$. So the convolution becomes:

$$g_\theta * x \approx \theta_0 x + \theta_1(L - I)x = \theta_0 x - \theta_1 D^{-\frac{1}{2}} A D^{-\frac{1}{2}} x, \tag{2.22}$$

where θ_0 and θ_1 are two parameters. Setting $\theta = \theta_0 = -\theta_1$ renders:

$$g_\theta * x \approx \theta(I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}})x. \tag{2.23}$$

Stacking this operation is not stable, thus renormalization is needed at each layer: $I + D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \rightarrow \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}}$, with $\tilde{A} = A + I$ and $\tilde{D}_{\mathrm{ii}} = \sum_{\mathrm{j}} \tilde{A}_{\mathrm{ij}}$. So for a signal $X \in \mathbb{R}^{N \times C}$ with C channels, a convolution layer can be summarized by the following matrix multiplications:

$$Z = \tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta \tag{2.24}$$

where $\Theta \in \mathbb{R}^{C \times F}$ is a matrix of filter parameters and $Z \in \mathbb{R}^{N \times F}$ is the output of the layer. In Chapter 4, we apply a spatial-temporal graph convolutional network (STGCN) [170] to detect diffusion source(s) on networks. In specific, we adopt both Eq. 2.21 and Eq. 2.24 for graph convolution.

3. LEARNING NETWORK EVENT SEQUENCES USING LONG SHORT-TERM MEMORY AND SECOND-ORDER STATISTIC LOSS

A version of this chapter was previously published by Statistical Analysis and Data Mining: The ASA Data Science Journal. Sha, H, Al Hasan, M, Mohler, G. Learning network event sequences using long short-term memory and second-order statistic loss. Stat Anal Data Min: The ASA Data Sci Journal. 2021; 14: 61– 73. [124]. DOI: 10.1002/sam.11489.

3.1 Introduction

Modeling event sequences is essential in many areas, such as earthquake forecasting [108], crime prevention [99, 133], and user-behavior study in social networks [173, 98]. Earthquakes tend to occur as sequences or clusters in close spatial and temporal proximity. Modeling earthquake sequences helps forecast future earthquakes and mitigate the seismic hazard. Space-time clustering event sequences are also observed in certain types of crime data, such as burglary and gang violence [133]. Studying these sequences may help identify crime patterns and prevent crimes from happening. Social networking services allow users to share content, and widely popular content can be shared by hundreds of thousands of users. Content sharing events form a sequence that spreads through social networks, as such, modeling this sequences could help predict content popularity, and provide useful information for content ranking and aggregation. It also helps better understanding of influence, fake-news or rumor propagation over the social networks.

An event sequence is a series of timestamp and mark pairs organized in time ascending order. The timestamps denote the time when events occur; while the marks indicate the identity of events. For instance, an event sequence could be a series of time and user ID pairs indicating when and who posts a photo. It may also be a sequence of time and longitude/latitude coordinates indicating when and where earthquakes occur. In many cases, sequential events occur at the vertices of an existing network or the events can be mapped to one of the vertices of a constructed network. For instance, earthquakes predominantly

occur along fault lines, where tectonic movements are active. Therefore, earthquakes can be assigned membership to a large-scale fault-line cluster [15]. By inserting vertices on a fault line and then connecting nearby vertices, an earthquake network can be constructed. A sequence of earthquakes can thus be viewed as a series of events occurring on the nodes of this network. As another example, posting messages on an online social network (such as, Facebook or Twitter) can be viewed as an event occurring at a node of that network. In this work, we focus on modeling the event sequences in a network with an objective to predict the nodes where the future events are likely to occur and to generate sequences that closely resemble the real sequences.

The occurrence of an event may be spontaneous and independent of other events. On the other hand, it may also be triggered by the previous events (self-excitation). Traditionally, event sequences are modeled by various point processes [106, 45, 60]. In particular, Hawkes processes [60] model the spontaneity by a based intensity and the self-excitation by a time-varying conditional intensity. However, these models rely on some predefined parametric forms, thus limiting their capability of modeling arbitrarily distributed event data. To remedy this problem, EM-based nonparametric models [85, 176] are proposed. However, these models are still under the framework of Hawkes processes and may suffer from model mis-specification when the underlining event generation mechanism is not known a priori.

Alternatively, one could ignore the space-time coupling of event sequence and model an event's time and location independently. For instance, one can turn to Recurrent Neural Net (RNN) [121], specifically, Long Short-term Memory (LSTM) [61, 50] to model a general point process. Various models [33, 157, 95] along this line are proposed, but in these models, the events' locations are not considered, and they are not necessarily occurring in a network node. One can also emphasize the network on which the events occur, ignoring their timestamp and consider event prediction as a node classification task, which can be solved by using a network embedding model [171]; however, such a direction ignores the temporal dependency of successive events, resulting in poor performance. Some existing works on network embedding consider temporal change in network [181, 93], but they are not appropriate for our task because in our task the underlying network is fixed, and we are merely interested in the temporal sequence of nodes where the future events will occur.

In this work, we propose a novel method for modeling temporal event sequence in the vertices of a network. Our proposed model uses LSTM to minimize the cross-entropy between the generated event probability and the one-hot encoding of the real event, which essentially enforces the generated sequences to have a first-order statistics similar to the real sequence. However, a potential next event may occur at any of the neighbors of the current event node and does not have to be the exact neighbor in the real sequence. The first-order statistic loss (cross-entropy loss) does not take into account this aspect. In contrast, the second-order loss penalizes the network distance between events rather than their identity, thus favoring the events that are within the correct hop distance in the network. For implementation, we combine two LSTMs—our first LSTM takes long-term event counts as inputs, while the second LSTM takes short-term event marks as inputs. For instance, we may feed the monthly event counts during the past 30 months to the first LSTM and the latest 30 events to the second LSTM. The first LSTM thus learns the slowly varying characteristics, while the second LSTM learns the fast-changing characteristics. Unlike existing works on point process [157], our model does not require domain-specific features and solely relies on the event sequences.

The contributions of this chapter are: We combine two LSTMs to model both the slowly varying base intensity and the fast varying conditional intensity of an event sequence on a network. We introduce a new loss function using the network distance distribution of consecutive events. We compare our model with various baselines on both synthetic and real-world datasets to show its superiority for event prediction and sequence generation.

The rest of the chapter is organized as follows: In Section 3.2, we provide some backgrounds regarding the important components of our model. In Section 3.3, we discuss related works in learning event sequence. In Section 3.4, we present a detailed description of our method. In Section 3.5, we describe the experiments and present the results. Finally, we summarize our work in Section 3.6.

3.2 Background

In this section, we briefly go through the major building components of this work and provide necessary background information.

3.2.1 Hawkes Process

Hawkes processes [60] are self-exciting point processes where the occurrence of an event increases the likelihood of the occurrence of future events. Hawkes processes are characterized by an intensity function

$$\lambda_v(t) = \mu_v(t) + \sum_{\{(v_\mathrm{i},t_\mathrm{i})|t>t_\mathrm{i}\}} g_v(v_\mathrm{i}, t - t_\mathrm{i}), \tag{3.1}$$

where $\lambda_v(t)$ is the intensity of an event v at time t. $\mu_v(t)$ represents the base intensity. The triggering kernels $g_v(v_\mathrm{i}, t - t_\mathrm{i})$ are accumulated over the historical events $\{v_\mathrm{i}, t_\mathrm{i}\}$. It is common to assume that the base intensity is constant over time $\mu_v(t) = \mu_v$ when the triggering kernels $g_v(v_\mathrm{i}, t - t_\mathrm{i})$ vary significantly faster than $\mu_v(t)$. Furthermore, one may assume that the kernels have some predefined functional forms, such as the exponential function or the power-law function. With a constant base intensity and exponential kernels, Eq. 3.1 becomes

$$\lambda_v(t) = \mu_v + \sum_{\{(v_\mathrm{i},t_\mathrm{i})|t>t_\mathrm{i}\}} W_{v,v_\mathrm{i}} \mathrm{e}^{-w(t-t_\mathrm{i})}, \tag{3.2}$$

where $\mu_v \geq 0$ is the time-independent base intensity, w is the decay rate, and $W_{v,v_\mathrm{i}} \geq 0$ is a measurement by which v_i initially excites v. Let $G = \{V, E\}$ denote a graph, where V and E are collections of vertices and edges. Let the subscript v in Eq. 3.2 denote an event occurring on node $v \in V$. If we assume that an event at node v is either spontaneous (determined by μ_v) or triggered homogeneously by its neighbors v_i, then we have $W_{v,v_\mathrm{i}} = A_{v,v_\mathrm{i}}$, where A is the adjacency matrix of G. We use the Hawkes process of Eq. 3.2 to generate synthetic sequences in experiments [163]. In addition, one of the baselines, multidimensional Hawkes process model [163], is also based on this formula.

3.2.2 Long Short-term Memory Architecture

Recurrent Neural Net (RNN) [121] is a neural network model designed for modeling time series data. Besides taking input at each time step, it passes down the hidden state of the previous time step. Thus, it is capable of exhibiting temporal dynamic behaviors. However, as pointed out by [7, 111], traditional RNNs suffer from vanishing (exploding) gradient problem, preventing them from learning relationships separated by an extended period. To remedy this problem, authors of [61] propose a Long Short-term Memory (LSTM) architecture where a cell state is introduced. Now the output and the intermediate states at time t are collectively determined by the input, the hidden state and the cell state at time $t-1$. Such design effectively reduces the multiplicative effect of the small gradients. In this work, we adopt the version of LSTM implemented by the following equations:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{\mathrm{i}}\mathbf{x}_t + \mathbf{U}_{\mathrm{i}}\mathbf{h}_{t-1} + \mathbf{V}_{\mathrm{i}}\mathbf{c}_{t-1} + \mathbf{b}_{\mathrm{i}}),\\
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{x}_t + \mathbf{U}_f\mathbf{h}_{t-1} + \mathbf{V}_f\mathbf{c}_{t-1} + \mathbf{b}_f),\\
\mathbf{c}_t &= \mathbf{f}_t\mathbf{c}_{t-1} + \mathbf{i}_t \odot tanh(\mathbf{W}_c\mathbf{x}_t + \mathbf{U}_c\mathbf{h}_{t-1} + \mathbf{b}_c),\\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{x}_t + \mathbf{U}_o\mathbf{h}_{t-1} + \mathbf{V}_o\mathbf{c}_t + \mathbf{b}_o),\\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t),
\end{aligned}
\tag{3.3}
$$

where σ is the sigmoid function and $\odot$ is the Hadamard product. $\mathbf{i}_t$, $\mathbf{f}_t$, and $\mathbf{o}_t$ are vectors of intermediate states; while $\mathbf{x}_t$, $\mathbf{h}_t$, and $\mathbf{c}_t$ are vectors of input, hidden, and cell states, respectively. Matrices $\mathbf{W}_*$, $\mathbf{U}_*$, $\mathbf{V}_*$, and vectors $\mathbf{b}_*$ are trainable parameters.

In recent years, LSTM has become a popular choice for modeling sequential data, along with other methods such as dilated causal convolutions [109] and Gated Recurrent Unit (GRU) [17]. LSTM has been successfully applied to solve various sequence related learning problems, such as, speech recognition [51], language translation [140], handwriting synthesis [50], and image generation [52].

3.2.3 Second-order Statistics of Sequential Events

Besides correlating in time, sequential events may also correlate in types or space. For instance, aftershocks commonly occur in the vicinity of a major earthquake. A picture posted by a user is more likely to be liked by a friend than a stranger. The general spatial correlation could be measured by Ripley's K-function [120, 26] which is defined as:

$$\begin{aligned} K(d) =& \lambda^{-1} E[number\ of\ extra\ events\ within \\ & distance\ d\ of\ a\ randomly\ chosen\ event], \end{aligned} \tag{3.4}$$

where λ is the event density (number per unit area). If edge effects are ignored, $K(d)$ can be estimated by [26]:

$$\hat{K}(d) = \frac{\sum_{\mathrm{i,j}} \mathbb{1}(d_{\mathrm{i,j}} < d)}{N\lambda}, \tag{3.5}$$

where N is the number of events and $d_{\mathrm{i,j}}$ is the distance between the i'th and j'th events. $\mathbb{1}(x)$ is the indicator function with the value 1 if x is true and 0 otherwise. For network events, $d_{\mathrm{i,j}}$ can be defined as the length of the shortest path between node v_{i} and v_{j} where the i'th and j'th events occur. Based on this concept, we propose using a normalized K_1 estimate to describe the second-order property of a sequence in a network. In specific, given a sequence $S = \{t_{\mathrm{i}}, v_{\mathrm{i}}\}$ in a network $G = \{V, E\}$ with $v_{\mathrm{i}} \in V$, we define the following K_1 estimate:

$$K_1(d_m|S, G) = \frac{\sum_{\mathrm{i}=1}^{N-1} \mathbb{1}(d_{\mathrm{i,i+1}} < d_m)}{\sum_{d_m=0}^{d} \sum_{\mathrm{i}=1}^{N-1} \mathbb{1}(d_{\mathrm{i,i+1}} < d_m)} \tag{3.6}$$

where N is the size of the sequence and d is the diameter of the network. d_m can take discrete values from 0 to d, representing the range of all possible network distances between any two nodes. $K_1(d_m|S, G)$ is essentially an estimate of the distribution of the distance between two consecutive events in the network G. Assuming $S' = \{t'_{\mathrm{i}}, v'_{\mathrm{i}}\}$ is an artificial sequence generated by some model in the same network G, we can enforce S' to have the same distance distribution as S by minimizing $D(K_1(d_m|S, G)||K_1(d_m|S', G))$, where $D(P||Q)$ measures the

distance between distribution P and Q (such as Kullback-Leibler divergence, Jensen-Shannon divergence and L2-norm).

3.3 Related Works

Event sequences have been primarily modeled by point processes [106], where predefined intensity functions are used to capture the generative mechanisms. For instance, Hawkes processes [60] model the self/mutual-excitation effect by a time-varying conditional intensity, which is boosted upon the arrival of a new event (Eq. 3.1). Point processes have been widely applied to the areas such as seismology [108, 94], criminology [99, 133], social activity analysis [40, 173], and information diffusion [34, 35]. Despite their success, parametric point process models (such as Eq. 3.2) have some limitations. Most notably, their intensities rely on explicitly defined kernel functions. An inappropriate selection of the kernel function may cause significant degradation of the model. To remedy the problem, [85] proposes an EM-based nonparametric model to estimate the intensity functions without prior knowledge of their form. Furthermore, [176] and [91] extend this method to the multidimensional Hawkes processes. Despite their enhanced flexibility, these models still rely on the assumption that the events are generated by a Hawkes process, and may not perform well for other point processes.

For better generalization, [33, 157] resort to RNN to model arbitrarily distributed event data. In particular, [157] combines two LSTMs that take synchronized time series as well as asynchronous event sequences as inputs to predict both timestamps and marks. But, in their model, the event process does not occur in the vertices of a network. Besides, their model relies on domain-specific features such as ATM logs, which are often not available. Alternatively, [158] replaces RNN with Generative Adversary Net (GAN) [49, 48], specifically, Wasserstein GAN [3]. However, their model is specialized in unmarked temporal point processes and is not capable of predicting event types. Another line of researches [160] focuses on inferring the causal network among different types of events. Our model is different from these models as our networks are constructed beforehand; besides, our causal networks are significantly larger.

3.4 Methods

3.4.1 Problem Description

Given a network $G = \{V, E\}$ where V and E are the collections of vertices and edges, we can represent a sequence of N events on the vertices V before time T by a series of vertex-time pairs, $S = \{(v_i, t_i)|v_i \in V, t_i \in [0, T), t_i < t_{i+1}, i = 1, ..., N\}$. Furthermore, we can extract the vertices from S to form a vertex sequence $\hat{S} = \{v_i|v_i \in V, i = 1, ..., N\}$. In this work, we focus on learning a model to predict the vertex where the next event is most likely to occur based on the previous events. Formally, given a graph $G = \{V, E\}$ and a sequence $\hat{S} = \{v_i|v_i \in V, i = 1, ..., N\}$, we denote

$$u^v_{N+1} = P(v|\hat{S}) \tag{3.7}$$

where u^v_{N+1} is the probability for the $(N+1)$'th event to occur on vertex $v \in V$, and $P(v|\hat{S})$ is the model. We can sample the next event via $v \sim P(v|\hat{S}), v \in V$, and we call this task **event prediction**. Assuming that v'_{N+1} is the $(N+1)$'s event predicted by the model, we can append it to the end of $\hat{S}$, to obtain an extended sequence $\hat{S}' = [\hat{S}, v'_{N+1}]$. We then feed $\hat{S}'$ to Eq. 3.7 to predict the $(N+2)$'th event. By repeating this process, we can generate an artificial sequence of arbitrary length on the graph, and we call this task **sequence generation**. Some key variables adopted in the following section are listed in Table 3.1.

3.4.2 Model Formulation

We propose an architecture consisting of two LSTMs, denoted by LSTM1 and LSTM2. Fig. 3.1 is an illustration of this architecture. Both LSTMs (shaded areas in Fig. 3.1) are deep (multi-layer) and contain a stack of recurrently connected memory cells, each of which performs the calculations in Eq. 3.3. Note that the number of layers and cells in either LSTM are adjustable and not necessarily the same as those in Fig. 3.1. Summarized in Eq. 3.8, a cell in LSTM1 (LSTM2) takes an input $\mathbf{x}^l_i$ ($\mathbf{x}^s_i$), a hidden state $\mathbf{h}^l_i$ ($\mathbf{h}^s_i$) and a cell state $\mathbf{c}^l_i$ ($\mathbf{c}^s_i$), and outputs a new hidden state $\mathbf{h}^l_{i+1}$ ($\mathbf{h}^s_{i+1}$) and a new cell state $\mathbf{c}^l_{i+1}$ ($\mathbf{c}^s_{i+1}$). As shown in Fig. 3.1, these new states are then passed horizontally to the next cell in the

Table3.1. Key variables in this work. $|V|$ is the number of vertices in the graph.

symbol	size	description
k_l	1	number of cells in an LSTM1 layer
k_s	1	number of cells in an LSTM2 layer
d_h	1	LSTM hidden dimension
$\mathbf{y}$	$\|V\| \times 1$	one-hot encoding of a real event
$\mathbf{x}^l$	$k_l \times 1$	event count vector
$\mathbf{x}^s$	$k_s \times 1$	embedding vector of an event
$\mathbf{u}$	$\|V\| \times 1$	predictive distribution
$\mathbf{h}$	$d_h \times 1$	LSTM hidden state
$\mathbf{c}$	$d_h \times 1$	LSTM cell state
$\mathbf{D}$	$\|V\| \times \|V\|$	network distance matrix
$\mathbf{W}$	$\|V\| \times (\|V\| + d_h)$	weight (fully-connected layer)
$\mathbf{b}$	$\|V\| \times 1$	bias (fully-connected layer)

same layer and vertically to the cell in the next layer. The final outputs of the two LSTMs are concatenated and fed to a fully-connected layer with a Softmax activation, to generate a predictive distribution $\mathbf{u}_{\mathrm{i}+1}$ over the vertices (V) for the next event (Eq. 3.8).

$$
\begin{aligned}
(\mathbf{h}^l_{\mathrm{i}+1}, \mathbf{c}^l_{\mathrm{i}+1}) &= cell_{LSTM1}(\mathbf{x}^l_{\mathrm{i}}, \mathbf{h}^l_{\mathrm{i}}, \mathbf{c}^l_{\mathrm{i}}) \\
(\mathbf{h}^s_{\mathrm{i}+1}, \mathbf{c}^s_{\mathrm{i}+1}) &= cell_{LSTM2}(\mathbf{x}^s_{\mathrm{i}}, \mathbf{h}^s_{\mathrm{i}}, \mathbf{c}^s_{\mathrm{i}}) \\
\mathbf{e}_{\mathrm{i}} &= [\mathbf{h}^l_f, \mathbf{h}^s_{f,\mathrm{i}}] \\
\mathbf{u}_{\mathrm{i}+1} &= Softmax(\mathbf{W}\mathbf{e}_{\mathrm{i}} + \mathbf{b})
\end{aligned}
\tag{3.8}
$$

LSTM1 takes long-term event counts as input and learns the slowly varying background rate (similar to $\mu_v(t)$ in Eq. 3.1). We define an event count vector $\mathbf{x}^l_{\mathrm{i}} = [x^l_{\mathrm{i},v}], v \in V$, with $x^l_{\mathrm{i},v}$ representing the number of events on node v during $[(\mathrm{i}-1)\Delta t, \mathrm{i}\Delta t)$. The inputs of LSTM1 are $\mathbf{x}^l_{\mathrm{i}}$, $\mathbf{x}^l_{\mathrm{i}+1}$, $\mathbf{x}^l_{\mathrm{i}+2}$, ..., $\mathbf{x}^l_{\mathrm{i}+k_l-1}$, corresponding to the latest k_l intervals. Note these pre-computed vectors are not trainable. LSTM2, on the other hand, learns the short-range correlation (resembling the triggering kernels in Eq. 3.1). Given that the latest k_s events occur on nodes v_{i}, $v_{\mathrm{i}+1}$, ..., $v_{\mathrm{i}+k_s-1}$, the inputs of LSTM2 are their embedding vectors $\mathbf{x}^s_{\mathrm{i}}$, $\mathbf{x}^s_{\mathrm{i}+1}$, $\mathbf{x}^s_{\mathrm{i}+2}$, ..., $\mathbf{x}^s_{\mathrm{i}+k_s-1}$. Note these embedding vectors are learned during training by back-propagation.

Let $\mathbf{h}^l_f$ denote the hidden state of the last cell in LSTM1, and $\{\mathbf{h}^s_{f,\mathrm{i}}, ..., \mathbf{h}^s_{f,\mathrm{i}+k_s-1}\}$ denote the hidden states in the last layer of LSTM2. Summarized in the last two equations in

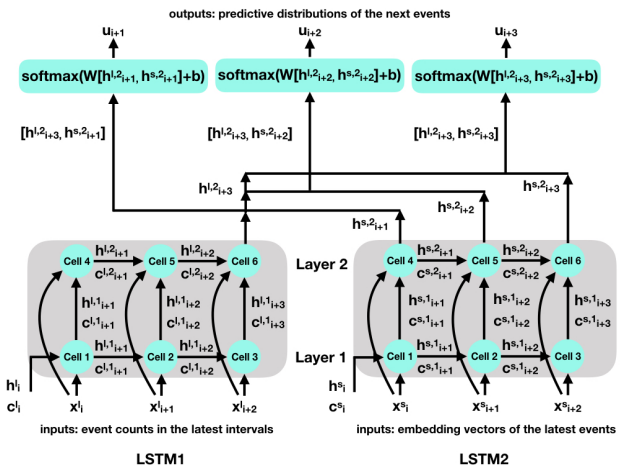


Figure3.1. Network Architecture. The shaded areas represent LSTMs; while the cyan circles represent cells. Each LSTM contains two layers.

Eq. 3.8, concatenating $\mathbf{h}_f^l$ to each element in $\{\mathbf{h}_{f,\mathrm{i}}^s, \mathbf{h}_{f,\mathrm{i+1}}^s, ..., \mathbf{h}_{f,\mathrm{i}+k_s-1}^s\}$, we obtain a series of vectors $\{\mathbf{e}_\mathrm{i}, \mathbf{e}_{\mathrm{i}+1}, .., \mathbf{e}_{\mathrm{i}+k_s-1}\}$, which are then fed to the Softmax layer to generate a series of vectors $\{\mathbf{u}_{\mathrm{i}+1}, \mathbf{u}_{\mathrm{i}+2}, ..., \mathbf{u}_{\mathrm{i}+k_s}\}$, representing the probability distributions over the vertices for the next events. Let $\{\mathbf{y}_{\mathrm{i}+1}, \mathbf{y}_{\mathrm{i}+2}, ..., \mathbf{y}_{\mathrm{i}+k_s}\}$ denote the one-hot encoding of the nodes where the real events occur. The cross-entropy loss can thus be written as the following:

$$L_C = -\frac{1}{k_s}\sum_{k=1}^{k_s}\sum_{\mathrm{j}=1}^{|V|} y_{\mathrm{i}+k}^{\mathrm{j}} log(u_{\mathrm{i}+k}^{\mathrm{j}}), \tag{3.9}$$

where $y_{\mathrm{i}+k}^{\mathrm{j}}$ and $u_{\mathrm{i}+k}^{\mathrm{j}}$ are the j'th elements of $\mathbf{y}_{\mathrm{i}+k}$ and $\mathbf{u}_{\mathrm{i}+k}$, respectively. The summation is over the number of steps k_s and the number of nodes $|V|$ in the network.

As mentioned in Sec. 3.2.3, by minimizing $D(K_1(d_m|S,G)||K_1(d_m|S',G))$, we could ensure that the generated sequence S' has the same distance distribution as the real sequence S. However, implementing K_1 in a neural network setting is not trivial. In particular, the right-hand side of Eq. 3.6 is not differentiable. If our loss function contains Eq. 3.6, we would not be able to perform gradient descent. To overcome this problem, we smooth Eq. 3.6 using Kernel Density Estimation (KDE) with a Gaussian kernel:

$$K_1(d_m|S,G) = \frac{\sum_{\mathrm{i}=1}^{N-1} exp(-(\frac{d_{\mathrm{i},\mathrm{i}+1}-d_m}{h})^2)}{\sum_{d_m=0}^{d}\sum_{\mathrm{i}=1}^{N-1} exp(-(\frac{d_{\mathrm{i},\mathrm{i}+1}-d_m}{h})^2)}, \tag{3.10}$$

where $h > 0$ is the bandwidth. With a small enough h, Eq. 3.10 is a good approximation to Eq. 3.6. Most importantly, Eq. 3.10 is differentiable and thus can be used to construct a loss function.

Furthermore, we can calculate $d_{\mathrm{i,i+1}}$ efficiently by matrix multiplication. We pre-compute a distance matrix $\mathbf{D}$ with entry $D_{\mathrm{i,j}}$ representing the distance between node v_{i} and v_{j}. Let $d_{\mathrm{i,i+1}}$ denote the distance between the real events v_{i} and $v_{\mathrm{i+1}}$ and $d'_{\mathrm{i,i+1}}$ denote the distance between the generated events v'_{i} and $v'_{\mathrm{i+1}}$, then we have

$$\begin{aligned} d_{\mathrm{i,i+1}} &= \mathbf{y}_{\mathrm{i}}^T \mathbf{D} \mathbf{y}_{\mathrm{i+1}}, \\ d'_{\mathrm{i,i+1}} &= \mathbf{u}_{\mathrm{i}}^T \mathbf{D} \mathbf{u}_{\mathrm{i+1}}, \end{aligned} \tag{3.11}$$

where $\mathbf{y}_{\mathrm{i}}$ and $\mathbf{y}_{\mathrm{i+1}}$ are the one-hot encoding of the real events at step i and $\mathrm{i}+1$, and $\mathbf{u}_{\mathrm{i}}$ and $\mathbf{u}_{\mathrm{i+1}}$ are predictive distributions over events at step i and $\mathrm{i}+1$.

Plugging Eq. 3.11 into Eq. 3.10, we obtain

$$K_1(d_m|S,G) = \frac{\sum_{\mathrm{i}=1}^{N-1} exp(-(\frac{\mathbf{y}_{\mathrm{i}}^T \mathbf{D} \mathbf{y}_{\mathrm{i+1}} - d_m}{h})^2)}{\sum_{d_m=0}^{d} \sum_{\mathrm{i}=1}^{N-1} exp(-(\frac{\mathbf{y}_{\mathrm{i}}^T \mathbf{D} \mathbf{y}_{\mathrm{i+1}} - d_m}{h})^2)}, \tag{3.12}$$

for a real sequence S, and

$$K_1(d_m|S',G) = \frac{\sum_{\mathrm{i}=1}^{N-1} exp(-(\frac{\mathbf{u}_{\mathrm{i}}^T \mathbf{D} \mathbf{u}_{\mathrm{i+1}} - d_m}{h})^2)}{\sum_{d_m=0}^{d} \sum_{\mathrm{i}=1}^{N-1} exp(-(\frac{\mathbf{u}_{\mathrm{i}}^T \mathbf{D} \mathbf{u}_{\mathrm{i+1}} - d_m}{h})^2)}, \tag{3.13}$$

for a generated sequence S'.

Accordingly, we define the following loss function:

$$L_K(S,S'|G) = D(K_1(d_m|S',G)||K_1(d_m|S,G)), \tag{3.14}$$

Algorithm 2: Composite LSTM with Second-order Statistic Loss

model M with optimal set of parameters Θ;
for *epoch = 1 to n* **do**
  for *batch = 1 to m* **do**
    for *i = 1 to o* **do**
      Initialize $\mathbf{h}_0^l = \mathbf{c}_0^l = \mathbf{h}_0^s = \mathbf{c}_0^s = \mathbf{0}$;
      $\mathbf{H}_{\mathrm{i}}^l \leftarrow LSTM1(\mathbf{x}_{\mathrm{i}}^l, ..., \mathbf{x}_{\mathrm{i}+k_l-1}^l, \mathbf{h}_0^l, \mathbf{c}_0^l)$;
      $\mathbf{h}_{\mathrm{i}+1}^l, ..., \mathbf{h}_{\mathrm{i}+k_l}^l \leftarrow \mathbf{H}_{\mathrm{i}}^l$;
      $\mathbf{H}_{\mathrm{i}}^s \leftarrow LSTM2(\mathbf{x}_{\mathrm{i}}^s, ..., \mathbf{x}_{\mathrm{i}+k_s-1}^s, \mathbf{h}_0^s, \mathbf{c}_0^s)$;
      $\mathbf{h}_{\mathrm{i}+1}^s, ..., \mathbf{h}_{\mathrm{i}+k_s}^s \leftarrow \mathbf{H}_{\mathrm{i}}^s$;
      for *j = 1 to* k_s **do**
        $\mathbf{e}_{\mathrm{i+j}} \leftarrow Concatenate(\mathbf{h}_{\mathrm{i}+k_l}^l, \mathbf{h}_{\mathrm{i+j}}^s)$;
        $\mathbf{u}_{\mathrm{i+j}} \leftarrow Softmax(\mathbf{W}\mathbf{e}_{\mathrm{i+j}} + \mathbf{b})$;
        $d_{\mathrm{i+j-1,i+j}} \leftarrow \mathbf{y}_{\mathrm{i+j-1}}^T \mathbf{D} \mathbf{y}_{\mathrm{i+j}}$;
        $d'_{\mathrm{i+j-1,i+j}} \leftarrow \mathbf{u}_{\mathrm{i+j-1}}^T \mathbf{D} \mathbf{u}_{\mathrm{i+j}}$;
      end
    end
    Calculate loss L using Eq. 3.15 and update $\mathbf{W}$, $\mathbf{b}$, embedding, and the trainable parameters in Eq. 3.3 using gradient descent.
  end
end

where $D(P||Q)$ could be KL-divergence, JS-divergence or L2-norm. Overall, we propose a total loss function of the following form:

$$\begin{aligned} L &= L_C + \lambda L_K \\ &= -\frac{1}{k_s} \sum_{k=1}^{k_s} \sum_{\mathrm{j}=1}^{|V|} y_{\mathrm{i}+k}^{\mathrm{j}} log(u_{\mathrm{i}+k}^{\mathrm{j}}) \\ &+ \lambda D(K_1(d_m|S', G)||K_1(d_m|S, G)), \end{aligned} \tag{3.15}$$

where λ is a hyper-parameter for regularization. k_s is the number of cells in a layer of LSTM2. The second term on the right (regularization) is obtained by plugging Eq. 3.12 and 3.13 into Eq. 3.14.

3.4.3 Training Protocol

In Algorithm 2, we show the pseudo-code of our training procedure. A sample of input contains k_s embedding vectors $\mathbf{x}_{\mathrm{i}}^s$ representing the current window of events. It also contains k_l event count vectors $\mathbf{x}_{\mathrm{i}}^l$ that provide the background information of the current window. At the beginning of each window, the hidden states and the cell states are initialized as zero vectors, which are passed to the LSTMs along with the inputs at various steps (intervals). The final hidden state ($\mathbf{h}_{\mathrm{i}+k_l}^l$) of LSTM1 is concatenated with each of the k_s hidden state of LSTM2. The resulting vectors $\mathbf{e}_{\mathrm{i}+\mathrm{j}}$ then go through a fully-connected layer with Sofmax activation, to generate the predictive distributions $\mathbf{u}_{\mathrm{i}+\mathrm{j}}$. Meanwhile, the network distances between consecutive events, $d_{\mathrm{i}+\mathrm{j}-1,\mathrm{i}+\mathrm{j}}$ (real) and $d'_{\mathrm{i}+\mathrm{j}-1,\mathrm{i}+\mathrm{j}}$ (predicted) are retrieved from a precomputed hop-distance matrix $\mathbf{D}$. At the end of a batch, the total loss function is computed using Eq. 3.15 and its gradient is evaluated and averaged over the entire batch. The gradient descent optimizer is adopted to update the trainable parameters, including the embedding, the parameters in the LSTMs and those in the fully-connected layer.

3.5 Experiment

We run several experiments to validate the effectiveness of our proposed model and to compare the performance of our model over a set of competing models. In our first experiment, we perform **event prediction**, using the proposed model to predict the vertices of a network in which future events will occur. In detail, we feed event counts at each vertex in the past 30 ($k_l = 30$) successive intervals $\{\mathbf{x}_{\mathrm{i}}^l, \mathbf{x}_{\mathrm{i}+1}^l, ..., \mathbf{x}_{\mathrm{i}+29}^l\}$ to LSTM1 and the embedding vectors of 32 ($k_s = 32$) historical events $\{\mathbf{x}_{\mathrm{i}}^s, \mathbf{x}_{\mathrm{i}+1}^s, ..., \mathbf{x}_{\mathrm{i}+31}^s\}$ to LSTM2. Note that the selections of k_l and k_s are heuristic. In general, using values too small might reduce the model's capacity, whereas using values too large might make the training very expensive. The output $\mathbf{u}_{\mathrm{i}+32}$ gives a probability distribution for an event to occur at each vertex at step $\mathrm{i} + 32$. To evaluate our predictions, we calculate a series of hit rates (hit@10, hit@20, and hit@30). For instance, if the true event $v_{\mathrm{i}+32}$ is among the top 10 most probable vertices given by $\mathbf{u}_{\mathrm{i}+32}$, we consider it as a hit. We slide a window of 32 events through the entire test set and obtain the average hit rates to indicate the prediction performance.

Table3.2. Dataset properties. $|V|$ and $|E|$ are the number of nodes and number of edges of each network, respectively. Sequence size gives the number of events in each sequence.

dataset	$\|V\|$	$\|E\|$	sequence size
Rand-1	291	290	10000
Rand-2	1599	1940	10000
Earthquake	648	41744	17381
Email	2634	6458	14092
Twitter	393	777	18879

Our second experiment is **sequence generation**. Similar to what we did in event prediction, 30 ($k_l = 30$) event count vectors and 32 ($k_s = 32$) embedding vectors are fed to LSTM1 and LSTM2, to generate a predictive distribution for the next event. However, here we sample the next event at step $\mathrm{i}+32$ from a multinomial distribution given by $\mathbf{u}_{\mathrm{i}+32}$. Assuming that the event that we draw is $v'_{\mathrm{i}+32}$, we then append it to the past 31 events, resulting a new list of 32 events $\{v_{\mathrm{i}+1}, v_{\mathrm{i}+2}, ..., v'_{\mathrm{i}+32}\}$. Here we essentially keep a queue of 32 successive events, where the oldest event leaves the queue when a newly sampled event enters the queue. We then feed the embedding vectors and event count vectors of the current queue to the model to obtain the probability distribution for the next event at step $\mathrm{i}+33$. By repeating this process, we can generate a fake sequence of arbitrary length. To see how realistic the generated sequences are, we compare them with the real sequences in terms of diffusion pattern.

We also perform additional experiments for validating model convergence, and robustness, which are presented in the Supplement. Below, we first discuss the datasets on which we run our experiments and the competing models with which we compare our model.

3.5.1 Data Description

We use three real-world datasets, Earthquake, Email and Twitter, and two synthetic datasets, Rand-1, and Rand-2. Each of these is a composition of a graph and a node sequence on which an event occurred. For all datasets, 70% of the sequence from its prefix

is used for training and the remaining part of the sequence is used for test. Statistics of the dataset is provided in Table 6.1. More discussion of the datasets is provided below:

Earthquake: [123] contains the location, time and magnitude of earthquakes that occurred in Southern California. We construct a network based on the Community Fault Model 3.0 [115], which is a 3D representation (latitude, longitude, and elevation) of faults in Southern California. Specifically, we sample every 100 points from each fault line and add an edge between two points if their distance is less than 40 kilometers. The resulting network contains 648 nodes and 41744 edges (Table 6.1). We collect earthquakes from 1997 to 2018 with a magnitude of at least 2.5 and map them to the nearest location (node) in the network. Consequently, we obtain a sequence of 17381 earthquakes that occur on the nodes of the fault network.

Email: Enron email is a publicly available dataset that contains $\sim 500,000$ emails generated by employees of the Enron Corporation. We use email addresses owned by Enron employees as nodes, and add an edge between two nodes if at least one email has been exchanged between the corresponding addresses. Since the resulting network is not connected, we select the largest connected component that contains 2634 nodes and 6458 edges as our final network (Table 6.1). We extract the sender address and timestamp from each email across the entire corpse. The sender-time pairs are then sorted in time ascending order. The resulting sequence contains 14092 email sending events (sender-time pairs).

Twitter: This dataset contains Twitter data collected during the presidential election in South Africa in 2014. We define a tweet as popular if it is retweeted more than 10 times, and a user as popular if she has posted a popular tweet. We use the popular users as nodes to construct a network. We add an edge (undirected) between two nodes if one of them has mentioned the other in a popular tweet. The resulting network is not connected. We thus select the largest connected component that has 393 nodes and 777 edges (Table 6.1). We consider that an event occurs on a node when the corresponding user posts a tweet. We gather a series of such events in time ascending order. The resulting sequence contains 18879 events (user-time pairs).

Rand-1 & Rand-2: We use Erdős-Rényi model to generate random graphs $G(n, p)$ with n nodes, where an edge is connected randomly with probability p independent of every

other edge. First, we generate two random graphs $G(n = 2000, p = 0.0001)$ and $G(n = 2000, p = 0.001)$, then we extract the largest connected components from these random graphs, rendering two connected graphs with ($|V| = 291$, $|E| = 290$) and ($|V| = 1599$, $|E| = 1940$), respectively (Table 6.1). Next, we simulate two event sequences on these two graphs using multi-dimensional Hawkes Process [163]. Specifically, these sequences are generated with predefined base intensities $\mu_0^v = 10^{-3}u/|V|, u \sim U(0,1), v \in V$ and decay rate $w = 1$. We use adjacency matrix for W_{v,v_i} in Eq. 3.2. Both sequences contain $N = 10^4$ events.

3.5.2 Competing Methods

We compare our proposed model with the following competing methods. All methods were given the identical experimental setup (same data input) to maintain fairness.

DeepWalk: DeepWalk [113] learns latent representations of nodes in a network using truncated random walks. We apply DeepWalk to each of the networks with the embedding dimension in $\{64, 128, 256\}$, the walk length in $\{20, 40, 60\}$, and the window size in $\{5, 10, 15\}$. We adopt Multilayer Perceptron (MLP) (hidden dimension $\{256, 128\}$) with Softmax activation as the classifier in our **DeepWalk-dense** model. The classifier takes the embedding vectors of 32 historical events, and outputs the probability distribution of the next event. Additionally, we use a Convolutional Neural Network (CNN) as the classifier in our **DeepWalk-cnn** model. The classifier contains two CNN layers (hidden dimension $\{256, 128\}$), and two fully-connected layers (hidden dimension $\{100, |V|\}$). We use a filter size of 3, a stride of 1, and a max-pooling size of 2. We apply dropout of 0.5 in both DeepWalk-dense and DeepWalk-cnn.

Node2Vec: Node2Vec [53] uses biased random walks to learn node embeddings in a network. We apply Node2Vec to each of the networks with the embedding dimension in $\{64, 128, 256\}$, and the return parameter p and the in-out parameter q in $\{0.5, 1.0., 1.5\}$. Similar to DeepWalk, we build classifiers using MLP (**Node2Vec-dense**) and CNN (**Node2Vec-cnn**). The classifiers have the same architecture and hyper-parameters as those of DeepWalk.

Random Walk: Given the current event, the next event is predicted by performing a random walk. Precisely, let v_{i} denotes the current node, the next node $v_{\mathrm{i}+1}$ is chosen uniformly at random from $N(v_{\mathrm{i}}) \bigcup \{v_{\mathrm{i}}\}$, where $N(v_{\mathrm{i}})$ is the set of neighbors of v_{i}.

Hawkes Processes (Hawkes-Exp): We fit a multi-dimensional Hawkes process using Eq. 3.2 with constant base intensity μ_0 and exponential kernels. In specific, we use a maximum likelihood estimator (MLE) with a sparse-group-lasso regularizer [163]. We test all combinations of hyper-parameters given decay rate w in $\{0.1, 0.5, 1.0\}$ and regularizer in $\{0.1, 1.0, 10.0\}$. For better performance, the original sequences are cut into subsequences based on constant time intervals. For a fair comparison, we feed the Hawkes process model with 32 historical events, the same as the number of cells in LSTM2. Therefore, the results of Hawkes-Exp should not be considered as the upper limit for Rand-1 or Rand-2, despite that they were generated using Hawkes processes.

RNN for Point Processes (RNNPP): [157] is an RNN model consist of two LSTMs that combine synchronized time series and asynchronous event sequences. The model is primarily designed for maintenance support services problem and relies on domain-specific features such as ATM logs. Without domain-specific features, we use sequences of constant features (e.g. vectors of ones) instead.

Logistic Classification (Logistic): The model is a multi-class logistic regression classifier with cross-entropy loss. It virtually fits $Softmax(\mathbf{W}\mathbf{x} + \mathbf{b}) = \mathbf{y}$ and resembles the last layer of our LSTM model (Fig. 3.1); $\mathbf{x}$ contains event counts on each node in a given window, and $\mathbf{y}$ indicates the identity of the next event.

3.5.3 Hyper-parameter Tuning and Sensitivity

In the experiments, we tune the following hyper-parameters across all the datasets: λ in $\{0.1, 1, 10, 100, 1000\}$; learning-rate in $\{0.01, 0.1, 1.0, 10\}$ with a decay rate of 0.9; embedding dimension in $\{64, 128, 256, 512\}$. For the LC+LK model, the results shown in Table 3.5 are obtained using the optimal parameters. For the LC model, the hit rates are obtained using the same parameters as the LC+LK model without the second-order constraint.

Table3.3. Embedding Dimension Sensitivity. The values are Hit@10 rates with different embedding dimensions on each dataset.

dataset	64	128	256	512
rand-1	0.531	**0.532**	0.527	0.527
rand-2	0.234	**0.241**	0.233	0.234
earthquake	**0.648**	0.591	0.560	0.503
email	0.302	0.301	**0.306**	0.284
twitter	0.649	0.652	**0.655**	0.654

Table3.4. Learning Rate Sensitivity. The values are Hit@10 rates with different learning rates on each dataset.

dataset	0.01	0.1	1.0	10
rand-1	**0.532**	0.509	0.524	0.227
rand-2	**0.241**	0.142	0.139	0.081
earthquake	0.627	0.628	**0.648**	0.538
email	**0.306**	0.272	0.186	0.193
twitter	**0.655**	0.607	0.585	0.607

The rest of the parameters are fixed. We use a batch-size of 32 and a dropout of 0.5. The number of layers in each LSTM is 2 and the numbers of cells in a layer of LSTM1 and LSTM2 are 30 and 32, respectively. Event counts are calculated using time intervals of 0.01 seconds (Rand-1 and Rand-2), 1 day (Email and Twitter), and 30 days (Earthquake). We adopt KL-divergence for all the datasets to evaluate the distance between two distributions (Eq. 3.14).

We examine the impact on the performance of changing some key hyper-parameters. For each dataset, we vary these hyper-parameters while keeping the other hyper-parameters as their optimal values. We use Hit@10 rate as an indicator to illustrate the performance. In Table 3.3, we show Hit@10 values under different embedding dimensions. The results show that our model is insensitive to the change of the embedding dimension. We also test the effect of changing the learning rates in Table 3.4. Here we can see that a large learning rate may cause significant degradation of the performance; for instance, using a learning rate of 10 in the rand-2 dataset reduces the performance by more than 66%.

Table3.5. Experimental Results.

Model	Rand-1			Rand-2			Earthquake			Email			Twitter		
	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30	Hit@10	Hit@20	Hit@30
LC	0.527	0.768	0.879	0.237	0.352	0.437	0.638	0.715	0.756	0.293	**0.412**	0.465	0.652	0.708	0.740
LC + LK	**0.532**	**0.769**	**0.886**	**0.241**	**0.358**	**0.442**	**0.648**	**0.722**	**0.768**	**0.306**	0.411	**0.467**	**0.655**	**0.711**	**0.742**
Node2Vec-dense	0.420	0.655	0.786	0.148	0.251	0.337	0.494	0.589	0.672	0.176	0.248	0.298	0.533	0.603	0.650
Node2Vec-cnn	0.449	0.724	0.872	0.165	0.307	0.410	0.452	0.585	0.649	0.162	0.233	0.282	0.507	0.555	0.605
DeepWalk-dense	0.403	0.627	0.756	0.150	0.245	0.331	0.526	0.625	0.677	0.182	0.256	0.309	0.540	0.610	0.655
DeepWalk-cnn	0.423	0.697	0.863	0.168	0.292	0.384	0.478	0.579	0.667	0.148	0.233	0.293	0.511	0.580	0.631
Random-Walk	0.013	0.016	0.066	0.005	0.015	0.016	0.300	0.364	0.370	0.003	0.009	0.013	0.143	0.159	0.175
Hawkes-Exp	0.503	0.721	0.827	0.196	0.302	0.362	0.622	0.672	0.697	0.257	0.325	0.355	0.599	0.640	0.664
RNNPP	0.348	0.717	0.826	0.208	0.308	0.375	0.376	0.493	0.554	0.249	0.359	0.406	0.587	0.649	0.692
Logistic	0.005	0.113	0.147	0.015	0.029	0.037	0.122	0.197	0.200	0.002	0.004	0.005	0.008	0.025	0.045

3.5.4 Model Convergence

In Fig. 3.2, we plot the cost for LC model and LC + LK model with the five datasets. The cost is the cross-entropy loss (Eq. 3.9) for the LC model and the combination of the cross-entropy loss and the second-order statistic loss (Eq. 3.15) for the LC + LK model. The curves here indicate a convergence of the training. A few spikes appear during the early stage of the LC + LK curves, which is likely due to a finite batch size (= 32).

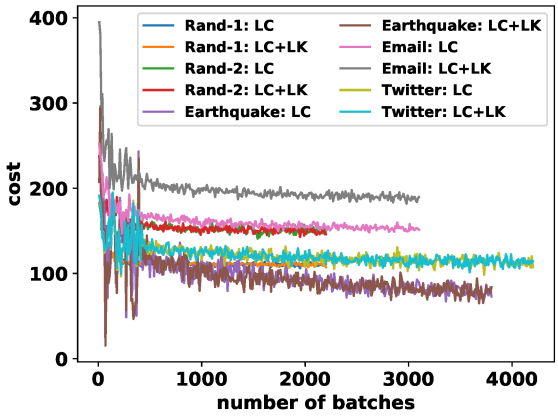
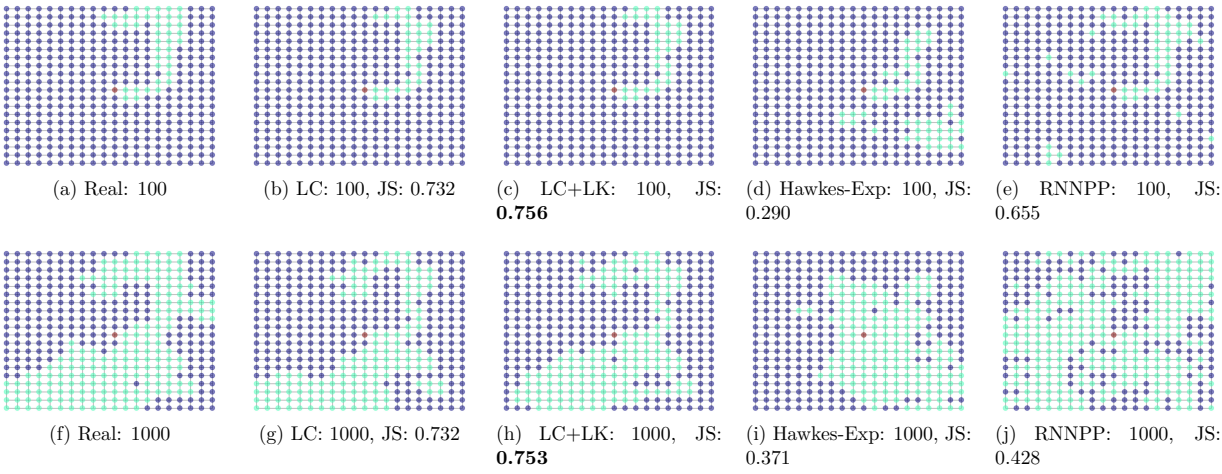


Figure3.2. Cost for LC and LC + LK with different datasets.

3.5.5 Results

Event prediction. In Table 3.5, we show the comparison results of the event prediction task for ten methods over five datasets using hit rate (@10, @20, and @30) as the evaluation metric. In the table, LC is our basic LSTM model, while LC + LK is our LSTM model with the second-order statistic loss. Remaining eight are competing methods. As we can see from



(a) Real: 100 (b) LC: 100, JS: 0.732 (c) LC+LK: 100, JS: **0.756** (d) Hawkes-Exp: 100, JS: 0.290 (e) RNNPP: 100, JS: 0.655

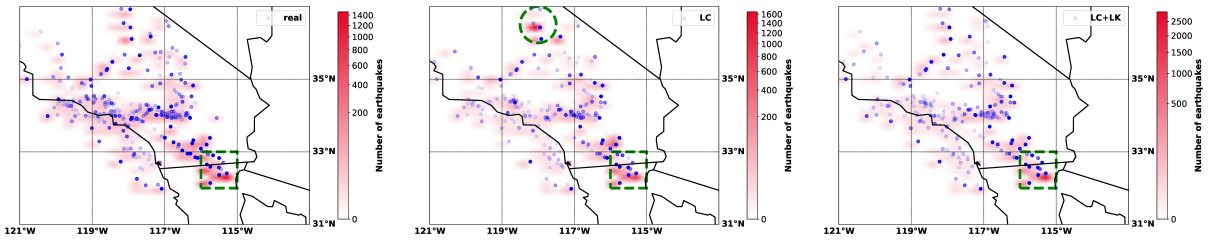
(f) Real: 1000 (g) LC: 1000, JS: 0.732 (h) LC+LK: 1000, JS: **0.753** (i) Hawkes-Exp: 1000, JS: 0.371 (j) RNNPP: 1000, JS: 0.428

Figure3.3. Diffusion on grids. The sequences start from the center (red) of the grids. The green dots represent the nodes where events have occurred; while the purple dots represent the grid. The top and bottom rows are snapshots taken at time step 100 and 1000, respectively. Images from left to right represent different models. JS denotes the Jaccard Similarity between a generated sequence and a real sequence for each snapshot.

Table3.6. Jaccard Similarity between a generated sequence and a real sequence for snapshots at different steps. The scores are in the form of mean +/- standard deviation, which are estimated over 100 experiments.

Step	LC	LC+LK	Hawkes-Exp	RNNPP
100	**0.754+/-0.108**	0.747+/-0.098	0.298+/-0.030	0.656+/-0.019
500	0.661+/-0.061	**0.673+/-0.058**	0.282+/-0.058	0.333+/-0.018
1000	0.755+/-0.039	**0.756+/-0.045**	0.435+/-0.066	0.414+/-0.018

the table, for all the datasets and for all different hit rates, LC + LK is the best method (except for Email for hit@20). Our proposed LC and LC + LK models win over Hawkes-Exp even for the random datasets in which events actually follow a Hawkes distribution. We can also see that our models outperform the embedding based methods where latent representations of the nodes are learned from the network topology. Our LC+LK model's performance is better than that of the LC model in all cases (except for Email for hit@20), which suggests that adding the second-order statistic loss can improve the model for predicting the future event.



(a) Real (b) LC, corr: 0.515 (c) LC+LK, corr: **0.936**

Figure3.4. Earthquakes in Southern California. Blue circles represent earthquake locations; red heatmaps indicate the number of earthquakes at each location. The correlation is between the event count distributions of the real and generated sequences.

Sequence generation. We also test if our model can learn the correct diffusion of the real sequence. We simulate a sequence on a 20×20 grid such that the event marks (nodes) are determined by a simple symmetric random walk and the event timestamps are determined by a homogeneous Poisson process with a rate of 10. We train various models with this sequence and generate fake sequences using them. We illustrate the diffusion processes in Fig. 3.3. The snapshots are taken at the 100'th and 1000'th time-step. The green nodes represent the nodes that have been visited by the sequences, while the purple nodes represent the grid. We calculate the Jaccard Similarity (JS under each image) between a generated sequence and a real sequence for each snapshot. A higher score indicates that the generated sequence better mimics the real sequence. In the snapshots, LC (2nd column) and LC+LK (3rd column) show similar patterns as the real sequence (1st column). In contrast, the last two columns suggest that Hawkes-Exp may be too conservative; while RNNPP may be too aggressive. In Table 3.6, we list the mean and standard deviation of Jaccard Similarity scores over 100 experiments. The snapshots are sampled at step 100, 500, and 1000. We can see that our LC and LC+LK models outperform the other two methods significantly. Notably, the LC+LK model scores the highest mean Jaccard Similarity in two out of the three cases. Overall, our models generate more realistic sequences than the competing methods, and by adding the second-order statistic constraint, the LC+LK model better captures the characteristics of the real sequence than the LC model.

Case study. We show the effectiveness of our model in the earthquake forecasting task. As mentioned in the previous section, we map earthquakes in Southern California during $1997 \sim 2018$ to a network constructed based on the fault lines (CFM3). After training, we use both LC and LC+LK models to generate fake sequences of the same length as the test data. In Fig. 3.4, we mark the locations of earthquakes (blue circles) on the map of Southern California. From left to right, the figures represent the real sequence (test data), the fake sequences generated by the LC model and the LC+LK model. On the same map, we indicate the number of earthquakes at each location using a heatmap (in red). Notably, the LC+LK heatmap captures the hot spot at $(32.5°N, 115.5°W)$ (green square). In contrast, the LC heatmap shows an additional hot spot at $(36°N, 118°W)$ (green circle) which is not in the real heatmap. We also estimate the correlation between the event count distributions of the real and the generated sequences. The LC+LK model renders a 0.936 Pearson correlation, while the LC model only scores 0.515. Therefore, using second-order statistic constraint can significantly enhance the model for generating more realistic sequences.

Model reproducibility. We make our datasets and code available at https://github.com/daDiz/LSTM2-2ndStat.

3.6 Chapter Summary

In this work, we proposed an LSTM based model for the task of modeling event sequences on network vertices. We integrated structural information of the network into conventional LSTM models to achieve improved performance. Specifically, we introduced a second-order statistic loss that measures the difference between distance distributions of the generated sequence and the target sequence. Moreover, we proposed an architecture that combines two LSTMs to learn both the slowly varying base intensities and the fast varying triggering kernels. We tested our model on synthetic and real-world datasets and illustrated its superior performance in forecasting future events.

4. SOURCE DETECTION ON NETWORKS USING SPATIAL TEMPORAL GRAPH CONVOLUTIONAL NETWORKS

A version of this chapter is pending publication in Data Science and Advanced Analytics for Smart & Connected Communities - a Special Session of the 8th IEEE International Conference on Data Science and Advanced Analytics (DSAA 2021) with Hao Sha, Mohammad Al Hasan, George Mohler.

4.1 Introduction

By early January 2021, the number of confirmed COVID-19 cases has reached 83.6 millions world-wide, and over 1.8 million people have lost their lives. One important method for limiting transmission of an infectious disease consists of identifying epidemic cluster sources and isolating them from the population. Epidemiologists conduct source detection by analysing the genetic evolution of virus strains [143] or by contact tracing [4], which can be time-consuming and labor-demanding. However, COVID-19 has demonstrated limits to contact tracing when prevalence is widespread, for example in the United States, and methods are needed for source detection in such situations.

In real life, most individuals have high probability of contact with only a small portion of the population. It is thus realistic to model an epidemic as a spreading process on an interpersonal network, where infection can only transmit from an individual to its neighbors. Moreover, the interpersonal network can be obtained through IoT technology [66]. For example, [41] reconstructs contact networks through mobile phone communication data. So in this work, we solve the source detection problem in a network setting and assume the availability of the network.

Source detection on networks is a well studied problem with wide applications [67]. Existing solutions include centrality based methods [127, 116, 180], Dynamic Message Passing (DMP) [89] and Label Propagation based Source Identification (LPSI) [155]. Recently, [126] proposed using graph convolutional network (GCN) to solve source detection with improved efficiency and accuracy. The method takes as input a snapshot of the spreading and out-

puts a probability for each node in the graph as to whether the node is the source. There appears to be some controversy [110] as to whether a GCN approach is valid compared to more standard (non-deep learning based) methods for source detection. In particular, some discussion on the 2021 ICLR open review portal questions whether a GCN approach can handle diverse and realistic transmission dynamics (beyond SIR), diverse network topology, and can even be solved using a single snapshot.

In practice, there can possibly be multiple snapshots observed at different stages of an epidemic; these snapshots can help revealing the temporal dynamics of the disease propagation. As suggested in [154], multiple independent snapshots can enhance detectability. So an ideal model should be able to take advantage of richer observations and at the same time exploit the underlining connectivity of the network. This motivates us to adopt a spatial temporal graph convolutional network (STGCN) architecture that combines the best of the two worlds. STGCNs were originally developed for the tasks of traffic forecasting [170] and action recognition [165], where the data contains sequences of temporal snapshots of route networks and skeleton networks, respectively. We adapt the particular form of STGCN proposed in [170] for source detection, and name it Source-Detection-STGCN (SD-STGCN). In this work we show that single snapshot GCNs [126] do not perform significantly better than simpler message passing algorithms, however multi-snapshot STGCNs do lead to significant accuracy improvements. We validate these findings using more realistic models of transmission, including non-Markovian epidemic simulations with delay, and more realistic network topology, including both synthetic and empirical contact networks. Additionally, we apply our model to real COVID-19 case data. The experiments demonstrate the superior performance of SD-STGCN.

4.2 Background

4.2.1 Epidemic Models

SD-STGCN is a deep learning model that requires abundant data to train. However, real infection records are very limited in the public domain. So we resort to epidemic models to generate simulations that resemble the real contagion processes. It is worth noting that

although SD-STGCN is trained on simulation data, it is independent with the particular epidemic models, and thus can be applied to more complex and realistic cases.

SIR [69, 31] splits the population into three compartments - susceptible (S), infectious (I), and recovered (R). Let $S(t)$, $I(t)$, and $R(t)$ denote the ratios of individuals who are susceptible, infectious, and recovered, respectively, at time t. They follow the transformation rule: $S \to I \to R$, and satisfy $S(t) + I(t) + R(t) = 1$, assuming a close system. The ordinary differential equations of the system are given by:

$$\frac{dS}{dt} = -\beta IS, \qquad \frac{dI}{dt} = \beta IS - \gamma I, \qquad \frac{dR}{dt} = \gamma I \tag{4.1}$$

where β is the transmission rate ($S \to I$) and γ is the recovery rate ($I \to R$). Given β, γ, and initial conditions (S_0, I_0, R_0), one can solve $S(t)$, $I(t)$, and $R(t)$ at any t from Eq. 4.1.

Network SIR [103, 126] assumes that the population form a static contact network G of N nodes, with each node representing an individual. An infectious node i can transmit the disease to a node j if and only if j is susceptible and is a neighbor of i, i.e. $\mathrm{j} \in \mathcal{N}(\mathrm{i})$. Let A be the adjacency matrix of G, with $A_{\mathrm{ij}} = 1$ if $\mathrm{j} \in \mathcal{N}(\mathrm{i})$, $A_{\mathrm{ij}} = 0$ otherwise. Let $S_{\mathrm{i}}(t)$, $I_{\mathrm{i}}(t)$, and $R_{\mathrm{i}}(t)$ be the probabilities of node i being in each of the states at time t, with $S_{\mathrm{i}}(t) + I_{\mathrm{i}}(t) + R_{\mathrm{i}}(t) = 1$. Let I_0 denote the initial infected persons at time $t = 0$, where $I_{0,\mathrm{i}} = 1$ if i is initially infectious (i.e. patient zero), otherwise $I_{0,\mathrm{i}} = 0$. Let $\mathcal{P}(t)$ denote the probability of remaining infectious at later time t after becoming infectious. $\mathcal{P}(t)$ is monotonic decreasing with $\mathcal{P}(0) = 1$ and $\lim_{t\to\infty} \mathcal{P}(t) = 0$. The probability for node i being in state I at time t is:

$$I_{\mathrm{i}}(t) = I_{0,\mathrm{i}}\mathcal{P}(t) + \beta \int_0^t \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(x) I_{\mathrm{j}}(x) \mathcal{P}(t-x) dx \tag{4.2}$$

where β is the transmission rate. So the derivative of I_{i} is:

$$\begin{aligned} \frac{dI_{\mathrm{i}}(t)}{dt} &= \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) + I_{0,\mathrm{i}} \mathcal{P}'(t) \\ &\quad + \beta \int_0^t \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(x) I_{\mathrm{j}}(x) \mathcal{P}'(t-x) dx. \end{aligned} \tag{4.3}$$

As $\mathcal{P}(t)$ is non-increasing, $\mathcal{P}'(t)$ is non-positive, so the last two terms reduce the increase of infection, which corresponds to the increase of the recovered:

$$\frac{dR_{\mathrm{i}}(t)}{dt} = -I_{0,\mathrm{i}}\mathcal{P}'(t) - \beta \int_0^t \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(x) I_{\mathrm{j}}(x) \mathcal{P}'(t-x) dx. \tag{4.4}$$

Given that $\frac{dS_{\mathrm{i}}(t)}{dt} + \frac{dI_{\mathrm{i}}(t)}{dt} + \frac{dR_{\mathrm{i}}(t)}{dt} = 0$, we also have:

$$\frac{dS_{\mathrm{i}}(t)}{dt} = -\beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t). \tag{4.5}$$

Let $F_{\mathrm{i}}(t)$ be the probability of node i being in either I state or R state, i.e. $F_{\mathrm{i}}(t) = I_{\mathrm{i}}(t) + R_{\mathrm{i}}(t)$ and $S_{\mathrm{i}}(t) + F_{\mathrm{i}}(t) = 1$. We further have the derivative of $F_{\mathrm{i}}(t)$ as:

$$\begin{aligned} \frac{dF_{\mathrm{i}}(t)}{dt} &= \frac{dI_{\mathrm{i}}(t)}{dt} + \frac{dR_{\mathrm{i}}(t)}{dt} \\ &= \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} (1 - F_{\mathrm{i}}(t))(F_{\mathrm{j}}(t) - R_{\mathrm{j}}(t)) \end{aligned} \tag{4.6}$$

In summary, S_{i}, I_{i}, and R_{i} obey the following differential equations:

$$\begin{aligned} \frac{dS_{\mathrm{i}}(t)}{dt} &= -\beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) \\ \frac{dI_{\mathrm{i}}(t)}{dt} &= \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) + I_{0,\mathrm{i}}\mathcal{P}'(t) \\ &\quad + \beta \int_0^t \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(x) I_{\mathrm{j}}(x) \mathcal{P}'(t-x) dx \\ \frac{dR_{\mathrm{i}}(t)}{dt} &= -I_{0,\mathrm{i}}\mathcal{P}'(t) - \beta \int_0^t \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(x) I_{\mathrm{j}}(x) \mathcal{P}'(t-x) dx \end{aligned} \tag{4.7}$$

where β is the transmission rate. Note that a similar derivation can be found in [25], but there they assume that the population is homogeneously mixed and an individual can have contact with anyone else. In contrast, here we derive the system assuming that the population is in a network and an individual has only limited contacts.

Standard network SIR model Let $\mathcal{P}(t) = \mathrm{e}^{-\gamma t}$, i.e. the probability for an I node to stay infectious decays exponentially [25]. Eq. 4.7 becomes:

$$\begin{aligned}
\frac{dS_{\mathrm{i}}(t)}{dt} &= -\beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) \\
\frac{dI_{\mathrm{i}}(t)}{dt} &= \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) - \gamma I_{\mathrm{i}}(t) \\
\frac{dR_{\mathrm{i}}(t)}{dt} &= \gamma I_{\mathrm{i}}(t)
\end{aligned} \tag{4.8}$$

In the early stage, we have $S_{\mathrm{i}}(t) \approx 1$. Plugging $S_{\mathrm{i}}(t) \approx 1$ and $\mathcal{P}(t) = \mathrm{e}^{-\gamma t}$ in Eq. 4.3, $\frac{dI_{\mathrm{i}}(t)}{dt}$ becomes:

$$\begin{aligned}
\frac{dI_{\mathrm{i}}(t)}{dt} &= \sum_{\mathrm{j}} (\beta A_{\mathrm{ij}} - \gamma \delta_{\mathrm{ij}}) I_{\mathrm{j}}(t) \\
&= \sum_{\mathrm{j}} (\beta A - \gamma 1)_{\mathrm{ij}} I_{\mathrm{j}}(t)
\end{aligned} \tag{4.9}$$

where δ_{ij} is Kronecker delta. Solving Eq. 4.9, we have

$$\begin{aligned}
I(t) &= \mathrm{e}xp((\beta A - \gamma 1)t) I_0 \\
&= \mathrm{e}xp((\beta Q \Lambda Q^T - \gamma Q Q^T)t) I_0 \\
&= \mathrm{e}xp(Q(\beta \Lambda t - \gamma 1 t) Q^T) I_0 \\
&= Q \mathrm{e}xp(\beta \Lambda t - \gamma 1 t) Q^T I_0 \\
&\approx \psi_1 \mathrm{e}xp((\beta \lambda_1 - \gamma)t) \psi_1^T I_0 \\
&= \mathrm{e}xp((\beta \lambda_1 - \gamma)t)(\psi_1^T I_0) \psi_1
\end{aligned} \tag{4.10}$$

where we expand A using the eigenvalue decomposition $A = Q \Lambda Q^T$ with Q and Λ being the eigen-vector and eigen-value matrices. λ_1 and ψ_1 are the largest eigen-value and the corresponding eigen-vector, respectively. Eq. 4.10 gives the basic reproduction number

$$R_0 = \frac{\beta \lambda_1}{\gamma}, \tag{4.11}$$

and we can see that when $R_0 > 1$ the disease will spread to form an epidemic.

Delay network SIR model For delay SIR, following [25], we assume $\mathcal{P}(t) = \Theta(t - T)$, a step function, with $\mathcal{P}(t) = 1$ for $0 \leq t \leq T$ and $\mathcal{P}(t) = 0$ for $t > T$, then $\mathcal{P}'(t - x)$ in Eq. 4.7 becomes $-\delta(t - x - T)$. Here T is the delay time for recovery, i.e. an infectious node would recover after T units of time. Now Eq. 4.7 becomes

$$\begin{aligned}
\frac{dS_{\mathrm{i}}(t)}{dt} &= -\beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) \\
\frac{dI_{\mathrm{i}}(t)}{dt} &= \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) - \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t - T) I_{\mathrm{j}}(t - T) \\
\frac{dR_{\mathrm{i}}(t)}{dt} &= \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t - T) I_{\mathrm{j}}(t - T)
\end{aligned} \tag{4.12}$$

Such delay differential equations are known to be associated with non-Markovian dynamics [25, 76]. Plugging $\mathcal{P}(t) = \Theta(t - T)$ in Eq. 4.3 and combing the result with $\frac{dF_{\mathrm{i}}(t)}{dt} = \frac{dR_{\mathrm{i}}(t)}{dt} + \frac{dI_{\mathrm{i}}(t)}{dt} = \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t)$, we have $\frac{dR_{\mathrm{i}}(t)}{dt} = \frac{dF_{\mathrm{i}}(t-T)}{dt}$, which leads to $R_{\mathrm{i}}(t) = F_{\mathrm{i}}(t) + C$ where C is a constant. Since $R_{\mathrm{i}}(t)$ and $F_{\mathrm{i}}(t)$ are both monotonic increasing and $F_{\mathrm{i}}(t)$ saturates at $t \to +\infty$, therefore $C = 0$, namely $R_{\mathrm{i}}(t) = F_{\mathrm{i}}(t - T)$. Plugging this into Eq. 4.6, we have

$$\frac{dF_{\mathrm{i}}(t)}{dt} = T\beta \sum_{\mathrm{j}} A_{\mathrm{ij}} \frac{F_{\mathrm{j}}(t) - F_{\mathrm{j}}(t - T)}{T}, \tag{4.13}$$

or equivalently

$$\begin{aligned}
\frac{dF(t)}{dt} &= T\beta Q \Lambda Q^T \frac{F(t) - F(t - T)}{T} \\
&\approx T\beta \psi_1 \lambda_1 \psi_1^T \frac{F(t) - F(t - T)}{T} \\
&= T\beta \lambda_1 (\psi_1^T \frac{F(t) - F(t - T)}{T}) \psi_1,
\end{aligned} \tag{4.14}$$

which gives the basic reproduction number

$$R_0 = \beta \lambda_1 T. \tag{4.15}$$

SEIR is another popular epidemic model [69, 31] that is well-suited for modeling the infectious diseases with an exposed/latent period. During such period, the pathogen in the host is in low numbers, so that the host is infected but cannot transmit it to others. In

addition to S, I, and R, the model further assumes an exposed (E) compartment, to account for the latent period. The compartments follow the transformation rule: $S \to E \to I \to R$, and obey the following differential equations:

$$\begin{aligned}
\frac{dS_{\mathrm{i}}(t)}{dt} &= -\beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) \\
\frac{dE_{\mathrm{i}}(t)}{dt} &= \beta \sum_{\mathrm{j}} A_{\mathrm{ij}} S_{\mathrm{i}}(t) I_{\mathrm{j}}(t) - \alpha E_{\mathrm{i}}(t) \\
\frac{dI_{\mathrm{i}}(t)}{dt} &= \alpha E_{\mathrm{i}}(t) - \gamma I_{\mathrm{i}}(t) \\
\frac{dR_{\mathrm{i}}(t)}{dt} &= \gamma I_{\mathrm{i}}(t),
\end{aligned} \tag{4.16}$$

where β is the transmission rate, γ is the recovery rate, and α is the rate from E to I (or equivalently $1/\alpha$ is the mean latent period). The basic reproduction number is of the same form as Eq. 4.11.

4.2.2 STGCN

Although STGCN is separately developed in [165] and [170], here we adopt the one designed for traffic forecasting [170]. The model combines graph convolutional network (GCN) [24, 75] and convolutional neural network (CNN) for extracting spatial and temporal information. The model takes a traffic network (i.e. a network of sensor stations) and a sequence of sensor data (e.g. traffic speed, volume, density) as input, and predicts the future traffic status. The core of STGCN is a stack of the so-called spatio-temporal convolution (ST-Conv) blocks. An ST-Conv block consists of a spatial layer sandwiched by two temporal layers. A temporal layer contains a 1-D CNN along time axis followed by a gated linear unit (GLU), to capture the temporal dynamics. The spatial layer is a GCN implemented using the Chebyshev polynomials approximation [24] or the 1st-order approximation [75]. As suggested in [170], such sandwich architecture allows jointly processing graph-structured time series; the spatial layer in the middle can serve as a bottleneck to help achieve scale compression and feature squeezing.

4.3 Methodologies

Problem description In general, a contact network during a contagion process can be temporal, directed, and weighted, but here we concentrate on a static undirected and non-weighted graph $G = (V, E)$, where V and E are the sets of nodes and edges; the number of nodes in G is $N = |V|$. For a contagion process on G, we observe a sequence of k snapshots $\mathcal{X} = \{\mathbf{x}_{t_1}, \dots \mathbf{x}_{t_k}\}$ at time steps $\{t_1, t_2, \dots t_k\}$. Note that the snapshots are not necessarily consecutive. A snapshot $\mathbf{x}_{t_k}$ contains the states of all the nodes of G at t_k, i.e. $\mathbf{x}_{t_k} = \{x_{t_k,1}, \dots x_{t_k,N}\}$, with each node's state $x_{t_k,\mathrm{i}} \in \{S, I, R\}$ for the SIR model or $\in \{S, E, I, R\}$ for the SEIR model. The problem of source detection is to find the set of initially infected nodes $\mathcal{Y} = \{\mathrm{i} | x_{t=0,\mathrm{i}} = I, \mathrm{i} \in V\}$ by solving the following objective:

$$\mathcal{Y}^{\star} = argmax_{\mathcal{Y}} P(\mathcal{X}|\mathcal{Y}, G) \tag{4.17}$$

where $P(\mathcal{X}|\mathcal{Y}, G)$ is the likelihood of observing $\mathcal{X}$ with $\mathcal{Y}$ being the source.

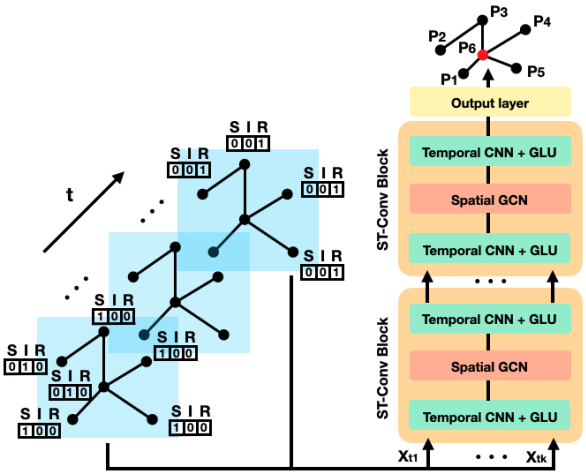


Figure4.1. SD-STGCN architecture. The blue areas on the left represent the input snapshots, which are one-hot encoded network states at multiple time steps. The orange areas on the right illustrate the model architecture consisting of a stack of ST-Conv blocks followed by an output layer. The output is a list of probabilities of each node being the source.

Model description Our SD-STGCN model is built upon the STGCN architecture. Fig. 4.1 is an illustration of SD-STGCN. The input is a sequence of k snapshots $\mathcal{X} = \{\mathbf{x}_{t_1}, \dots \mathbf{x}_{t_k}\}$ (the blue areas in Fig. 4.1). A snapshot $\mathbf{x}_{t_k} = \{x_{t_k,1}, \dots x_{t_k,N}\}$ contains the states of all the nodes of G, with $x_{t_k,\mathrm{i}}$ being a one-hot encoded vector of the states. So the input $\mathcal{X}$ is of the shape $k \times N \times C_{\mathrm{in}}$, where k is the number of snapshots, N is the number of nodes, C_{in} is the number of channels (= 3 for SIR or 4 for SEIR). $\mathcal{X}$ then goes through a series of ST-Conv blocks (the orange areas in Fig. 4.1), each consisting of two temporal layers and one spatial layer.

A temporal layer contains a 1-D CNN followed by a GLU. The CNN has a kernel of size K_t and applies to every node of G without padding, thus compressing $\mathcal{X}$ along time axis by $K_t - 1$. This part of the operation can be summarized as:

$$\mathcal{Z}_1 = GLU(CNN(\mathcal{X})) \in \mathbb{R}^{(k-K_t+1)\times N \times C_h}, \tag{4.18}$$

where $\mathcal{Z}_1$ is the output of the layer and C_h is the number of channels.

$\mathcal{Z}_1$ is then fed to the spatial layer, which is a GCN followed by a rectified linear unit (ReLU). The GCN has a kernel of size K_s. If the GCN is implemented by Chebyshev polynomials approximation [24], then $K_s - 1$ is the order of the truncated expansion; if the GCN is implemented by 1st-order approximation [75], then K_s is the number of successive convolutional layers. The spatial layer can effectively encode the information in the spatial domain by aggregating the signals in the K_s neighborhood of each node in each snapshot. Let the graph convolution preserve the channel dimension, the spatial layer can be summarized as:

$$\mathcal{Z}_2 = ReLU(GCN(\mathcal{Z}_1)) \in \mathbb{R}^{(k-K_t+1)\times N \times C_h}, \tag{4.19}$$

where $\mathcal{Z}_2$ is the output of the layer.

Following the spatial layer is another temporal layer. It performs the same operation as the first temporal layer, and further reduces the time dimension by $K_t - 1$. Unlike the

spatial layer, this layer does not preserve the channel dimension, but instead magnifies it, so the spatial layer becomes a bottleneck. The formula for this layer is:

$$\mathcal{Z}_3 = GLU(CNN(\mathcal{Z}_2)) \in \mathbb{R}^{(k-2K_t+2)\times N\times C_{out}}, \tag{4.20}$$

where $\mathcal{Z}_3$ is the output of the ST-Conv block and C_{out} is the number of channels ($C_{out} > C_h$). As in Fig. 4.1, $\mathcal{Z}_3$ then goes through more ST-Conv blocks that repeat the same sequence of operations. Assuming that M ST-Conv blocks are used, the output of the last ST-Conv block has the shape $\mathcal{Z}_3 \in \mathbb{R}^{(k-2MK_t+2M)\times N\times C_{out}}$.

Finally, we send $\mathcal{Z}_3$ to an output layer, which contains a temporal layer followed by a fully-connected (dense) layer with softmax activation. The temporal layer is a 1-D CNN with a kernel of size $K_o = k - 2MK_t + 2M$, so that the time dimension becomes 1. The fully-connected layer further reduces the channel dimension from C_{out} to 1, and the softmax function normalizes the output across the nodes to represent the source probability distribution. The output layer can be summarized as:

$$\mathcal{P} = Softmax(Dense(CNN(\mathcal{Z}_3))) \in \mathbb{R}^N, \tag{4.21}$$

where $\mathcal{P} = \{P_1, \ldots, P_N\}$, with $\sum_{\mathrm{i}=1}^{N} P_{\mathrm{i}} = 1$. We then pick the set of nodes with top P_{i} as $\mathcal{Y}^\star$. For single-source detection (i.e. $|\mathcal{Y}^\star| = 1$), we select the node $\mathrm{i}^\star = argmax_{\mathrm{i}} P_{\mathrm{i}}$ as the source. We perform single-source detection in Sec. 4.5.1-4.5.2, and multi-source detection in Sec. 4.5.4.

The model parameters Θ are learned in a supervised manner. In specific, we generate S(E)IR simulations on G with random sources $\mathcal{Y}_{true}$ (the ground truth), and sample k random snapshots $\mathcal{X}$ from the simulations. We then minimize the following cross-entropy loss:

$$\Theta = argmin_\Theta - \sum_{\mathrm{i}\in V} y_{\mathrm{i}} log(P_{\mathrm{i}}) \tag{4.22}$$

where $\mathcal{Y}_{true} = \{y_1, \ldots y_N\}$ is the one-hot encoded source set; $\mathcal{P} = \{P_1, \ldots, P_N\}$ is the output of SD-STGCN.

Training protocol We adopt two ST-Conv blocks in SD-STGCN. We set the block dimensions as $(C^1_{\text{in}}, C^1_h, C^1_{out}) = (3, 36, 144)$ and $(C^2_{\text{in}}, C^2_h, C^2_{out}) = (144, 36, 72)$ for SIR model; $(C^1_{\text{in}}, C^1_h, C^1_{out}) = (4, 36, 144)$ and $(C^2_{\text{in}}, C^2_h, C^2_{out}) = (144, 36, 72)$ for SEIR model. To train the model, we perform batch gradient descent with RMSProp optimizer, a batch size of 16, and a learning rate of 0.001. In each pass, a series of $k = 16$ snapshots are sampled uniformly at random from every simulation. We examine different configurations - Chebyshev polynomials approximation [170] vs. 1st order approximation [75], spatial kernel size K_s in $\{2, 3, 4\}$, and temporal kernel size K_t in $\{2, 3, 4\}$. In our experiments, for each graph, we generate $2,000$ simulations and split into 80% training, 10% validation, and 10% testing. Grid search on validation data suggests that using 1st order approximation with $K_s = 4$ and $K_t = 3$ renders the best performance. In the following experiments, we adhere to this setup.

4.4 Related works

There have been a surge of works on various topics related to the COVID-19 pandemic [16, 25, 125]. To the best of our knowledge, there was only one work [126] before us using deep learning to solve the source detection problem of COVID-19, although the general problem of identifying the propagation sources in networks [67] is well studied. Early methods resort to graph-centrality measures such as rumor center [127, 128, 29, 92], eigenvector center [116, 117, 44], and Jordan center [180]. However, these methods are heuristic and only provide suboptimal solutions. Alternatively, [89] proposes a method named dynamic message passing (DMP) that provides near-optimal solution. However, this method has high computational complexity and requires the propagation time, which is in general not available in practise. [155] performs multi-source detection based on the idea of source prominence and label propagation, nonetheless, its convergent version has $O(N^3)$ complexity. Recently, [126] proposes using graph neural networks (GCN) [24, 75] for single-source detection without knowing the propagation time. But this method utilizes only one snapshot, while multiple snapshots may be observed in reality. Moreover, it has been shown in [154] that using multiple independent snapshots can improve source detection accuracy. In the fields of traffic forecasting [170] and action recognition [165], different forms of spatial temporal graph

Table4.1. Network statistics. The columns from left to right are the network name, number of nodes $|V|$, number of edges $|E|$, average degree $\overline{d}$, and clustering coefficient C [104].

network	$\|V\|$	$\|E\|$	$\overline{d}$	C
ER	1000	10128	20.3	0.020
BA	1000	9900	19.8	0.062
BA-Tree	1000	999	2.0	0
RGG	1000	9326	18.7	0.618
Frat	58	967	33.3	0.747
Conf	403	9565	47.5	0.282
High	774	7992	20.7	0.186

convolutional networks were developed for prediction involving spatial and temporal signals. Inspired by these works, especially [170], we propose SD-STGCN, a spatial temporal graph convolutional network for source detection that utilizes multiple snapshots.

4.5 Experiments

Data We run standard and delay S(E)IR simulations with synthetic random graphs [104] and empirical contact networks. We generate different types of random graphs using Erdös–Rényi (**ER**), Barabási–Albert (**BA** & **BA-Tree**) [2], and Random Geometric Graph (**RGG**) [21] models. We adopt three empirical contact networks - **Bernard and Killworth Fraternity** (Frat) [70, 36], **SFHH conference** (Conf) [46], and **High school** (High) [122], which are static networks obtained by aggregating dynamic contact sequences. Fraternity network describes the interactions between students living in a fraternity at a West Virginia college. An edge is added if two students are spotted engaged in a conversation. Conference network describes the face-to-face interactions of participants of the SFHH 2009 conference. High-school data contains close proximity records of students, teachers, and other persons in an American high school. We add an edge if two people are in close proximity for more than 5 minutes. The statistics of the networks are listed in Table 4.1. Note in addition to the entries in Table 4.1, we also test our model on ER graphs with $5,000$ and $10,000$ nodes. We defer the description of the **real COVID-19 cases data** to Sec. 4.5.4.

Baseline methods We compare our SD-STGCN with two popular baseline methods - Dynamic Message Passing (DMP) [89] and a graph convolutional network based model (GCN) [126]. We choose DMP as a representative of the non-deep learning based methods, as it has been shown in [89] to considerably outperform other popular methods like rumor center [127] and Jordan center [180]. We select the GCN model as the second baseline, as it also utilizes deep learning for source detection. We show that by leveraging multiple snapshots, our model can achieve significant improvement over this model. We notice that the algorithm proposed in [155] can perform multi-source detection, however their code is not publicly available.

DMP [89] is a source detection model based on the message passing framework [68]. For a single source SIR on a network $G = \{V, E\}$, it predicts the source given a snapshot $\mathcal{O}$ at time t. Assuming an arbitrary node i as the source, it first estimates the marginal probabilities for any node to be in each of the three states S, I, R, at time t. It then approximates the joint likelihood $P(\mathcal{O}|\mathrm{i})$ as the product of the marginal probabilities of the observed nodes being in the observed states. It then goes through all $\mathrm{i} \in V$ and picks the one that maximizes $P(\mathcal{O}|\mathrm{i})$ as the source. The time complexity of the model is $\mathcal{O}(tN^2\overline{d})$, where t is the time step of the observations, N is the number of nodes, and $\overline{d}$ is the average degree of the network. It is thus computationally expensive for very large networks that are strongly connected [126, 67]. Besides, the method requires the propagation time t of the observations, which is generally unknown.

GCN [126] is a graph convolutional network [75] based model for source detection. Similar to DMP, it predicts the source using one snapshot of the network. But unlike DMP, it does not need to know the time of the snapshot. Moreover, it can be applied to both SIR and SEIR. The model takes one-hot encoded node states as features and outputs the probabilities of each node being the source. In [126], the authors actually proposed three GCN models (i.e. GCN-S, GCN-R, GCN-M) varied by propagation rules (i.e. symmetric, random walk, mixture). We adopt GCN-S as it appears to have the best performance among the three.

Performance metrics For single-source detection, we adopt three types of metrics - top-1 accuracy, mean reciprocal rank (MRR), and hit rates. Top-1 accuracy (Top-1 Acc)

Table4.2. Performance of SD-STGCN and GCN trained and tested on SIR simulations using $R_0 = 2.5$ and $\gamma = 0.4$, over random graphs of different types. The scores are evaluated over five graphs per type and five runs per graph. The format is mean (standard deviation).

Type	Model	Top-1 Acc	MRR	Hit@5
ER	SD-STGCN	**0.694** (0.017)	**0.816** (0.012)	**0.974** (0.006)
	GCN	0.302 (0.029)	0.375 (0.025)	0.450 (0.028)
	DMP	0.258 (0.009)	0.291 (0.008)	0.328 (0.006)
BA	SD-STGCN	**0.818** (0.027)	**0.892** (0.016)	**0.981** (0.005)
	GCN	0.523 (0.035)	0.598 (0.029)	0.685 (0.031)
	DMP	0.383 (0.013)	0.415 (0.011)	0.451 (0.010)
BA-Tree	SD-STGCN	**0.908** (0.042)	**0.948** (0.023)	**0.994** (0.004)
	GCN	0.753 (0.029)	0.834 (0.019)	0.935 (0.018)
	DMP	0.781 (0.016)	0.854 (0.011)	0.949 (0.014)
RGG	SD-STGCN	**0.724** (0.020)	**0.839** (0.012)	**0.984** (0.008)
	GCN	0.413 (0.041)	0.517 (0.037)	0.629 (0.043)
	DMP	0.362 (0.042)	0.439 (0.050)	0.529 (0.057)

examines whether the highest ranking node is aligned with the true source. MRR is the average reciprocal ranks of the true source given by the model, and a greater MRR indicates better performance. In addition, we evaluate hit rates at k (Hit@k) to see if the true source is among the top k candidates given by the model. For multi-source detection, besides Hit@k, we adopt Jaccard Similarity (JS) and normalized discounted cumulative gain (nDCG). JS and nDCG are estimated between the true source set $\mathcal{Y}^\star$ and the predicted set $\mathcal{Y}$ of the same size, i.e. $|\mathcal{Y}^\star| = |\mathcal{Y}|$. These metrics can measure the model's ranking quality.

Model reproducibility All experiments were conducted on a server with Nvidia Tesla V100-PCIE-16GB GPU. We make our datasets and code available at https://github.com/daDiz/SD-STGCN.

4.5.1 Experiments with standard S(E)IR simulations

In this section, we examine our SD-STGCN model on standard S(E)IR simulations obeying Eq. 4.8 and 4.16 over random graphs and empirical contact networks.

SIR on random graphs We generate standard SIR simulations (Eq. 4.8) with $R_0 = 2.5$ (the basic reproduction number of COVID-19 [16]), $\gamma = 0.4$ on ER, BA, BA-tree, and RGG

Table4.3. Performance of SD-STGCN and GCN trained and tested on SEIR simulations using $R_0 = 2.5$, $\gamma = 0.4$ and $\alpha = 0.5$, over random graphs of different types. The scores are evaluated over five graphs per type and five runs per graph. The format is mean (standard deviation).

Type	Model	Top-1 Acc	MRR	Hit@5
ER	SD-STGCN	**0.775** (0.028)	**0.849** (0.019)	**0.954** (0.012)
	GCN	0.126 (0.019)	0.192 (0.019)	0.252 (0.023)
BA	SD-STGCN	**0.802** (0.029)	**0.870** (0.021)	**0.959** (0.012)
	GCN	0.154 (0.024)	0.226 (0.025)	0.291 (0.033)
BA-Tree	SD-STGCN	**0.983** (0.011)	**0.990** (0.007)	**0.997** (0.003)
	GCN	0.439 (0.053)	0.583 (0.043)	0.754 (0.035)
RGG	SD-STGCN	**0.775** (0.032)	**0.846** (0.026)	**0.942** (0.025)
	GCN	0.207 (0.038)	0.319 (0.042)	0.435 (0.059)

networks of $1,000$ nodes. The transmission rate β can be calculated using Eq. 4.11. To compare with the results in [126], we adopt a similar setup with the simulation length fixed at 30 time steps. SD-STGCN infers the source using 16 randomly sampled snapshots, while DMP and GCN use only one. We evaluate the average performance across five independent runs per graph, and five graphs per type. The results are shown in Table 4.2. We can see that SD-STGCN outperforms DMP and GCN by a significant margin, which indicates the advantage of leveraging multiple snapshots. Like the two baselines, the performance of SD-STGCN varies across graph types: the highest top-1 accuracy ~ 0.908 in BA-Tree; the lowest ~ 0.694 in ER.

SEIR on random graphs In addition to SIR, we evaluate SD-STGCN with SEIR simulations on ER, BA, BA-Tree and RGG random graphs with $1,000$ nodes. The simulations are generated using $R_0 = 2.5$, $\gamma = 0.4$, and $\alpha = 0.5$. Note that DMP is designed for SIR and not applicable for SEIR [89], so we evaluate SD-STGCN against GCN only. The results across different types of random graphs are listed in Table 4.3. We can see that SD-STGCN is the clear winner over GCN by more than two folds in most of the cases. We also observe a similar trend like in the SIR case, where the performance varies across graph types, with BA-Tree the easiest to predict. Overall, the results here highlight that by using multiple snapshots, our SD-STGCN outperforms GCN for not only SIR but also SEIR.

Table4.4. Performance of SD-STGCN evaluated against baseline methods over standard SIR simulations on empirical contact networks. The scores are evaluated across five runs per network. The format is mean (standard deviation).

Data	Model	Top-1 Acc	MRR	Hit@5
Frat	SD-STGCN	**0.664** (0.002)	**0.805** (0.009)	**0.976** (0.004)
	GCN	0.457 (0.035)	0.561 (0.031)	0.670 (0.040)
	DMP	0.546 (0.161)	0.642 (0.157)	0.738 (0.180)
Conf	SD-STGCN	**0.540** (0.006)	**0.708** (0.003)	**0.926** (0.002)
	GCN	0.480 (0.038)	0.549 (0.031)	0.623 (0.027)
	DMP	0.414 (0.341)	0.448 (0.356)	0.475 (0.382)
High	SD-STGCN	**0.644** (0.022)	**0.781** (0.013)	**0.953** (0.007)
	GCN	0.408 (0.029)	0.482 (0.027)	0.564 (0.034)
	DMP	0.346 (0.310)	0.376 (0.326)	0.404 (0.359)

SIR on empirical contact networks Empirical contact networks can exhibit different characteristics from random graphs. For example, as shown in Table 4.1, the empirical contact networks under consideration show higher clustering coefficients than the random graphs except for RGG, which indicate that the nodes in these empirical networks are more likely to form clusters. It is thus important to examine SD-STGCN over contagion processes on empirical contact networks.

For data generation, we run standard SIR simulations on Frat, Conf and High networks. The training set is generated using random $R_0 \in [1, 15)$ and $\gamma \in [0.1, 0.2)$ for Frat and $\gamma \in [0.1, 0.3)$ for Conf and High. The ranges are determined such that the simulations are longer than 20 iterations. The test set is generated using $R_0 = 2.5$ and $\gamma = 0.2$ for Frat and Conf, and $\gamma = 0.4$ for High. We evaluate the performance of SD-STGCN against GCN and DMP on the test set, and the results are listed in Table 4.4. We can see that SD-STGCN outperforms the competing methods by a significant margin, in particular SD-STGCN achieves above 90% Hit@5 rates higher than the runner-up by about 30%. This demonstrates that our method is effective for not only random graphs but also empirical contact networks.

Table4.5. Performance of SD-STGCN evaluated against GCN over delay SIR simulations. The scores are evaluated across five runs per network. The format is mean (standard deviation).

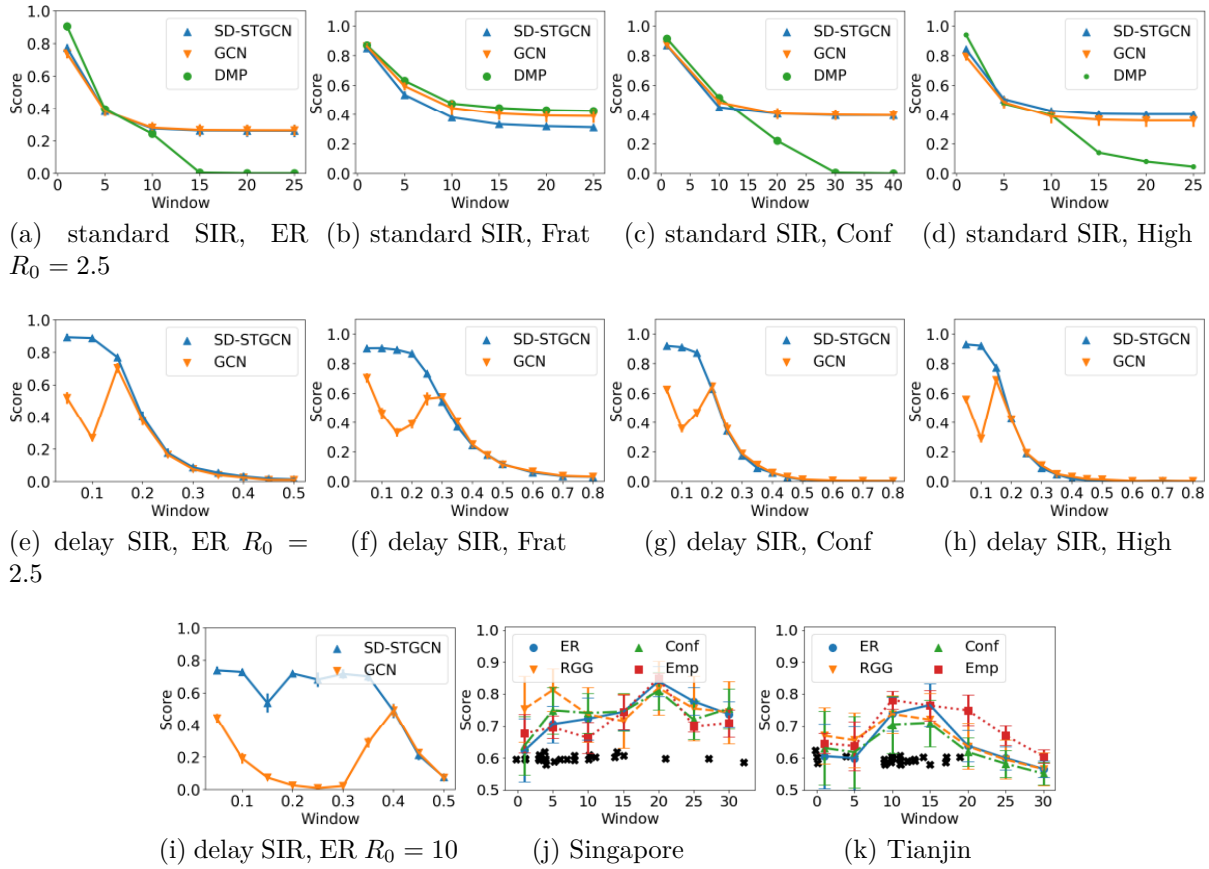
Data	Model	Top-1 Acc	MRR	Hit@5
ER	SD-STGCN	**0.573** (0.165)	**0.708** (0.149)	**0.881** (0.139)
	GCN	0.130 (0.025)	0.203 (0.025)	0.285 (0.029)
Frat	SD-STGCN	**0.773** (0.012)	**0.875** (0.005)	**0.995** (0.000)
	GCN	0.231 (0.033)	0.367 (0.037)	0.504 (0.047)
Conf	SD-STGCN	**0.719** (0.007)	**0.833** (0.004)	**0.982** (0.002)
	GCN	0.145 (0.044)	0.230 (0.044)	0.306 (0.039)
High	SD-STGCN	**0.722** (0.013)	**0.828** (0.006)	**0.953** (0.005)
	GCN	0.141 (0.027)	0.214 (0.024)	0.280 (0.027)

4.5.2 Experiments with delay SIR simulations

The experiments up to this point are based on simulations that are Markovian, while in reality the disease diffusion process is better described as non-Markovian [130]. To generate non-Markovian simulations, we adopt a delay SIR model [77, 97] that follows the dynamics in Eq. 4.12. In particular, it assumes that a constant delay period T between infectious and recovery states. Although, in real epidemics, T can vary from one individual to another, for simplicity, we adopt a constant T representing the average length of delay across population, and most importantly, using constant T already induces non-Markovian property.

We run delay SIR with $R_0 = 2.5$ and $T = 14$ (e.g. days) on random graphs (ER with $1,000$ nodes) and empirical contact networks (Frat, Conf, and High). Since no tractable form of DMP is known for the non-Markovian simulations [89], we only evaluate SD-STGCN against GCN. The results are shown in Table 4.5. We can see that SD-STGCN significantly outperforms GCN. Comparing to Table 4.2, for ER, we find that the performance of GCN reduces significantly, whereas the performance of SD-STGCN only reduces slightly. For example, GCN's top-1 accuracy drops by $\sim 56\%$, in contrast, SD-STGCN's top-1 accuracy only decreases by $\sim 17\%$. Comparing to Table 4.4, for Frat, Conf and High, we observe that the performance of SD-STGCN improves, whereas the performance of GCN reduces by more than 50%. Overall, we can see that SD-STGCN remains effective for non-Markovian

simulations, while GCN deteriorates, which highlights the fact that SD-STGCN is better than GCN by utilizing multiple snapshots instead of only one.



(a) standard SIR, ER $R_0 = 2.5$ (b) standard SIR, Frat (c) standard SIR, Conf (d) standard SIR, High

(e) delay SIR, ER $R_0 = 2.5$ (f) delay SIR, Frat (g) delay SIR, Conf (h) delay SIR, High

(i) delay SIR, ER $R_0 = 10$ (j) Singapore (k) Tianjin

Figure4.2. Top-1 accuracy across sliding windows with standard and delay SIR simulations on different networks. (a)-(d) Top-1 accuracy for standard SIR simulations. The horizontal axis represents the first frame in each window. (e)-(h) Top-1 accuracy for delay SIR simulations. The horizontal axis represents percentage windows. (i) Top-1 accuracy for delay SIR on ER graph with $R_0 = 10$. (j)-(k) nDCG across sliding windows on Singapore and Tianjin datasets. The black crosses represent the source cases, which are jittered to avoid overlapping.

4.5.3 Sliding windows

In the previous sections, we perform source detection based on randomly sampled snapshots distributed over different stages of the spread. In real life contact tracing, we may observe consecutive snapshots in a time window of limited length. Therefore, in this section, we mimic such scenario and place sliding windows across standard and delay SIR simulations

on ER random graph and Frat, Conf and High contact networks, and examine SD-STGCN's performance at these windows.

For standard SIR, we train SD-STGCN using simulations with random $R_0 \in [1, 15)$ and $\gamma \in [0.1, 0.4)$. For testing, we generate simulations using $R_0 = 2.5$ and $\gamma = 0.4$, with a fix size of 30 iterations. We then place sliding windows of size 16 at step 1, 5, 10, 15, 20, and 25, and evaluate the top-1 accuracy. As comparison, we apply GCN and DMP at the same windows with the first frame of each window as input. The results are illustrated in Fig. 4.2 (a)-(d) (top-1 accuracy). We can see that the performances of all the three models decrease as the window moving away from the starting point. Nevertheless, SD-STGCN and GCN outperform DMP significantly after 10 iterations except for the Frat case in Fig. 4.2 (b) where the network size (58 nodes) is smaller than the others (Table 4.1). Also notice that SD-STGCN and GCN give very close scores. This is because the standard SIR simulations following Eq. 4.8 are Markovian - the subsequent snapshots are conditionally independent of the source given the first snapshot in the window [154]. Therefore, SD-STGCN essentially operates like a GCN in this case. In the following, we will demonstrate that this is not the case for delay SIR simulations which are non-Markovian.

For delay SIR, we run simulations using $R_0 = 2.5$ and $T = 14$. Since the simulations generated in this way have different sizes, we place windows at fix percentages rather than at fix time steps, to better represent the various stages of the spread. Note that DMP is not available for non-Markovian simulations, so we only compare SD-STGCN against GCN. The results are illustrated in Fig. 4.2 (e)-(h) (top-1 accuracy). We can see that GCN suffers a performance degradation in the early stage (near the 10% window). In contrast, our SD-STGCN achieves above 80% top-1 accuracy in this period. This is because in the early stage many newly infected nodes emerge while none is yet recovered, so GCN has to pick the source out of many I nodes in one snapshot. In contrast, SD-STGCN can look ahead for multiple snapshots (e.g. 16 frames) in which the source and other early infected nodes recover, so it only needs to select the source from the fewer recovered nodes. In the later stage, as more and more nodes recover, the difference between one snapshot and multiple snapshots becomes less significant, and therefore we can see that SD-STGCN and GCN have similar performance.

Table4.6. COVID-19 case data network statistics. The columns from left to right are the network name, number of nodes $|V|$, number of edges $|E|$, clustering coefficient (C) [104].

network	$\|V\|$	$\|E\|$	C
Emp-HighSchool	774	7992	0.172
Singapore-ER	1000	9999 (84)	0.021 (0.000)
Singapore-RGG	1000	9463 (75)	0.601 (0.009)
Singapore-Conf	1000	10311 (79)	0.029 (0.001)
Tianjin-ER	1000	10074 (88)	0.020 (0.000)
Tianjin-RGG	1000	9439 (104)	0.589 (0.004)
Tianjin-Conf	1000	10348 (79)	0.029 (0.001)

Table4.7. Performance of SD-STGCN over real COVID-19 cases. The cases are projected onto random networks generated by ER, RGG, and the Configuration (Conf) models, and an empirical contact network (Emp). The scores are evaluated across five runs and five networks per model. The format is mean (standard deviation).

		Singapore			Tianjin		
Data	Model	Hit@10	JS	nDCG	Hit@10	JS	nDCG
ER	SD-STGCN	**0.624** (0.076)	**0.398** (0.034)	**0.738** (0.058)	**0.400** (0.000)	**0.198** (0.076)	**0.644** (0.095)
	GCN	0.528 (0.151)	0.280 (0.099)	0.724 (0.089)	0.312 (0.170)	0.138 (0.077)	0.609 (0.086)
RGG	SD-STGCN	**0.632** (0.112)	**0.425** (0.044)	**0.802** (0.060)	**0.428** (0.053)	**0.219** (0.079)	**0.688** (0.102)
	GCN	0.464 (0.176)	0.248 (0.085)	0.724 (0.077)	0.216 (0.164)	0.130 (0.092)	0.579 (0.056)
Conf	SD-STGCN	**0.656** (0.070)	**0.393** (0.061)	**0.771** (0.051)	**0.400** (0.000)	**0.185** (0.090)	**0.625** (0.118)
	GCN	0.524 (0.166)	0.287 (0.099)	0.757 (0.088)	0.284 (0.138)	0.129 (0.086)	0.601 (0.085)
Emp	SD-STGCN	0.560 (0.150)	**0.396** (0.034)	0.712 (0.042)	0.400 (0.000)	**0.276** (0.013)	**0.733** (0.038)
	GCN	**0.572** (0.137)	0.261 (0.065)	**0.797** (0.062)	**0.468** (0.122)	0.193 (0.080)	0.684 (0.069)

To verify this reasoning, we further compare SD-STGCN against GCN on sliding windows using delay SIR with $R_0 = 10$ on ER graph. The results are shown in Fig. 4.2 (i). With a greater R_0, more nodes become infectious in the early stage, making it more difficult for GCN to predict the source. Therefore, we can see that the difference between the two curves here is even more significant. Before the 40% window, SD-STGCN maintains high accuracy, whereas the one of GCN drops to near zero.

4.5.4 Case study: real COVID-19 case data

In this experiment, we assess SD-STGCN on two real world datasets of COVID-19 cases in Singapore and Tianjin. The **Singapore** dataset comprises 93 confirmed COVID-19 cases in Singapore from Jan 19, 2020 - Feb 26, 2020; the **Tianjin** dataset contains 135 confirmed

cases in Tianjin, a city in the northeast of China, from Jan 21, 2020 - Feb 27, 2020 [147]. In both datasets, the initial cases were imported from Wuhan (or Hubei province), with later cases being caused by local transmission. The datasets provide temporal information like date of onset symptoms, date of confirmation, and date discharged for those who recovered (or date of death). Assuming SIR type, we use the date of onset symptoms as the step when an individual turns from S to I; while the date of onset symptoms is not available, we use the date of confirmation instead. For the recovered/death cases, we assume that they turned from I to R on the date discharged or date of death.

The datasets also provide links between the cases that are related (e.g. by family or location). We can thus connect these cases and form a network G_0. However, G_0 is likely a subset of a larger network G, as the confirmed cases may have unidentified contacts. To model this, we overlay G_0 onto a greater network G of $1,000$ nodes. Note that the size of G is arbitrary, and a different value can be used. We generate G using ER, RGG, and a configuration model [103] with the degree distribution of the High school network. In addition, we simply adopt the high school network as G. The network statics are listed in Table 4.6.

For training, we generate $2,000$ SIR simulations per network, using random $R_0 \in [1, 15)$ and $\gamma \in [0.1, 0.4)$. For testing, we project the states of the confirmed cases to a sequence of daily snapshots of the network (treating the unknown cases as susceptible), rendering 38 and 40 consecutive snapshots for Singapore and Tianjin, respectively. For prediction, we take 16 random snapshots as input, and rank the nodes by the probability of being the source. We take the set of the top k candidates and evaluate how much it overlaps with the set of the initial cases from Wuhan (or Hubei province). The results are shown in Table 4.7. Note the scores are evaluated over five runs and five networks for each graph type. As the sources here are more than one, we utilize Hit@10, Jaccard Similarity (JS), and normalized discounted cumulative gain (nDCG) as performance measure. The Hit@10 scores indicate that $\sim 60\%$ (Singapore) and $\sim 40\%$ (Tianjin) of the top 10 predictions are overlapped with the reported sources. The JS scores (between the set of sources and the set of top predictions of the same size) are around 0.4 for Singapore and 0.2 for Tianjin. The nDCG scores are above 0.7 for Singapore and 0.6 for Tianjin. We can also see that the performance does not vary

Table4.8. Performance of SD-STGCN trained and tested on SIR simulations using $R_0 = 2.5$ and $\gamma = 0.4$, over ER graphs of different sizes. The scores are evaluated over five graphs per size and five runs per graph. The format is mean (standard deviation).

$\|V\|$	Top-1 Acc	MRR	Hit@5
1000	0.541 (0.038)	0.704 (0.022)	0.928 (0.018)
5000	0.532 (0.044)	0.689 (0.030)	0.896 (0.019)
10000	0.441 (0.018)	0.616 (0.016)	0.845 (0.032)

significantly between different networks. In Fig. 4.2 (j)-(k), we plot nDCG scores at different sliding windows. The curves here are not monotonically decreasing as time increases. This is likely because multiple sources emerge at different time steps. The black crosses in Fig. 4.2 (s)-(t) mark the time when a source emerges. We can see that the nDCG scores are high at the steps when the source cases cluster, especially in the Tianjin case.

4.5.5 Impact of graph and simulation related factors

Effect of graph sizes We also examine SD-STGCN on ER graphs of different sizes. In addition to graphs of $1,000$ nodes, we generate ER graphs of $5,000$ and $10,000$ nodes. We train and test SD-STGCN using standard SIR simulations with $R_0 = 2.5$ and $\gamma = 0.4$ on these graphs. The performance metrics are shown in Table 4.8. We observe a slightly decrease in performance as the graph size increases, which is understandable as the model has to pick the correct source out of more candidates. Nonetheless, SD-STGCN achieves above 44% top-1 accuracy for a network of $10,000$ nodes.

Effect of basic reproduction numbers R_0 In this experiment, we evaluate SD-STGCN with standard SIR and SEIR simulations using different R_0. [31] provides a list of estimated reproduction numbers for some well-known diseases, ranging from 1.5 (Spring wave) to 14.5 (Measles in Ghana). We thus train and test SD-STGCN over simulations with $R_0 = 1.5$, 2.5, 5 and 10 on ER graphs of $1,000$ nodes. For fair comparison, we keep the simulation length roughly the same for different R_0 (~ 40 for SIR and ~ 70 for SEIR), by adjusting γ and α. The results are shown in Table 4.9. We can see that the performance

Table4.9. Performance of SD-STGCN trained and tested on SIR and SEIR simulations using different R_0. The scores are evaluated over five ER graphs and five runs per graph. The format is mean (standard deviation).

	R_0	Top-1 Acc	MRR	Hit@5
SIR	1.5	0.597 (0.035)	0.735 (0.024)	0.924 (0.008)
	2.5	0.541 (0.038)	0.704 (0.022)	0.928 (0.018)
	5	0.483 (0.029)	0.655 (0.019)	0.885 (0.024)
	10	0.369 (0.036)	0.533 (0.028)	0.735 (0.032)
SEIR	1.5	0.883 (0.019)	0.929 (0.011)	0.988 (0.004)
	2.5	0.887 (0.012)	0.933 (0.007)	0.993 (0.005)
	5	0.837 (0.021)	0.902 (0.014)	0.986 (0.006)
	10	0.814 (0.023)	0.878 (0.015)	0.965 (0.009)

reduces as R_0 increases. This is likely because when R_0 is large, the number of I nodes reaches a large value in a relatively short period of time, thus making the back-tracking more difficult.

4.5.6 Training without pre-knowledge of epidemics

In real-world scenarios, we may not know the true R_0 and γ during training. In this case, we train our model with bunch of R_0 and γ combinations in the range of the well-known diseases [31]. In specific, we train SD-STGCN on simulations generated using R_0 and γ sampled uniformly at random in $[1, 15)$ and $[0.1, 0.4)$, respectively. The range of R_0 is based on the estimated reproduction numbers of well-known diseases [31]; the range of γ is determined such that the simulations are at least 20 iterations. To evaluate the model trained in this way against that trained with known parameters, we use the same test sets described in the previous section. The results are listed in Table 4.10. Compared to Table 4.9, the results here are slightly better, except for the case when $R_0 = 10$. Therefore, in practise, we can train SD-STGCN in this way and apply it to real epidemics with unknown parameters. It is worth pointing out that in some of the experiments earlier in this work, we have already adopted this approach.

Table4.10. Performance of SD-STGCN trained on SIR simulations using random R_0 and γ, and tested on simulations with different R_0. The scores are evaluated over five ER graphs and five runs per graph. The format is mean (standard deviation).

R_0	Top-1 Acc	MRR	Hit@5
1.5	0.619 (0.029)	0.749 (0.026)	0.930 (0.021)
2.5	0.568 (0.029)	0.725 (0.016)	0.931 (0.013)
5	0.504 (0.035)	0.675 (0.021)	0.896 (0.014)
10	0.359 (0.039)	0.526 (0.027)	0.738 (0.014)

4.6 Discussion

In our SD-STGCN model, we use GCN for spatial convolution. A layer of GCN performs a 1-hop neighborhood aggregation. So after L layers of GCN, each node learns a representation vector that captures the topological information within its L-hop neighborhood. For a connected network, when $L \geq d$ (the network diameter), the representation vector of each node can comprehend the entire network. This is the rationale behind the use of GCN in our SD-STGCN model. On the other hand, a layer of GCN can be seen as a step of diffusion, and a stack of L layers of GCN is equivalent to L steps of diffusion [126]. The forward dynamics of an epidemic on a network until time T can be viewed as a diffusion starting from some source(s) after T iterations, and thus can be modeled by T layers of GCNs. So if we can somehow reverse these GCNs, we might be able to reverse the diffusion and recover the source(s). In a recent publication [126], the authors use GCN as its own inverse to perform source detection, without proving it. This leads us to ask: can GCN actually model the reverse dynamics? In specific, if $GCN(A, X, W) = \sigma(AXW) = Y$ is the forward step, we want to find A' and W' such that $GCN(A', Y, W') = \sigma(A'YW') = X$, where σ is some activation function. Note that the A' and W' in the backward GCN are not necessarily the same as the A and W in the forward GCN. Also note that we can directly solve for X using gradient descent, which should be a performance upper bound for the inverse GCN approach. In the following, we show that in general the inverse of a GCN cannot be learned by another GCN with an arbitrarily small error. We perform a series of experiments to validate our findings.

Gradient Descent: GCN can be summarized by:

$$\sigma(AXW) = Y, \tag{4.23}$$

where σ is a non-linear function. $A \in \mathcal{R}^N \times \mathcal{R}^N$ is the normalized adjacency matrix; $X \in \mathcal{R}^N \times \mathcal{R}^c$ is the input features; $W \in \mathcal{R}^c \times \mathcal{R}^c$ is the filter; $Y \in \mathcal{R}^N \times \mathcal{R}^c$ is the output features. N and c are the number of nodes and the number of channels, respectively. In our experiments, we use a leaky-ReLU activation, as it is reversible. The leaky-ReLU function is defined as:

$$\sigma(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases} \tag{4.24}$$

where α is a constant between 0 and 1.

For regression, we evaluate the sum of squared error (SSE):

$$\begin{aligned} SSE &= \sum_{\mathrm{i}} \sum_{l} (Y_{\mathrm{i}l} - \hat{Y}_{\mathrm{i}l})^2 \\ &= \sum_{\mathrm{i}} \sum_{l} [\sigma(\sum_{\mathrm{j}} \sum_{k} A_{\mathrm{ij}} X_{\mathrm{j}k} W_{kl}) - \hat{Y}_{\mathrm{i}l}]^2 \end{aligned} \tag{4.25}$$

where $\hat{Y}$ is the true value. The derivative of SSE w.r.t $X_{\mathrm{j}k}$ is:

$$\begin{aligned} \frac{\partial SSE}{\partial X_{\mathrm{j}k}} &= \frac{\partial}{\partial X_{\mathrm{j}k}} \sum_{\mathrm{i}} \sum_{l} [\sigma(\sum_{\mathrm{j}} \sum_{k} A_{\mathrm{ij}} X_{\mathrm{j}k} W_{kl}) - \hat{Y}_{\mathrm{i}l}]^2 \\ &= 2 \sum_{\mathrm{i}} \sum_{l} [\sigma(\sum_{\mathrm{j}} \sum_{k} A_{\mathrm{ij}} X_{\mathrm{j}k} W_{\mathrm{i}l}) - \hat{Y}_{\mathrm{i}l}](\beta A_{\mathrm{ij}} W_{kl}) \end{aligned} \tag{4.26}$$

where $\beta = 1$ when the term in σ is positive; $\beta = \alpha$ when it is negative. To update $X_{\mathrm{j}k}$, we have:

$$X_{\mathrm{j}k}^t = X_{\mathrm{j}k}^{t-1} - lr \times \frac{\partial SSE}{\partial X_{\mathrm{j}k}}, \tag{4.27}$$

where lr is the learning rate.

Given $\hat{Y}$ and A, it is impossible to fully recover X for some W. For example, with $W = ones(c, c)$, we can see from Eq. 4.26, that $\frac{\partial SSE}{\partial X_{\mathrm{j}k_1}} = \frac{\partial SSE}{\partial X_{\mathrm{j}k_2}}$ for $\forall k_1, k_2 \in [0, c]$, since $W_{k_1 l} = W_{k_2 l} = 1$. The updates to $X_{\mathrm{j}k}$ are therefore identical across k, i.e. $X_{\mathrm{j}*} \propto ones(c)$. So gradient descent would never get to the true $X_{\mathrm{j}k}$, unless the initial guess $X^0_{\mathrm{j}k}$ satisfies $X_{\mathrm{j}k_1} - X^0_{\mathrm{j}k_1} = X_{\mathrm{j}k_2} - X^0_{\mathrm{j}k_2}$ for $\forall k_1, k_2 \in [0, c]$. **Therefore, gradient descent cannot recover X of an arbitrary form, when $W = ones(c, c)$.**

Inverse GCN: The inverse GCN can be written as:

$$\sigma^{-1}(AYW') = X \tag{4.28}$$

where W' is a trainable filter, which is not necessarily the same as the W in the forward pass (Eq. 4.23). σ^{-1} is the inverse σ. In our experiments, σ is leaky-ReLU, so σ^{-1} is:

$$\sigma^{-1}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \frac{1}{\alpha}x. & \text{otherwise} \end{cases} \tag{4.29}$$

Let $\hat{X}$ denote the true X that generates Y. We want to see if Eq. 4.28 can recover $\hat{X}$ by solving the objective:

$$argmin_{W'} \sum_{\mathrm{i}=1}^{N} \sum_{\mathrm{j}=1}^{c} (X_{\mathrm{ij}} - \hat{X}_{\mathrm{ij}})^2, \tag{4.30}$$

where N is the number of nodes and c is the number of channels. We can solve this objective by gradient descent. However, X might not be arbitrarily close to $\hat{X}$ in some cases. For example, when $W = ones(c, c)$, from Eq. 4.23, for an arbitrary $\hat{X}$, we can see that Y is channel-wise constant, i.e. $Y_{\mathrm{ij}_1} = Y_{\mathrm{ij}_2}$ for $\forall \mathrm{j}_1, \mathrm{j}_2 \in [0, c]$. Plugging Y into Eq. 4.28, we can see that X's rows are only differed by a scaling factor, regardless W'. So X cannot approximate $\hat{X}$ arbitrarily well. **Therefore, the inverse GCN in Eq. 4.28 cannot recover an arbitrary X arbitrarily well, when the forward pass in Eq. 4.23 uses a filter $W = ones(c, c)$.**

Experiments To validate our findings, we run inverse GCN with synthetic data. In specific, we generate $5,000$ $X \in \mathcal{R}^{1000} \times \mathcal{R}^{16}$ sampled from a uniform distribution $U(0, 1)$ and a

standard normal distribution $N(0,1)$. We use the forward GCN in Eq. 4.23 to generate Y. For the inverse, we add Gaussian noise $\sim 0.1 \times N(0,1)$ to Y, to resemble the real-world applications where the observations are usually noisy. So, we essentially recover X from $Y' = Y + 0.1 \times N(0,1)$. We adopt a ER random graph of $1,000$ nodes, with A normalized by row, i.e. $A_{ij}/\sum_j A_{ij} + I_{ij}$. On top of the model described in Eq. 4.28, we replace A by its inverse i.e. A^{-1} and a matrix constructed using page-rank [159]. The page-rank matrix is defined as $S = \alpha(I - (1-\alpha)A_{norm})^{-1}$, where A_{norm} is the normalized adjacency matrix. According to [159], one layer of page-rank GCN can represent infinite steps of random walks and thus cover higher order neighborhoods. Besides the inverse GCNs, we also perform gradient descent (GD) to directly learn X. Note that we regularize the norm of X in gradient descent, i.e. $0.1 \times ||X||^2$, as our experiments indicate better results with it. As a baseline, we guess X randomly from $U(0,1)$ and $N(0,1)$ for the respective datasets. To evaluate the performance, we measure mean squared error (MSE) and mean absolute error (MAE). We split the data into 80% train, 10% validation, and 10% test sets. The results evaluated on the test sets are listed in Table 4.11.

Table4.11. Recovering X from Y with noise using different methods, with $W = ones(16,16)$. The graph is an ER random graph. The true $X \in \mathcal{R}^{1000} \times \mathcal{R}^{16}$ is drawn from a uniform distribution $U(0,1)$ and a normal distribution $N(0,1)$.

Model	U(0,1)		N(0,1)	
	MSE	MAE	MSE	MAE
A	0.0831 (0.0001)	0.2496 (0.0001)	0.9770 (0.0000)	0.7886 (0.0000)
A^{-1}	0.1686 (0.0038)	0.3272 (0.0031)	0.9992 (0.0001)	0.7975 (0.0000)
pageRank	0.0834 (0.0001)	0.2500 (0.0001)	0.9964 (0.0001)	0.7964 (0.0000)
GD	**0.0817** (0.0000)	**0.2439** (0.0000)	**0.9560** (0.0000)	**0.7802** (0.0000)
random guess	0.1667 (0.0000)	0.3333 (0.0001)	1.9995 (0.0010)	1.1283 (0.0004)

From Table 4.11, we can see that GD gives the best estimations in both datasets. But even GD cannot achieve zero MSE or zero MAE, which as stated earlier, is due to $W = ones(16,16)$. We also observe that the inverse GCN with A (the first row) is the close runner-up in both datasets. Page rank is slightly behind using just A. Using A^{-1} is close to random guess for the uniform data ($\sim U(0,1)$), but is not far behind page rank for the normal data

($\sim N(0,1)$). In addition, we can see that the uniform dataset is easier to learn than the normal dataset, which is likely because the normal dataset has a greater range. Overall, the results here confirm our findings that GCN cannot be reversed by itself with a different W nor by gradient descent. Although not perfect, GCN can give performance close to gradient descent for recovering X from Y. We also perform experiments with $W = I(16,16)$ (an identity matrix) and $W = U(0,1)^{16} \times U(0,1)^{16}$ (a random uniform matrix). The results are listed in Table 4.12 and Table 4.13, respectively. Similar to the previous case, we can see that GD is the winner in these two cases, and the inverse GCN with A is the runner-up. However, the gap between them becomes more significant.

Table4.12. Recovering X using different methods, with $W = I(16,16)$. The graph is an ER random graph. The true $X \in \mathcal{R}^{1000} \times \mathcal{R}^{16}$ is drawn from a uniform distribution $U(0,1)$ and a normal distribution $N(0,1)$.

Model	U(0,1)		N(0,1)	
	MSE	MAE	MSE	MAE
A	0.0827 (0.0000)	0.2491 (0.0000)	0.8587 (0.0000)	0.7392 (0.0000)
A^{-1}	0.3313 (0.0000)	0.4972 (0.0000)	1.0001 (0.0000)	0.7978 (0.0000)
pageRank	0.0832 (0.0000)	0.2498 (0.0000)	0.9886 (0.0000)	0.7932 (0.0000)
GD	**0.0670** (0.0000)	**0.2184** (0.0000)	**0.7865** (0.0000)	**0.7066** (0.0000)
random guess	0.1667 (0.0000)	0.3333 (0.0000)	2.0004 (0.0010)	1.1284 (0.0003)

Table4.13. Recovering X using different methods, with $W \in U(0,1)^{16} \times U(0,1)^{16}$. The graph is an ER random graph. The true $X \in \mathcal{R}^{1000} \times \mathcal{R}^{16}$ is drawn from a uniform distribution $U(0,1)$ and a normal distribution $N(0,1)$.

Model	U(0,1)		N(0,1)	
	MSE	MAE	MSE	MAE
A	0.0829 (0.0000)	0.2494 (0.0000)	0.9020 (0.0002)	0.7577 (0.0001)
A^{-1}	0.2562 (0.0001)	0.4132 (0.0001)	0.9985 (0.0000)	0.7972 (0.0000)
pageRank	0.0833 (0.0000)	0.2499 (0.0000)	0.9875 (0.0000)	0.7928 (0.0000)
GD	**0.0629** (0.0000)	**0.2108** (0.0000)	**0.7892** (0.0000)	**0.7083** (0.0000)
random guess	0.1667 (0.0001)	0.3333 (0.0001)	1.9989 (0.0008)	1.1281 (0.0003)

4.7 Chapter Summary

In this work, we tackled the problem of identifying the source(s) of epidemics. We considered the problem in the framework of source detection on networks and solved it using SD-STGCN - a model that extracts both spatial and temporal features of a contagion process. We conducted a series of experiments using standard and non-Markovian epidemic simulations, on synthetic and empirical contact networks. We compared our model with two state-of-the-art baselines - DMP and GCN. The results suggest that SD-STGCN outperforms the baselines for randomly sampled snapshots and consecutive snapshots at sliding windows with non-Markovian simulations. Lastly, we applied SD-STGCN to two real COVID-19 cases datasets with multiple sources, and we found that the prediction was well aligned with the ground truth. For future work, we like to replace the graph convolutional network by some architecture that is better suited for modeling the inverse process of the forward propagation. Moreover, we like to examine SD-STGCN on simulations not limited to fix reproduction number and extend SD-STGCN to directed acyclic graph (DAG) and subgraph with a fraction of the nodes observed.

5. DYNAMIC TOPIC MODELING OF THE COVID-19 TWITTER NARRATIVE AMONG U.S. GOVERNORS AND CABINET EXECUTIVES

A version of this chapter was previously published by Workshop Proceedings of the 14th International AAAI Conference on Web and Social Media. H. Sha, M. Al Hasan, P.J. Brantingham, and G. Mohler. (2020). Dynamic topic modeling of the COVID-19 Twitter narrative among U.S. governors and cabinet executives. 5th International Workshop on Social Sensing (SocialSens 2020). [125]. DOI: 10.36190/2020.21.

5.1 Introduction

By mid-April 2020, the number of active COVID-19 cases has reached over 2 million and the number of deaths is over 140,000 world-wide. The United States has the largest share of confirmed cases (over 670,000) and confirmed deaths (over 27,000). Without a vaccine yet available, states throughout the U.S. are attempting to control transmission and reduce strain on the healthcare system through school and business closings, along with shelter-in-place orders. Careful planning and coordination is needed both to minimize risk from the disease, and to minimize the long-term economic impact.

In the U.S., a combination of federal and state-level decision making has shaped the country's response to COVID-19. The response is quickly evolving, making it difficult to understand how decision makers have influenced each other, and whom among the decision makers have emerged as leaders on different topics. To overcome this difficulty, we analyze the Twitter narrative of various decision makers through dynamic topic modeling. Specifically, we analyze a dataset of all COVID-19 related tweets by U.S. Governors, the President, and his cabinet members between January 1st 2020 and April 7th 2020. We use a Hawkes binomial topic model (HBTM) [100] to track evolving sub-topics around risk, testing and vaccination/treatment. The model also allows for estimation of Granger causality [161] that we use to construct influence networks amongst government officials.

Our work contributes to the growing body of literature on social media analytics and COVID-19. A summary of the most related work is as follows. In [20], general COVID-19 related topic diffusion across different social media platforms is analyzed. In [168], the authors study COVID-19 discussions on Chinese microblogs. Gender differences in COVID-19 related tweeting is investigated in [144] and in [145] the authors analyze consensus and dissent in attitudes towards COVID-19. Geolocated tweets are used to estimate mobility indices for tracking social distancing in [164].

5.2 Hawkes Binomial Topic Model

We analyze COVID-19 related tweets by U.S. governors and cabinet members using a network Hawkes binomial topic model[1] (HBTM) [100] with intensity $\lambda_s(t, \vec{m})$ at node s in the network determined by,

$$\lambda_s(t, \vec{m}) = \mu_s(t) J_0(\vec{m}|p_0^s) + \sum_{t>t_\mathrm{i}} \theta_{ss_\mathrm{i}} \omega_{ss_\mathrm{i}} \mathrm{e}^{-\omega_{ss_\mathrm{i}}(t-t_\mathrm{i})} J_1(\vec{m}, \vec{m}_\mathrm{i}|p_{off}^{ss_\mathrm{i}}, p_{on}^{ss_\mathrm{i}}). \tag{5.1}$$

A Hawkes process is a model for contagion in social media where the occurrence of a post increases the likelihood of more posts in the near future. In the HBTM, tweets are represented as bags of words following a Binomial distribution. When viewed as a branching process, the daughter event bag of words is generated by randomly turning on/off parent words through independent Bernoulli random variables.

In Equation 5.1 events at time t_i are associated with a mark $\vec{m}_\mathrm{i}$, a vector of size W, the number of words in the overall dictionary across events. The binary variables indicate whether each word is present or absent in the event at time t_i. Spontaneous events occur according to a Poisson process with rate $\mu_s(t)$ at node s in the network (here a node is either a governor or cabinet member). Unlike in [100], we let the spontaneous rate vary in time to reflect the exponential increase in overall COVID-19 related Twitter activity (for estimation

[1]↑Code and data available at: https://github.com/daDiz/hbtm_covid19_twitter

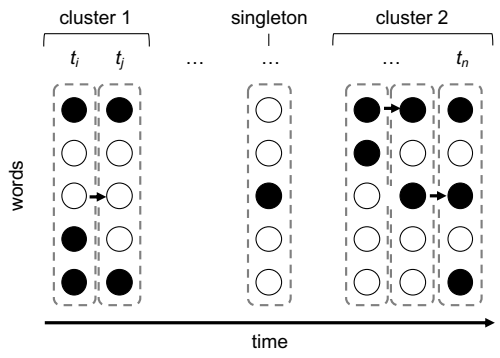


Figure5.1. In the HTBM, spontaneous events occur with marks generated by a binomial random variable over the dictionary of keywords contained in the data set. Events then trigger offspring events whose marks are generated by switching parent event words off (white circle) with probability p_{off} and on (black circle) with probability p_{on}. Unique events are delineated with dashed lines. Clusters are groups of parent daughter events connected by triggering.

we use a non-parametric histogram). The mark vector of spontaneous events is determined by,

$$J_0(\vec{m}|p_0^s) = p_0^{s\sum_{\mathrm{j}=1}^{W} m_{\mathrm{j}}}(1-p_0^s)^{W-\sum_{\mathrm{j}=1}^{W} m_{\mathrm{j}}}, \tag{5.2}$$

which is the product of W independent Bernoulli random variables with parameters p_0^s

The parameter $\theta_{ss'}$ determines the expected number of tweets by individual s triggered by a tweet by individual s' and can be viewed as a measure of influence. The expected waiting time between a parent-daughter event pair is given by $\omega_{ss'}^{-1}$. The mark of a daughter event is determined by two independent Bernoulli processes. Each word absent, or "turned off," in the parent bag of words is added to the bag of words of the child event with probability $p_{on}^{ss'}$. Each word present in the parent bag of words is deleted with probability $p_{off}^{ss'}$. Thus J_1 is given by,

$$J_1(\vec{m},\vec{m}_{\mathrm{i}}|p_{off}^{ss'},p_{on}^{ss'}) = \tag{5.3}$$

$$(p_{on}^{ss'})^{W_1^{\vec{m},\vec{m}_{\mathrm{i}}}}(1-p_{on}^{ss'})^{W_2^{\vec{m},\vec{m}_{\mathrm{i}}}}(p_{off}^{ss'})^{W_3^{\vec{m},\vec{m}_{\mathrm{i}}}}(1-p_{off}^{ss'})^{W_4^{\vec{m},\vec{m}_{\mathrm{i}}}},$$

where $W_1^{\vec{m},\vec{m}_{\mathrm{i}}}$ is the number of words present in the child vector and absent in the parent vector, $W_2^{\vec{m},\vec{m}_{\mathrm{i}}}$ is the number of words absent in both vectors, $W_3^{\vec{m},\vec{m}_{\mathrm{i}}}$ is the number of words

in the parent vector absent in the child vector, and $W_4^{\vec{m},\vec{m}_i}$ is the number of words present in both vectors.

After removing stop words we restrict the dictionary to the W most frequent words, on the order of several hundred most frequent words across tweets. The Model given by Eq. 5.1 can be viewed as a branching process and is estimated using Expectation-Maximization (EM) [100]. Using the EM algorithm for estimation has the added benefit that branching probabilities, estimates of the likelihood that tweet i was triggered by tweet j, are jointly estimated with the model:

$$q_{ij} = \frac{\theta_{s_i s_j} \omega_{s_i s_j} e^{-\omega_{s_i s_j}(t_i - t_j)} J_1(\vec{m}_i, \vec{m}_j | p_{off}^{s_i s_j}, p_{on}^{s_i s_j})}{\lambda(t_i, \vec{m}_i)}. \tag{5.4}$$

These branching probabilities can then be clustered to generate families of dynamic topics over time [100].

5.2.1 Related work

We note that Hawkes branching point processes in general are a popular model for mimicking viral processes on social media. Previous studies have utilized temporal point processes to model Twitter [174, 135], Dirichlet Hawkes processes [32, 162, 81], joint models of information diffusion and evolving networks [39], Hawkes topic modeling for detecting fake retweeters [37], and Latent influencers are modeled in [141] using an Indian buffet Hawkes process. For a review of point process modeling of social media data see [71].

Compared to standard LDA-type Hawkes processes, the HBTM has the advantage that it jointly estimates a network that can be used to measure influence; additionally, HBTM automatically detects the number of clusters. The temporal aspect of HBTM-like dynamic topic models tend to improve topic coherence in relation to LDA (see Figure 5.2).

5.3 Data

We first collected the verified Twitter handles of all U.S. state governors, presidential cabinet members, and the president (a total of 73 politicians, see Fig. 5.5 for their handles).

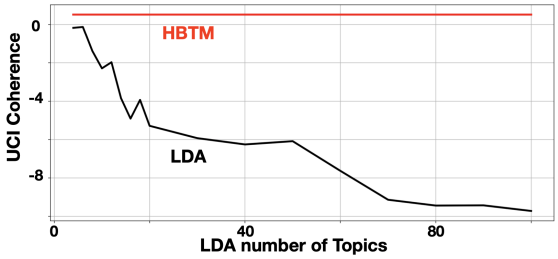


Figure5.2. UCI coherence of HBTM vs. LDA when applied to COVID-19 related tweets by governors and cabinet members.

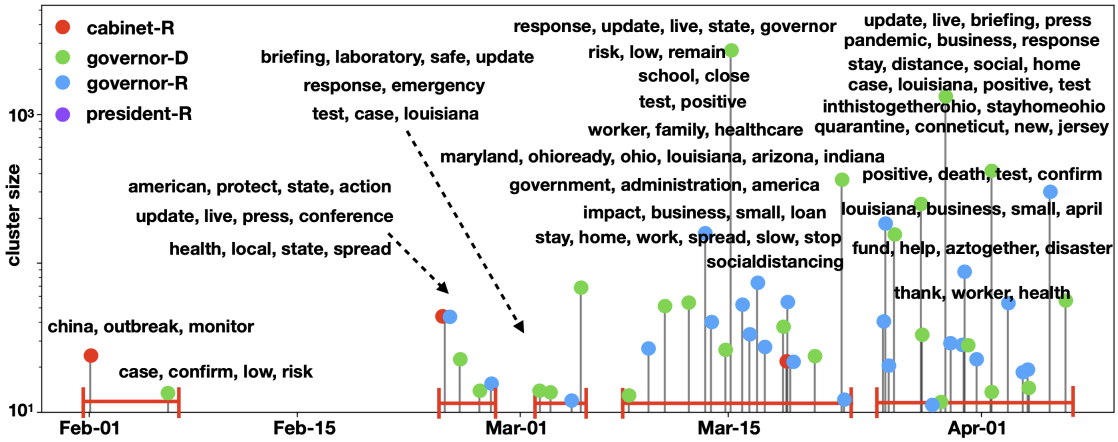


Figure5.3. Topic timeline. Clusters with size greater than 10 are pinned. Keywords indicate the topic of the clusters. The marker color indicates the dominant component of the cluster.

Next, we used the Twitter API to query all tweets by these users during the period of January 1, 2020 to April 7, 2020. We then performed a keyword expansion [12, 100] to extract a list of keywords related to COVID-19. This method iteratively adds keywords to a query list whose frequencies in the set of matching tweets are significantly higher than in the general sample. We then scanned the corpus with the expanded keyword list, obtaining a set of 7881 COVID-19 related tweets by these politicians. These tweets were further sorted in time-ascending order and converted to a bag-of-word representation. The vocabulary was then restricted to the top 425 words according to frequency.

5.4 Results

We cluster the data into space-time topics by sampling the branching probabilities q_{ij} in Equation 5.4. In particular, we assign tweets to the same group when a link between tweet

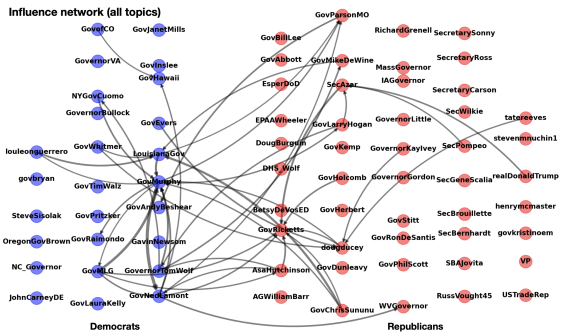


Figure5.4. Granger causality [161] influence network. Democrats (blue), Republicans (red). Weights of the edges of the directed graph correspond to the fraction of events estimated to be triggered across the edge. Edges with weights less than 10 are removed.

i and tweet j is sampled. In Fig. 5.3, we show topic clusters over time consisting of more than 10 tweets. Each marker height represents the size of the cluster and the most frequent keywords per marker indicate the topics of the clusters.

The clusters show roughly four phases in time, with a significant gap between the first phase and the rest. In the first phase (early February), the federal government (most frequent handle @SecAzar, Alex Azar, Sec. of Health) informed the public of the **outbreak** in **China** and claimed to closely **monitor** the situation. Also in this phase several state governors (most frequent handle @NYGovCuomo, Andrew Cuomo, Gov. of New York) started reporting **confirmed cases**, but stated that the **risk** was **low**, as the number of cases was limited.

The second and the third phases (early March) appeared almost a month later. From the keywords in these two phases, we can see that the government started to take **action** to **protect** the **American** citizens (possibly overseas in the regions of the outbreak). We can also see that **live updates** and **press conferences** were given to **brief** the public. Keywords like **spread** and **emergency** indicate that the outbreak was getting worse in the U.S. Meanwhile, the keyword **test** was mentioned frequently alongside **laboratory**, as limitations in U.S. testing was driving some of the narrative.

The fourth phase starts around mid-March, when clusters became larger and denser. In this phase, **live updates** were held by many governors on a regular basis (the highest peak

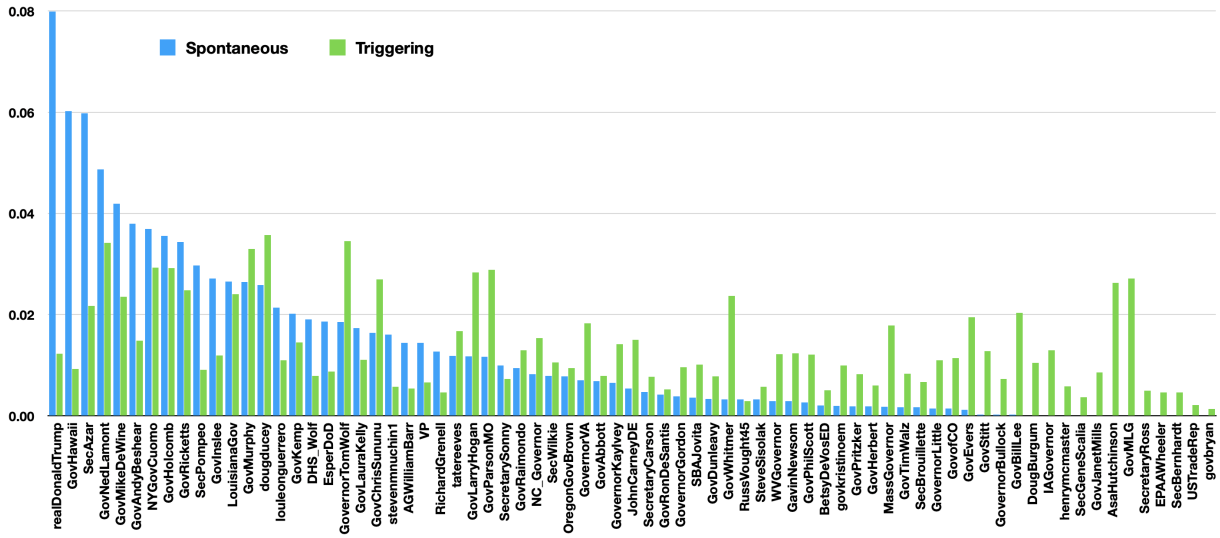


Figure5.5. Spontaneous vs. triggering effects of politicians on Twitter. Vertical axis: base intensities (spontaneous) and effective influences (triggering) are normalized over politicians; horizontal axis: Twitter handles of politicians. To save space, vertical axis is truncated at 0.08, rendering President Trump's spontaneous rate off the chart (~ 0.16).

in Fig. 5.3). We also see the separation between the federal and state governments, as the clusters divided into **government, administration, america** and the various states (**maryland, ohio, louisiana, arizona, indiana**). The Louisiana governor John Bel Edwards (@LouisianaGov) and the Ohio governor Mike DeWine (@GovMikeDeWine) were among the most active on Twitter sending information to the people in their respective states.

The topic of **risk** appears in this phase, and the message is that risk **remains low**. New topics also emerged on social distancing policies such as **school close**, **stay home**, and **work (from) home**. During the third phase the government began addressing problems like **healthcare** for **workers** and **families**, and **loan**(s) for **small businesses** due to the **impact** of the pandemic. The slogan **socialdistancing** was widely adopted in this phase.

In the most recent phase, a cluster with frequent words **live update**, **press conference**, and **briefing** is the largest, alongside a narrative around the number of **tested**, **confirmed positive** and **death cases** in different states. The Louisiana and Ohio governors continued to be the most active. Also **small businesses** remained a concern during this phase and the

keyword **disaster** indicates the negative impact of COVID-19. Meanwhile, **quarantine** and **stay home** were encouraged and reiterated on Twitter. The sacrifices of **health workers** were acknowledged (**thank**).

In Figure 5.4, we show inferred influence among governors and cabinet members by plotting a network where each edge weight from i $\rightarrow$ j is determined by the total estimated number of tweets triggered at node j by tweets from node i. The network shows influence across party lines, with Democrat governors **GovNedLamont**, **GovernorTomWolf**, **GovMurphy** and **LouisianaGov** highly connected with Republican governors **GovRicketts**, **GovLarryHogan** and **GovParsonMO**. We caution that this network captures Granger causality [161], and does not control for confounding effects. In Figure 5.5, we plot the estimated baseline rate of spontaneous tweets per governor and cabinet member (i.e. $\mu_s(t)$ averaged across time), along with each individual's estimated influence (average number of subsequent tweets in the network directly triggered by a Tweet, i.e. $\theta_{ss'}$ summed over s' and scaled by the number of tweets by s). Here we observe that President Trump has the highest rate of spontaneous tweets, followed by the Governor of Hawaii and Secretary Azar. Governors Ducey, Wolf and Lamont are the largest estimated influencers.

5.4.1 Risk, treatment and testing sub-topics

In addition to applying the HBTM to all COVID-19 related tweets, we also apply the model separately to three sub-categories. We first apply HBTM to tweets containing the word "risk". A sequence of clusters are illustrated in the top row of Fig. 5.6. The emergence of this sub-category coincides with the start of the second phase of the general timeline, and it appears that the **CDC** was among the first to mention how **serious** the risk was and asked for **immediate** actions. However, the subsequent clusters in early March indicate that both state and federal governments (Republicans and Democrats) were telling the public that the risk **remains low**. Also in this period, we observe calls for **washing hands** to **reduce** risk, and that **seniors** were identified to be the most vulnerable. After March 15, the narrative changes and the **high** risk to the general **population** is acknowledged. Keywords like **age** and **adult** indicate the high risk across **age** groups, even for young **adults**. The word **high**

frequently co-occurs with **test** and **quarantine**; due to the **high** risk of transmission, state governments increased **testing** and enforced **quarantine**(s). Overall, from left to right, the sequence of clusters show a clear trend in the narrative from low risk in late February to high risk in April.

Next, we apply HBTM to tweets containing the words "vaccine" and "treatment". The resulting clusters are illustrated in the middle row of Fig. 5.6. In mid-March, keywords **launch**, **trial**, **clinicaltrial**, **phase**, and **candidate** indicate that vaccine **candidates** were identified and entered the **clinical trial phase**. We can also see the National Institute of Health (**NIH**) **partner** with the pharmaceutical industry in developing the vaccine. Later in March, we start to see clusters where state governors (mainly Democrats) commented on the lack of **resources**, **equipment**, **ventilators**, and **hospital beds**. We also see cabinet members (specifically Sec. of Health @SecAzar) giving updates about vaccine development (**genetic sequence** and **clinical trial**). Another narrative is around an agreement (**agree**) with insurance companies to **ease** the **burden** of the pandemic on their **customers**. Additionally, we see the request to **create global researcher team** in developing a vaccine. In general, the clusters here suggest that the search for a vaccine has been a collective effort that crosses political parties and national boundaries.

Table5.1. Officials ranked by in-degree (most influenced) and out-degree (most influential) in influence networks.

Topic	In-degree	Out-degree
all	GovMurphy, GovRicketts, LouisianaGov	GovNedLamont, GovMurphy, GovMLG
risk	GovMikeDeWine, NYGovCuomo, GovMLG	GovMikeDeWine, GovPritzker, SecAzar
treatment	SecAzar, GovNedLamont, GovofCO	GovofCO, GovChrisSununu, GovNedLamont
test	GovNedLamont, GovMikeDeWine, LouisianaGov	NYGovCuomo, GovHerbert, GovKemp

In the bottom row of Fig. 5.6, we show clusters found by applying HBTM after filtering the dataset on the keyword "test". In early March, we see that **new** test **kits** were **available**. Tweets mention (**negative**) test results of some individuals by the Democrat governors and cabinet members. Concern about the **capacity** of testing **facilities** and **hospitals** is also discussed in early March. In mid-March, testing is expanded to the **community**, followed by requests for **expanding facility capacity** and **increasing laboratories**. During this

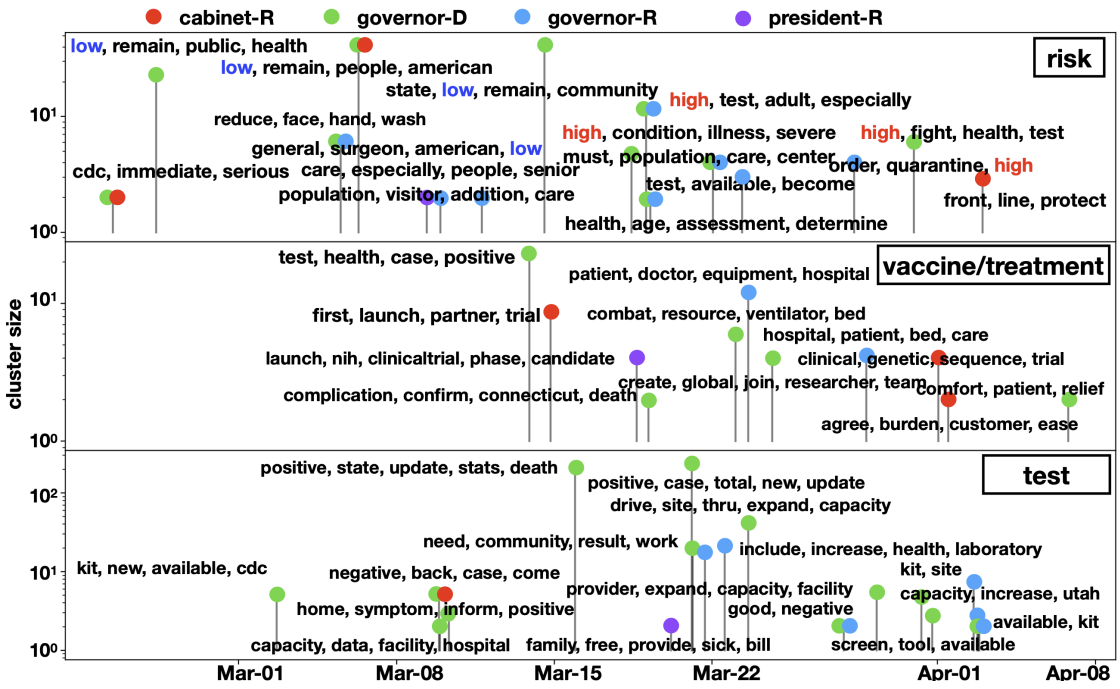


Figure5.6. Timeline of sub-topics on risk, treatment and testing. Clusters with size at least 2 are pinned. Keywords indicate the topic of the clusters. The marker color indicates the dominant component of the cluster.

period, state governors (especially Democrats, the two highest green markers in Fig. 5.6) start updating test results (in particular number of **positive cases**) and providing **stats** in their press conferences. The HBTM model identifies a cluster in which **drive thru site** is suggested as a way to **expand** testing **capacity**. In early April, we observe that the narrative has shifted away from a lack of testing resources; keywords indicate that **screen tools**, test **kits**, and test **sites** are available, and the testing **capacity** has **increased**.

In Figure 5.7, we plot Granger causality influence networks for the risk, treatment and testing sub-topics. Again we see connections crossing party lines. In the case of testing, the network is characterized by a dense set of connections between a select set of governors. The risk and treatment networks are characterized by more active nodes with fewer connections. In Table 5.1 we also list the most influential officials by sub-topic along with those officials most influenced.

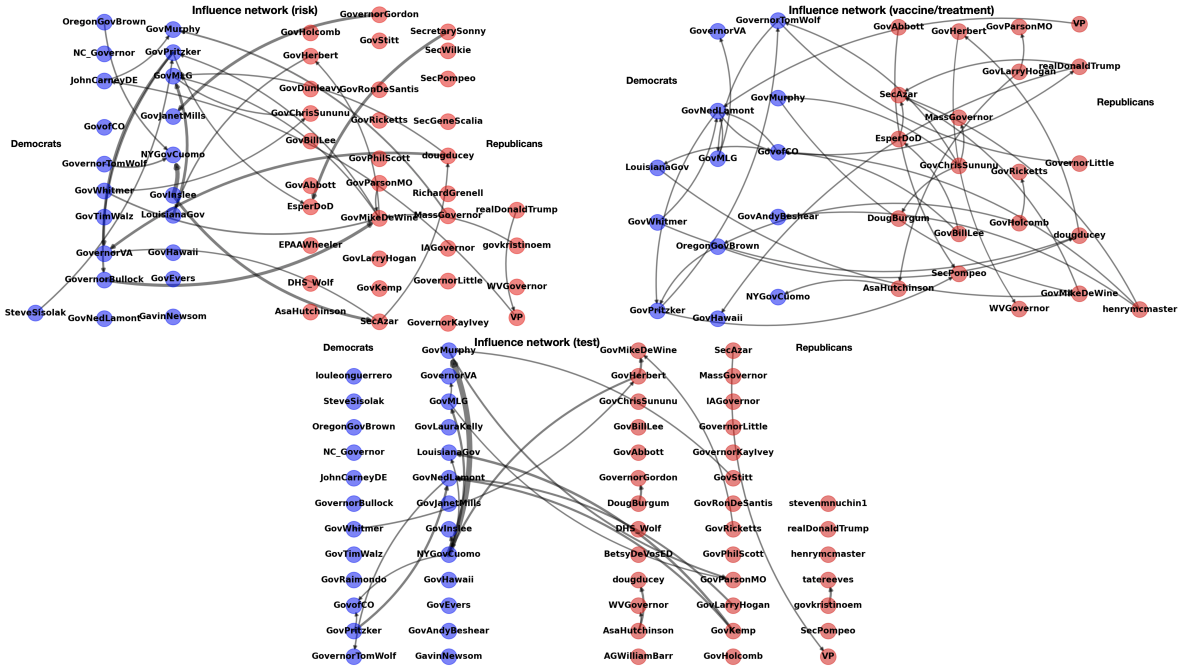


Figure5.7. Granger causality influence network for "risk" (top), "treatment" (middle) and "test" (bottom) sub-topics.

5.5 Chapter Summary

We analyzed the COVID-19 Twitter narrative among U.S. governors and presidential cabinet members using a Hawkes binomial topic model. We observed several narratives between January 1st and early April 2020, including a shift in the assessment of risk from low to high, discussion of a lack of testing resources which later subsided, and sub-topics around the impact of COVID-19 on businesses, efforts to create treatments and a vaccine, and calls for social distancing and staying at home. We also constructed influence networks amongst government officials using Granger causality inferred from the network Hawkes process. President Trump stands out for spontaneity, yet appears to have little influence with respect to network cross-excitation. Polarization is not obvious in the Granger influence networks; we observe a high level of cross party event triggering and influence seems more geographically clustered and related to state size.

We see several potential directions for future work. Here we limited the analysis to only COVID-19 related tweets among U.S. government officials. The HBTM can be used

to explore the COVID-19 narrative among the general population and may highlight issues around trust in institutions, adherence to social distancing, and economic impacts. Furthermore, analyzing non-COVID related tweets by government officials prior to the pandemic and constructing an evolving influence network may provide insights into how bi-partisan cooperation changes during national emergencies.

6. GROUP LINK PREDICTION USING CONDITIONAL VARIATIONAL AUTOENCODER

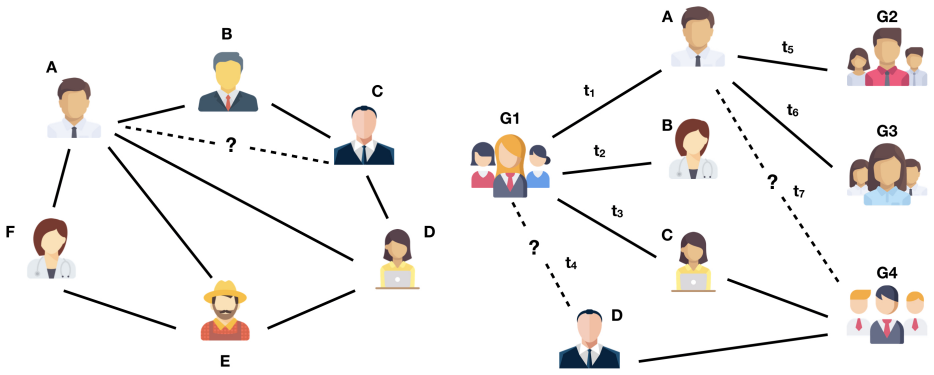
A version of this chapter was previously published by Proceedings of the Fifteenth International AAAI Conference on Web and Social Media (ICWSM). H. Sha, M. Al Hasan, and G. Mohler. Group link prediction Using Convolutional Variational Autoencoder. Accepted in the Fifteenth International AAAI Conference on Web and Social Media (ICWSM 2021).

6.1 Introduction

Link prediction [87] is a widely studied problem with successful applications in social networks [87], co-authorship networks [59], protein-protein interactions [84] and item recommendation [14]. As illustrated in Fig. 6.1a, given the current state of a network (say, a friendship network), the conventional link prediction task looks at a pair of disconnected nodes (such as A and C in Fig. 6.1a) and predicts if a link will form between them at a future time. Numerous machine learning models have been proposed for solving this task; for a comprehensive listing of the models, see the following surveys [90, 1].

In many real-world networks, patterns of link formation are not exclusively limited to two nodes. For instance, in a co-authorship network, more than two people may co-author an article. Likewise, in a network of online groups, where members sharing a group are connected by edges, the addition of a new individual to a group creates links between the individual and all of the existing members of that group. For accurately predicting links in such networks, one would need to consider the collective link formation between an individual and a group of individuals, a task which we refer to as "group link prediction".

Fig. 6.1 illustrates the difference between traditional link prediction and group link prediction. In the top graph, we are simply interested to know whether A and C, two disconnected nodes, will be connected in the future. In the bottom graph, we have two types of nodes: individuals and groups. Observing that a group G_1 connects with individuals A, B, and C at time t_1, t_2, and t_3, we would like to know the likelihood that G_1 will link with



(a) Conventional Link Prediction (b) Group Link Prediction

Figure6.1. Conventional link prediction and group link prediction in social networks. (a) In link prediction, links are between a pair of individuals. (b) In group link prediction, links are between a group and an individual. The goal is to predict future links given the current state of the network.

individual D at time t_4. Conversely, knowing that individual A joins groups G_2 and G_3 at time t_5 and t_6, we would like to predict if she will join group G_4 at time t_7.

Many real-life problems related to social media can be modeled as a task of group link prediction. For instance, Facebook makes suggestions for potential users to join certain Facebook Groups (e.g. sports enthusiast group, animal lover group)—a task of predicting/recommending a potential group to a user. The same is true for LinkedIn's recommendation of potential employers to a job-seeker. The above are examples of an individual connecting to a group given a list of possible groups—a task, what we also name as **group-recommendation**. The role of group and individual can be reversed in other applications. For instance, given a group (a partial list of emails), Gmail uses auto-complete to recommend additional email recipients. In this case, a group is creating a link to an individual out of many possible choices, which we call **member-recommendation**. Other examples of such tasks are below: a jobseeker is recommended to a potential employers, based on how much she aligns with the current employees at the company; Meetup.com (an event-based social network platform) recommends a user for an event, based on who else are participating in that event.

The existing link prediction solutions [87, 59, 14] make use of a pair-wise score $r(u, v)$ to measure the similarity between entities u and v. Often $r(u, v)$ is constructed based on

some topological properties of the network, including neighbor-based scores like Common Neighbor, Jaccard Index, Adamic/Adar and Preferential Attachment, and path-based scores like Graph Distance, Katz$_\beta$, and hitting time, or scores from dot product of latent representation of u and v obtained through matrix factorization [79]. Given the current state of the network, one predicts the pair (u, v) with the highest $r(u, v)$ to form a link in the future.

Link prediction can also be posed as a binary classification problem with the labels indicating whether two entities are connected [59]. In this approach, the features may be node embeddings obtained through static methods like DeepWalk [112], Node2Vec [54], and LINE [142]; dynamic methods like TNE [182], DynamicTriad [179], CTDNE [105], and HTNE [183]; attribute methods like Graph2Gauss [10], Neural-Brane [22], and DANE [86]; graph neural network (GNN) methods like Graph Convolutional Networks (GCN) [74], GraphSAGE [56], and Deep Graph Infomax (DGI) [150]

One possible adaptation of existing methods for solving group link prediction is to aggregate the pair-wise similarity scores $r(u, v)$ of traditional link prediction over a group, through functions like sum, mean, max, and min, to obtain the corresponding score $r(g, v)$ between a group g and an individual v. For instance, we can use the maximum $r(u, v)$ for $u \in g$ as $r(g, v)$. Similarly, for adapting the classification-based approach, we can aggregate (e.g. sum, max-pool, etc) the embedding vectors of all group members to represent the entire group.

Group link prediction can also be cast as link prediction on heterogeneous networks [131]. With individuals and groups as nodes, we can construct a bipartite heterogeneous network as illustrated in Fig. 6.1b. In this context, without aggregation, $r(g, v)$ can be directly obtained from measures such as random walk with restart [82] and meta-path similarity [139]. Moreover, nodal representations can be learned for link prediction, via methods like meta-path guided random walks [30] and matrix factorization [132].

While potentially easy to implement, these adaptations are not specifically designed for group link prediction and, as we will show in experiments to follow, are not consistently effective for group link prediction. In this work, we propose a Conditional Variational Auto-encoder (**CVAE**) [136] based model specially designed for solving both the **member-recommendation** task and the **group-recommendation** task. In addition, we provide a

variant of the CVAE model - Conditional Variational Auto-encoder with History (**CVAEH**) to incorporate the temporal characteristics, where the historical links are considered.

The contributions of this work are three-fold: We reframe a special case of link prediction on heterogeneous networks that considers the links between an individual and a group, which we call "group link prediction". We propose a CVAE-based model to solve group link prediction. We also introduce a second CVAE model (named CVAEH) that considers the temporal effect by incorporating the historical links. We examine the group link prediction problem in five real-world datasets and show the superiority of our CVAE/CVAEH models in comparison with various competing methods.

The rest of the chapter is organized as follows. In Section 6.2, we give a detailed description of our model. In Section 6.3, we review other works related to group link prediction. In Section 6.4.1, we provide details on the datasets used to assess our approach. In Section 6.4.2, we describe the various baseline models used for comparison. We present several group link prediction experiments in Section 6.4.3 and the results in Section 6.4.4. Finally, we summarize our work in Section 6.5.

6.2 Method

In this section, we first formally define group link prediction. We then provide background information about the building blocks of our model. Lastly, we give a detailed description on how to implement our model.

6.2.1 Problem Description

First we describe the terminology. Given a set of n individuals $G = \{v_1, v_2, \cdots, v_n\}$, a subset of them form a group $g_{\mathrm{i}} \subseteq G$ to participate in an event $\mathrm{e_i}$ at time t_{i}, i.e. $\mathrm{e_i} = (t_{\mathrm{i}}, g_{\mathrm{i}})$. We denote a sequence of historical events up to time t_{i} as $\mathcal{H}(t_{\mathrm{i}}) = \{\mathrm{e}_1, \mathrm{e}_2, \cdots, \mathrm{e}_{\mathrm{i}-1}\} = \{(t_1, g_1), (t_2, g_2), \cdots, (t_{\mathrm{i}-1}, g_{\mathrm{i}-1})\}$. Note, throughout the paper, v denotes an individual and g denotes a group. We use the subscription "ob" to indicate that the subject is observed, and "unob" to indicate the opposite.

Group link prediction is to predict the temporal association between an individual v and a group g in an event e at time t. It can be decomposed into two sub-problems - member-recommendation and group-recommendation. In essence, they differ in what is observed and what is to predict. If we observe a group of individuals in an event, and want to predict an individual to join them, then we have a member-recommendation problem; if we observe an individual, and want to predict a group for the individual to join an event with, then we have a group-recommendation problem. To facilitate ground truth based evaluation, we formalize them as the following:

Definition 6.2.1. ***Member-recommendation*** *Given an event* $\mathrm{e_i} = (t_\mathrm{i}, g_\mathrm{i})$ *and the history* $\mathcal{H}(t_\mathrm{i})$*, we randomly hold out a member* $v_{unob,\mathrm{i}} \in g_\mathrm{i}$ *from* g_i *as unobserved, and denote the rest of the group* $g_{ob,\mathrm{i}} = g_\mathrm{i} \setminus \{v_{unob,\mathrm{i}}\}$ *as observed. The member-recommendation task is to recommend an individual* $v \in G$ *for* $g_{ob,\mathrm{i}}$ *such that* $v = v_{unob,\mathrm{i}}$*, provided that* $g_{ob,\mathrm{i}}$ *and* $\mathcal{H}(t_\mathrm{i})$ *are known.*

Definition 6.2.2. ***Group-recommendation*** *Given an event* $\mathrm{e_i} = (t_\mathrm{i}, g_\mathrm{i})$ *and the history* $\mathcal{H}(t_\mathrm{i})$*, we randomly select an individual* $v_{ob,\mathrm{i}} \in g_\mathrm{i}$ *as observed, and denote the rest of the group* $g_{unob,\mathrm{i}} = g_\mathrm{i} \setminus \{v_{ob,\mathrm{i}}\}$ *as unobserved. The group-recommendation task is to recommend a group* g *for* $v_{ob,\mathrm{i}}$ *such that* $g = g_{unob,\mathrm{i}}$*, provided that* $v_{ob,\mathrm{i}}$ *and* $\mathcal{H}(t_\mathrm{i})$ *are known.*

Remark. For group-recommendation, in general, we need to rank every group g for a given individual $v_{ob,\mathrm{i}}$, which is intractable as the number of possible g is 2^n. Therefore, we simplify group-recommendation to picking the positive group $g_{pos} = g_{unob,\mathrm{i}}$ which $v_{ob,\mathrm{i}}$ actually joins at time t_i, out of a set of m negative groups $\{g_{neg,\mathrm{i}} | \mathrm{i} = 1, 2, \cdots, m\}$, which $v_{ob,\mathrm{i}}$ does not join.

Remark. Although our proposed models can be easily adapted to the more general task of recommending one group for another (or given a partial group, predicting the rest of the group), we leave this exciting direction to future work.

6.2.2 Preliminaries

Variational Autoencoder A variational autoencoder (VAE) [72] is an unsupervised neural network model for embedding high-dimensional data into a latent space. Unlike a traditional

autoencoder, this is accomplished by approximating the true distribution $P_{real}(\mathbf{x})$ for the data $\mathbf{x}$ through a variational Bayes approach. VAEs have the added advantage of being capable of generating artificial data that resembles the real data. The generative process starts by sampling a latent variable $\mathbf{z}$ from a prior Gaussian distribution $P(\mathbf{z})$. The VAE then generates a data point $\mathbf{x}$ conditioning on $\mathbf{z}$, using a generative distribution $P_\theta(\mathbf{x}|\mathbf{z})$, where θ are the parameters of the generative model. Usually $P(\mathbf{z})$ is assumed to be standard Gaussian and a neural network is used to model $P_\theta(\mathbf{x}|\mathbf{z})$. To obtain the parameters θ, one can maximize the log-likelihood $logP_\theta(\mathbf{x}) = \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}\left[logP_\theta(\mathbf{x}|\mathbf{z})P(\mathbf{z})\right]$. In practice, directly maximizing this log-likelihood is inefficient, as for most $\mathbf{z}$, $P_\theta(\mathbf{x}|\mathbf{z})$ will be nearly zero, and contribute almost nothing to the estimate of $P_\theta(\mathbf{x})$. Instead, it would be more efficient to sample from the posterior distribution $P(\mathbf{z}|\mathbf{x})$. However, in general, the posterior distribution inference is intractable. To alleviate the difficulty, one can approximate $P(\mathbf{z}|\mathbf{x})$ by a proposal distribution $Q_\phi(\mathbf{z}|\mathbf{x})$, where ϕ denotes the model parameters. Furthermore, [72] proposed to maximize a variational lower bound of the log-likelihood as the following:

$$\begin{aligned} logP_\theta(\mathbf{x}) &= \mathop{\mathbb{E}}_{Q_\phi(\mathbf{z}|\mathbf{x})}\left[log\frac{P_\theta(\mathbf{x},\mathbf{z})}{Q_\phi(\mathbf{z}|\mathbf{x})}\right] + KL(Q_\phi(\mathbf{z}|\mathbf{x})||P(\mathbf{z}|\mathbf{x})) \\ &\geq \mathop{\mathbb{E}}_{Q_\phi(\mathbf{z}|\mathbf{x})}\left[logP_\theta(\mathbf{x}|\mathbf{z})\right] - KL(Q_\phi(\mathbf{z}|\mathbf{x})||P(\mathbf{z})) \\ &= L_{VAE}(\mathbf{x};\phi,\theta), \end{aligned} \tag{6.1}$$

where KL denotes the Kullback–Leibler divergence. In Eq. 6.1, $Q_\phi(\mathbf{z}|\mathbf{x})$ is essentially an encoder, mapping a data point $\mathbf{x}$ to $\mathbf{z}$ in the latent space. The generative model $P_\theta(\mathbf{x}|\mathbf{z})$, on the other hand, acts as a decoder, converting the sampled latent vector $\mathbf{z}$ back to the data space. In practise, $Q_\phi(\mathbf{z}|\mathbf{x})$ is a neural network that maps a data point to two vectors - mean μ and the diagonal elements of the standard deviation σ. The reparameterization trick is used to sample $\mathbf{z}$ from $\mu + \sigma \odot \mathbf{z_0}$, where $\mathbf{z_0} \sim \mathcal{N}(0, I)$. The lower bound $L_{VAE}(\mathbf{x};\phi,\theta)$ can thus be optimized using stochastic gradient ascent w.r.t ϕ and θ. The optimal θ should render $P_\theta(\mathbf{x})$ resembling the true data distribution $P_{real}(\mathbf{x})$.

Conditional Variational Autoencoder A conditional variational auto-encoder (CVAE) [136] approximates the conditional probability distribution $P_{real}(\mathbf{x}|\mathbf{y})$, where $\mathbf{x}$ is the data and $\mathbf{y}$ is the observed features. For instance, if $\mathbf{x}$ is an MNIST handwritten digit image and

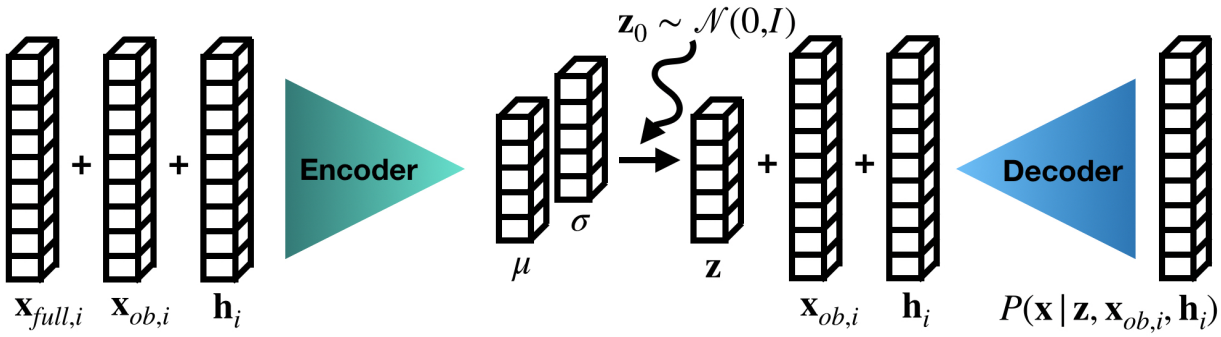


Figure6.2. Network architecture. $\mathbf{x}_{full,\mathrm{i}}$ and $\mathbf{x}_{ob,\mathrm{i}}$ are encoding vectors representing the entire group and observed member(s), which are concatenated as the input to the encoder (green). $\mathbf{h}_\mathrm{i}$ contains the historical event counts, which is also concatenated with the input for the CVAEH model. Mean μ and the diagonal elements of standard deviation σ are encoder outputs, with which we sample the latent vector $\mathbf{z}$. $\mathbf{z}$ is concatenated with $\mathbf{x}_{ob,\mathrm{i}}$ and $\mathbf{h}_\mathrm{i}$ (CVAEH only) and fed to the decoder (blue). The output $P(\mathbf{x}|\mathbf{z}, \mathbf{x}_{ob,\mathrm{i}}, \mathbf{h}_\mathrm{i})$ is the conditional probability indicating the likelihood for $\mathbf{x}$ to join $\mathbf{x}_{ob,\mathrm{i}}$.

$\mathbf{y}$ encodes a number, then CVAE would generate an image of that number. In contrast, VAE would generate images of any number between 0 and 9. Like the VAE model, a variational lower bound defined as the following can be maximized for training:

$$\begin{aligned} logP_\theta(\mathbf{x}|\mathbf{y}) \geq & \mathop{\mathbb{E}}_{Q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})}[logP_\theta(\mathbf{x}|\mathbf{y},\mathbf{z})] \\ & - KL(Q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})||P(\mathbf{z}|\mathbf{y})) \\ & = L_{CVAE}(\mathbf{x},\mathbf{y};\phi,\theta), \end{aligned} \tag{6.2}$$

where $Q_\phi(\mathbf{z}|\mathbf{x},\mathbf{y})$, $P_\theta(\mathbf{x}|\mathbf{y},\mathbf{z})$, and $P(\mathbf{z}|\mathbf{y})$ represent the encoder, decoder, and conditional prior networks, respectively. Note, in our implementation, we do not explicitly construct a conditional prior network. Instead, we adopt the reparameterization trick to model the conditional prior $P(\mathbf{z}|\mathbf{y})$. Also note, unlike the existing applications of CVAE focusing on image generation/reconstruction [136, 65], we use CVAE to learn a conditional probability for the unobserved group member(s) given the observed one(s).

6.2.3 Member-recommendation

In this section, we provide detailed description of our CVAE model and its variant—the CVAEH model, in the context of member-recommendation. We will explain how to adapt them for group-recommendation in the next section. The network architectures of these two models are illustrated in Fig. 6.2.

CVAE In Fig. 6.2, $\mathbf{x}_{full,\mathrm{i}}$ is an n-dimensional many-hot vector that encodes the entire group g_{i}, whereas $\mathbf{x}_{ob,\mathrm{i}}$ is another n-dimensional many-hot vector that only encodes the observed members $g_{ob,\mathrm{i}}$. Here we should ignore $\mathbf{h}_{\mathrm{i}}$, as it is only intended for the CVAEH model. The input of the encoder is the concatenation of $\mathbf{x}_{full,\mathrm{i}}$ and $\mathbf{x}_{ob,\mathrm{i}}$, whilst the outputs are the Gaussian mean μ and standard deviation σ (the diagonal elements). We then sample a latent vector $\mathbf{z}$ using the reparameterization trick [72], i.e. $\mathbf{z} = \mu + \sigma \odot \mathbf{z_0}$, where $\mathbf{z_0} \sim \mathcal{N}(0, I)$. The process up to now is equivalent to $Q_\phi(\mathbf{z}|\mathbf{x}, \mathbf{y})$ and $P(\mathbf{z}|\mathbf{y})$ in Eq. 6.2. Next, $\mathbf{z}$ and $\mathbf{x}_{ob,\mathrm{i}}$ are concatenated and fed to the decoder. The output of the decoder is a conditional probability $P(v|g_{ob,\mathrm{i}}) = P(\mathbf{x}|\mathbf{z}, \mathbf{x}_{ob,\mathrm{i}})$, where $\mathbf{x}$ is the one-hot encoding of $v \in G$. **Now we can recommend the individual v with the highest $P(v|g_{ob,\mathrm{i}})$ for the observed group $g_{ob,\mathrm{i}}$.**

CVAEH is similar to CVAE, except that it introduces an additional vector $\mathbf{h_i}$ whose v's entry represents the number of events that an individual v has attended during $[t_{\mathrm{i}-m}, t_{\mathrm{i}})$, with $t_{\mathrm{i}-m}$ being the time m events prior to the current time t_{i}. $\mathbf{h_i}$ is thus a representation of the event history $\mathcal{H}(t_{\mathrm{i}})$. As shown in Fig. 6.2, $\mathbf{x}_{full,\mathrm{i}}$, $\mathbf{x}_{ob,\mathrm{i}}$ and $\mathbf{h}_{\mathrm{i}}$ are concatenated at the encoder; while $\mathbf{z}$, $\mathbf{x}_{ob,\mathrm{i}}$ and $\mathbf{h}_{\mathrm{i}}$ are concatenated at the decoder. Accordingly, the output of the decoder is a conditional probability $P(v|g_{ob,\mathrm{i}}, \mathcal{H}(t_{\mathrm{i}})) = P(\mathbf{x}|\mathbf{z}, \mathbf{x}_{ob,\mathrm{i}}, \mathbf{h}_{\mathrm{i}})$, where $\mathbf{x}$ is again the one-hot encoding of an individual $v \in G$. By introducing $\mathbf{h}_{\mathrm{i}}$, the CVAEH model is able to take advantage of the temporal correlation in the data. As will be shown in our experiments, for the datasets where samples are time-correlated, the CVAEH model is generally better than the CVAE model in group link prediction.

Algorithm 3: CVAE/CVAEH learning process

1 learning rate lr, batch size m randomly initialize ϕ, θ;
2 **while** *not converge* **do**
3 sample batch of $\mathbf{x}_{full,\mathrm{i}}$, $\mathbf{x}_{ob,\mathrm{i}}$, $\mathbf{h}_{\mathrm{i}}$;
4 **for** i $\leftarrow 1$ **to** m **do**
5 $\mathbf{c}_{\mathrm{i}} \leftarrow concat(\mathbf{x}_{full,\mathrm{i}}, \mathbf{x}_{ob,\mathrm{i}}, \mathbf{h}_{\mathrm{i}})$;
6 $\mu, \sigma \leftarrow encoder(\mathbf{c}_{\mathrm{i}})$;
7 $\mathbf{z} \leftarrow \mu + \sigma \odot \mathbf{z_0}$ where $\mathbf{z_0} \sim \mathcal{N}(0, I)$;
8 $\mathbf{c}'_{\mathrm{i}} \leftarrow concat(\mathbf{z}, \mathbf{x}_{ob,\mathrm{i}}, \mathbf{h}_{\mathrm{i}})$;
9 $P_\theta(\mathbf{x}|\mathbf{z}, \mathbf{x}_{ob}, \mathbf{h}_{\mathrm{i}}) \leftarrow decoder(\mathbf{c}'_{\mathrm{i}})$;
10 calculate $\mathcal{L}(\phi, \theta)$ using Eq. 6.3;
11 $\partial\mathcal{L}_{\phi,\mathrm{i}} \leftarrow \frac{\partial\mathcal{L}(\phi,\theta)}{\partial\phi}$ and $\partial\mathcal{L}_{\theta,\mathrm{i}} \leftarrow \frac{\partial\mathcal{L}(\phi,\theta)}{\partial\theta}$;
12 **end**
13 $\phi \leftarrow \phi - lr \times \frac{1}{m}\sum_{\mathrm{i}=1}^{m} \partial\mathcal{L}_{\phi,\mathrm{i}}$;
14 $\theta \leftarrow \theta - lr \times \frac{1}{m}\sum_{\mathrm{i}=1}^{m} \partial\mathcal{L}_{\theta,\mathrm{i}}$;
15 **end**

Loss function To train CVAE and CVAEH, we maximize the variational lower bound defined in Eq. 6.2. More concretely, we minimize the following loss function using mini-batch gradient descent:

$$\begin{aligned}\mathcal{L}(\phi, \theta) = \sum_{v=1}^{n} [&- x_{full,v} log(P_\theta(x_v|\mathbf{z}, \mathbf{x}_{ob}, \mathbf{h})) \\ &- (1 - x_{full,v}) log(1 - P_\theta(x_v|\mathbf{z}, \mathbf{x}_{ob}, \mathbf{h}))] \\ &+ \frac{1}{2}\sum_{v=1}^{n}(exp(\sigma_v) + \mu_v^2 - 1 - log(\sigma_v))\end{aligned} \tag{6.3}$$

where x_v, $x_{full,v}$, σ_v and μ_v are the v'th element of $\mathbf{x}$, $\mathbf{x}_{full}$, σ and μ, respectively. Note, $\mathbf{h}$ here is only for the CVAEH model. Although not shown explicitly, μ, σ, and $\mathbf{z}$ are dependent on ϕ. The second summation on the right is the closed form of the KL-divergence in Eq. 6.2, as the result of the reparameterization trick [72].

Learning process is outlined in Alg. 3. Note $\mathbf{h}_{\mathrm{i}}$ is only intended for the CVAEH model. The algorithm starts by initializing the encoder and decoder parameters ϕ and θ. We then sample a batch of data (line 3), and for each data we go through the encode-decode process (line $5-9$), estimate the loss $\mathcal{L}(\phi, \theta)$ in Eq. 6.3 (line 10), and obtain the gradients w.r.t ϕ

and θ (line 11). At the end of each batch, we calculate the mean gradients and update ϕ and θ (line $13-14$). This process (line $3-14$) repeats until the model converges.

6.2.4 Group-recommendation

The architecture shown in Fig. 6.2 remains the same for group-recommendation. However, $\mathbf{x}_{ob,\mathrm{i}}$ is now a one-hot encoded vector of the observed individual $v_{ob,\mathrm{i}}$, and $\mathbf{x}$ is a many-hot encoded vector of a group. Consequently, the output of the decoder gives a conditional probability $P(g|v_{ob,\mathrm{i}}) = P(\mathbf{x}|\mathbf{z}, \mathbf{x}_{ob,\mathrm{i}})$, for $g \subseteq G \setminus \{v_{ob,\mathrm{i}}\}$. **We thus recommend the group g with the highest $P(g|v_{ob,\mathrm{i}})$ for the observed individual $v_{ob,\mathrm{i}}$.** The learning process is also the same except that the contents of $\mathbf{x}_{ob,\mathrm{i}}$, and $\mathbf{x}$ have changed.

6.3 Related works

Recommender systems Traditionally, recommender systems are mostly based on collaborative filtering [138], especially matrix factorization (MF) [79]. To reduce cold start, many works [42, 172] leverage additional information to improve performance. For example, [172] adopts an extended MF approach that incorporates location features and social features. Since we do not use additional features, our approach is different, nevertheless we compare our methodology with a matrix factorization baseline in Sec. 6.4.3. There exists a few works on "group recommendation" [88, 118], which refers to recommending *items* for a group of users, which is not the same as our group link prediction task, where we recommend a *user* for a group or vice-versa.

Heterogeneous networks The proposed group link prediction problem can be posed as link prediction on (bipartite) heterogeneous networks [131] with nodes of individuals and groups. Node proximity on heterogeneous networks can be evaluated via path-based similarity measures, such as random walk with restart [82, 114] and meta-path similarity [139]. Furthermore, nodal representations can be learned from meta-path guided random walks [30]. In addition, matrix factorization can be adopted for heterogenous networks [132]. We compare our models with two state-of-the-art models - metapath2vec [30] and HERec [132] in Sec. 6.4.3.

Dynamic graph embedding As many real-world networks evolve over time, many dynamic graph embedding models [179, 182, 105, 183] are proposed to learn time-respecting representations. For example, [105] leverages temporal random walks to learn node embeddings in continuous-time dynamic networks. [183] combines the Hawkes process [60] and the attention mechanism [5] to learn temporal embeddings via neighborhood formation. We compare our models (also dynamic) with these two models as well as a Long Short-Term Memory (LSTM) based model [61] in Sec. 6.4.3.

Graph neural networks Recent years have witnessed the surge of graph neural networks (GNN) [57]. Many GNN models, such as graph convolutional networks (GCN) [74] and GraphSAGE [56], generate nodal representations by aggregating local neighborhood features or attributes. Alternatively, [150] proposes a unsupervised node embedding method by maximizing mutual information. Furthermore, [169] extends GNN-based representation learning to web-scale bipartite networks of billions of nodes, by combining random walks and GCN. We compare our models with a state-of-the-art GNN model in Sec 6.4.3.

6.4 Experimental Results

In this section, we present experimental results to validate the effectiveness of our proposed group link prediction models for solving member and group recommendation tasks on five real-world datasets. We also compare the performance of our models with several baseline models. Below we first discuss the datasets and the baseline methods before presenting the experimental results.

6.4.1 Data Description

We use five real-life datasets. Their statistics is provided in Table 6.1. For data preprocessing, we organized the datasets into sequences of time-group pairs $\{(t_i, g_i)\}$, sorted in time ascending order. We then split each dataset into training (80%), validation (10%), and test (10%) sets. Any group member that appears in the validation or test split, but not in the training split, are removed from the corresponding data-split. Also, a group consists of less than three group members are also removed from the group-list. Let M_{train}, M_{valid},

Table6.1. Data set properties. N denotes the number of events (groups). $|V|$ and $|E|$ are the number of nodes and number of edges for each network, respectively.

Data set	N	$\|V\|$	$\|E\|$
Enron	31145	1946	47164
HT09	3703	113	9317
SFHH	3331	403	120507
Meetup NYC	11136	25458	340751
Meetup CA	15717	36799	407584

and M_{test} denote the set of individuals in the training, validation, and test splits, respectively. The processed datasets then satisfy (1) $M_{valid} \subseteq M_{train}$, (2) $M_{test} \subseteq M_{train}$, and (3) $G = M_{train}$. More discussion of each datasets are below:

Enron Email: This dataset [78] contains emails collected by the Federal Energy Regulatory Commission (FERC) during the investigation of the Enron Corporation. It contains $\sim 600,000$ emails from the year of 1999 to 2002, involving more than three thousands email addresses, most of which belong to the employees of the company. We organize the emails in time ascending order and extract the sending and receiving addresses owned by the employees (i.e. addresses ending with "@enron.com"). For member-recommendation, we only keep the emails with $3 \sim 10$ people, as the emails that include too many people are mostly messages broadcasted to everyone without distinguishing the groups. After the processing described earlier, we obtain a sequence of 31145 samples involving 1949 email addresses (Table 6.1). For group-recommendation, we keep the people who have involved in at least 1000 emails so that we can obtain enough negative samples. This also helps identify the major players behind the Enron scandal as discussed later in the case study. This treatment renders us a sequence of 15801 emails involving 114 email addresses.

Dynamic Contact Networks: These two datasets contain spatial temporal contact information of attendees during the ACM Hypertext 2009 conference (**HT09**) [64] and the 2009 SFHH conference (**SFHH**) [46]. The attendees of the conferences voluntarily wore radio

badges that monitored their face-to-face proximity. If a group of individuals engaged in a conversation, the time-stamp and their IDs would be recorded. During the course of the conferences, such events were collected, rendering a sequence of (time, attendee-list) samples. **HT09** dataset has 3703 conversational groups and 113 attendees, and **SFHH** has 3331 conversations groups involving 403 attendees (Table 6.1).

Event-based Social Networks: These datasets [114] were crawled from an event-based social network platform - Meetup.com. This platform allows its users to create/join groups and announce events for the group members. For instance, a user in the sports-enthusiastic group may invite other group members to a football watch-party. In this work, we adopt the **Meetup-NYC** dataset and the **Meetup-CA** dataset containing events taking place in New York City and California, respectively. Note although a user can be part of multiple groups and join various events, she cannot join the events created by a group of which she is not a member. For member-recommendation, we only keep the events with $3 \sim 20$ participants due to performance consideration. This renders a sequence of 11136 events involving 25458 users (Meetup-NYC) and a sequence of 15717 events involving 36799 users (Meetup-CA) (Table 6.1). Also notice that for a user who is only in one group, it would be trivial to recommend an event for her, as she would join the events exclusively of this group. Therefore, to effectively evaluate our model, for group-recommendation, we only keep users who are member of at least 10 groups, which leads to a sequence of 4500 events including 2286 users (Meetup-NYC) and a sequence of 2392 events including 1649 users (Meetup-CA).

6.4.2 Baseline Methods

In this section, we discuss the various competing methods.

Graph topology-based Link Prediction We convert our data into an undirected graph where nodes are event participants (e.g. email addresses, conference attendees, and so on) and edges are formed between them if they are in an event together. We define some graph-based score functions $r(u, v)$ to measure the similarity between nodes u and v. A larger $r(u, v)$ indicates a higher likelihood for linking the nodes. Here we examine an array of such

score functions [87]: **Common Neighbors**, **Jaccard Index**, **Adamic/Adar**, **Preferential Attachment**, and **Katz**$_\beta$. Since these scores are evaluated in a pair-wise manner, we resort to some aggregation methods to obtain the scores between a group g and an individual v. Formally, we have $r_F(g, v) = F(r(u, v)), \forall u \in g$, for $F \in \{sum, max, min, mean\}$.

Homogeneous Network Embedding-based Link Prediction We adopt a random walk based model - **Node2Vec** [54] and a recent GNN based model - **Deep Graph Infomax (DGI)** [150], to learn node representations on homogeneous networks consisting of individuals. We aggregate individual-individual cosine-similarities to obtain group-individual similarities.

Heterogeneous Network Embedding-based Link Prediction The proposed group link prediction can be cast as link prediction on heterogeneous networks with mixed nodes of individuals and groups. We adopt **metapath2vec** [30] and **HERec** [132] for the task. metapath2vec utilizes meta-path based random walks and skip-gram [96], to generate nodal representations. We apply this method with "individual-group-individual" meta-path and use dot product as similarity measure. HERec leverages fusion functions and matrix factorization to integrate multiple embeddings w.r.t different meta-paths. We use the recommendation ratings given by the method for link prediction.

Dynamic Network Embedding-based Link Prediction We can construct dynamic networks of individuals utilizing event timestamps. In specific, we use **CTDNE** [105] and **HTNE** [183] to learn time-respecting nodal representations. CTDNE uses random walks in time-ascending order to learn temporal embeddings; while HTNE combines the Hawkes process [60] and the attention mechanism [5] to learn embeddings via neighborhood formation. With the embeddings at each time step, we use an aggregation approach as in the static case, to predict the links between individuals and groups.

Matrix Factorization Methods Our datasets can also be converted to matrices $\mathbf{X} \in \mathbb{R}^{|G|} \times \mathbb{R}^{N}$, with the rows representing individuals and the columns representing groups (or equivalently events). If an individual v was part of a group g, then the entry $X_{v,g}$ would be one, otherwise zero. This setup resembles the user-item rating matrix widely adopted in the context of recommender system [79]. By matrix factorization (MF), an individual v is associated with a vector $\mathbf{p}_v \in \mathbb{R}^d$ and a group g is associated with a vector $\mathbf{q}_g \in \mathbb{R}^d$ in some

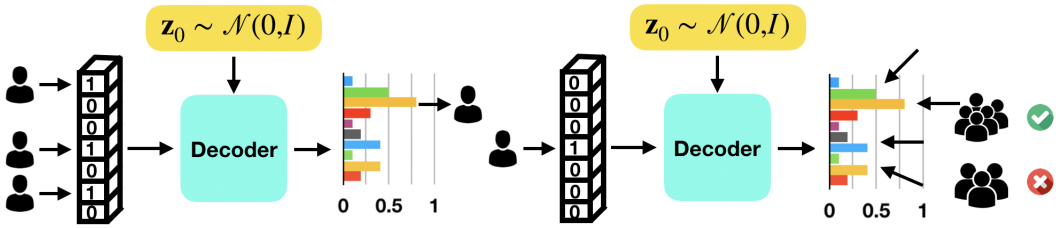
d-dimensional latent space, such that $X_{v,g} = \mathbf{p}_v^T \mathbf{q}_g$. To factorize $\mathbf{X}$, we adopt two methods - non-negative matrix factorization (**NMF**) [19, 43] and singular value decomposition (**SVD**) [55]. We can thus define the score function as $r(g, v) = \mathbf{p}_v^T \mathbf{q}_g$.

Neural Network Methods Aforementioned link prediction techniques (graph topology-based, embedding-based or MF-based) rank the candidates according to certain score functions $r(g, v)$. In contrast, our proposed model and the following Neural Network based models (except set2vec BPR) can provide a probability distribution for the links. In specific, we propose the following baseline Neural Network models for comparison:

One-hot MLP The first model combines Multi-layer Perceptron (MLP) with one-hot encoding (one-hot MLP). The input of the model is the one-hot encoded vector of the observed participant(s), $g_{ob,\mathrm{i}}$ in member-recommendation or $v_{ob,\mathrm{i}}$ in group-recommendation, and the output is a probability distribution for the unobserved participant(s), $v_{unob,\mathrm{i}}$ in member-recommendation or $g_{unob,\mathrm{i}}$ in group-recommendation. This is essentially a multivariate logistic regression, where whether an individual would join an event is determined by a Binomial distribution given by the output probabilities. To learn the model, the cross entropy loss is minimized using mini-batch gradient descent.

Set2vec MLP Set2vec is proposed by [151] for encoding a set of elements. The set encoding is essentially a weighted sum of the individual encodings of the set members. We use set2vec to encode a group as a single vector $\mathbf{x}_s = \sum_{\mathrm{i}=1}^{k} \alpha_{\mathrm{i}} \mathbf{x}_{\mathrm{i}}$, where the weights $\alpha_1, \alpha_2, ..., \alpha_k$ and the individual encoding vectors $\{\mathbf{x}_{\mathrm{i}}, \mathrm{i} = 1, 2, ..., k\}$ are learned by coupling set2vec with a downstream MLP classifier. The output of the classifier is a probability distribution, like in the one-hot MLP model.

Set2vec BPR We can also replace the cross entropy loss with the Bayesian Personalized Ranking (BPR) loss [119], leading to the set2vec-BPR model. BPR is proposed by [119] as a ranking framework for item recommendation. We use BPR to learn the embedding $\mathbf{x}_v$ for any individual v and the embedding $\mathbf{x}_g$ for any group g. We can then define the score function as $r(g, v) = \mathbf{x_g}^T \mathbf{x_v}$, which measures the similarity between the two embeddings.



(a) Recommend an individual for a group. (b) Recommend a group for an individual.

Figure6.3. Prediction Illustration. The one-hot encoding of the observed group member(s) is fed to the decoder. $\mathbf{z}_0$ is sampled multiple times, generating a multiplicity of outputs, which are averaged to obtain the final conditional probability. (a) For member-recommendation, we predict the one most probable member. (b) For group-recommendation, we find a group that best aligns with the probability distribution.

LSTM Long Short-Term Memory (LSTM) [61] is efficient for modeling sequence related problems. We compare our model with an LSTM-based model which takes in a sequence of embedding vectors $\{\mathbf{x}_{t_\mathrm{i}}, \mathbf{x}_{t_{\mathrm{i}+1}}, \cdots, \mathbf{x}_{t_{\mathrm{i}+\mathrm{j}}}\}$ of the observed member-lists at time-steps $\{t_\mathrm{i}, t_{\mathrm{i}+1}, \cdots, t_{\mathrm{i}+\mathrm{j}}\}$, and outputs a conditional probability $P(v|\mathbf{x}_{t_\mathrm{i}}, \mathbf{x}_{t_{\mathrm{i}+1}}, ..., \mathbf{x}_{t_{\mathrm{i}+\mathrm{j}}}), v \in G$ conditioning on all the observations in the window $[t_\mathrm{i}, t_{\mathrm{i}+\mathrm{j}})$. However, this model only predicts an individual rather than a group, therefore we only apply it to member-recommendation. Specifically, we use the output probability to rank the candidates at time $t_{\mathrm{i}+\mathrm{j}}$ given the current observed members and the members of the previous events.

6.4.3 Experimental Setup

Model details For the proposed **CVAE/CVAEH**, the encoder and the decoder consist of two fully-connected (FC) layers, with the hidden dimension (hd) in $\{128, 256, 512, 1024\}$. We also tune the dimension of $\mathbf{z}$, zd in $\{32, 64, 128, 256\}$. In addition, for CVAEH, we use different window sizes, ts in $\{2, 4, 8\}$ for calculating $\mathbf{h}_\mathrm{i}$. To learn the model, we minimize the loss in Eq. 6.3 using mini-batch gradient descent, with the batch size = 64 and the learning rate, lr in $\{0.0001, 0.001, 0.01\}$.

Prediction process In Fig. 6.3a, we illustrate the prediction process for **member-recommendation**. For each g_{ob}, we sample $\mathbf{z_0} \sim \mathcal{N}(0, I)$ m times, generating a series of conditional probabilities $P_\mathrm{i}(v|g_{ob}), \mathrm{i} = 1, 2, ..., m$. The final conditional probability

used for prediction is the sample mean of these m conditional probabilities, $P(v|g_{ob}) = \frac{1}{m}\sum_{\mathrm{i}=1}^{m} P_{\mathrm{i}}(v|g_{ob}), v \in G$. We then rank the individuals v by $P(v|g_{ob})$.

In Fig. 6.3b, we illustrate the prediction process for **group-recommendation**. We take out an individual v_{ob} from an event and regard the rest of the member-list as unobserved g_{unob}. Similarly, for each v_{ob}, we sample $\mathbf{z_0} \sim \mathcal{N}(0, I)$ m times, and calculate the sample mean of these m conditional probabilities, $P(g|v_{ob}) = \frac{1}{m}\sum_{\mathrm{i}=1}^{m} P_{\mathrm{i}}(g|v_{ob}), g \subseteq G \setminus \{v_{ob}\}$. However, we do not scan over all the possible g as there are exponential number of possible groups. Instead, we denote the true g_{unob} as a positive sample, $g_{pos} = g_{unob}$, and randomly sample a set of 200 negative samples $S_{neg} = \{g_{neg,\mathrm{i}}|v_{ob} \notin g_{neg,\mathrm{i}}, \mathrm{i} = 1, 2, ..., 200\}$ from the training set. We then rank $g \in \{g_{pos}\} \cup S_{neg}$ by the average probability $\frac{\sum_{v \in g} P(v|v_{ob})}{|g|}$. Note, here the numerator prefers a group that contains individuals with high $P(v|v_{ob})$; while the denominator penalizes a group that includes too many irrelevant individuals. Overall, this score measures to what extent a group g aligns with the conditional probability $P(g|v_{ob})$.

Performance metric As performance metrics, we use hit rate@m (Hit@m) and mean reciprocal rank@m (MRR@m). For each query, if the target is among the top-m recommendations, we have a hit and record the reciprocal of the rank at which the target is retrieved. When averaged across queries, we obtain Hit@m and MRR@m, with the former indicating how good the model is at finding the target and the later indicating how high the model ranks the target.

For **member-recommendation**, during validation and testing, for each sample, we hold out the last member $v_{\mathrm{i},k}$ from a group g_{i} of size k, rendering $v_{unob} = v_{\mathrm{i},k}$ and $g_{ob} = \{v_{\mathrm{i},1}, \cdots, v_{\mathrm{i},k-1}\}$. If the true v_{unob} is among the top m predictions, we then register a hit for the Hit@m score. We perform parameter tuning over the validation set and select the optimal parameters based on the mean Hit@10 rate. The results discussed in the next section are obtained using the optimal hyper-parameters as, Enron-CVAE: $zd = 256, hd = 512, lr = 0.0001$; HT09-CVAEH: $zd = 128, hd = 1024, ts = 2, lr = 0.0001$; SFHH-CVAEH: $zd = 128, hd = 1024, ts = 2, lr = 0.0001$; Meetup-NYC-CVAE: $zd = 32, hd = 1024, lr = 0.001$; Meetup-CA-CVAE: $zd = 256, hd = 128, lr = 0.001$.

For **group-recommendation**, we similarly hold out the last member of each sample g_{i} in the validation set and the test set, which leads to $v_{ob} = v_{\mathrm{i},k}$, $g_{pos} = g_{unob} =$

$\{v_{\mathrm{i},1}, v_{\mathrm{i},2}, \cdots, v_{\mathrm{i},k-1}\}$, and $S_{neg} = \{g_{neg,\mathrm{i}} | v_{\mathrm{i},k} \notin g_{neg,\mathrm{i}}, \mathrm{i} = 1, 2, ..., 200\}$. If g_{pos} is among the top m highest ranking groups in $\{g_{pos}\} \cup S_{neg}$, we register a hit for the Hit@m rate. We perform hyper-parameter tuning over the validation set using the mean Hit@10 rate, and obtain the optimal hyper-parameters as, Enron-CVAE: $zd = 256, hd = 1024, lr = 0.01$; HT09-CVAEH: $zd = 32, hd = 128, ts = 2, lr = 0.0001$; SFHH-CVAEH: $zd = 64, hd = 1024, ts = 2, lr = 0.0001$; Meetup-NYC-CVAE: $zd = 64, hd = 1024, lr = 0.01$; Meetup-CA-CVAE: $zd = 32, hd = 256, lr = 0.01$.

Setup for baseline methods For fair comparison, we also perform hyper-parameter tuning for the baseline methods on the same validation set. For the embedding-based methods, we test different embedding dimensions in $\{64, 128, 256, 512\}$. For the neural network models, we also tune the hidden dimension in $\{128, 256, 512, 1024\}$, learning rate in $\{0.0001, 0.001, 0.01\}$ and the batch size in $\{32, 64\}$. For set2vec MLP and LSTM, we test time-step in $\{5, 10, 15\}$ and $\{2, 4, 8, 16, 32\}$, respectively. To convert $r(u, v)$ to $r(g, v)$, we try aggregation in $\{sum, mean, max, min\}$. For the hyper-parameters not mentioned here, we adopt the same values as suggested in the original papers publishing the methods. We use Hit@10 rate for determining the optimal hyper-parameters. The optimal models are compared with CVAE/CVAEH on the same test set.

Hyper-parameter tuning For CVAE and CVAEH model, the encoder and the decoder are both consist of two fully-connected (FC) layers. We tune the hidden dimension (hd) in $\{128, 256, 512, 1024\}$, the dimension (zd) of the latent vector $\mathbf{z}$ in $\{32, 64, 128, 256\}$, and the learning rate (lr) in $\{1\mathrm{e}{-4}, 1\mathrm{e}{-3}, 1\mathrm{e}{-2}\}$. For **CVAEH**, we also tune the window size (ts) for $\mathbf{h}_{\mathrm{i}}$ in $\{2, 4, 8\}$. For the member-recommendation task, we have the optimal hyper-parameters as, Enron-CVAE: $zd = 256, hd = 512, lr = 1\mathrm{e}-4$; HT09-CVAEH: $zd = 128, hd = 1024, ts = 2, lr = 1\mathrm{e}-4$; SFHH-CVAEH: $zd = 128, hd = 1024, ts = 2, lr = 1\mathrm{e}-4$; Meetup-NYC-CVAE: $zd = 32, hd = 1024, lr = 1\mathrm{e}-3$; Meetup-CA-CVAE: $zd = 256, hd = 128, lr = 1\mathrm{e}-3$. For the group-recommendation task, we have the optimal hyper-parameters as, Enron-CVAE: $zd = 256, hd = 1024, lr = 1\mathrm{e}-2$; HT09-CVAEH: $zd = 32, hd = 128, ts = 2, lr = 1\mathrm{e}-4$; SFHH-CVAEH: $zd = 64, hd = 1024, ts = 2, lr = 1\mathrm{e}-4$; Meetup-NYC-CVAE: $zd = 64, hd = 1024, lr = 1\mathrm{e}-2$; Meetup-CA-CVAE: $zd = 32, hd = 256, lr = 1\mathrm{e}-2$.

For one-hot mlp and set2vec mlp, we tune hidden dimension in $\{128, 256, 512, 1024\}$ and learning rate in $\{1e-4, 1e-3, 1e-2, 1e-1\}$. We also try different embedding dimension in $\{32, 64, 128, 256\}$ for set2vec mlp. For set2vec BPR, we tune embedding dimension in $\{128, 256, 512\}$, time-step (LSTM-encoder) in $\{5, 10, 15\}$, and learning rate in $\{1e-4, 1e-3, 1e-2\}$. For the above models, we use a batch size of 64. For LSTM, we tune time-step in $\{2, 4, 8, 16, 32\}$ and hidden dimension in $\{128, 256, 512\}$. We use a batch size of 32, a learning rate of 1.0, a dropout of 0.5 and a 2 layer LSTM architecture. For the Node2Vec model, we tune the embedding dimension in $\{32, 64, 128\}$, p and q in $\{0.1, 0.5, 1.0\}$. For Node2Vec, Common Neighbor, Jaccard Index, Adar and Preferential Attachment, we try aggregation in $\{sum, mean, max, min\}$. For Katz_β, we tune l in $\{1, 2, 3, 4, 5\}$ and β in $\{1e-4, 1e-3, 1e-2, 1e-1\}$. For NMF and SVD, we tune the latent dimension in $\{32, 64, 128, 256\}$. For NMF, we also tune α in $\{0.0, 0.1, 1.0, 10.0\}$ and l_1 ration in $\{0.1, 0.5, 0.9\}$.

6.4.4 Results

Table6.2. Member recommendation hit rates. The highest and the second highest hit rates are bold.

Model	Enron		HT09		SFHH		Meetup-NYC		Meetup-CA	
	Hit@10	Hit@20	Hit@10	Hit@20	Hit@10	Hit@20	Hit@10	Hit@20	Hit@10	Hit@20
Common Neighbors	0.349	0.433	0.075	0.108	0.027	0.186	0.309	0.393	0.359	0.461
Jaccard Index	0.412	0.522	0.057	0.100	0.129	0.302	0.282	0.360	0.320	0.432
Adamic/Adar	0.366	0.453	0.067	0.111	0.033	0.189	0.317	0.384	0.367	**0.474**
Preferential Attachment	0.036	0.046	0.075	0.111	0.039	0.401	0.015	0.047	0.004	0.011
Katz β	0.446	0.561	0.124	0.148	0.165	0.314	**0.318**	**0.409**	**0.387**	**0.483**
Node2Vec	0.188	0.345	0.140	0.232	0.000	0.000	0.070	0.178	0.076	0.206
CTDNE	0.390	0.526	0.154	0.240	0.275	0.290	0.170	0.274	0.193	0.291
HTNE	0.130	0.222	0.412	0.496	0.168	0.222	0.041	0.063	0.071	0.132
metapath2vec	0.277	0.442	0.350	0.588	0.027	0.153	0.107	0.231	0.150	0.275
HERec	0.036	0.074	0.412	0.507	0.192	0.216	0.004	0.010	0.015	0.031
NMF	0.257	0.350	0.466	0.561	0.308	0.356	0.112	0.187	0.124	0.228
SVD	0.278	0.393	0.555	0.677	0.177	0.254	0.126	0.208	0.159	0.266
one-hot MLP	0.462	0.536	0.553	0.712	0.371	0.440	0.326	0.392	0.339	0.428
set2vec MLP	0.434	0.514	0.561	0.617	0.207	0.296	0.261	0.336	0.312	0.381
set2vec BPR	0.374	0.494	0.501	0.590	0.069	0.111	0.173	0.258	0.214	0.316
LSTM	0.438	0.522	0.503	0.621	0.269	0.323	0.253	0.315	0.301	0.385
DGI	0.275	0.368	0.051	0.213	0.051	0.141	0.193	0.258	0.200	0.281
CVAE	**0.521**	**0.601**	**0.615**	**0.728**	**0.428**	**0.500**	**0.341**	**0.406**	**0.400**	0.455
CVAEH	**0.485**	**0.578**	**0.709**	**0.806**	**0.509**	**0.560**	0.275	0.345	0.267	0.331

Table6.3. Member recommendation mean reciprocal ranks (MRR). The highest and the second highest MRR scores are bold.

Model	Enron		HT09		SFHH		Meetup-NYC		Meetup-CA	
	Mrr@10	Mrr@20	Mrr@10	Mrr@20	Mrr@10	Mrr@20	Mrr@10	Mrr@20	Mrr@10	Mrr@20
Common Neighbors	0.186	0.192	0.022	0.024	0.026	0.032	0.161	0.167	0.208	0.215
Jaccard Index	0.210	0.218	0.012	0.015	0.070	0.081	0.127	0.132	0.157	0.165
Adamic/Adar	0.200	0.206	0.036	0.039	0.006	0.017	0.167	0.172	0.213	0.220
Preferential Attachment	0.022	0.023	0.042	0.044	0.005	0.028	0.002	0.005	0.001	0.001
Katz β	0.238	0.246	0.108	0.110	0.043	0.053	0.166	0.173	**0.218**	**0.225**
Node2Vec	0.033	0.044	0.025	0.032	0.000	0.000	0.009	0.017	0.010	0.019
CTDNE	0.107	0.117	0.024	0.030	0.126	0.127	0.041	0.048	0.065	0.071
HTNE	0.032	0.039	0.085	0.0916	0.034	0.037	0.007	0.008	0.015	0.019
metapath2vec	0.068	0.079	0.082	0.100	0.007	0.016	0.020	0.028	0.041	0.049
HERec	0.009	0.012	0.091	0.101	0.026	0.032	0.001	0.001	0.023	0.026
NMF	0.102	0.108	0.410	0.416	0.144	0.147	0.037	0.039	0.026	0.028
SVD	0.114	0.121	0.214	0.222	0.045	0.050	0.027	0.029	0.032	0.034
one-hot MLP	0.276	0.281	0.456	0.461	0.208	0.211	**0.193**	**0.198**	0.210	0.215
set2vec MLP	0.277	0.283	**0.470**	**0.474**	0.096	0.102	0.137	0.142	0.171	0.176
set2vec BPR	0.098	0.106	0.172	0.179	0.010	0.013	0.047	0.053	0.056	0.063
LSTM	0.262	0.267	0.393	0.401	0.080	0.084	0.135	0.140	0.177	0.182
DGI	0.111	0.119	0.018	0.022	0.012	0.019	0.039	0.039	0.087	0.092
CVAE	**0.304**	**0.311**	0.387	0.397	**0.260**	**0.266**	**0.213**	**0.218**	**0.222**	**0.227**
CVAEH	**0.311**	**0.317**	**0.535**	**0.542**	**0.275**	**0.278**	0.154	0.159	0.148	0.152

Member recommendation In Table 6.2 and 6.3, we show the hit rates and MRR scores respectively, for the member recommendation task, using the various models over the five datasets. The methods are arranged in row clusters based on their approaches. Our proposed methods, CVAE and CVAEH are in the last row cluster. For every method, the results are obtained using the optimal hyper-parameters. For the graph topology-based and the embedding-based methods, the results are shown for the best aggregation function among sum, min, max, and mean. For easy comparison, we highlight the highest and the second highest scores.

Hit rates can indicate the models' ability to include the target in the top candidate list. From Table 6.2, we can see that our models perform significantly better than the competing methods in hit rates. Specifically, for the HT09 and SFHH datasets, our CVAEH model's performance is around 20% better than the best competing method in both Hit@10 and Hit@20 metrics. For other datasets, our models' performance is generally 4% to 8% better over the best of the competing methods. Only for the Hit@20 rate for the Meetup-NYC and the Meetup-CA datasets, our method came in a close second and fourth position, out of the 17 competing methods.

Table6.4. Group recommendation hit rates. The highest and the second highest hit rates are bold.

Model	Enron		HT09		SFHH		Meetup-NYC		Meetup-CA	
	Hit@10	Hit@20	Hit@10	Hit@20	Hit@10	Hit@20	Hit@10	Hit@20	Hit@10	Hit@20
Common Neighbors	0.318	0.422	0.054	0.084	0.009	0.030	0.123	0.200	0.159	0.358
Jaccard Index	0.531	0.603	0.264	0.380	0.204	0.231	0.215	0.300	0.332	0.487
Adamic/Adar	0.202	0.357	0.054	0.092	0.015	0.027	0.179	0.242	0.224	0.379
Preferential Attachment	0.078	0.129	0.008	0.038	0.009	0.045	0.099	0.141	0.043	0.095
Katz β	0.150	0.307	0.016	0.046	0.021	0.051	0.267	0.352	0.336	0.478
Node2Vec	0.469	0.565	0.067	0.089	0.024	0.039	0.184	0.271	**0.362**	0.509
CTDNE	0.641	0.719	0.194	0.358	0.195	0.195	**0.294**	0.379	0.293	0.440
HTNE	0.698	0.775	0.318	0.412	0.006	0.021	0.222	0.314	0.345	0.517
metapath2vec	0.562	0.734	**0.515**	**0.574**	**0.497**	**0.572**	**0.410**	**0.500**	**0.418**	**0.608**
HERec	0.141	0.231	0.089	0.092	0.063	0.120	0.209	0.272	0.142	0.228
NMF	0.579	0.820	0.429	0.482	0.222	0.311	0.233	0.332	0.293	0.461
SVD	0.700	0.782	0.434	0.520	0.269	0.362	0.258	0.357	0.315	0.470
one-hot MLP	**0.751**	0.826	0.488	0.536	0.042	0.120	0.278	0.372	0.284	0.517
set2vec MLP	0.737	**0.837**	0.474	0.531	0.030	0.072	0.274	0.352	0.306	0.448
set2vec BPR	0.727	0.818	0.488	0.585	0.153	0.344	0.274	0.368	0.323	0.500
DGI	0.643	0.739	0.173	0.280	0.290	0.437	0.200	0.242	0.203	0.336
CVAE	**0.757**	**0.835**	0.501	0.571	0.066	0.093	0.287	**0.381**	0.332	**0.522**
CVAEH	0.706	0.796	**0.768**	**0.873**	**0.512**	**0.743**	0.220	0.332	0.181	0.349

MRR scores can indicate how high the models rank the target in the candidate list. From Table 6.3, we can see that for all datasets, the highest MRR scores are achieved by either CVAE or CVAEH. In particular, for the HT09 and SFHH datasets, our CVAEH model outperforms the competing models by $\sim 14\%$ and $\sim 32\%$, respectively. For the other datasets, our method is better than the best competing method by at least $\sim 10\%$, except for Meetup-CA ($\sim 1\%$ to 2%).

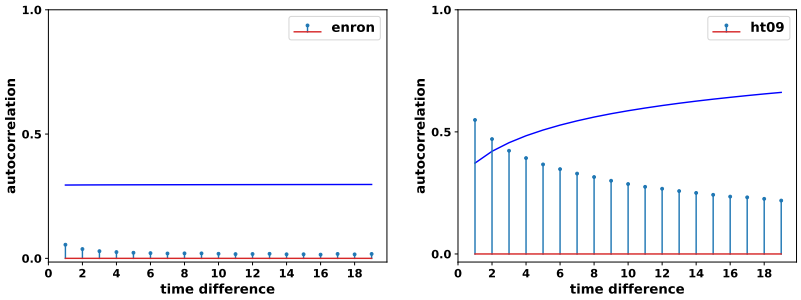
Combining the two metrics, we can see that our method not only predicts the target members more accurately, but also ranks them higher than the competing methods. Moreover, our methods show consistently great performance over the different datasets, as opposed to that, for example, the Katz β method varies from comparable to ours in Meetup-NYC and Meetup-CA to significantly poorer in HT09 and SFHH. This is likely because our model does not explicitly rely on the network topology. Our method is also better than the dynamic network embedding methods (CTDNE and HTNE) and heterogeneous network embedding methods (metapath2vec and HERec), as our method combines the advantages of both approaches, incorporating the dynamic and heterogeneous characteristics of the data.

Table6.5. Group recommendation mean reciprocal ranks (MRR). The highest and the second highest MRR scores are bold.

Model	Enron		HT09		SFHH		Meetup-NYC		Meetup-CA	
	Mrr@10	Mrr@20	Mrr@10	Mrr@20	Mrr@10	Mrr@20	Mrr@10	Mrr@20	Mrr@10	Mrr@20
Common Neighbors	0.135	0.143	0.020	0.022	0.005	0.006	0.052	0.057	0.071	0.083
Jaccard Index	0.264	0.269	0.114	0.122	0.066	0.069	0.132	0.138	**0.196**	**0.208**
Adamic/Adar	0.062	0.073	0.013	0.015	0.007	0.009	0.077	0.081	0.084	0.094
Preferential Attachment	0.019	0.023	0.003	0.005	0.000	0.000	0.005	0.007	0.011	0.014
Katz β	0.050	0.061	0.005	0.007	0.010	0.012	0.109	0.115	0.162	0.171
Node2Vec	0.184	0.191	0.015	0.017	0.013	0.014	0.089	0.095	0.179	0.188
CTDNE	0.320	0.326	0.069	0.081	0.115	0.115	0.160	0.166	0.125	0.135
HTNE	0.360	0.366	0.176	0.182	0.001	0.002	0.100	0.106	0.123	0.134
metapath2vec	0.298	0.310	0.347	0.351	**0.296**	**0.301**	**0.242**	**0.249**	**0.192**	**0.205**
HERec	0.069	0.075	0.024	0.025	0.025	0.029	0.124	0.130	0.062	0.069
NMF	0.317	0.330	**0.411**	**0.421**	0.184	0.188	0.145	0.152	0.120	0.131
SVD	0.430	0.438	0.382	0.387	0.164	0.178	0.146	0.155	0.141	0.149
one-hot MLP	**0.467**	**0.472**	0.392	0.395	0.017	0.022	0.153	0.160	0.103	0.120
set2vec MLP	0.457	0.464	0.393	0.397	0.010	0.013	**0.167**	**0.172**	0.115	0.125
set2vec BPR	0.419	0.425	0.379	0.386	0.111	0.127	0.154	0.160	0.143	0.155
DGI	0.403	0.409	0.027	0.034	0.095	0.096	0.125	0.128	0.077	0.086
CVAE	**0.472**	**0.478**	0.403	0.407	0.025	0.026	0.161	0.168	0.132	0.145
CVAEH	0.389	0.395	**0.537**	**0.545**	**0.258**	**0.275**	0.124	0.132	0.073	0.085

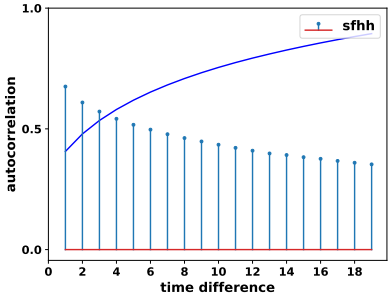
Group recommendation In Table 6.4 and 6.5, we show the hit rates and MRR scores for the group recommendation task, using different methods over the five datasets. Identical to the member recommendation, we show the results of the best hyper-parameters and the best aggregation function (for the graph topology-based and the embedding-based methods).

The results suggest that our CVAEH model outperforms the other models for the HT09 dataset, in both hit rates and MRR scores, leading the runner up by $\sim 50\%$ and $\sim 30\%$, respectively. For the SFHH dataset, CVAEH achieves the highest hit rates and the second highest MRR scores. For the Enron dataset, our CVAE model outperforms the competing baselines except for Hit@20 in a close second position. For the Meetup-NYC and Meetup-CA datasets, our methods score the second or third highest hit rates and MRR. Notice that the static methods like metapath2vec and Jaccard Index perform very well in the Meetup-NYC and Meetup-CA datasets. This is not surprising, as in these datasets, who joins an event with whom is very static, given that the users on meetup.com do not frequently change their interest groups, and per the platform policy, all the events are exclusively joined by the members of the same interest group. In contrast, for the other datasets where group membership is constantly changing, our models are better.

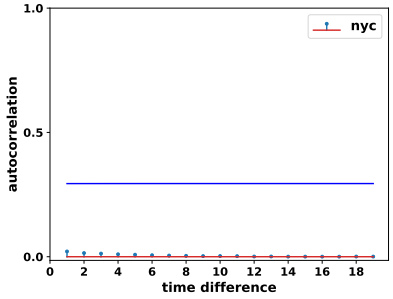


(a) Enron

(b) HT09



(c) SFHH



(d) Meetup-nyc

Figure6.4. Correlograms for the samples in the datasets. The horizontal axis is the time difference; the vertical bars (light blue) are the autocorrelations; the correlation bands (dark blue) indicate the 90% confidence interval. Note, as the correlograms for Meetup-NYC and Meetup-CA are very similar, we only show the one for Meetup-NYC. The plots suggest that the samples in HT09 and SFHH are correlated in ~ 2 steps; while the samples in the other datasets are time independent.

Comparison between CVAE and CVAEH Notice that in Table 6.2 - 6.5, CVAEH outperforms CVAE in HT09 and SFHH; whereas we observe the opposite in the other datasets. This is because the samples (member-list) in these two datasets are correlated in time, while those in the others are not. As the CVAEH model uses the history vector $\mathbf{h}_i$ to take account of the temporal effect, it performs better when the data are time-correlated. Conversely, if the data are time-independent, the history vector $\mathbf{h}_i$ may introduce noise, making the CVAEH model less effective than the CVAE model. In Fig. 6.4, we plot the correlograms for the datasets. The horizontal axis is the time difference (window size); the vertical bars are the autocorrelations; the correlation bands indicate the 90% confidence interval. The plots (the middle two) suggest that the samples in HT09 and SFHH are correlated in ~ 2 steps; while

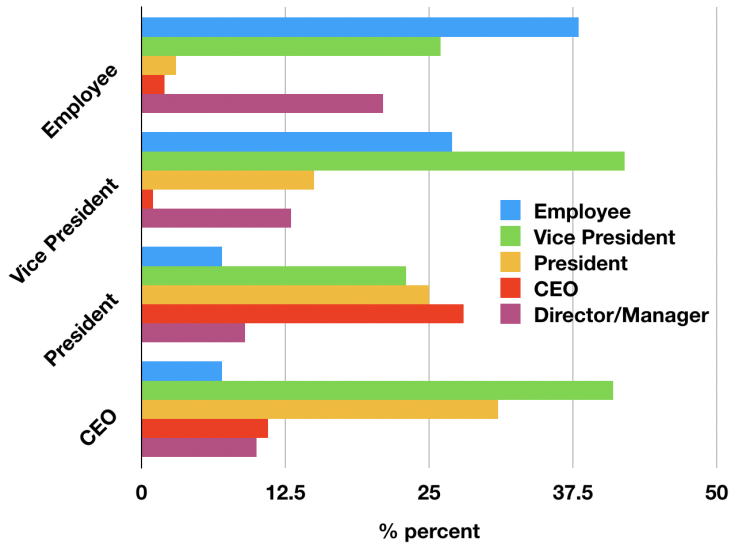


Figure6.5. Case study: Ernon email member composition. The color bars indicate the percent composition of critical members ("influencers") grouped by job titles.

the samples in the other datasets are time independent. In fact, the optimal performance is achieved using a window size of 2 to calculate the history vector $\mathbf{h}_{\mathrm{i}}$ for CVAEH.

Case study: Enron email member composition In this experiment we investigate why particular individuals may be included in an email thread, based upon their relationship with other individuals also receiving the email. Given that an individual v_{ob} is in a group g_{i}, is there a way to auto-detect the key members that lead to the inclusion of v_{ob} in an email? Our modeling framework is well suited for this task. Recall that in group-recommendation, our model provides a probability $P(v|v_{ob}), v \in G$. We therefore consider the group g_{i} and identify a group of "influencers" $\{v|P(v|v_{ob}) \geq c, v \in g_{\mathrm{i}}\}$, where c is a threshold. In particular, we apply this analysis to the Enron dataset and examine the relationship between v_{ob}'s job title and the influenecers' job title [78]. Here we use a threshold $c = 0.1$.

In Fig. 6.5, we plot a bar chart showing the composition of the influencers for v_{ob} (vertical axis) based on her job titles. The horizontal axis indicates the percent composition of influencers' job titles. Moving up the hierarchy, the blue bars shorten, indicating that regular employees are less likely to influence the higher level management people. A similar trend can be observed for the director/manager (purple bars). Conversely, the yellow bars

lengthen, indicating that presidents become more and more important to the people on the top. We also see that regular employees primarily join a conversation because their coworkers or managers are in it. The green bars are long for all roles, suggesting that vice presidents are generally influential. If v_{ob} is a president, she would primarily join a group involving other management people. Lastly, we can see that a CEO is in an email mainly because the presidents or vice presidents are in it. Overall, our model reveals that the individual-influencer relationship obeys the hierarchical structure of the company.

Model reproducibility. We make our datasets and code available at https://github.com/daDiz/cvae-glp.

6.5 Chapter Summary

In this work, we proposed a new problem - group link prediction. Unlike the traditional link prediction which predicts the link between two individuals, our task was to predict the link between an individual and a group. We decompose the problem into member-recommendation and group-recommendation tasks. To solve these two tasks, we proposed a CVAE model and a CVAEH model that takes account of the historical events. The models learned a conditional probability distribution over the unobserved members given the observed ones. We compared our model with a series of competing methods over five real-world datasets. Our CVAEH model shows superior performance for the datasets where the group organizations are correlated in time; while our CVAE model are better in the other cases.

7. Conclusions and future work

In this thesis, we studied the interplay between network topology and event sequence. We divided the problems under consideration into two tracks - given a network, predicting events, and given events, predicting the network. On the first track, we performed two tasks: in Chapter 3, we predicted future events occurring on the nodes of a known network, based on historical events on the same network, using a composite LSTM model with a second-order statistic loss; in Chapter 4, we detected the source(s) of a propagation process on a network, given multiple snapshots of the network states, using a neural network model built on spatial-temporal graph convolution. The second track also included two tasks: in Chapter 5, we revealed the influence network among a group of Twitter users, based on a time series of their tweets, using the Hawkes Binomial Topic Model (HBTM); in Chapter 6, we studied the link formation process in a heterogeneous network, and predicted links between a group of nodes and an individual node, from their previous connections, using a conditional variational autoencoder (CVAE) based model. In the respective chapters, we evaluated our models against a series of state-of-the-art baselines, over synthetic and real-world datasets, and the results proved the superiority of our models.

We see several potential directions for future work. For the composite LSTM in Chapter 3, we can leverage the technique introduced in the neural Hawkes process [95], to incorporate the continuous timestamps, such that the model not only predicts where, but also when the next event occurs. We like to extend the source detection model in Chapter 4, so that it can recover the entire process, a task that is intuitively more difficult than predicting just the source(s). For the Hawkes Binomial Topic Model in Chapter 5, we like to construct an evolving influence network using the tweets by government officials prior to and during the pandemic, to find out how bi-partisan cooperation shifts during national emergencies. Last but not least, for group link prediction in Chapter 6, we like to explore other options for encoding the historical events, for example, combining CVAE with RNN [18].

REFERENCES

[1] Mohammad Al Hasan and Mohammed J Zaki. "A survey of link prediction in social networks". In: *Social network data analytics*. 2011, pp. 243–275.

[2] Réka Albert and Albert-László Barabási. "Statistical mechanics of complex networks". In: *Rev. Mod. Phys.* 74 (1 Jan. 2002), pp. 47–97.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein GAN". In: *arXiv e-prints*, arXiv:1701.07875 (Jan. 2017), arXiv:1701.07875. arXiv: 1701.07875 [stat.ML].

[4] D.M. Auerbach et al. "Cluster of Cases of the Acquired Immune Deficiency Syndrome. Patients Linked by Sexual Contact". In: *Journal of Urology* 132.2 (1984), pp. 421–421.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *ICLR*. 2015.

[6] Baruch Barzel and Albert-László Barabási. "Network link prediction by global silencing of indirect correlations". In: *Nature Biotechnology* 31.8 (2013), pp. 720–725. DOI: 10.1038/nbt.2601.

[7] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (Mar. 1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181.

[8] Ranran Bian et al. "Network Embedding and Change Modeling in Dynamic Heterogeneous Networks". In: SIGIR'19. Paris, France: Association for Computing Machinery, 2019, pp. 861–864. ISBN: 9781450361729. DOI: 10.1145/3331184.3331273. URL: https://doi.org/10.1145/3331184.3331273.

[9] Niklas Boers et al. "Complex networks reveal global pattern of extreme-rainfall teleconnections". In: *Nature* 566.7744 (2019), pp. 373–377. DOI: 10.1038/s41586-018-0872-x. URL: https://doi.org/10.1038/s41586-018-0872-x.

[10] Aleksandar Bojchevski and Stephan Günnemann. "Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking". In: *ICLR '18*. 2018.

[11] Joan Bruna et al. "Spectral Networks and Locally Connected Networks on Graphs". In: *arXiv e-prints*, arXiv:1312.6203 (Dec. 2013), arXiv:1312.6203. arXiv: 1312.6203 [cs.LG].

[12] Cody Buntain, Erin McGrath, and Brandon Behlendorf. "Sampling social media: Supporting information retrieval from microblog data resellers with text, network, and spatial analysis". In: *Proc. of the Hawaii Intl. Conf. on System Sciences*. 2018.

[13] Meeyoung Cha et al. “Measuring User Influence in Twitter: The Million Follower Fallacy”. In: *Fourth International AAAI Conference on Weblogs and Social Media*. May 2010. URL: http://aaai.org/ocs/index.php/ICWSM/ICWSM10/paper/view/1538/0.

[14] H. Chen, X. Li, and Z. Huang. “Link prediction approach to collaborative filtering”. In: *JCDL ’05*. June 2005, pp. 141–142.

[15] Yicheng Cheng, Murat Dundar, and George Mohler. “A coupled ETAS-I 2 GMM point process with applications to seismic fault detection”. In: *Ann. Appl. Stat.* 12.3 (Sept. 2018), pp. 1853–1870. DOI: 10.1214/18-AOAS1134. URL: https://doi.org/10.1214/18-AOAS1134.

[16] Matteo Chinazzi, Jessica T. Davis, et al. “The effect of travel restrictions on the spread of the 2019 novel coronavirus (COVID-19) outbreak”. In: *Science* 368.6489 (2020), pp. 395–400. ISSN: 0036-8075.

[17] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[18] Junyoung Chung et al. “A Recurrent Latent Variable Model for Sequential Data”. In: *Advances in Neural Information Processing Systems*. Vol. 28. Curran Associates, Inc., 2015.

[19] Andrzej Cichocki and Anh Huy Phan. “Fast Local Algorithms for Large Scale Nonnegative Matrix and Tensor Factorizations”. In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 92-A (2009), pp. 708–721.

[20] Matteo et al Cinelli. “The covid-19 social media infodemic”. In: *arXiv:2003.05004* (2020).

[21] Jesper Dall and Michael Christensen. “Random geometric graphs”. In: *Phys. Rev. E* 66 (1 July 2002), p. 016121.

[22] Vachik S. Dave et al. “Neural-Brane: Neural Bayesian Personalized Ranking for Attributed Network Embedding”. In: *Data Science and Engineering* (2018).

[23] Eric H. Davidson et al. “A Genomic Regulatory Network for Development”. In: *Science* 295.5560 (2002), pp. 1669–1678. ISSN: 0036-8075. DOI: 10.1126/science.1069883.

[24] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering”. In: *Advances in Neural Information Processing Systems*. Vol. 29. 2016, pp. 3844–3852.

[25] Luca Dell'Anna. "Solvable delay model for epidemic spreading: the case of Covid-19 in Italy". In: *medRxiv* (2020).

[26] Philip M. Dixon. "Ripley's K Function". In: *Wiley StatsRef: Statistics Reference Online*. 2014. ISBN: 9781118445112. DOI: 10.1002/9781118445112.stat07751. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118445112.stat07751. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118445112.stat07751.

[27] Benjamin Doerr, Mahmoud Fouz, and Tobias Friedrich. "Why Rumors Spread so Quickly in Social Networks". In: *Commun. ACM* 55.6 (June 2012), pp. 70–75. ISSN: 0001-0782.

[28] Carl Doersch. *Tutorial on Variational Autoencoders*. 2021. arXiv: 1606.05908 [stat.ML].

[29] W. Dong, W. Zhang, and C. W. Tan. "Rooting out the rumor culprit from suspects". In: *2013 IEEE International Symposium on Information Theory*. 2013, pp. 2671–2675.

[30] Yuxiao Dong, Nitesh V. Chawla, and Ananthram Swami. "Metapath2vec: Scalable Representation Learning for Heterogeneous Networks". In: *KDD '17*. 2017.

[31] Pauline van den Driessche. "Reproduction numbers of infectious disease models". In: *Infectious Disease Modelling* 2.3 (2017), pp. 288–303.

[32] Nan Du et al. "Dirichlet-hawkes processes with applications to clustering continuous-time document streams". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 219–228.

[33] Nan Du et al. "Recurrent Marked Temporal Point Processes: Embedding Event History to Vector". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA, 2016, pp. 1555–1564. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939875. URL: http://doi.acm.org/10.1145/2939672.2939875.

[34] Nan Du et al. "Scalable Influence Estimation in Continuous-Time Diffusion Networks". In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. 2013, pp. 3147–3155.

[35] Nan Du et al. "Uncover Topic-Sensitive Information Diffusion Networks". In: *AISTATS*. 2013.

[36] Christopher L. DuBois et al. "netdata: A Collection of Network Data". In: (2003).

[37] Hridoy Sankar Dutta et al. "HawkesEye: Detecting Fake Retweeters using Hawkes Process and Topic Modeling". In: *IEEE Transactions on Information Forensics and Security* (2020).

[38] Michael Eichler, Rainer Dahlhaus, and Johannes Dueck. "Graphical Modeling for Multivariate Hawkes Processes with Nonparametric Link Functions". In: *Journal of Time Series Analysis* 38.2 (2017), pp. 225–242. DOI: https://doi.org/10.1111/jtsa.12213.

[39] Mehrdad Farajtabar et al. "Coevolve: A joint point process model for information diffusion and network evolution". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 1305–1353.

[40] Mehrdad Farajtabar et al. "Shaping Social Activity by Incentivizing Users". In: *CoRR* abs/1408.0406 (2014). arXiv: 1408.0406. URL: http://arxiv.org/abs/1408.0406.

[41] Katayoun Farrahi, Rémi Emonet, and Manuel Cebrian. "Epidemic contact tracing via communication traces". In: *PloS one* 9.5 (May 2014), e95133–e95133.

[42] Wei Feng and Jianyong Wang. "Incorporating Heterogeneous Information for Personalized Tag Recommendation in Social Tagging Systems". In: *KDD '12*. 2012.

[43] C. Févotte and J. Idier. "Algorithms for Nonnegative Matrix Factorization with the -Divergence". In: *Neural Computation* 23.9 (2011), pp. 2421–2456.

[44] Vincenzo Fioriti and Marta Chinnici. "Predicting the sources of an outbreak with a spectral technique". In: *Appl. Math. Sci.* 8.135 (2014), pp. 6775–6782.

[45] John Frank and Charles Kingman. *Poisson processes*. 1993.

[46] Mathieu G'enois and Alain Barrat. "Can co-location be used as a proxy for face-to-face contacts?" In: *EPJ Data Science* 7.1 (May 2018), p. 11.

[47] Timothy S. Gardner et al. "Inferring Genetic Networks and Identifying Compound Mode of Action via Expression Profiling". In: *Science* 301.5629 (2003), pp. 102–105. ISSN: 0036-8075. DOI: 10.1126/science.1081900.

[48] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *arXiv e-prints*, arXiv:1412.6572 (Dec. 2014), arXiv:1412.6572. arXiv: 1412.6572 [stat.ML].

[49] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada, 2014, pp. 2672–2680. URL: http://dl.acm.org/citation.cfm?id=2969033.2969125.

[50] Alex Graves. "Generating Sequences With Recurrent Neural Networks". In: *arXiv* (2013). arXiv: 1308.0850.

[51] Alex Graves and Navdeep Jaitly. “Towards End-to-end Speech Recognition with Recurrent Neural Networks”. In: *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*. ICML’14. Beijing, China, 2014, pp. II-1764–II-1772. URL: http://dl.acm.org/citation.cfm?id=3044805.3045089.

[52] Karol Gregor et al. “DRAW: A Recurrent Neural Network For Image Generation”. In: *CoRR* abs/1502.04623 (2015). arXiv: 1502.04623. URL: http://arxiv.org/abs/1502.04623.

[53] Aditya Grover and Jure Leskovec. “Node2Vec: Scalable Feature Learning for Networks”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’16. San Francisco, California, USA, 2016, pp. 855–864. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939754. URL: http://doi.acm.org/10.1145/2939672.2939754.

[54] Aditya Grover and Jure Leskovec. “node2vec: Scalable Feature Learning for Networks”. In: *KDD ’16*. 2016.

[55] N. Halko, P. G. Martinsson, and J. A. Tropp. “Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions”. In: *SIAM Rev.* 53.2 (May 2011), pp. 217–288. ISSN: 0036-1445.

[56] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *NIPS ’17*. 2017.

[57] William L. Hamilton, Rex Ying, and Jure Leskovec. “Representation Learning on Graphs: Methods and Applications”. In: *IEEE Data Engineering Bulletin* (2017).

[58] David K. Hammond, Pierre Vandergheynst, and Rémi Gribonval. “Wavelets on graphs via spectral graph theory”. In: *Applied and Computational Harmonic Analysis* 30.2 (2011), pp. 129–150. ISSN: 1063-5203. DOI: https://doi.org/10.1016/j.acha.2010.04.005. URL: http://www.sciencedirect.com/science/article/pii/S1063520310000552.

[59] Mohammad Al Hasan et al. “Link prediction using supervised learning”. In: *In Proc. of SDM 06 workshop on Link Analysis, Counterterrorism and Security*. 2006.

[60] Alan G. Hawkes. “Spectra of Some Self-Exciting and Mutually Exciting Point Processes”. In: *Biometrika* 58.1 (1971), pp. 83–90. ISSN: 00063444. URL: http://www.jstor.org/stable/2334319.

[61] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9 (1997), pp. 1735–1780.

[62] Petter Holme and Jari Saramäki. "Temporal networks". In: *Physics Reports* 519.3 (Oct. 2012), pp. 97–125. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2012.03.001. URL: http://dx.doi.org/10.1016/j.physrep.2012.03.001.

[63] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[64] Lorenzo Isella et al. "What's in a Crowd? Analysis of Face-to-Face Behavioral Networks". In: *Journal of Theoretical Biology* 271.1 (2011), pp. 166–180.

[65] Oleg Ivanov, Michael Figurnov, and Dmitry Vetrov. "Variational Autoencoder with Arbitrary Conditioning". In: *ICLR '19*. 2019.

[66] Vicnesh Jahmunah, Vidya K. Sudarshan, et al. "Future IoT tools for COVID-19 contact tracing and prediction: A review of the state-of-the-science". In: *International Journal of Imaging Systems and Technology* 31.2 (2021), pp. 455–471.

[67] J. Jiang et al. "Identifying Propagation Sources in Networks: State-of-the-Art and Comparative Studies". In: *IEEE Communications Surveys Tutorials* 19.1 (2017), pp. 465–481.

[68] Brian Karrer and M. E. J. Newman. "Message passing approach for general epidemic models". In: *Phys. Rev. E* 82 (1 July 2010), p. 016101.

[69] W. O. Kermack and A. G. McKendrick. "A Contribution to the Mathematical Theory of Epidemics". In: *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character* 115.772 (1927), pp. 700–721.

[70] B. Killworth and H. Bernard. "Informant accuracy in social network data". In: *Human Organization* 35 (1976), pp. 269–286.

[71] Minkyoung Kim, Dean Paini, and Raja Jurdak. "Real-world diffusion dynamics based on point process approaches: A review". In: *Artificial Intelligence Review* 53.1 (2020), pp. 321–350.

[72] Diederik P Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *ICLR '14*. 2014.

[73] Durk P Kingma et al. "Semi-supervised Learning with Deep Generative Models". In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014.

[74] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *ICLR*. 2017.

[75] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[76] Istvan Z. Kiss, Gergely Röst, and Zsolt Vizi. "Generalization of Pairwise Models to non-Markovian Epidemics on Networks". In: *Phys. Rev. Lett.* 115 (7 Aug. 2015), p. 078701.

[77] Istvan Z Kiss, Joel C Miller, and Peter Simon. *(Book) Mathematics of epidemics on networks: from exact to approximate models*. Springer, 2017.

[78] Bryan Klimt and Yiming Yang. "The Enron Corpus: A New Dataset for Email Classification Research". In: *Lecture Notes in Artificial Intelligence (Subseries of Lecture Notes in Computer Science)* 3201 (Sept. 2004), pp. 217–226.

[79] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (Aug. 2009), pp. 30–37. ISSN: 0018-9162.

[80] Haewoon Kwak et al. "What is Twitter, a social network or a news media?" In: *Proceedings of the 19th international conference on World wide web*. ACM. 2010, pp. 591–600. ISBN: 9781605587998.

[81] Eric Lai et al. *Topic Time Series Analysis of Microblogs*. Tech. rep. DTIC Document, 2014.

[82] Ni Lao and William W. Cohen. "Relational retrieval using a combination of path-constrained random walks". In: *Machine Learning* 81 (2010), pp. 53–67.

[83] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. ISSN: 0018-9219. DOI: 10.1109/5.726791.

[84] Chengwei Lei and Jianhua Ruan. "A novel link prediction algorithm for reconstructing protein–protein interaction networks by topological similarity". In: *Bioinformatics* 29.3 (Dec. 2012), pp. 355–364.

[85] Erik A. Lewis and George O. Mohler. "RESEARCH ARTICLE A Nonparametric EM algorithm for Multiscale Hawkes Processes". In: 2011.

[86] Jundong Li et al. "Attributed Network Embedding for Learning in a Dynamic Environment". In: *CIKM '17*. 2017.

[87] David Liben-Nowell and Jon Kleinberg. "The Link Prediction Problem for Social Networks". In: *CIKM '03*. 2003.

[88] Xingjie Liu et al. “Exploring Personal Impact for Group Recommendation”. In: *CIKM ’12*. 2012.

[89] Andrey Y. Lokhov et al. “Inferring the origin of an epidemic with a dynamic message-passing algorithm”. In: *Phys. Rev. E* 90 (1 July 2014), p. 012801.

[90] Linyuan Lü and Tao Zhou. “Link prediction in complex networks: A survey”. In: *Physica A: statistical mechanics and its applications* 390.6 (2011), pp. 1150–1170.

[91] Dixin Luo et al. “Multi-task Multi-dimensional Hawkes Processes for Modeling Event Sequences”. In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI’15. Buenos Aires, Argentina, 2015, pp. 3685–3691. ISBN: 978-1-57735-738-4. URL: http://dl.acm.org/citation.cfm?id=2832747.2832763.

[92] W. Luo, W. P. Tay, and M. Leng. “Identifying Infection Sources and Regions in Large Networks”. In: *IEEE Transactions on Signal Processing* 61.11 (2013), pp. 2850–2865.

[93] Yao Ma et al. “Streaming Graph Neural Networks”. In: *CoRR* abs/1810.10627 (2018). arXiv: 1810.10627. URL: http://arxiv.org/abs/1810.10627.

[94] David Marsan and Olivier Lengliné. “Extending Earthquakes’ Reach Through Cascading”. In: *Science* 319.5866 (2008), pp. 1076–1079. ISSN: 0036-8075. DOI: 10.1126/science.1148783. eprint: http://science.sciencemag.org/content/319/5866/1076.full.pdf. URL: http://science.sciencemag.org/content/319/5866/1076.

[95] H. Mei and J. Eisner. “The Neural Hawkes Process: A Neurally Self-Modulating Multivariate Point Process”. In: *CoRR* abs/1612.09328 (2016). arXiv: 1612.09328. URL: http://arxiv.org/abs/1612.09328.

[96] Tomas Mikolov et al. “Distributed Representations of Words and Phrases and their Compositionality”. In: *NIPS ’13*. 2013.

[97] Joel C. Miller and Tony Ting. “EoN (Epidemics on Networks): a fast, flexible Python package for simulation, analytic approximation, and analysis of epidemics on networks”. In: *Journal of Open Source Software* 4.44 (2019), p. 1731.

[98] Lawrence Mitchell and Michael E. Cates. “Hawkes process as a model of social interactions: a view on video dynamics”. In: *Journal of Physics A Mathematical General* 43, 045101 (Jan. 2010), p. 045101. DOI: 10.1088/1751-8113/43/4/045101. arXiv: 0907.3864 [physics.soc-ph].

[99] G. O. Mohler et al. “Self-Exciting Point Process Modeling of Crime”. In: *Journal of the American Statistical Association* 106.493 (2011), pp. 100–108. DOI: 10.1198/jasa.2011.ap09546. eprint: https://doi.org/10.1198/jasa.2011.ap09546.

[100] George Mohler et al. “Hawkes binomial topic model with applications to coupled conflict-Twitter data”. In: *DOI: 10.13140/RG.2.2.13638.83527* (2016).

[101] GO Mohler et al. “Self-exciting point process modeling of crime”. In: *Journal of the American Statistical Association* 106.493 (2011), pp. 100–108.

[102] M. E. J. Newman. “Coauthorship networks and patterns of scientific collaboration”. In: *Proceedings of the National Academy of Sciences* 101.suppl 1 (2004), pp. 5200–5205.

[103] M. E. J. Newman. *Networks: an introduction*. 2010.

[104] M. E. J. Newman, D. J. Watts, and S. H. Strogatz. “Random graph models of social networks”. In: *Proceedings of the National Academy of Sciences* 99.suppl 1 (2002), pp. 2566–2572.

[105] Giang Hoang Nguyen et al. “Continuous-Time Dynamic Network Embeddings”. In: *WWW ’18*. 2018.

[106] Ornulf Borgan Odd Aalen and Hakon Gjessing. *Survival and event history analysis: a process point of view*. 2008.

[107] Yosihiko Ogata. “On Lewis’ simulation method for point processes”. In: *IEEE Trans. Information Theory* 27 (1981), pp. 23–30.

[108] Yosihiko Ogata. “Space-Time Point-Process Models for Earthquake Occurrences”. In: *Annals of the Institute of Statistical Mathematics* 50.2 (1998), pp. 379–402. URL: https://EconPapers.repec.org/RePEc:spr:aistmt:v:50:y:1998:i:2:p:379-402.

[109] Aäron van den Oord et al. “WaveNet: A Generative Model for Raw Audio”. In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

[110] Openreview. https://openreview.net/forum?id=xQnvyc6r3LL. 2021.

[111] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the Difficulty of Training Recurrent Neural Networks”. In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML’13. Atlanta, GA, USA, 2013, pp. III-1310–III-1318. URL: http://dl.acm.org/citation.cfm?id=3042817.3043083.

[112] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “DeepWalk: Online Learning of Social Representations”. In: *KDD ’14*. 2014.

[113] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. "DeepWalk: Online Learning of Social Representations". In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '14. New York, New York, USA, 2014, pp. 701–710. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623732. URL: http://doi.acm.org/10.1145/2623330.2623732.

[114] T. N. Pham et al. "A general graph-based model for recommendation in event-based social networks". In: *ICDE '15*. 2015, pp. 567–578.

[115] Andreas Plesch et al. "Community Fault Model (CFM) for Southern California." In: *Bulletin of the Seismological Society of America* 97(6) (2007).

[116] B. A. Prakash, J. Vreeken, and C. Faloutsos. "Spotting Culprits in Epidemics: How Many and Which Ones?" In: *2012 IEEE 12th International Conference on Data Mining*. 2012, pp. 11–20.

[117] B. Aditya Prakash, Jilles Vreeken, and Christos Faloutsos. "Efficiently spotting the starting points of an epidemic in a large graph". In: *Knowl. Inf. Syst.* 38.1 (2014), pp. 35–59.

[118] Juan A. Recio-Garcia et al. "Personality Aware Recommendations to Groups". In: *RecSys '09*. 2009.

[119] Steffen Rendle et al. "BPR: Bayesian Personalized Ranking from Implicit Feedback". In: *UAI '09*. 2009.

[120] B. D. Ripley. "The Second-Order Analysis of Stationary Point Processes". In: *Journal of Applied Probability* 13.2 (1976), pp. 255–266. ISSN: 00219002. URL: http://www.jstor.org/stable/3212829.

[121] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Neurocomputing: Foundations of Research". In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA, 1988. Chap. Learning Representations by Back-propagating Errors, pp. 696–699. ISBN: 0-262-01097-6.

[122] Marcel Salathé et al. "A high-resolution human contact network for infectious disease transmission". In: *Proceedings of the National Academy of Sciences* 107.51 (2010), pp. 22020–22025.

[123] SCEDC. *Southern California Earthquake Center*. Caltech.Dataset. 2013. DOI: 10.7909/C3WD3xH1. URL: https://service.scedc.caltech.edu/eq-catalogs/.

[124] Hao Sha, Mohammad Al Hasan, and George Mohler. "Learning network event sequences using long short-term memory and second-order statistic loss". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 14.1 (2021), pp. 61–73. DOI: https://doi.org/10.1002/sam.11489.

[125] Hao Sha et al. "Dynamic topic modeling of the COVID-19 Twitter narrative among U.S. governors and cabinet executives". In: *arXiv e-prints* (Apr. 2020).

[126] Chintan Shah et al. "Finding Patient Zero: Learning Contagion Source with Graph Neural Networks". In: (2020).

[127] Devavrat Shah and Tauhid Zaman. "Detecting Sources of Computer Viruses in Networks: Theory and Experiment". In: *SIGMETRICS Perform. Eval. Rev.* 38.1 (June 2010), pp. 203–214.

[128] Devavrat Shah and Tauhid Zaman. "Rumors in a Network: Who's the Culprit?" In: *IEEE Trans. Inf. Theory* 57.8 (Aug. 2011), pp. 5163–5181.

[129] Zhesi Shen et al. "Reconstructing propagation networks with natural diversity and identifying hidden sources". In: *Nature Communications* 5.1 (2014), p. 4323. DOI: 10.1038/ncomms5323.

[130] N. Sherborne et al. "Mean-field models for non-Markovian epidemics on networks: from edge-based compartmental to pairwise models". In: *Journal of Mathematical Biology* 76.3 (2018), pp. 755–778.

[131] C. Shi et al. "A survey of heterogeneous information network analysis". In: *IEEE Transactions on Knowledge and Data Engineering* 29.1 (2017), pp. 17–37.

[132] C. Shi et al. "Heterogeneous Information Network Embedding for Recommendation". In: *IEEE Transactions on Knowledge and Data Engineering* 31.2 (2019), pp. 357–370.

[133] M. B. Short et al. "Gang rivalry dynamics via coupled point process networks". In: *Discrete & Continuous Dynamical Systems - B* 19.1531-3492_2014_5_1459 (2014), p. 1459. ISSN: 1531-3492.

[134] Hava T. Siegelmann and Eduardo D. Sontag. "Turing computability with neural nets". In: *Applied Mathematics Letters* 4.6 (1991), pp. 77–80. ISSN: 0893-9659.

[135] Aleksandr Simma and Michael I Jordan. "Modeling events with cascades of Poisson processes". In: *arXiv preprint arXiv:1203.3516* (2012).

[136] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *NIPS '15*. 2015.

[137] Joshua M. Stuart et al. "A Gene-Coexpression Network for Global Discovery of Conserved Genetic Modules". In: *Science* 302.5643 (2003), pp. 249–255. DOI: 10.1126/science.1087447.

[138] Xiaoyuan Su and Taghi M. Khoshgoftaar. "A Survey of Collaborative Filtering Techniques". In: *Adv. in Artif. Intell.* 2009 (Jan. 2009).

[139] Yizhou Sun and Jiawei Han. *Mining Heterogeneous Information Networks: Principles and Methodologies*. Morgan Claypool Publishers, 2012.

[140] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215.

[141] Xi Tan, Vinayak Rao, and Jennifer Neville. "The Indian Buffet Hawkes Process to Model Evolving Latent Influences." In: *UAI*. 2018, pp. 795–804.

[142] Jian Tang et al. "LINE: Large-Scale Information Network Embedding". In: *WWW '15*. 2015.

[143] Xiaolu Tang, Changcheng Wu, et al. "On the origin and continuing evolution of SARS-CoV-2". In: *National Science Review* 7.6 (Mar. 2020), pp. 1012–1023.

[144] Mike Thelwall and Saheeda Thelwall. "Covid-19 Tweeting in English: Gender Differences". In: *aarXiv:2003.11090* (2020).

[145] Mike Thelwall and Saheeda Thelwall. "Retweeting for COVID-19: Consensus building, information sharing, dissent, and lockdown life". In: *arXiv:2004.02793* (2020).

[146] Marc Timme. "Revealing Network Connectivity from Response Dynamics". In: *Phys. Rev. Lett.* 98 (22 May 2007), p. 224101. DOI: 10.1103/PhysRevLett.98.224101. URL: https://link.aps.org/doi/10.1103/PhysRevLett.98.224101.

[147] Lauren Tindale et al. "Transmission interval estimates suggest pre-symptomatic spread of COVID-19". In: (2020).

[148] Alejandro Veen and Frederic P Schoenberg. "Estimation of space–time branching process models in seismology using an em–type algorithm". In: *Journal of the American Statistical Association* 103.482 (2008), pp. 614–624.

[149] Theresa A. Velden, Asif-ul Haque, and Carl J. Lagoze. "A New Approach to Analyzing Patterns of Collaboration in Co-authorship Networks - Mesoscopic Analysis and Interpretation". In: *arXiv e-prints* (2009).

[150] Petar Veličković et al. “Deep Graph Infomax”. In: *ICLR ’19*. 2019.

[151] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. “Order Matters: Sequence to sequence for sets”. In: *ICLR ’15*. 2015.

[152] Wen-Xu Wang et al. “Predicting Catastrophes in Nonlinear Dynamical Systems by Compressive Sensing”. In: *Phys. Rev. Lett.* 106 (15 Apr. 2011), p. 154101.

[153] Y. Wang et al. “Modeling the Propagation of Worms in Networks: A Survey”. In: *IEEE Communications Surveys Tutorials* 16.2 (2014), pp. 942–960. DOI: 10.1109/SURV.2013.100913.00195.

[154] Zhaoxu Wang et al. “Rumor Source Detection with Multiple Observations: Fundamental Limits and Algorithms”. In: SIGMETRICS ’14. 2014, pp. 1–13.

[155] Zheng Wang et al. “Multiple Source Detection without Knowing the Underlying Propagation Model”. In: AAAI’17. 2017, pp. 217–223.

[156] P. J. Werbos. “Backpropagation through time: what it does and how to do it”. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560.

[157] Shuai Xiao et al. “Modeling The Intensity Function Of Point Process Via Recurrent Neural Networks”. In: *AAAI*. 2017.

[158] Shuai Xiao et al. “Wasserstein Learning of Deep Generative Point Process Models”. In: *CoRR* abs/1705.08051 (2017). arXiv: 1705.08051. URL: http://arxiv.org/abs/1705.08051.

[159] Ze Xiao and Yue Deng. “Graph embedding-based novel protein interaction prediction via higher-order graph convolutional network”. In: *PLOS ONE* 15.9 (Sept. 2020), pp. 1–18. DOI: 10.1371/journal.pone.0238915.

[160] Hongteng Xu, Mehrdad Farajtabar, and Hongyuan Zha. “Learning Granger Causality for Hawkes Processes”. In: *CoRR* abs/1602.04511 (2016). arXiv: 1602.04511. URL: http://arxiv.org/abs/1602.04511.

[161] Hongteng Xu, Mehrdad Farajtabar, and Hongyuan Zha. “Learning granger causality for hawkes processes”. In: *International Conference on Machine Learning*. 2016, pp. 1717–1726.

[162] Hongteng Xu and Hongyuan Zha. “A dirichlet mixture model of hawkes processes for event sequence clustering”. In: *Advances in Neural Info. Processing Systems*. 2017, pp. 1354–1363.

[163] Hongteng Xu and Hongyuan Zha. "THAP: A Matlab Toolkit for Learning with Hawkes Processes". In: *arXiv e-prints*, arXiv:1708.09252 (Aug. 2017), arXiv:1708.09252. arXiv: 1708.09252 [stat.ML].

[164] Paiheng Xu, Mark Dredze, and David A Broniatowski. "The Twitter Social Mobility Index: Measuring Social Distancing Practices from Geolocated Tweets". In: *arXiv* (2020). eprint: 2004.02397.

[165] Sijie Yan, Yuanjun Xiong, and Dahua Lin. "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition". In: *AAAI'18* (2018).

[166] Y. Yan et al. "A Survey on Smart Grid Communication Infrastructures: Motivations, Requirements and Challenges". In: *IEEE Communications Surveys Tutorials* 15.1 (2013), pp. 5–20. DOI: 10.1109/SURV.2012.021312.00034.

[167] Zhijun Yao et al. "A review of structural and functional brain networks: small world and atlas". In: *Brain Informatics* 2.1 (2015), pp. 45–52. DOI: 10.1007/s40708-015-0009-z. URL: https://doi.org/10.1007/s40708-015-0009-z.

[168] Fulian Yin et al. "COVID-19 information propagation dynamics in the Chinese Sina-microblog". In: *Math. Biosciences and Eng.* 17.3 (2020), p. 2676.

[169] Rex Ying et al. "Graph Convolutional Neural Networks for Web-Scale Recommender Systems". In: *KDD '18*. 2018.

[170] Bing Yu, Haoteng Yin, and Zhanxing Zhu. "Spatio-temporal Graph Convolutional Networks: A Deep Learning Framework for Traffic Forecasting". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018.

[171] Daokun Zhang et al. "Network representation learning: A survey". In: *IEEE transactions on Big Data* (2018).

[172] Wei Zhang, Jianyong Wang, and Wei Feng. "Combining latent factor model with location features for event-based group recommendation". In: *KDD*. 2013, pp. 910–918.

[173] Qingyuan Zhao et al. "SEISMIC: A Self-Exciting Point Process Model for Predicting Tweet Popularity". In: *CoRR* abs/1506.02594 (2015). arXiv: 1506.02594.

[174] Qingyuan Zhao et al. "Seismic: A self-exciting point process model for predicting tweet popularity". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 1513–1522.

[175] Jie Zhou et al. *Graph Neural Networks: A Review of Methods and Applications*. 2019. arXiv: 1812.08434.

[176] K. Zhou, H. Zha, and L. Song. "Learning Triggering Kernels for Multi-dimensional Hawkes Processes". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. Atlanta, GA, USA, 2013, pp. III-1301–III-1309.

[177] Ke Zhou, Hongyuan Zha, and Le Song. "Learning Social Infectivity in Sparse Low-rank Networks Using Multi-dimensional Hawkes Processes". In: *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*. 2013, pp. 641–649.

[178] L. Zhou et al. "Dynamic Network Embedding by Modelling Triadic Closure Process". In: *AAAI*. 2018.

[179] L. Zhou et al. "Dynamic Network Embedding by Modelling Triadic Closure Process". In: *AAAI*. 2018.

[180] K. Zhu and L. Ying. "Information Source Detection in the SIR Model: A Sample-Path-Based Approach". In: *IEEE/ACM Transactions on Networking* 24.1 (2016), pp. 408–421.

[181] L. Zhu et al. "Scalable Temporal Latent Space Inference for Link Prediction in Dynamic Social Networks". In: *IEEE Transactions on Knowledge and Data Engineering* 28.10 (Oct. 2016), pp. 2765–2777. ISSN: 1041-4347. DOI: 10.1109/TKDE.2016.2591009.

[182] Linhong Zhu, Greg Ver Steeg, and Aram Galstyan. "Scalable Link Prediction in Dynamic Networks via Non-Negative Matrix Factorization". In: *IEEE Transactions on Knowledge and Data Engineering 2016* (2016).

[183] Yuan Zuo et al. "Embedding Temporal Network via Neighborhood Formation". In: *KDD '18*. 2018.

VITA

Hao Sha received his BS in Physics from University of Illinois at Urbana-Champaign in December, 2013. He then received a MS in Physics from Indiana University Purdue University Indianapolis in December, 2016. He joined the Department of Computer and Information Science of Indiana University Purdue University Indianapolis in August, 2018 to pursue his Ph.D. degree in computer science under the supervision of Dr. George Mohler and Dr. Mohammad Al Hasan. His research focused on statistical and deep learning approaches to solving problems in time series prediction, networks, natural language processing, computer vision, and recommender systems.

PUBLICATIONS

H. Sha, M. Al Hasan, G. Mohler. Group Link Prediction Using Conditional Variational Autoencoder. Accepted at the International AAAI Conference on Web and Social Media (ICWSM 2021).

H. Sha, M. Al Hasan, P. J. Brantingham, and G. Mohler. Dynamic topic modeling of the COVID-19 Twitter narrative among U.S. governors and cabinet executives. 5th International Workshop on Social Sensing (SocialSens 2020).

Sha, H, Al Hasan, M, Mohler, G. Learning network event sequences using long short-term memory and second-order statistic loss. Stat Anal Data Min: The ASA Data Sci Journal. 2021; 14: 61– 73.

H. Sha, M. A. Hasan, J. Carter and G. Mohler, Interpretable Hawkes Process Spatial Crime Forecasting with TV-Regularization, 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 3228-3236, doi: 10.1109/BigData50022.2020.9377984.

J. Wong, H. Sha, M. A. Hasan, G. Mohler, S. Becker and C. Wiltse, Automated Corn Ear Height Prediction Using Video-Based Deep Learning, 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 2371-2374, doi: 10.1109/BigData50022.2020.9378115.

A. Stanhope, H. Sha, D. Barman, M. A. Hasan and G. Mohler, Group Link Prediction, 2019 IEEE International Conference on Big Data (Big Data), 2019, pp. 3045-3052, doi: 10.1109/BigData47090.2019.9006261.

A. Baas, F. Hung, H. Sha, M. A. Hasan and G. Mohler, Predicting Virality on Networks Using Local Graphlet Frequency Distribution, 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 2475-2482, doi: 10.1109/BigData.2018.8622605.