Georgia State University

ScholarWorks @ Georgia State University

Mathematics Dissertations

Department of Mathematics and Statistics

8-10-2021

Network Inference, Influence Estimation and Maximization via Neural Mean-field Dynamics on Diffusion Network

Shushan He

Follow this and additional works at: https://scholarworks.gsu.edu/math_diss

Recommended Citation

He, Shushan, "Network Inference, Influence Estimation and Maximization via Neural Mean-field Dynamics on Diffusion Network." Dissertation, Georgia State University, 2021.
doi: https://doi.org/10.57709/23984105

This Dissertation is brought to you for free and open access by the Department of Mathematics and Statistics at ScholarWorks @ Georgia State University. It has been accepted for inclusion in Mathematics Dissertations by an authorized administrator of ScholarWorks @ Georgia State University. For more information, please contact scholarworks@gsu.edu.

Network Inference, Influence Estimation and Maximization via Neural Mean-field Dynamics on Diffusion Network

by

Shushan He

Under the Direction of Xiaojing Ye, PhD

ABSTRACT

We propose two novel learning frameworks using neural mean-field (NMF) dynamics for inference and estimation problems on heterogeneous diffusion networks in discrete-time and continuous-time setting, respectively. The frameworks leverages the Mori-Zwanzig formalism to obtain an exact evolution equation of the individual node infection probabilities, which renders a delay differential equation with memory integral approximated by learnable time convolution operators. Directly using information diffusion cascade data, our frameworks can *simultaneously* learn the structure of the diffusion network and the evolution of node infection probabilities. Connections between parameter learning and optimal control are also established, leading to a rigorous and implementable algorithm for training NMF. Moreover, we show that the projected gradient descent method can be employed to solve the challenging influence maximization problem, where the gradient is computed extremely fast by integrating NMF forward in time just once in each iteration. Extensive empirical studies show that our approach is versatile and robust to variations of the underlying diffusion network models, and significantly outperform existing approaches in accuracy and efficiency on both synthetic and real-world data.

INDEX WORDS: Diffusion networks, Influence estimation, Network inference, Mori-Zwanzig formalism, Influence maximization.

Network Inference, Influence Estimation and Maximization via Neural Mean-field Dynamics on Diffusion Network

by

Shushan He

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2021

Copyright by
Shushan He
2021

NETWORK INFERENCE, INFLUENCE ESTIMATION AND MAXIMIZATION ON DIFFUSION NETWORK USING DEEP LEARNING TOOL AND OPTIMAL CONTROL THEORIES

by

SHUSHAN HE

Committee Chair:	Xiaojing Ye
Committee:	Guantao Chen
	Jonathan Shihao Ji
	Alexandra B Smirnova
	Michael Stewart

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

August 2021

DEDICATION

This dissertation is dedicated to my parents, my sister, and my husband.

ACKNOWLEDGEMENTS

When I start to write on this page, I am sitting in front of a bright window, a messy table with books, and my unfinished breakfast... I am trying to summarize the process for me to get here, but I realize it is so difficult and my tear suddenly comes out without control. One question jump to my mind: how to become a good mathematician? This is one question from my advisor, Dr. Guantao Chen, who likes to discuss it with us especially when there is new member joining the team. I can not remember any conclusions from these discussions, but this question appears in my mind often with a big question mark.

As a person who spend most of her best life time to learn the mathematics, I do not think I am a good mathematician, but I am sure I love it. I chose mathematics as my major when I was 18. At that moment, my only dream is becoming a teacher who can share knowledge, experience, kindness, positive energy and many others to the students. I did not even have the idea on what I should teach until my English teacher suggested me to go with maths. At that time, I could never know how important this decision is to me. Even I could not contribute too much, I learn a lot from my major which helps me becoming who I am now. So my first thanks I want to give to the mathematics.

One of the most important people during my PhD life is my advisor, **Dr. Guantao Chen**. Thank you for taking me on under your guidance. You are like a father who always have our back and support us unconditionally. You are also a reliable mentor who helps me to build my research mindset, and find my research interests. Without you, this thesis would not have been completed. Thank you for making me a Doctor.

Many thanks also to my another advisor **Dr. Xiaojing Ye**. Thank you for everything you have done for me. You opened the door of artificial intelligence for me and show me the magnificent view which extends my world and starts my new career dream. Now when I look back, all the memories look so peaceful, and only good ones left. But I know, it is not always the truth. In the process, I was almost beat by my bad emotions. You gave me the most of patience I have never got from other people. If without your help and support, I might be stuck in some fight with myself. Thank you for sharing the research topic of this thesis with me. Thank you for all the research training you did to me. Thank you for all the talks you gave to me. Thank you Dr. Ye.

I would also like to thank my other thesis committee members: **Dr. Jonathan Shihao Ji**, **Dr. Alexandra B Smirnova**, and **Dr. Michael Stewart**. Thank you for your invaluable feedback and mentorship.

At the end, please allow me to write down my special appreciation to my families and friends.

Mr. Xiaomei He and Mrs. Xiangkun Li, my parents, you are the best parents in the world. Thank you for bring me to the world and teach me to be kind and grateful. Thank you for all your sacrifices for sister and me.

Mrs. Shuhui He, my sister, thank you for your love, and support in every aspect throughout my life. I am so proud of you being the best sister. Same thanks to **Mr. Liang Wang**, my brother in law, for the support to the whole family.

Mr. Chun Wei Wang, my husband, in the adventure to USA, you are the best gift to me. Thank you for your love with patience, kind, endurance and support.

Mr. Ruei Chih Wang and Mrs. Shouli Wanglin, my parents in law, thank you for your love and support. Dear father, we miss you. Have a good trip in the heaven.

Dr. Yan Cao and Dr. Guangming Jing, my friends, thank you for your supports.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . . **v**

LIST OF TABLES . . . . **x**

LIST OF FIGURES . . . . **xi**

CHAPTER 1 INTRODUCTION **1**

1.1 Motivation 1

1.2 Diffusion model 2

1.3 Cascade data 3

1.4 Influence estimation and network inference 4

1.5 Influence maximization 5

1.6 Overview 5

CHAPTER 2 BACKGROUND **7**

2.1 Diffusion models 7

2.1.1 Triggering diffusion models 7

2.1.2 Continuous-time Independent Cascade (CIC) model . . . 9

2.2 Influence estimation 9

2.3 Network inference 13

2.4 Influence maximization 14

CHAPTER 3 NEURAL MEAN-FIELD (NMF) DYNAMICS APPLIED ON INFLUENCE ESTIMATION AND NETWORK INFERENCE **19**

3.1 Mean-field dynamics of diffusion 19

3.1.1 Notations 19

3.1.2 Modelling diffusion by stochastic jump processes 19

3.1.3 Mori-Zwanzig memory closure 23

3.1.4 Delay differential equation 26

3.2 Discrete NMF algorithm 28

3.2.1 Formulation of discrete NMF model 29

3.2.2 Learning the parameters of discrete NMF 30

3.2.3 Revisiting optimal control theory 30

3.2.4 Optimal control of parameter training 32

3.2.5 Implementation details 36

3.2.6 Comparison models 39

3.2.7 Experiment results 40

3.3 Continuous NMF algorithm 46

3.3.1 ODE system for continuous NMF 47

3.3.2 Evolutionary point processes and loss function 48

3.3.3 Optimal control formulation of parameter learning 52

3.3.4 Revisiting neural ODE (NODE) 53

3.3.5 Backpropagation in continuous NMF 55

3.3.6 Experiment evaluation 62

CHAPTER 4 INFLUENCE MAXIMIZATION WITH LEARNED CONTINUOUS NMF **70**

4.1 Optimal control formulation 70

4.1.1 On original influence maximization 70

4.1.2 On relaxed influence maximization 71

4.2 Projected gradient descent (PGD) 71

4.3 Gradients calculation for relaxed IM 72

4.4 Proposed algorithm NMF-InfMax 73

4.5 Experiment evaluation 74

4.5.1 Comparison algorithms 74

4.5.2 Experiment setting 75

4.5.3 Comparison results 76

4.5.4 Real data 77

CHAPTER 5 SUMMARY AND DISCUSSION **79**

5.1 Summary 79

5.2 Future work 79

5.2.1 Implementation for application 79

5.2.2 Dynamic diffusion network 80

5.2.3 Context-aware problems 81

REFERENCES **82**

LIST OF TABLES

3.1 Frequently used notations 20

3.2 Performance of structure inference using NetRate and the proposed discrete NMF on Random, Hierarchical, and Core-periphery networks with Rayleigh distribution as the diffusion time model on edges. Quality of the learned edge set $\mathcal{E}$ and distribution parameter $\boldsymbol{A}$ are measured by precision (Prc), recall (Rcl), accuracy (Acc), and correlation (Cor). 42

3.3 Performance of network structure inference using NetRate [35] and the proposed continuous NMF on Random, Hierarchical, and Core-periphery networks consisting of 128 nodes and 512 edges with Exponential and Rayleigh as diffusion distribution on edges. Quality of the learned edge set $\mathcal{E}$ and distribution parameter $\boldsymbol{A}$ are measured by precision (Prc), recall (Rcl), accuracy (Acc), and correlation (Cor). Larger value indicates higher accuracy. 66

LIST OF FIGURES

1.1 Example of a sample cascade on a diffusion network. The cascade was originated from the source set $\mathcal{S} = \{1\}$ and gradually propagates to other nodes through their directed edge connections. The time line below the network shows the wall-clock time t_i that each node i was infected during the cascade with $t_1 = 0$. The orange edges indicate whom each node got infection from, and $t_{ij} := t_j - t_i$ is the time that node i took to infect node j. 2

3.1 MAE of node infection probability by LSTM, InfluLearner, and discrete NMF on each of the 9 different combinations of Hierarchical (Hier), Core-periphery (Core) and Random (Rand) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 100 tests are shown. . 43

3.2 MAE of influence by LSTM, InfluLearner, and discrete NMF on each of the 9 different combinations of Hierarchical (Hier), Core-periphery (Core) and Random (Rand) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 100 tests are shown. 44

3.3 Ground truth $\boldsymbol{A}^*$ (left) and $\boldsymbol{A}$ inferred NetRate (middle) and discrete NMF (right) under the same color scale using cascaded data from a Hierarchical network with Rayleigh diffusion model. 45

3.4 (a)–(b) MAE of influence (Inf) and infection probability (Prob) estimated by discrete NMF for Hierarchical networks with increasing network sizes from 256 to 2048. (c) runtime (in seconds) for training versus network sizes in log-log scale. 46

3.5 (a) MAE of influence estimated by LSTM, InfluLearner on Weibo data; (b)–(c) MAE of influence and infection probability of discrete NMF for different network densities. 46

3.6 MAE of node infection probability (bottom) by InfluLearner [24] and continuous NMF on each of the 9 different combinations of Core-periphery (Core), Random (Rand) and Hierarchical (Hier) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 50 test source sets are shown. Each network has two configurations of (n, d): $(128, 4)$ and $(1024, 4)$, where n is the number of nodes in the diffusion network, and d is the average out-degree per node. 64

3.7 MAE of scaled influence by InfluLearner [24] and continuous NMF on each of the 9 different combinations of Core-periphery (Core), Random (Rand) and Hierarchical (Hier) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 50 test source sets are shown. Each network has two configurations of (n, d): $(128, 4)$ and $(1024, 4)$, where n is the number of nodes in the diffusion network, and d is the average out-degree per node. 65

3.8 Ground truth $\boldsymbol{A}^*$ (left) and $\boldsymbol{A}$ inferred by NetRate (middle) and continuous NMF (right) in same color scale using cascades from a Hierarchical network consisting of 128 nodes and 512 edges with exponential diffusion model. Darker pixel indicates larger value of an entry of $\boldsymbol{A}$. 67

3.9 MAE of infection probability (a) and influence (b) obtained by InfluLearner [24] and continuous NMF on Hierarchical networks of size $n = 128$ and increasing d from 4 to 6. 67

3.10 (a) Training time (in seconds) of continuous NMF versus density (average out-degree per node) d. (b) Training time (in seconds) versus network size n. 68

4.1 Influence of the source sets selected by the compared methods on three different types of networks: (a) Hierarchical, (b) Random, and (c) Core-periphery, with exponential diffusion model at $T = 10$ and varying source sizes n_0 from 1 to 10. Each network consists of 1024 nodes and 2048 edges (top) or 4096 edges(bottoms). 77

4.2 (a) Influence generated the source sets selected by NMF-InfMax trained using increasing number of cascades on Hierarchical networks with 1,024 nodes and 4,096 edges. (b) Influence generated by the source sets selected by IMINFECTOR and NMF-InfMax on the MemeTracker dataset at $T = 10$ hours. 78

CHAPTER 1

INTRODUCTION

1.1 Motivation

Continuous-time information diffusion on heterogenous networks is a prevalent phenomenon [7, 73, 79]. News spreading on social media [25, 30, 96], viral marketing [50, 51, 100], computer malware propagation, and epidemics of contagious diseases [6, 70, 79, 89] are all examples of diffusion on networks, among many others. For instance, a piece of information (such as a tweet) can be retweeted by users (nodes) with followee-follower relationships (edge) on the Twitter network. We call a user *infected* if she retweets and *healthy* otherwise. Her followers see her retweet and can also become infected if they retweet in turn, and so on. Such information diffusion mimics the epidemic spread where an infectious virus can spread to individuals (human, animal, or plant) and then to many others upon their close contact. This information diffusion impact our lives by transmitting information, behavior, news, advertisement, and many others. It enables companies to create a cascade of product adoptions via the word-of-month effect which has big potential commercial values in viral marketing by suggesting new strategies to advertising a product by a set of *influencial* users in some social network.

For a given social network $G = (\mathcal{V}, \mathcal{E})$, if the budget size k is given, the problem to identify the set $\mathcal{S} \subseteq \mathcal{V}$ with k most influencial users to maximize the *influence* $\delta(\mathcal{S})$, the expected number of follow-up adoptions by $\mathcal{S}$, is referred to as *influence maximization.* Practically, the advertising process is expected as soon as possible. That is, the company will expect to adopt the product in a short time. Thus, we suppose the time horizon T is finite and scheduled at the beginning. Intuitively, the node selection phase to find the solution of influence maximization should depend on one evaluation of the influence $\sigma(\cdot)$ which is the solution of the problem referred to as *influence estimation.* When the diffusion model is

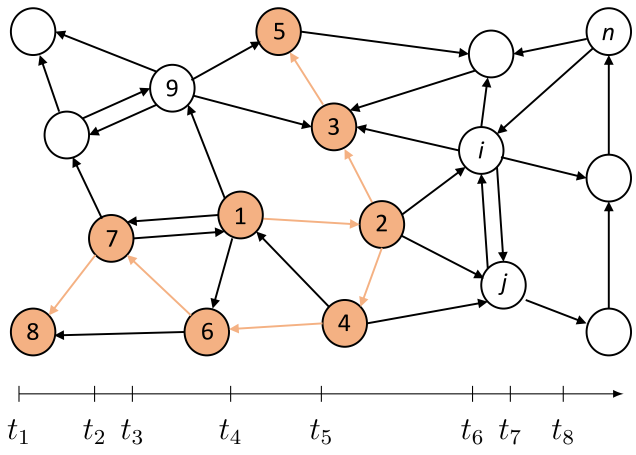


Figure (1.1). Example of a sample cascade on a diffusion network. The cascade was originated from the source set $\mathcal{S} = \{1\}$ and gradually propagates to other nodes through their directed edge connections. The time line below the network shows the wall-clock time t_i that each node i was infected during the cascade with $t_1 = 0$. The orange edges indicate whom each node got infection from, and $t_{ij} := t_j - t_i$ is the time that node i took to infect node j.

fully modeled, then the phase to calculate the expected influence of a source set have been solved by using Monte Carlo simulation which is simple but expansive on computation in many literature. Since the network structure is difficult to know in practice, algorithms to uncover the existence and strength of connections between nodes are also discussed, which is referred to *network structure inference*.

1.2 Diffusion model

The study of heterogeneous diffusion networks only emerged in the past decade and is considered very challenging, mainly because of the extremely large scale of modern networks, the heterogeneous inter-dependencies between the nodes, and the randomness exhibited in cascade data. In Figure 1.1, we illustrate one such cascade originated from a singleton source set $\mathcal{S} = \{1\}$, which spreads to other nodes during the propagation. The orange edges indicate whom a node got infection from, for example, node 4 succeeded in infecting node 6 before node 1 did. The time line below the network indicates the wall-clock time t_i

of each node i got infected in this cascade. In particular, $t_1 = 0$. Moreover, $t_{ij} := t_j - t_i$ is the time node i took to infect node j. Note that this is one sample cascade of $\mathcal{S} = \{1\}$, and a different sample cascade of the same source $\mathcal{S}$ may yield different infected nodes and infection times due to the randomness of t_{ij}.

In this thesis, we will only use *diffusion models* in continuous-time setting, which describes the distribution $p(t; \alpha_{ij})$ of the time t that an infected node i takes to infect her healthy neighbor $j \in \{j' : (i, j') \in \mathcal{E}\}$. Here α_{ij} is the infection rate of i on j which *vary across different edges.* That is, t_{ij} is a random variable following the density function $p(t; \alpha_{ij})$ for each $(i, j) \in \mathcal{E}$. We assume that the infection is *progressive*, i.e., a node will not be infected again nor recover once infected, since generalization to the case with recovery is straightforward. Then, given a source set $\mathcal{S}$ (a subset of $\mathcal{V}$) of nodes that are infected at time 0, they will infect their healthy neighbors with random infection times described above; and the infected neighbors will then infect their healthy neighbors, and so on. As such, the infection initiated by $\mathcal{S}$ at time 0 propagates to other nodes of the network. We call one course of such propagation a *cascade.* For simplicity, it is common to assume that the infection times across different edges are independent, known as the *continuous-time independent cascade* (CIC) model [25, 34, 39].

The standard diffusion model with exponential distribution $p(t; \alpha) = \alpha e^{-\alpha t}$ is mostly widely used in the literature. That is, $t_{ij} \sim p(t; \alpha_{ij})$ for each $(i, j) \in \mathcal{E}$. Note that the parameter $\alpha_{ij} > 0$ in the exponential distribution indicates the *strength* of impact node i has on j—the expectation of $t_{ij} \sim p(t; \alpha_i j)$ is $1/\alpha_{ij}$—and the larger α_{ij} is, the sooner node j will be infected by i on expectation. We focus on the diffusion model with exponential distribution in this work. Other distributions, such as Rayleigh and general Weibull distributions, are also experimented in our empirical studies in this work.

1.3 Cascade data

Observation data $\mathcal{D}$ of a diffusion network are often in the form of *sample cascades* $\mathcal{D} := \{\mathcal{C}_k = (\mathcal{S}_k, \boldsymbol{\tau}_k) \in \mathcal{V} \times \mathbb{R}^n_+ : k \in [K]\}$, where the kth cascade $\mathcal{C}_k$ records its source set $\mathcal{S}_k \subset \mathcal{V}$ and the time $(\boldsymbol{\tau}_k)_i \geq 0$ which indicates when node i was infected (if i was not infected during $\mathcal{C}_k$ then $(\boldsymbol{\tau}_k)_i = \infty$). See Figure 1.1 for one of such sample cascades, where we have $\boldsymbol{\tau} = \{t_1, \dots, t_8, \infty, \dots, \infty\}$ if no other nodes were infected in this cascade. Cascade data are collected from historical events for training purposes.

1.4 Influence estimation and network inference

Given the network $G = (\mathcal{V}, \mathcal{E})$, as well as the CIC diffusion model specified by a distribution $p(t; \alpha_{ij})$ for edge (i, j), and the transmission rate matrix $\boldsymbol{A}$, the *influence prediction* (or *influence estimation*) is to compute

$$\boldsymbol{x}(t; \boldsymbol{\chi}_{\mathcal{S}}) = [x_1(t; \boldsymbol{\chi}_{\mathcal{S}}), \dots, x_n(t; \boldsymbol{\chi}_{\mathcal{S}})]^\top \in [0, 1]^n \tag{1.1}$$

for all time $t \geq 0$ and any source set $\mathcal{S} \subset \mathcal{V}$. The probability $\boldsymbol{x}(t; \boldsymbol{\chi}_{\mathcal{S}})$ can also be used to compute the *influence function*

$$\sigma(t; \mathcal{S}) = \mathbf{1}_n^\top \boldsymbol{x}(t; \boldsymbol{\chi}_{\mathcal{S}}). \tag{1.2}$$

On the other hand, the objective of *network inference* problem is to learn the network connectivity $\mathcal{E}$ and $\boldsymbol{A}$ given cascade data $\mathcal{D}$. Influence prediction may also require network inference when only cascade data $\mathcal{D}$ are available, resulting in a *two-stage* approach: a network inference is performed first to learn the network structure $\mathcal{E}$ and the diffusion model parameters $\boldsymbol{A}$, and then an influence estimation is used to compute the influence for the source set $\mathcal{S}$. However, approximation errors and biases in the two stages will certainly accumulate. Alternatively, one can use a *one-stage* approach to directly estimate $\boldsymbol{x}(t; \boldsymbol{\chi}_{\mathcal{S}})$ of any $\mathcal{S}$ from the cascade data $\mathcal{D}$, which is more versatile and less prone to diffusion model misspecification. Our method is a such kind of one-stage method. Additionally, it allows knowledge of $\mathcal{E}$ and/or $\boldsymbol{A}$, if available, to be integrated for further performance improvement.

We now give a statement on the main goal to achieve by our models.

Problem 1 (Multi-task learning for influence estimation and network inference)**.** *Given a directed network $\mathcal{G}$ with node set $\mathcal{V}$, a time horizon T, and a cascade data $\mathcal{D}$, the objective is*

(i) to predict the infection probability function $\boldsymbol{x}(t; \boldsymbol{\chi}_{\mathcal{S}})$ for time t and source set $\boldsymbol{\chi}_{\mathcal{S}}$; and

(ii) to infer the network connection $\mathcal{E}$ and the transmission matrix $\boldsymbol{A}$.

1.5 Influence maximization

In a network $G = (\mathcal{V}, \mathcal{E})$, for any source set $\mathcal{S}$, influence $\sigma(t; \mathcal{S})$ captures the expected number of infected nodes. Let $n_0 \in [n]$ be the budget size to impose constraint on the size of the source set. The problem could be defined as follows.

Problem 2 (Influence maximization)**.** *Given network $G = (\mathcal{V}, \mathcal{E})$, budget size n_0, and time horizon T, the objective of influence maximization problem is to find the source set $\mathcal{S} \subseteq \mathcal{V}$ of size n_0 to maximize $\sigma(T; \mathcal{S})$. That is, the problem could be formulated as*

$$\max_{\mathcal{S}} \ \sigma(T; \mathcal{S}), \quad \text{s.t.} \quad \mathcal{S} \subset \mathcal{V}, \quad |\mathcal{S}| = n_0. \tag{1.3}$$

There are two main ingredients of an influence maximization method for solving (1.3): an influence prediction subroutine that evaluates the influence $\sigma(t; \mathcal{S})$ for any given source set $\mathcal{S}$, and an (approximate) combinatorial optimization solver to find the optimal set $\mathcal{S}$ of (1.3) that repeatedly calls the subroutine. The combinatorial optimization problem is NP-hard and is often approximately solved by greedy algorithms with guaranteed sub-optimality when $\sigma(t; \mathcal{S})$ is submodular in $\mathcal{S}$.

1.6 Overview

The rest of this thesis is organized as follows.

In Chapter 2, we first review the commonly discussed diffusion models which apply different mechanisms to capture how the node status change through the interaction between nodes, and then review the literature on influence estimation, network inference and influence maximization in order.

In Chapter 3, we develop the proposed framework of neural mean-field dynamics in both discrete and continuous setting for estimating the infection probability of nodes and uncovering the existence and strength of edges on diffusion networks, as well as an optimal control formulation for parameter learning. The performance of the proposed methods are demonstrated in empirical research on a variety of synthetic networks and real social networks.

By employing the proposed model in continuous setting as subroutine, we also propose an efficient algorithm for influence maximization and show its performance by numerical experiments on both synthetic and real data in Chapter 4.

In Chapter 5, we summarize our contributions, and discuss the possible extensions to this work.

CHAPTER 2

BACKGROUND

In this section, we first conduct a comprehensive review of the literature on topics involved in our proposed method, namely, influence estimation, network inference, and influence maximization, starting from the review of diffusion models commonly used for these problems.

2.1 Diffusion models

Consider a diffusion network model, which consists of a directed network $G = (\mathcal{V}, \mathcal{E})$ with node set $\mathcal{V} = [n]$ and directed edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and a diffusion model $\mathscr{M}$ to capture the propagation behaviors across through edges between nodes. The types of diffusion models could be divided into two big classes: discrete-time and continuous-time. Most of existing works on influence estimation and maximization have been considered for discrete-time models which assume node infections only occur at discrete time points. Independent Cascade (IC) [33] and Linear Threshold (LT) [41, 91] diffusion model are two most classic and general discussed discrete-time diffusion models. In contrast to discrete-time models, continuous-time diffusion models allow arbitrary event occurrence times and hence are more accurate in modeling real-world diffusion processes. In our work, we only propose algorithms under the continuous-time independent (CIC) model. However, in this section, we also introduce the Triggering model [50] which generalizes the IC and LT models for referring to other related works.

2.1.1 Triggering diffusion models

The process starts from a initially activated (of infected) node set $\mathcal{S}$. For each node $v \in \mathcal{V}$, it is associated with some triggering distribution $\mathcal{T}(v)$ over subsets of its incoming neighbor set $\mathcal{N}_I(v)$. According to the distribution, we take a sample $T(v)$ from $\mathcal{T}(v)$. At timestamp $t+1$, a healthy node v becomes infected if there is at least one node $u \in T(v)$ infected at timestamp t. The process is terminated when no more nodes can be infected. One equivalent formulation of Triggering model in term of edges is as follows: after we fix S and $T(v)$ for each node v, we can generate the induced subgraph g of G by vertex set $S \cup \{T(v) : v \in \mathcal{V}\}$. Then a node u is infected at timestamp t if there is a directed path from one node $w \in S$ to u contained in g such that the distance from S to u is t. Kempe et al. proved that

Theorem 2.1.1. *[50] In every instance of the Triggering model, the influence function $\sigma(\cdot)$ is submodular i.e., $\sigma(S_1 \cup \{v\}) - \sigma(S_1) \geq \sigma(S_2 \cup \{v\}) - \sigma(S_2)$ for any $S_1 \subseteq S_2 \subseteq \mathcal{V}$ and $v \in \mathcal{V} - S_2$.*

Independent Cascade (IC) model considers an edge weighted network $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ with $\mathcal{W} : \mathcal{E} \to [0,1]$ which identifies the probability of an infected node u to infect a healthy node v through an edge $u \to v$. It is important to note that each infected node u has only one try to infect its healthy outgoing neighbors, and after that, it can not infect any node. IC model is a special case of Triggering model by taking a triggering distribution $\mathcal{T}(v)$ for each node v such that any node u in v's incoming neighbor set independently selected for $T(v)$ with probability $\mathcal{W}(u \to v)$.

Linear Threshold (LT) model considers a node and edge weighted network $G = (\mathcal{V}, \mathcal{E}, \mathcal{W}_{node}, \mathcal{W}_{edge})$ with node weight $\mathcal{W}_{node} : \mathcal{V} \to [0,1]$ and edge weight $\mathcal{W}_{edge} : \mathcal{E} \to [0,1]$ such that $\sum_{u \in \mathcal{N}_I(v)} \mathcal{W}_{edge}(u \to v) \leq 1$. At timestamp $t+1$, a healthy node v is infected if

$$\sum_{u \in \mathcal{N}_I(v) \text{ and u in infected at } t} \mathcal{W}_{edge}(u \to v) \geq \mathcal{W}_{node}(v).$$

LT model is also a special case of Triggering model by assuming a node v switching from healthy to infected if the total weight of infected nodes in $T(v)$ is at least $\mathcal{W}_{node}(v)$.

2.1.2 Continuous-time Independent Cascade (CIC) model

The continuous-time diffusion model for cascade data is introduced in [35] to uncover the structure of social networks by considering the likelihood of pairwise propagation between nodes which is a continuous distribution of time. It is formulated by Du et al. in [25].

CIC model considers an edge weighted network $G = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ with $\mathcal{W} : \mathcal{E} \to \mathbb{R}^+$ which indicates the infection strength α_{ij} of an infected node i to impact on the node j through the edge from i to j. The model draw a distribution $p(t; \alpha_{ij})$ on the transmission time $t_{ij} = t_j - t_i$ for node j to infect i. That is, if a node j get infected at time t_j by another node i which was infected at time t_i, then $t_i < t_j, i \to j \in \mathcal{E}$ and $t_{ij} := t_i - t_j$ is a random variable following the density function $p(t; \alpha_{ij})$. we assume transmission times are independent. Then, we can sample an instance g of the CIC model, by generating the time t_{ij} weight following the diffusion distribution $p(t; \alpha_{ij})$ for each edge. Then for any infected node i and healthy node j (not necessary adjacent), the transmission time t_{ij} is equal to the distance between them in g. And we call the neighbor of i contained in the shortest path given the distance is the *father* of j. Clearly, a node might be infected by multiple neighbors. We only call the neighbor first infects the node as its *true father*. In this thesis, we only discuss our algorithms under CIC model.

2.2 Influence estimation

Most of existing methods on influence estimation are sampling based and used as subroutine of influence maximization. In [50], the IC and LT models are considered and the propagation spread of a source set $\mathcal{S}$ is simply estimated by the expected reachable set size of $\mathcal{S}$ taken over the randomness of the influence propagation process. That is

Theorem 2.2.1. *[50] Let $\mathcal{G}$ be the distribution of g induced by the randomness in sampling from each node's triggering distribution. For given network $G = (\mathcal{V}, \mathcal{E})$ and source set $\mathcal{S}$,*

$$\sigma(\mathcal{S}) = \mathbb{E}[|R(\mathcal{S})|]$$

where $R(\mathcal{S})$ is the set of nodes in $g \in \mathcal{G}$ that are reachable from nodes in $\mathcal{S}$.

Hence we could estimate the expected influence by taking the average measurement of $|R(\mathcal{S})|$ over multiple instances of g. To improve the efficiency of Monte Carlo simulations used here, a method with provable performance guarantee is developed which iterates over a sequence of guesses on the true influence until the verifier accepts in [67]. Insider, the influence is estimated by a standard Riemann sum as follows.

Proposition 2.2.2. *Let $I(\mathcal{S})$ be the random variable denoting the set of nodes influenced. Then we have*

$$\sigma(\mathcal{S}) = \mathbb{E}\left[|I(\mathcal{S})|\right] = \sum_{k=1}^{n} \Pr\left(|I(\mathcal{S})| > k\right).$$

Thus the influence can be approximated using a standard Riemann sum:

$$\sum_{k \in \{1,(1+\varepsilon),(1+\varepsilon)^2,\cdots,n\}} \frac{k\varepsilon}{1+\varepsilon} P_k,$$

where P_k is the fraction of L sampled graphs $g \in \mathcal{G}$ in which the set $\mathcal{S}$ reaches at least k nodes.

The major components of this method are:

- Iterate the guess on $\tau \in \left\{n, n/(1+\epsilon), n/(1+\epsilon)^2, \ldots, |S|\right\}$ which is from large to small.
- Sample L instances of graph $g \in \mathcal{G}$ and approximate the influence by Proposition 2.2.2.
- If the estimated influence is close to the guess, then accept it as estimated influence and stop the iteration.

In [8], the *reverse reachable (RR)* sets of nodes are adopted which is defined as follows

Definition 2.2.3 (Reverse Reachable Set). *[93] Let v be a node in $\mathcal{V}$. A reverse reachable (RR) set for v is the set of nodes in $g \in \mathcal{G}$ that can reach v. A random RR set is an RR set for a node selected uniformly at random from $\mathcal{V}$.*

Let $\mathcal{R}$ be a set of sampled random RR sets. Tang el at. [94] proved if the size of $\mathcal{R}$ is big enough, then we have

Theorem 2.2.4. *[94] Let $F_{\mathcal{R}}(\mathcal{S})$ be fraction of RR sets in $\mathcal{R}$ covered by $\mathcal{S}$. Then*

$$\mathbb{E}[n \cdot F_{\mathcal{R}}(\mathcal{S})] = \sigma(\mathcal{S}).$$

The sample size $|\mathcal{R}|$ can be controlled by a given threshold [8], a pre-calculated parameter [94], or some stop conditions [93] to achieve a balance between efficiency and accuracy.

Instead of using the full network structure as the methods, *sketch-based* approaches [12, 14, 19, 75, 76, 98] only characterize propagation instances for influence computation, such as the method in [19], which considers per-node summary structures defined by the bottom-k min-bash [18] sketch of the combined reachability set.

In CIC models, influence estimation can be reformulated as the problem of finding the least label list which contains information about the distance to the smallest reachable labels from the source [25, 39]. Compared to methods using a fixed number of samples, a more scalable approximation scheme with a built-in block is developed to minimize the number of samples needed for the desired accuracy [74]. The influence function can also be approximated by solving a jump stochastic differential equation [103] or a deterministic differential equation that governs the evolution of the influence counter [16]. Inspired by [93], algorithms proposed in [8, 93, 94] can be extended from the Triggering model to CIC models by generalizing the definition of RR sets as follows.

Definition 2.2.5 (Reverse Influence Set)**.** *[93] For a node $v \in \mathcal{V}$, a* reverse influence (RI) *set $\mathcal{R} \subseteq \mathcal{V}$ for v is a node set such that for any source set $\mathcal{S} \subseteq \mathcal{V}$, the probability that $\mathcal{R} \cap \mathcal{S} \neq \emptyset$ equals the probability that $\mathcal{S}$ can activate v in a diffusion process. A random RI set is an RI set for a node selected uniformly at random from $\mathcal{V}$.*

Combining with the definition of CIC model, we could generate one RI set $\mathcal{R}$ for v by the following steps: 1) generate a weighted graph $g = (\mathcal{V}, \mathcal{E}, \mathcal{W})$ where $\mathcal{W}(e) = t_{ij} \sim p(t; \alpha_{ij})$ with $e = i \to j \in \mathcal{E}$; 2) Invoke the Dijkstra's algorithm on g to identify the shortest distance from each node $u \in \mathcal{V}$ to v; 3) insert all nodes into $\mathcal{R}$ with distance to v no more than the maximum observation time.

The aforementioned methods require knowledge of cascade traces [20] or the diffusion networks, such as node connectivity and node-to-node infection rates, as well as various assumptions on the diffusion of interests. However, such knowledge about the diffusion networks may not be available in practice, and the assumptions on the propagation or data formation are often application-specific and do not hold in most other problems. InfluLearner [24] is a state-of-the-art method that does not require knowledge of the underlying diffusion network. InfluLearner estimates the influence directly from cascades data in the CIC models by learning the influence function with a parameterization of the coverage functions using random basis functions as follows.

Theorem 2.2.6. *[24] Let $R \in \{0,1\}^{n\times n}$ be the reachability matrix with $n = |\mathcal{V}|$ with*

$$(R)_{sj} = \begin{cases} 1, & j \textit{ is reachable from source } s, \\ 0, & \textit{otherwise.} \end{cases}$$

Then we could estimate the infection probability by the following expression.

$$x_i(t;\mathcal{S}) = \mathbb{E}_{r\sim p_i(r)}[\phi(\boldsymbol{\chi}_{\mathcal{S}}^T r)]$$

where $r := R_{:i}$, $p_i(r)$ is the marginal distribution of column j of R induced by p_R, and $\phi(\boldsymbol{\chi}_{\mathcal{S}}^T R_{:i})$ indicates the node i could be infected by $\mathcal{S}$ or not.

The distribution $p_i(r)$ is unknown and could be estimated by using the frequency of node i being infected by source node $r \in \mathcal{S}$. However, this estimation method requires knowledge of the original source node for every infection, which can be difficult or impossible to be tracked in real-world applications, such as epidemic spreads.

In recent years, deep learning techniques have been employed to improve the scalability of influence estimation on large networks. In particular, convolutional neural networks

(CNNs) [2] and attention mechanism are incorporated with both network structures and user specific features to learn users' latent feature representation in [84]. By piping represented cascade graphs through a gated recurrent unit (GRU), the future incremental influence of a cascade can be predicted [62]. RNNs and CNNs are also applied to capture the temporal relationships on the user-generated contents networks (e.g., views, likes, comments, reposts) and extract more powerful features in [107]. In methods based on graph structures, graph neural networks (GNNs) [90] and graph convolution networks (GCNs) [106] are widely applied. In particular, two coupled GNNs are used to capture the interplay between node activation states and the influence spread [9], while GCNs integrated with teleport probability from the domain of page rank in [61] enhanced the performance of method in [84]. However, these methods depend critically on the structure or content features of cascades which is not available in many real-world applications.

2.3 Network inference

Inference of diffusion network structure is an important problem closely related to influence estimation. In particular, if the network structure and infections rates are unknown, one often needs to first infer such information from a training dataset of sampled cascades, each of which tracks a series of infection times and locations on the network. Existing methods have been proposed to infer network connectivity [26, 36, 38, 65] and also the infection rates between nodes [35, 37, 71]. Submodular optimization is applied to infer network connectivity [36,38,65] by considering the most probable [36] or all [38,65] directed trees supported by each cascade. One of the early works that incorporate spatio-temporal factors into network inference is introduced in [65]. Utilizing convex optimization, transmission functions [26], the prior probability [71], and the transmission rate [35] over edges are inferred from cascades. In addition to static networks, the infection rates are considered but also in the unobserved dynamic network changing over time [37]. Besides cascades, other features of dynamical processes on networks have been used to infer the diffusion network structures. To avoid using predefined transmission models, the statistical difference of the infection time intervals between nodes in the same cascade versus those not in any cascade was considered in [85]. A given time series of the epidemic prevalence, i.e., the average fraction of infected nodes was applied to discover the underlying network. The recurrent cascading behavior is also explained by integrating a feature vector describing the additional features [97]. A graph signal processing (GSP) approach is developed to infer graph structure from dynamics on networks [23, 69].

2.4 Influence maximization

Influence maximization is an important but very challenging problem in real-world applications of diffusion networks, such as commercial advertising and epidemic controls. Influence maximization is shown to be an NP-hard problem under most of diffusion models [64] (e.g., LT, IC, CIC). It was first formulated in [50] as a combinatorial optimization problem. It also proves that the influence function $\sigma(\cdot)$ is a non-negative monotone submodular function under certain assumptions, thus a greedy hill-climbing algorithm [21, 32, 72] can be applied to obtain a provable sub-optimal solution [39, 50]. Specifically, we have

Theorem 2.4.1. *[21, 32, 72] For a non-negative, monotone submodular function f, let $\mathcal{S}$ be a set of size k obtained by selecting elements one at a time, each time choosing an element that provides the largest marginal increase in the function value. Let $\mathcal{S}^*$ be a set that maximizes the value of f over all k-element sets. Then*

$$f(\mathcal{S}) \geq (1 - 1/e) \cdot f(\mathcal{S}^*),$$

in other words, $\mathcal{S}$ provides a $(1 - 1/e)$ approximation.

It is important to note that Theorem 2.4.1 holds under the assumption that the underlying function f can be evaluated exactly, but this is not the story on the influence function $\sigma(\mathcal{S})$. This requires an arbitrarily close approximation to $\sigma(\mathcal{S})$ from the influence estimation subroutine. The main components of the greedy framework are: starts from an empty set $\mathcal{S}$, gradually add one node i that maximizes the marginal gain $\sigma(\mathcal{S} \cup \{i\}) - \sigma(\mathcal{S})$ to $\mathcal{S}$. With the influence $\sigma(\cdot)$ function in Theorem 2.2.1, Kempe et al.'s *Greedy* framework is general and effective. But it has computation overhead due to its $O(kmnr)$ time complexity over k iterations and r graphs for $\sigma(\cdot)$ with $m = |\mathcal{E}|$. There are two sources of inefficiency:

(S1) Spread estimation: the calculation of influence $\sigma(\cdot)$ dominated by $O(mr)$.

(S2) Node selection: the basic greedy algorithm dominated by $O(kn)$.

Many efforts are made to increase Greedy's efficiency to tackle (S1) [11, 12, 13, 52] or (S2) [40, 58].

Leskovec et al. first proposed the lazy greedy procedure called *CELF* in [58] by exploiting submodularity for an efficient algorithm. Let $\Delta_v(\mathcal{S}) := \sigma(\mathcal{S} \cup \{v\}) - \sigma(\mathcal{S})$ be the marginal increments by adding node $v \in \mathcal{V} - \mathcal{S}$. Clearly, each iteration of the standard greedy aims to select a node with the maximum $\Delta_v(\mathcal{S})$. The main idea of CELF is that the value of $\Delta_v(\mathcal{S})$ in the current iteration can not be bigger than in the previous iterations. That is, we have

Proposition 2.4.2. *Let $\mathcal{S}_i$ be the selected node set in the ith iteration. For any iteration i and node $v \in \mathcal{V} - \mathcal{S}_i$, we have*

$$\Delta_v(\mathcal{S}_{i+1}) \leq \Delta_v(\mathcal{S}_i).$$

Hence, in each iteration i, it is not necessary to calculate the marginal gain of all the nodes which has been done as follows.

- Let $R = \{u_1, u_2, \ldots, u_k\}$ be the set of top k nodes in the rank of marginal gain $\Delta_u(\mathcal{S}_{i-1})$ in the (i-1)-th iteration where k is a trainable hyper-parameter to balance the accuracy and efficiency. Then we will only consider nodes in R.

- We iteratively calculate $\Delta_u(\mathcal{S}_i)$ in order of R and it is broken in j-th iteration if $\Delta_{u_j}(\mathcal{S}_i) \geq \Delta_{u_{j+1}}(\mathcal{S}_{i-1})$.

- The node with the maximum marginal gain is selected.

CELF++ [40] further optimize CELF by exploiting submodularity with the idea: if the node u is picked as a seed in the current iteration, then the marginal gain of with respect to u does not need to be recomputed in the next iteration. Chen et al. optimizes CELF by a linear scan [27] of the triggering graph instances to calculate reachability of set $\mathcal{S}$ for spread estimation in [12].

CELF and CELF++ tackled (S2) by controlling $O(n)$ part in the complexity. Another way to tackle (S2) is to work on $O(r)$. In [50], Kempe et al. suggested setting $r = 10,000$ and proved

Theorem 2.4.3. *When each estimation of expected spread has ε related error, Greedy returns a $(1 - 1/e - \varepsilon')$-approximate solution for a certain ε'.*

Tang et al. provided a formal relationship between ε and r [94] which paved a way to control r.

Theorem 2.4.4. *Greedy returns a $(1 - 1/e - \varepsilon)$-approximate solution with at least $1 - n^{-\ell}$ probability, if*

$$r \geq (8k^2 + 2k\varepsilon)n\frac{(\ell + 1)\log n + \log k}{\varepsilon^2 \cdot OPT}.$$

where k is the budget size and OPT is the maximum number of infected nodes for any size-k node set $\mathcal{S}$.

Suppose OPT is known, then we can set r to be the smallest value satisfying Theorem 2.4.4, then the complexity to Greedy is

$$O(k^3 \ell m n^2 \varepsilon^{-2} \log n / OPT).$$

Some approximation ideas of OPT are shown in [93, 94].

To avoid the limitation of Greedy, Borgs et al. proposed a completed different method under IC model in [8] which is referred to Reverse Influence Sampling (RIS). It suns in two steps

1. Generate a certain number of random RR sets on G.

2. Consider the maximum coverage (MK) problem [95] of selecting k nodes to cover the maximum number of RR sets generated. A standard greedy algorithm [95] for MK problem can be employed for a $(1 - 1/e)$-approximate solution.

The number of random RR sets generated in the first step is controlled by a threshold-based approach in [8] which terminal the generation process until the total number of nodes and edges reaches a pre-defined threshold. This setting generates a sequence of correlated RR sets.

Tang et al. [94] optimized RIS by adding one step to estimate the parameter θ to control the number of generated RR sets.

0. Parameter Estimation. This phase computes a lower-bound of the maximum expected spread (OPT) among all size-k node sets, and then uses the lower-bound to derive a parameter θ.

More specifically, the choice of θ depends on the following theorem.

Theorem 2.4.5. *[94] Given a θ that satisfies*

$$\theta \geq (8 + 2\varepsilon)n \cdot \frac{\ell \log n + \log \binom{n}{k} + \log 2}{OPT \cdot \varepsilon^2}.$$

The algorithm returns a $(1 - 1/e - \varepsilon)$-approximate solution with at least $1 - n^{-\ell}$ probability.

Based on different strategies to approximate OPT, two algorithms are proposed which are referred to TIM and TIM+ in [94]. Tang et al. further improve TIM and TIM+ by the proposed method referred to as IMM [93] which derives a lower bound of OPT that is asymptotically tight. Comparing to TIM, the main different is that IMM optimizes the parameter estimation phase in TIM by a set of estimation techniques based on *martingales* [99].

The research of influence maximization also gets benefits from the development on deep learning tools. The deep reinforcement learning [48] is employed for the influence maximization to avoid an evaluation of influence [68]. GCOMB [68] trains on a Graph Convolutional

Network (GCN) [106] using a novel probabilistic greedy mechanism to predict the quality of a node, and utilizes a Q-learning [92] framework for efficiency through *importance sampling*. The importance of nodes are evaluated by considering one-hop and two-hop spread benefit measured on nodes in [46].

For instances without the information of network structure, the influence relationships between nodes are representation learned from cascade date initiated by a single node to derive a greedy solution in [77, 78]. CELFIE [78] utilizes Inf2Vec [31] to learn influence and susceptibility embeddings based on the co-occurence in diffusion cascades. IMINFECTOR [77] uses representations learned by an end-to-end linear model with activation function from diffusion cascades to perform model-independent influence maximization.

More algorithms can be seen in [4, 64].

CHAPTER 3

NEURAL MEAN-FIELD (NMF) DYNAMICS APPLIED ON INFLUENCE ESTIMATION AND NETWORK INFERENCE

In this research, a neural mean-field dynamics approach is proposed, which employs the Mori-Zwanzig (MZ) formalism [15] to derive the node infection probabilities in both discrete-time and continuous-time setting referred to as discrete NMF and continuous NMF respectively. In the same models, the strength of interactions between nodes through edges are inferred by placing them as trainable parameters in the NMF dynamics. This section is organized as follows. First, we develop the proposed framework of neural mean-field dynamics in section 3.1. Then, we propose the discrete and continuous NMF models and demonstrate their performances on a variety of synthetic and real-world networks in section 3.2 and section 3.3, respectively.

3.1 Mean-field dynamics of diffusion

3.1.1 Notations

Throughout this paper, we use boldfaced lower (upper) letter to denote vector (matrix) or vector-valued (matrix-valued) function, and $(\cdot)_k$ (or $(\cdot)_{ij}$) for its kth component (or (i,j)-th entry). All vectors are column vectors unless otherwise noted. We follow the Matlab syntax and use $[\boldsymbol{x};\boldsymbol{y}]$ to denote the vector that stacks $\boldsymbol{x}$ and $\boldsymbol{y}$ vertically. We denote inner product by $\boldsymbol{x} \cdot \boldsymbol{y}$ and component-wise multiplication by $\boldsymbol{x} \odot \boldsymbol{y}$. For ease of reference, more notations are listed in Table 3.1.

3.1.2 Modelling diffusion by stochastic jump processes

We begin with the jump process formulation of network diffusion. Given a source set $\boldsymbol{\chi}_{\mathcal{S}}$, let $X_i(t;\boldsymbol{\chi}_{\mathcal{S}})$ denote the infection status of the node i at time t which has value 1 if node

Table (3.1). Frequently used notations

Notations	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	a network $\mathcal{G}$ with a node set $\mathcal{V}$ and an edge set $\mathcal{E}$
n	the number of nodes in $\mathcal{V}$
(i, j)	directed edge from node i to j
$p(t; \alpha_{ij})$	diffusion distribution with parameters α_{ij} for edge (i, j)
$\boldsymbol{A}$	the transmission matrix with $(A)_{ij} = \alpha_{ij}$ which measures the strength of impact node i on j
t, T	time $t \in [0, T], T \in \mathbb{R}_+$ in continuous case or $t = 1, 2, \cdots, T, T \in \mathbb{N}$ in discrete case.
f'	derivative $'$ with respect to t for a function f
$\nabla_{\boldsymbol{x}} f$	gradient with respect to $\boldsymbol{x}$ for a function f
$\Pr(\cdot)$	probability of an event
$\mathbb{E}_X[\,\cdot\,]$	expectation with respect to X
$\boldsymbol{\chi}_{\mathcal{S}}$	a 0-1 vector such that $(\boldsymbol{\chi}_{\mathcal{S}})_i = 1$ if and only if node i is contained in the source set $\mathcal{S}$
$x_i(t; \boldsymbol{\chi}_{\mathcal{S}})$	the probability of node i being infected at time t given a source set $\mathcal{S}$
$\boldsymbol{x}(t; \boldsymbol{\chi}_{\mathcal{S}})$	the infection probability function at time t given source set $\mathcal{S}$
$\sigma(t; \mathcal{S})$	the expected number of infected nodes at time t given a source set $\mathcal{S}$
$\mathbf{1}_n, \mathbf{1}$	the vector or matrix with entries all ones
$\mathbf{0}_n, \mathbf{0}$	the vector or matrix with entries all zeros

i is infected by time t, and 0 otherwise. Then, $\{X_i(t;\boldsymbol{\chi}_{\mathcal{S}}) : i \in [n]\}$ is a set of n coupled jump processes with the following properties.

Proposition 3.1.1. *(i) For any time addition $\tau > 0$, we have*

$$\Pr(X_i(t+\tau) = 1, X_i(t) = 0 | \mathcal{H}(t)) = \mathbb{E}[X_i(t+\tau;\boldsymbol{\chi}_{\mathcal{S}}) - X_i(t;\boldsymbol{\chi}_{\mathcal{S}}) | \mathcal{H}(t)].$$

(ii) The infection probability of node i is the expectation of $X_i(t;\boldsymbol{\chi}_{\mathcal{S}})$ conditioning on $\mathcal{H}(t)$:

$$x_i(t;\boldsymbol{\chi}_{\mathcal{S}}) = \mathbb{E}_{\mathcal{H}(t)}[X_i(t;\boldsymbol{\chi}_{\mathcal{S}}) | \mathcal{H}(t)]. \tag{3.1}$$

(iii) Let $\lambda_i^*(t)$ be the conditional intensity of $X_i(t;\boldsymbol{\chi}_{\mathcal{S}})$ given history $\mathcal{H}(t) = \{X_i(s;\boldsymbol{\chi}_{\mathcal{S}}) : s \le t,\, i \in [n]\}$, i.e.,

$$\lambda_i^*(t) := \lim_{\tau \to 0^+} \frac{\mathbb{E}[X_i(t+\tau;\boldsymbol{\chi}_{\mathcal{S}}) - X_i(t;\boldsymbol{\chi}_{\mathcal{S}}) | \mathcal{H}(t)]}{\tau}. \tag{3.2}$$

Suppose the standard diffusion model is adopted, i.e., $p(t;\cdot)$ is the exponential distribution. Then we have

$$\lambda_i^*(t) = \sum_j \alpha_{ji} X_j(t)(1 - X_i(t)). \tag{3.3}$$

Since the standard diffusion model is mostly widely used and it indicates that the conditional intensity $\lambda_i^*(t)$ of a healthy node i is determined by the total infection rate of its infected neighbors j as shown in (3.3). Henceforth, we will derive the mean-field dynamics under the assumption that the diffusion model is standard. To this end, we adopt the following notations (for notation simplicity we temporarily drop $\boldsymbol{\chi}_{\mathcal{S}}$ in this subsection as the source set $\mathcal{S}$ is arbitrary but fixed):

$$x_I(t) = \mathbb{E}_{\mathcal{H}(t)}\left[\textstyle\prod_{i \in I} X_i(t;\boldsymbol{\chi}_{\mathcal{S}}) \middle| \mathcal{H}(t)\right], \quad y_I(t) = \textstyle\prod_{i \in I} x_i(t), \quad e_I(t) = x_I(t) - y_I(t) \tag{3.4}$$

for all $I \subset [n]$ and $|I| \ge 2$. Then we can derive the evolution of $\boldsymbol{z} := [\boldsymbol{x}; \boldsymbol{e}]$ as follows where

$\boldsymbol{x}(t) \in [0,1]^n$ is the *resolved* variable whose value is of interests and samples can be observed in cascade data $\mathcal{D}$, and $\boldsymbol{e}(t) = [\cdots; e_I(t); \dots]^T \in \mathbb{R}^{2^n-n-1}$ is the *unresolved* variable.

Theorem 3.1.2. *[47] The evolution of $\boldsymbol{z}(t) = [\boldsymbol{x}(t); \boldsymbol{e}(t)]$ follows the nonlinear differential equation:*

$$\boldsymbol{z}' = \bar{\boldsymbol{f}}(\boldsymbol{z}), \quad \textit{where} \quad \bar{\boldsymbol{f}}(\boldsymbol{z}) = \bar{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{e}) = \Big[\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{A}) - (\boldsymbol{A} \odot \boldsymbol{E})\mathbf{1};\ \cdots, f_I(\boldsymbol{x}, \boldsymbol{e}); \cdots\Big], \tag{3.5}$$

with initial value $\boldsymbol{z}_0 = [\boldsymbol{\chi}_{\mathcal{S}}; \mathbf{0}]$, $\boldsymbol{E} = [e_{ij}] \in \mathbb{R}^{n\times n}$, and

$$\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{A}) = \boldsymbol{A}\boldsymbol{x} - \operatorname{diag}(\boldsymbol{x})\boldsymbol{A}\boldsymbol{x}, \tag{3.6}$$

$$f_I(\boldsymbol{x}, \boldsymbol{e}) = \sum_{i\in I}\sum_{j\notin I} \alpha_{ji}(y_I - y_{I\cup\{j\}} + e_I - e_{I\cup\{j\}}) - \sum_{i\in I} y_{I\setminus\{i\}} \sum_{j\neq i} \alpha_{ji}(x_j - y_{ij} - e_{ij}). \tag{3.7}$$

Proof. Taking expectation $\mathbb{E}_{\mathcal{H}(t)}[\cdot]$ on both sides of (3.3), we obtain

$$\begin{aligned}\lambda_i(t) :=& \mathbb{E}_{\mathcal{H}(t)}[\lambda_i^*(t)] = \mathbb{E}_{\mathcal{H}(t)}\Big[\alpha_{ji} X_j(t)(1 - X_i(t)) \big| \mathcal{H}(t)\Big] \\ =& \sum_j \alpha_{ji}(x_j - x_{ij}) = \sum_j \alpha_{ji}(x_j - y_{ij} - e_{ij}).\end{aligned} \tag{3.8}$$

On the other hand, there is

$$\lambda_i(t)\,\mathrm{d}t = \mathbb{E}_{\mathcal{H}(t)}[\lambda_i^*(t)]\,\mathrm{d}t = \mathbb{E}_{\mathcal{H}(t)}[\mathrm{d}X_i(t)|\mathcal{H}(t)] = \mathrm{d}\mathbb{E}_{\mathcal{H}(t)}[X_i(t)|\mathcal{H}(t)] = \mathrm{d}x_i. \tag{3.9}$$

Combining (3.8) and (3.9) yields

$$x_i' = \frac{\mathrm{d}x_i(t)}{\mathrm{d}t} = \sum_j \alpha_{ji}(x_j - y_{ij} - e_{ij}) = (\boldsymbol{A}\boldsymbol{x})_i - (\operatorname{diag}(\boldsymbol{x})\boldsymbol{A}\boldsymbol{x})_i - \sum_j \alpha_{ji} e_{ij}$$

for every $i \in [n]$, which verifies the $\boldsymbol{x}$ part of (3.5). Similarly, we can obtain

$$x_I' = \sum_{i\in I}\sum_{j\notin I} \alpha_{ji}(x_I - x_{I\cup\{j\}}) = \sum_{i\in I}\sum_{j\notin I} \alpha_{ji}(y_I + e_I - y_{I\cup\{j\}} - e_{I\cup\{j\}}). \tag{3.10}$$

Moreover, by taking derivative on both sides of $x_I(t) = y_I(t) + e_I(t)$, we obtain

$$x_I' = \sum_{i \in I} y_{I \setminus \{i\}} x_i' + e_I' = \sum_{i \in I} y_{I \setminus \{i\}} \sum_{j \neq i} \alpha_{ji}(x_j - x_i x_j - e_{ij}) + e_I'. \tag{3.11}$$

Combining (3.10) and (3.11) yields the $\boldsymbol{e}$ part of (3.5).

It is clear that $\boldsymbol{x}_0 = \boldsymbol{\chi}_{\mathcal{S}}$. For every I, at time $t = 0$, there is $x_I(0) = \prod_{i \in I} X_i(0) = 1$ if $I \subset \mathcal{S}$ and 0 otherwise; and the same for $y_I(0)$. Hence $e_I(0) = x_I(0) - y_I(0) = 0$ for all I. Hence $\boldsymbol{z}_0 = [\boldsymbol{x}_0; \boldsymbol{e}_0] = [\boldsymbol{\chi}_{\mathcal{S}}; \boldsymbol{0}]$, which verifies the initial condition of (3.5). □

It is important to note that the evolution (3.5) is derived under the standard diffusion model assumption. However, it is approximately well for other distributions $p(t; \cdot)$, as shown in our empirical study. In any case, the dimension of $\boldsymbol{z}$ is $2^n - 1$, which grows exponentially fast in n and hence renders the computation infeasible in practice. To overcome this issue, we employ the Mori-Zwanzig formalism [15] to derive a reduced-order model of $\boldsymbol{x}$ that has dimensionality n only.

3.1.3 Mori-Zwanzig memory closure

We employ the Mori-Zwanzig (MZ) formalism [15] that allows to introduce a generalized Langevin equation (GLE) of the $\boldsymbol{x}$ part of the dynamics. The GLE of $\boldsymbol{x}$ is derived from the original equation (3.5) describing the evolution of $\boldsymbol{z} = [\boldsymbol{x}; \boldsymbol{e}]$, while maintaining the effect of the unresolved part $\boldsymbol{e}$. This is particularly useful in our case, as we only need $\boldsymbol{x}$ for infection probability and influence estimation.

Define the Liouville operator $\mathcal{L}$ such that $\mathcal{L}[g](\boldsymbol{z}) := \bar{\boldsymbol{f}}(\boldsymbol{z}) \cdot \nabla_{\boldsymbol{z}} g(\boldsymbol{z})$ for any real-valued function g of $\boldsymbol{z}$. Let $e^{t\mathcal{L}}$ be the Koopman operator associated with $\mathcal{L}$ such that $e^{t\mathcal{L}} g(\boldsymbol{z}(0)) = g(\boldsymbol{z}(s))$ where $\boldsymbol{z}(t)$ solves (3.5). Then $\mathcal{L}$ is known to satisfy the semi-group property for all g, i.e., $e^{t\mathcal{L}} g(\boldsymbol{z}) = g(e^{t\mathcal{L}} \boldsymbol{z})$. Now consider the projection operator $\mathcal{P}$ as the truncation such that $(\mathcal{P}g)(\boldsymbol{z}) = (\mathcal{P}g)(\boldsymbol{x}, \boldsymbol{e}) = g(\boldsymbol{x}, 0)$ for any $\boldsymbol{z} = (\boldsymbol{x}, \boldsymbol{e})$, and its orthogonal complement as $\mathcal{Q} = I - \mathcal{P}$ where I is the identity operator. The following theorem describes the *exact* evolution of $\boldsymbol{x}(t)$.

Theorem 3.1.3. *The evolution of $\boldsymbol{x}$ specified in* (3.5) *can also be described by the following GLE:*

$$\boldsymbol{x}' = \boldsymbol{f}(\boldsymbol{x};\boldsymbol{A}) + \int_0^t \boldsymbol{k}(t-s,\boldsymbol{x}(s))\,\mathrm{d}s, \tag{3.12}$$

where $\boldsymbol{f}$ is given in (3.6), *and $\boldsymbol{k}(t,\boldsymbol{x}) := \mathcal{P}\mathcal{L}e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{x}$.*

Proof. Consider the system (3.5) over a finite time horizon $[0,T]$, which evolves on a smooth manifold $\Gamma \subset \mathbb{R}^N$. For any real-valued phase (observable) space function $g : \Gamma \to \mathbb{R}$, the nonlinear system (3.5) is equivalent to the linear partial differential equation, known as the Liouville equation:

$$\begin{cases} \partial_t u(t,\boldsymbol{z}) = \mathcal{L}[u](t,\boldsymbol{z}) \\ u(0,\boldsymbol{z}) = g(\boldsymbol{z}) \end{cases} \tag{3.13}$$

where the Liouville operator $\mathcal{L}[u] := \bar{\boldsymbol{f}}(\boldsymbol{z}) \cdot \nabla_{\boldsymbol{z}} u$. The equivalency is in the sense that the solution of (3.13) satisfies $u(t,\boldsymbol{z}_0) = g(\boldsymbol{z}(t;\boldsymbol{z}_0))$, where $\boldsymbol{z}(t;\boldsymbol{z}_0)$ is the solution to (3.5) with initial value $\boldsymbol{z}_0$.

Denote $e^{t\mathcal{L}}$ the Koopman operator associated with $\mathcal{L}$ such that $e^{t\mathcal{L}}g(\boldsymbol{z}_0) = g(\boldsymbol{z}(t))$ where $\boldsymbol{z}(t)$ is the solution of (3.5). Then $e^{t\mathcal{L}}$ satisfies the semi-group property, i.e.,

$$e^{t\mathcal{L}}g(z) = g(e^{t\mathcal{L}}z) \tag{3.14}$$

for all g. On the right hand side of (3.14), $\boldsymbol{z}$ can be interpreted as $\boldsymbol{z} = \boldsymbol{\iota}(\boldsymbol{z}) = [\iota_1(\boldsymbol{z}),\ldots,\iota_N(\boldsymbol{z})]$ where $\iota_j(\boldsymbol{z}) = z_j$ for all j.

Now consider the projection operator $\mathcal{P}$ as the truncation such that $\mathcal{P}g(\boldsymbol{z}) = \mathcal{P}g(\boldsymbol{x},\boldsymbol{e}) = g(\boldsymbol{x},0)$ for any $\boldsymbol{z} = (\boldsymbol{x},\boldsymbol{e})$, and its orthogonal complement as $\mathcal{Q} = I - \mathcal{P}$ where I is the identity operator. Note that $\boldsymbol{z}'(t) = \frac{\mathrm{d}\boldsymbol{z}(t)}{\mathrm{d}t} = \frac{\partial}{\partial t}e^{t\mathcal{L}}\boldsymbol{z}_0$, and $\bar{\boldsymbol{f}}(\boldsymbol{z}(t)) = e^{t\mathcal{L}}\boldsymbol{f}(\boldsymbol{z}_0) = e^{t\mathcal{L}}\mathcal{L}\boldsymbol{z}_0$ since $\mathcal{L}\iota_j(\boldsymbol{z}) = \boldsymbol{f}_j(\boldsymbol{z})$ for all $\boldsymbol{z}$ and j. Therefore (3.5) implies that

$$\frac{\partial}{\partial t}e^{t\mathcal{L}}\boldsymbol{z}_0 = e^{t\mathcal{L}}\mathcal{L}\boldsymbol{z}_0 = e^{t\mathcal{L}}\mathcal{P}\mathcal{L}\boldsymbol{z}_0 + e^{t\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}_0. \tag{3.15}$$

Note that the first term on the right hand side of (3.15) is

$$e^{t\mathcal{L}}\mathcal{P}\mathcal{L}\boldsymbol{z}_0 = \mathcal{P}\mathcal{L}e^{t\mathcal{L}}\boldsymbol{z}_0 = \mathcal{P}\mathcal{L}\boldsymbol{z}(t). \tag{3.16}$$

For the second term in (3.15), we recall that the well-known Dyson's identity for the Koopman operator $\mathcal{L}$ is given by

$$e^{t\mathcal{L}} = e^{t\mathcal{Q}\mathcal{L}} + \int_0^t e^{s\mathcal{L}}\mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\,\mathrm{d}s. \tag{3.17}$$

Applying (3.17) to $\mathcal{Q}\mathcal{L}\boldsymbol{z}_0$ yields

$$\begin{aligned} e^{t\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}_0 &= e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}_0 + \int_0^t e^{s\mathcal{L}}\mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}_0\,\mathrm{d}s \\ &= e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}_0 + \int_0^t \mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}e^{s\mathcal{L}}\boldsymbol{z}_0\,\mathrm{d}s \\ &= e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}_0 + \int_0^t \mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}(s)\,\mathrm{d}s. \end{aligned} \tag{3.18}$$

Substituting (3.16) and (3.18) into (3.15), we obtain

$$\frac{\partial}{\partial t}e^{t\mathcal{L}}\boldsymbol{z}_0 = \mathcal{P}\mathcal{L}\boldsymbol{z}(t) + e^{t\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}_0 + \int_0^t \mathcal{P}\mathcal{L}e^{(t-s)\mathcal{Q}\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}(s)\,\mathrm{d}s, \tag{3.19}$$

where we used the fact that $e^{t\mathcal{L}}\mathcal{P}\mathcal{L}\boldsymbol{z}_0 = \mathcal{P}\mathcal{L}e^{t\mathcal{L}}\boldsymbol{z}_0 = \mathcal{P}\mathcal{L}\boldsymbol{z}(t)$. Denote $\boldsymbol{\phi}(t,\boldsymbol{z}) := e^{t\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}$, then we simplify (3.19) into

$$\frac{\partial}{\partial t}e^{t\mathcal{L}}\boldsymbol{z}_0 = \mathcal{P}\mathcal{L}\boldsymbol{z}(t) + \boldsymbol{\phi}(t,\boldsymbol{z}_0) + \int_0^t \boldsymbol{k}(t-s,\boldsymbol{z}(s))\,\mathrm{d}s, \tag{3.20}$$

where $\boldsymbol{k}(t,\boldsymbol{z}) := \mathcal{P}\mathcal{L}\boldsymbol{\phi}(t,\boldsymbol{z}) = \mathcal{P}\mathcal{L}e^{t\mathcal{L}}\mathcal{Q}\mathcal{L}\boldsymbol{z}$.

Now consider the evolution of $\boldsymbol{\phi}(t,\boldsymbol{z})$, which is given by

$$\partial_t\boldsymbol{\phi}(t,\boldsymbol{z}_0) = \mathcal{Q}\mathcal{L}\boldsymbol{\phi}(t,\boldsymbol{z}_0), \tag{3.21}$$

with initial condition $\boldsymbol{\phi}(0, \boldsymbol{z}_0) = \mathcal{Q}\mathcal{L}\boldsymbol{z}_0 = \mathcal{L}\boldsymbol{z}_0 - \mathcal{P}\mathcal{L}\boldsymbol{z}_0 = \bar{\boldsymbol{f}}(\boldsymbol{x}_0, \boldsymbol{e}_0) - \bar{\boldsymbol{f}}(\boldsymbol{x}_0, \boldsymbol{0}) = \boldsymbol{0}$ since $\boldsymbol{e}_0 = \boldsymbol{0}$. Applying $\mathcal{P}$ on both sides of (3.21) yields

$$\partial_t \mathcal{P}\boldsymbol{\phi}(t, \boldsymbol{z}_0) = \mathcal{P}\mathcal{Q}\mathcal{L}\boldsymbol{\phi}(t, \boldsymbol{z}_0) = \boldsymbol{0},$$

with initial $\mathcal{P}\boldsymbol{\phi}(0, \boldsymbol{z}_0) = \boldsymbol{0}$. This implies that $\mathcal{P}\boldsymbol{\phi}(t, \boldsymbol{z}_0) = \boldsymbol{0}$ for all t. Hence, applying $\mathcal{P}$ to both sides of (3.19) yields

$$\frac{\partial}{\partial t}\mathcal{P}\boldsymbol{z}(t) = \frac{\partial}{\partial t}\mathcal{P}e^{t\mathcal{L}}\boldsymbol{z}_0 = \mathcal{P}\mathcal{L}\boldsymbol{z}(t) + \int_0^t \mathcal{P}\boldsymbol{k}(t-s, \boldsymbol{z}(s))\,\mathrm{d}s. \tag{3.22}$$

Restricting to the first n components, $\mathcal{P}\boldsymbol{z}(t)$ reduces to $\boldsymbol{x}(t)$ and $\mathcal{P}\boldsymbol{k}(t-s, \boldsymbol{z}(s))$ reduces to $\boldsymbol{k}(t-s, \boldsymbol{x}(s))$. Recalling that $\mathcal{P}\mathcal{L}\boldsymbol{z}(t) = \mathcal{P}\bar{\boldsymbol{f}}(\boldsymbol{z}(t)) = \bar{\boldsymbol{f}}(\boldsymbol{x}(t), \boldsymbol{0}) = \boldsymbol{f}(\boldsymbol{x}(t))$ completes the proof. □

Note that, (3.12) is *not* an approximation—it is an *exact* representation of the $\boldsymbol{x}$ part of the original problem (3.5). The equation (3.12) can be interpreted as a *mean-field* equation, where the two terms on the right hand side are called the *streaming term* (corresponding to the mean-field dynamics) and *memory term*, respectively. The mean-field dynamics provide the *main drift* of the evolution, and the memory term in a convolution form is for vital *adjustment*. This inspires us to approximate the memory term as a time convolution on $\boldsymbol{x}$, which naturally yields a delay differential equation, as shown in the next subsection.

3.1.4 Delay differential equation

To compute the evolution (3.12) of $\boldsymbol{x}$, we consider an approximation of the Mori-Zwanzig memory term by a neural net $\boldsymbol{\varepsilon}$ with time convolution of $\boldsymbol{x}$ as follows,

$$\int_0^t \boldsymbol{k}(t-s, \boldsymbol{x}(s))\,\mathrm{d}s \approx \boldsymbol{\varepsilon}(\boldsymbol{x}(t), \boldsymbol{h}(t); \boldsymbol{\eta}), \tag{3.23}$$

where $\boldsymbol{h}(t) = \int_0^t \boldsymbol{K}(t-s; \boldsymbol{w})\boldsymbol{x}(s)\,\mathrm{d}s$. In (3.23), $\boldsymbol{K}(\cdot; \boldsymbol{w})$ is a convolutional operator with parameter $\boldsymbol{w}$, and $\boldsymbol{\varepsilon}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\eta})$ is a deep neural network with $(\boldsymbol{x}, \boldsymbol{h})$ as input and $\boldsymbol{\eta}$ as parameter.

Both $\boldsymbol{w}$ and $\boldsymbol{\eta}$ are to be trained by the cascade data $\mathcal{D}$. Hence, (3.12) reduces to the *delay differential equation* which involves a time integral $\boldsymbol{h}(t)$ of past $\boldsymbol{x}$:

$$\boldsymbol{x}' = \tilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\theta}) := \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{A}) + \boldsymbol{\varepsilon}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\eta}). \tag{3.24}$$

The initial condition of (3.24) with source set $\mathcal{S}$ is given by

$$\boldsymbol{x}(0) = \boldsymbol{\chi}_{\mathcal{S}}, \quad \boldsymbol{h}(0) = \mathbf{0}, \quad \text{and} \quad \boldsymbol{x}(t) = \boldsymbol{h}(t) = \mathbf{0}, \quad \forall\, t < 0. \tag{3.25}$$

We call the system (3.24) with initial (3.25) the *neural mean-field* (NMF) dynamics.

The delay differential equation (3.24) is equivalent to a coupled system of $(\boldsymbol{x}, \boldsymbol{h})$ which is shown in the following theorem.

Theorem 3.1.4. *The delay differential equation* (3.24) *is equivalent to the following coupled system:*

$$\boldsymbol{x}' = \tilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{A}, \boldsymbol{\eta}) = \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{A}) + \boldsymbol{\varepsilon}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\eta}) \tag{3.26a}$$

$$\boldsymbol{h}' = \textstyle\int_0^t \boldsymbol{K}(t-s; \boldsymbol{w}) \tilde{\boldsymbol{f}}(\boldsymbol{x}(s), \boldsymbol{h}(s); \boldsymbol{A}, \boldsymbol{\eta}) \, \mathrm{d}s \tag{3.26b}$$

with initial condition (3.25)*. In particular, if* $\boldsymbol{K}(t; \boldsymbol{w}) = \sum_{l=1}^{L} \boldsymbol{B}_l e^{-\boldsymbol{C}_l t}$ *for some* $L \in \mathbb{N}$ *with* $\boldsymbol{w} = \{(\boldsymbol{B}_l, \boldsymbol{C}_l)_l : \boldsymbol{B}_l \boldsymbol{C}_l = \boldsymbol{C}_l \boldsymbol{B}_l, \forall\, l \in [L]\}$*, then* (3.26) *can be solved by a non-delay system with* $\boldsymbol{h}' = \sum_{l=1}^{L} (\boldsymbol{B}_l \boldsymbol{x} - \boldsymbol{C}_l \boldsymbol{h})$*.*

Proof. From the definition of $\boldsymbol{h}(t)$ in (3.27), we obtain

$$\boldsymbol{h} = \int_0^t \boldsymbol{K}(t-s; \boldsymbol{w}) \boldsymbol{x}(s) \, \mathrm{d}s = \int_{-\infty}^t \boldsymbol{K}(t-s; \boldsymbol{w}) \boldsymbol{x}(s) \, \mathrm{d}s = \int_0^\infty \boldsymbol{K}(s; \boldsymbol{w}) \boldsymbol{x}(t-s) \, \mathrm{d}s \tag{3.27}$$

where we used the fact that $\boldsymbol{x}(t) = 0$ for $t < 0$. Taking derivative on both sides of (3.27) yields

$$\begin{aligned}\boldsymbol{h}' &= \int_0^\infty \boldsymbol{K}(s;\boldsymbol{w})\boldsymbol{x}'(t-s)\,\mathrm{d}s = \int_0^\infty \boldsymbol{K}(s;\boldsymbol{w})\tilde{\boldsymbol{f}}(\boldsymbol{x}(t-s),\boldsymbol{h}(t-s);\boldsymbol{A},\boldsymbol{\eta})\,\mathrm{d}s \\ &= \int_{-\infty}^t \boldsymbol{K}(t-s;\boldsymbol{w})\tilde{\boldsymbol{f}}(\boldsymbol{x}(s),\boldsymbol{h}(s);\boldsymbol{A},\boldsymbol{\eta})\,\mathrm{d}s = \int_0^t \boldsymbol{K}(t-s;\boldsymbol{w})\tilde{\boldsymbol{f}}(\boldsymbol{x}(s),\boldsymbol{h}(s);\boldsymbol{A},\boldsymbol{\eta})\,\mathrm{d}s\end{aligned}$$

where we used the fact that $\boldsymbol{x}'(t) = \tilde{\boldsymbol{f}}(\boldsymbol{x}(t),\boldsymbol{h}(t);\boldsymbol{A},\boldsymbol{\eta}) = 0$ for $t < 0$ in the last equality.

If $\boldsymbol{K}(t;\boldsymbol{w}) = \sum_l \boldsymbol{B}_l e^{-\boldsymbol{C}_l t}$, then we can take derivative of (3.27) and obtain

$$\begin{aligned}\boldsymbol{h}'(t) &= \sum_{l=1}^L \frac{\mathrm{d}}{\mathrm{d}t}\Big(\int_{-\infty}^t \boldsymbol{B}_l e^{-\boldsymbol{C}_l t}\boldsymbol{x}(s)\,\mathrm{d}s\Big) = \sum_{l=1}^L \Big(\boldsymbol{B}_l\boldsymbol{x}(t) - \int_{-\infty}^t \boldsymbol{B}_l\boldsymbol{C}_l e^{-\boldsymbol{C}_l t}\boldsymbol{x}(s)\,\mathrm{d}s\Big) \\ &= \sum_{l=1}^L \Big(\boldsymbol{B}_l\boldsymbol{x}(t) - \boldsymbol{C}_l\int_{-\infty}^t \boldsymbol{B}_l e^{-\boldsymbol{C}_l t}\boldsymbol{x}(s)\,\mathrm{d}s\Big) = \sum_{l=1}^L(\boldsymbol{B}_l\boldsymbol{x}(t) - \boldsymbol{C}_l\boldsymbol{h}(t)).\end{aligned}$$

□

As shown in Theorem 3.1.4, NMF (3.24) reduces to a non-deday ODE system of $(\boldsymbol{x},\boldsymbol{h})$ with (3.26a) and $\boldsymbol{h}' = \boldsymbol{B}\boldsymbol{x} - \boldsymbol{C}\boldsymbol{h}$. There are two different ways to solve such a system, which derives the two different models proposed in this thesis:

- **Discrete NMF**: solve the system by an Euler discretization over a grid of T time steps.
- **Continuous NMF**: solve the system by a modified neural ODE (NODE) [10] for any $t \in [0,T]$.

More detail on the proposed models are shown in the following sections.

3.2 Discrete NMF algorithm

Based on Theorem 3.1.4, by finite difference in time with normalized step size 1 and proper scaling of the network parameters $\boldsymbol{\theta}$, we could obtain the following discretization system which derives the framework of discrete NMF model.

Theorem 3.2.1. *The discretization of the system* (3.26) *reduces to a recurrent neural network (RNN) with hidden layers* $(\boldsymbol{x}_t, \boldsymbol{h}_t)$ *for* $t = 0, 1, \ldots, T-1$*:*

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{f}(\boldsymbol{x}_t; \boldsymbol{A}) + \boldsymbol{\varepsilon}(\boldsymbol{x}_t, \boldsymbol{h}_t; \boldsymbol{\eta}) \tag{3.28a}$$

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \textstyle\sum_{l=1}^{L}(\boldsymbol{B}_l \boldsymbol{x}_{t+1} - \boldsymbol{C}_l \boldsymbol{h}_t) \tag{3.28b}$$

where the input is given by $\boldsymbol{x}_0 = \chi_S$ *and* $\boldsymbol{h}_0 = \boldsymbol{0}$*.*

In this section, we will propose the discrete NMF model derived by Theorem 3.2.1 and show its efficiencies and robustness by empirical study.

3.2.1 Formulation of discrete NMF model

In stead of the exponential kernel in Theorem 3.1.4, we consider a more general convolution kernel $\boldsymbol{K}(\cdot; \boldsymbol{w})$. One benefit of this change is that the convolution weight $\boldsymbol{K}$ on older state $\boldsymbol{x}$ in (3.23) rapidly diminishes in practice, and hence the memory kernel $\boldsymbol{K}$ can be well approximated with a truncated history of finite length $\tau > 0$, or $\tau \in \mathbb{N}$ after discretization. Hence, we could substitute (3.28b) by

$$\boldsymbol{h}_t = \boldsymbol{K}^{\boldsymbol{w}} \boldsymbol{m}_t \tag{3.29}$$

where $\boldsymbol{K}^{\boldsymbol{w}} = [\boldsymbol{K}_0^{\boldsymbol{w}}, \ldots, \boldsymbol{K}_\tau^{\boldsymbol{w}}]$ and $\boldsymbol{m}_t = [\boldsymbol{x}_t; \ldots; \boldsymbol{x}_{t-\tau}]$. Together with Theorem 3.2.1, we could formulate a single evolution of $\boldsymbol{m}_t$ for time $t = 0, \ldots, T-1$:

Definition 3.2.2 (Formulation of discrete NMF model)**.** *Let* $\boldsymbol{J}_s := [\cdots, \boldsymbol{I}, \cdots] \in \mathbb{R}^{n \times (\tau+1)n}$ *has identity* $\boldsymbol{I}$ *as the* $(s+1)$*th block for* $s = 0, \ldots, \tau-1$*. We define the evolution of* $\boldsymbol{m}_t$ *as*

$$\boldsymbol{m}_{t+1} = \boldsymbol{g}(\boldsymbol{m}_t; \boldsymbol{\theta}) \tag{3.30}$$

where $\boldsymbol{g}(\boldsymbol{m}; \boldsymbol{\theta}) := [\boldsymbol{J}_0 \boldsymbol{m} + \tilde{\boldsymbol{f}}(\boldsymbol{J}_0 \boldsymbol{m}, \boldsymbol{K}^{\boldsymbol{w}} \boldsymbol{m}; \boldsymbol{\theta}); \boldsymbol{J}_0 \boldsymbol{m}; \ldots; \boldsymbol{J}_{\tau-1} \boldsymbol{m}]$*.*

Note that $\boldsymbol{J}_s \boldsymbol{m}_t$ extracts the $(s+1)$th block $\boldsymbol{x}_{t-s}$ of $\boldsymbol{m}_t$, which indicates the equivalence of system (3.28) and (3.30).

3.2.2 Learning the parameters of discrete NMF

With the discrete NMF formed by Definition 3.2.2 with parameter $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{w})$, we get the infection probability function for each $t \in [T]$ as output. Moreover, the support and magnitude of $\boldsymbol{A}$ implies the network structure and strength of interaction between nodes, respectively.

Loss function In order to train the network parameters $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{w})$ of (3.30) using cascade data $\mathcal{D}$ in a fully supervised setting, we apply the following cross entropy loss function:

$$\ell(\boldsymbol{x}, \hat{\boldsymbol{x}}) = \textstyle\sum_{t=1}^{T} \hat{\boldsymbol{x}}_t \cdot \log \boldsymbol{x}_t + (\boldsymbol{1} - \hat{\boldsymbol{x}}_t) \cdot \log(\boldsymbol{1} - \boldsymbol{x}_t), \tag{3.31}$$

where $\hat{\boldsymbol{x}} = \{\hat{\boldsymbol{x}}_t \in \{0,1\}^n : t \in [T]\}$ which records the infection status of nodes in the cascade data and the logarithm is taken componentwisely. Maximizing the log-likelihood of $\hat{\boldsymbol{x}}$ for the dynamics $\boldsymbol{x}_t = \boldsymbol{x}_t(\boldsymbol{\theta}) \in [0,1]^n$ induced by $\boldsymbol{\theta}$ is equivalent to minimizing the loss function $\ell(\boldsymbol{x}, \hat{\boldsymbol{x}})$. We can add a regularization term $r(\boldsymbol{\theta})$ to (3.31) to impose prior knowledge or constraint on $\boldsymbol{\theta}$. In particular, if $\mathcal{E}$ is known, we can enforce a constraint such that $\boldsymbol{A}$ must be supported on $\mathcal{E}$ only. Otherwise, we can add $\|\boldsymbol{A}\|_1$ or $\|\boldsymbol{A}\|_0$ (the l_1 or l_0 norm of the vectorized $\boldsymbol{A}$) if $\mathcal{E}$ is expected to be sparse.

3.2.3 Revisiting optimal control theory

We will show that the parameters learning in discrete NMF can be reduced to an optimal control problem. Before that, we revisit the optimal control problem and theory in this subsection.

Background The problem of optimal control has been well studied in both continuous and discrete cases in the past decades [5]. In particular, the discrete optimal control with nonlinear difference equations and the associated maximum principle have been extensively exploited. Recently, an optimal control viewpoint of deep learning has been proposed [63]—the network parameters of a neural network play the role of control variable in a discretized differential equation, and the training of these parameters for the network output to minimize the loss function can be viewed as finding the optimal control to minimize the objective function at the terminal state.

Overview of optimal control problem Consider an ordinary differential equation (ODE) having the form

$$\begin{cases} \boldsymbol{x}'(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{\alpha}(t)), & t > 0, \\ \boldsymbol{x}(0) = \boldsymbol{x}^0, & \text{otherwise.} \end{cases} \tag{3.32}$$

We call a function $\boldsymbol{\alpha} : [0, \infty) \to \boldsymbol{A}$ a *control* since the system may behave quite differently as the parameters $\boldsymbol{\alpha}$ changing. Define the *payoff function* as

$$P[\boldsymbol{\alpha}(\cdot)] := \int_0^T r(\boldsymbol{x}(t), \boldsymbol{\alpha}(t))\,\mathrm{d}t + g(\boldsymbol{x}(T)) \tag{3.33}$$

where r and g are given running and terminal payoff, respectively. The goal of the problem is to find an optimal control α^* to maximize the payoff.

Example Let $\alpha(t)$ be the fraction of colony effort devoted to increase work force which is constrained by $0 \le \alpha(t) \le 1$. Suppose the population of workers and queens change according to the following systems, respectively.

$$\begin{cases} w'(t) = -\mu w(t) + b s(t)\alpha(t) w(t) \\ w(0) = w^0 \end{cases} \quad \text{and} \quad \begin{cases} q'(t) = -\nu q(t) + c(1 - \alpha(t)) s(t) w(t) \\ q(0) = q^0 \end{cases}$$

The problem to maximize the number of queens at time T could be modeled as one optimal control problem as follows.

$$\begin{aligned} \max_{\alpha} \quad & P(\alpha(\cdot)) := q(T) \\ \text{s.t.} \quad & \boldsymbol{x}'(t) = [w'(t); q'(t)], \quad x^0 = [w^0; q^0], \end{aligned}$$

In that case, the running payoff $r = 0$ and the terminal payoff $g(\boldsymbol{x}(T)) = q(T)$.

Maximum principle The following maximization principle is useful for characterizing an optimal control.

Theorem 3.2.3 (Pontryagin maximum principle (PMP)). *[29] Assume $\alpha^*(\cdot)$ is optimal for the problem defined by* (3.32) *and* (3.33)*. Let $x^*(\cdot)$ be the corresponding trajectory. Consider the* Hamiltonian *function*

$$\boldsymbol{H}(x, p, a) = \boldsymbol{f}(x, a) \cdot \boldsymbol{p} + r(x, a).$$

Then there exists a function $\boldsymbol{p}^* : [0, T] \to \mathbb{R}^n$ such that

(i) $(\boldsymbol{x}^*)'(t) = \nabla_{\boldsymbol{p}} \boldsymbol{H}\left(\boldsymbol{x}^*(t), \boldsymbol{p}^*(t), \boldsymbol{\alpha}^*(t)\right)$.

(ii) $(\boldsymbol{p}^*)'(t) = -\nabla_{\boldsymbol{x}} \boldsymbol{H}\left(\boldsymbol{x}^*(t), \boldsymbol{p}^*(t), \boldsymbol{\alpha}^*(t)\right)$.

(iii) $\boldsymbol{H}\left(\boldsymbol{x}^*(t), \boldsymbol{p}^*(t), \boldsymbol{\alpha}^*(t)\right) = \max_{\boldsymbol{\alpha}} \boldsymbol{H}\left(\boldsymbol{x}^*(t), \boldsymbol{p}^*(t), a\right)$ *for $t \in [0, T]$.*

(iv) The mapping $t \mapsto \boldsymbol{H}\left(\boldsymbol{x}^*(t), \boldsymbol{p}^*(t), \boldsymbol{\alpha}^*(t)\right)$ is constant.

(v) The terminal condition $\boldsymbol{p}^*(T) = \nabla g\left(\boldsymbol{x}^*(T)\right)$ holds.

3.2.4 Optimal control of parameter training

Formulation as optimal control problem The optimal parameter $\boldsymbol{\theta}$ can be obtained by minimizing the loss function in (3.31) subject to the NMF dynamics (3.30). This procedure can also be cast as an optimal control problem to find $\boldsymbol{\theta}$ that steers $\boldsymbol{m}_t$ to fit data $\mathcal{D}$ through the NMF in (3.30):

$$\min_{\boldsymbol{\theta}} \quad \mathcal{J}(\boldsymbol{\theta}) := (1/K) \cdot \textstyle\sum_{k=1}^{K} \ell(\boldsymbol{x}^{(k)}, \hat{\boldsymbol{x}}^{(k)}) + r(\boldsymbol{\theta}) \tag{3.34a}$$

$$\text{s.t.} \quad \boldsymbol{m}_{t+1}^{(k)} = \boldsymbol{g}(\boldsymbol{m}_t^{(k)}; \boldsymbol{\theta}), \quad \boldsymbol{m}_0^{(k)} = [\boldsymbol{\chi}_{\mathcal{S}_k}, \boldsymbol{0}, \dots, \boldsymbol{0}], \quad t \in [T] - 1,\ k \in [K], \tag{3.34b}$$

where $\boldsymbol{x}_t^{(k)} = \boldsymbol{J}_0 \boldsymbol{m}_t^{(k)}$ for all t and k.

Modified Pontryagin's Maximum Principle on total Hamiltonian The Pontryagin's Maximum Principle (PMP) provides an important optimality condition of the optimal control [5,63]. In standard optimal control, the control variable can be chosen freely in the allowed set at any given time t, which is a key in the proof of PMP. However, the NMF dynamics derived in (3.26) or (3.28) require a time invariant control $\boldsymbol{\theta}$ throughout. This is necessary since $\boldsymbol{\theta}$ corresponds to the network parameter and needs to be shared across different layers of the RNN, either from the linear kernel case with state $[\boldsymbol{x};\boldsymbol{h}]$ in (3.28) or the general case with state $\boldsymbol{m}$ in (3.30). Therefore, we need to modify the original PMP and the optimality condition for our NMF formulation. To this end, consider the *Hamiltonian* function

$$H(\boldsymbol{m},\boldsymbol{p};\boldsymbol{\theta}) = \boldsymbol{p}\cdot\boldsymbol{g}(\boldsymbol{m};\boldsymbol{\theta}) - \tfrac{1}{T}r(\boldsymbol{\theta}), \tag{3.35}$$

and define the *total Hamiltonian* of the system (3.28) as $\sum_{t=0}^{T-1} H(\boldsymbol{m}_t,\boldsymbol{p}_{t+1};\boldsymbol{\theta})$. Then we can show that the optimal solution $\boldsymbol{\theta}^*$ is a time invariant control satisfying a *modified* PMP as follows.

Theorem 3.2.4. *Let $\boldsymbol{x}^*$ be the optimally controlled state process by $\boldsymbol{\theta}^*$, then there exists a co-state (adjoint) $\boldsymbol{p}^*$ which satisfies the backward differential equation*

$$\boldsymbol{m}^*_{t+1} = \boldsymbol{g}(\boldsymbol{m}^*_t;\boldsymbol{\theta}^*), \qquad \boldsymbol{m}^*_0 = [\boldsymbol{\chi}_{\mathcal{S}_k};\mathbf{0};\dots;\mathbf{0}], \quad t=0,\dots,T-1, \tag{3.36a}$$

$$\boldsymbol{p}^*_t = \boldsymbol{p}^*_{t+1}\cdot\nabla_{\boldsymbol{m}}\boldsymbol{g}(\boldsymbol{m}^*_t;\boldsymbol{\theta}^*), \quad \boldsymbol{p}^*_T = -\nabla_{\boldsymbol{m}_T}\ell, \quad t=T-1,\dots,0. \tag{3.36b}$$

Moreover, the optimal $\boldsymbol{\theta}^*$ maximizes the total Hamiltonian: for any $\boldsymbol{\theta}$, there is

$$\textstyle\sum_{t=0}^{T-1} H(\boldsymbol{m}^*_t,\boldsymbol{p}^*_{t+1};\boldsymbol{\theta}^*) \ge \sum_{t=0}^{T-1} H(\boldsymbol{m}^*_t,\boldsymbol{p}^*_{t+1};\boldsymbol{\theta}). \tag{3.37}$$

In addition, for any given $\boldsymbol{\theta}$, there is

$$\nabla_{\boldsymbol{\theta}}\mathcal{J}(\boldsymbol{\theta}) = -\sum_{t=0}^{T-1}\partial_{\boldsymbol{\theta}}H(\boldsymbol{m}^{\boldsymbol{\theta}}_t,\boldsymbol{p}^{\boldsymbol{\theta}}_{t+1};\boldsymbol{\theta}),$$

where $\{\boldsymbol{m}_t^{\boldsymbol{\theta}}, \boldsymbol{p}_t^{\boldsymbol{\theta}} : 0 \le t \le T\}$ *are obtained by the forward and backward passes* (3.36a)-(3.36b) *with* $\boldsymbol{\theta}$*.*

Proof. We consider the augmented state $\boldsymbol{\xi}$ and nonlinear dynamics $\bar{\boldsymbol{g}}(\cdot;\boldsymbol{\theta})$ associated with $\boldsymbol{m}$ and $\boldsymbol{g}(\cdot;\boldsymbol{\theta})$, defined as follows:

$$\boldsymbol{\xi}_0 = \begin{bmatrix} \boldsymbol{m}_0 \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad \boldsymbol{\xi}_1 = \bar{\boldsymbol{g}}(\boldsymbol{\xi}_0;\boldsymbol{\theta}) := \begin{bmatrix} \boldsymbol{g}^1(\boldsymbol{m}_0;\boldsymbol{\theta}) \\ \boldsymbol{g}^2(\boldsymbol{m}_0;\boldsymbol{\theta}) \\ \vdots \\ \boldsymbol{g}^T(\boldsymbol{m}_0;\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{m}_1 \\ \boldsymbol{m}_2 \\ \vdots \\ \boldsymbol{m}_T \end{bmatrix}, \tag{3.38}$$

where $\boldsymbol{g}^t$ stands for the composition of $\boldsymbol{g}(\cdot;\boldsymbol{\theta})$ for t times.

Without overloading the notations, we reuse $\mathcal{J}$ and ℓ of the objective function (3.34a) and loss function (3.31) of $\boldsymbol{m}$ respectively for the augmented state $\boldsymbol{\xi}$. In addition, following [63], we further simpify the notation by combining the K training data into a single variable $\hat{\boldsymbol{x}} := [\hat{\boldsymbol{x}}^{(1)}, \ldots, \hat{\boldsymbol{x}}^{(K)}]$; similar for the state variable $\boldsymbol{x}$. In this case, the dynamics $\boldsymbol{g}$ is applied to each column of $\boldsymbol{x}$, and the loss function ℓ is to be interpreted as the average loss as in (3.31). Furthermore, we temporarily assume the regularization $r(\boldsymbol{\theta}) = 0$ as it is simple to append $\boldsymbol{\theta}$ to the state $\boldsymbol{\xi}$ and merge $r(\boldsymbol{\theta})$ into the loss function $\ell(\boldsymbol{\xi}, \hat{\boldsymbol{\xi}})$. Then the optimal control problem (3.34) is rewritten as

$$\min_{\boldsymbol{\theta}} \quad \mathcal{J}(\boldsymbol{\theta}) := \ell(\boldsymbol{\xi}, \hat{\boldsymbol{\xi}}) + r(\boldsymbol{\theta}) \tag{3.39a}$$

$$\text{s.t.} \quad \boldsymbol{\xi}_1 = \bar{\boldsymbol{g}}(\boldsymbol{\xi}_0;\boldsymbol{\theta}), \quad \boldsymbol{\xi}_0 = [\boldsymbol{m}_0; \mathbf{0}; \ldots; \mathbf{0}]. \tag{3.39b}$$

Note that (3.39) is a one-step optimal control with $\bar{\boldsymbol{g}}(\cdot;\boldsymbol{\theta})$. Now by the discrete Pontryagin's Maximum Principle [5], for the state $\boldsymbol{\xi}^*$ optimally controlled by $\boldsymbol{\theta}^*$, there exists a co-state

$\boldsymbol{\psi}^*$, such that $\boldsymbol{\xi}^*$ and $\boldsymbol{\psi}^*$ satisfy the following forward and backward equations for $\boldsymbol{\theta} = \boldsymbol{\theta}^*$:

$$\boldsymbol{\xi}_1^* = \bar{\boldsymbol{g}}(\boldsymbol{\xi}_0^*; \boldsymbol{\theta}^*), \qquad \boldsymbol{\xi}_0^* = [\boldsymbol{m}_0; \mathbf{0}; \dots; \mathbf{0}], \tag{3.40a}$$

$$\boldsymbol{\psi}_0^* = \boldsymbol{\psi}_1^* \cdot \nabla_{\boldsymbol{\xi}} \bar{\boldsymbol{g}}(\boldsymbol{\xi}_1^*; \boldsymbol{\theta}^*), \quad \boldsymbol{\psi}_1^* = -\nabla_{\boldsymbol{\xi}} \ell(\boldsymbol{\xi}_1^*, \hat{\boldsymbol{\xi}}), \tag{3.40b}$$

where

$$\boldsymbol{\xi}_1^* = [\boldsymbol{m}_1^*; \dots; \boldsymbol{m}_T^*] \quad \text{and} \quad \boldsymbol{\psi}_1^* = [\partial_{\boldsymbol{m}_1} \ell(\boldsymbol{\xi}_1^*, \hat{\boldsymbol{\xi}}); \dots; \partial_{\boldsymbol{m}_T} \ell(\boldsymbol{\xi}_1^*, \hat{\boldsymbol{\xi}})] = [\boldsymbol{p}_1^*; \dots; \boldsymbol{p}_T^*]. \tag{3.41}$$

In addition, $\boldsymbol{\theta}^*$ maximizes the Hamiltonian $\mathcal{H}$ associated with (3.40):

$$\mathcal{H}(\boldsymbol{\xi}^*, \boldsymbol{\psi}^*; \boldsymbol{\theta}^*) \geq \mathcal{H}(\boldsymbol{\xi}^*, \boldsymbol{\psi}^*; \boldsymbol{\theta}), \quad \forall \boldsymbol{\theta}, \quad \text{where} \quad \mathcal{H}(\boldsymbol{\xi}, \boldsymbol{\psi}; \boldsymbol{\theta}) := \boldsymbol{\psi}_1 \cdot \bar{\boldsymbol{g}}(\boldsymbol{\xi}_0; \boldsymbol{\theta}) - r(\boldsymbol{\theta}). \tag{3.42}$$

Combining (3.41), (3.42), and the definition of H in (3.35) yields the maximization of total Hamiltonian at the optimal control $\boldsymbol{\theta}^*$:

$$\textstyle\sum_{t=0}^{T-1} H(\boldsymbol{m}_t^*, \boldsymbol{p}_{t+1}^*; \boldsymbol{\theta}^*) \geq \sum_{t=0}^{T-1} H(\boldsymbol{m}_t^*, \boldsymbol{p}_{t+1}^*; \boldsymbol{\theta}), \quad \forall \boldsymbol{\theta}.$$

For any control $\boldsymbol{\theta}$ and its state and co-state variables $\boldsymbol{\xi}^{\boldsymbol{\theta}}$ and $\boldsymbol{\psi}^{\boldsymbol{\theta}}$ following (3.40) with $\boldsymbol{\theta}$ (also corresponding to $\boldsymbol{m}_t^{\boldsymbol{\theta}}$ and $\boldsymbol{p}_t^{\boldsymbol{\theta}}$ for $t = 0, \dots, T$), we have

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\xi}} \ell(\boldsymbol{\xi}_1^{\boldsymbol{\theta}}, \hat{\boldsymbol{\xi}}) \cdot \nabla_{\boldsymbol{\theta}} \boldsymbol{\xi}_1^{\boldsymbol{\theta}} + \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}) \\
&= [\partial_{\boldsymbol{m}_1} \ell(\boldsymbol{\xi}_1^{\boldsymbol{\theta}}, \hat{\boldsymbol{\xi}}); \dots; \partial_{\boldsymbol{m}_T} \ell(\boldsymbol{\xi}_1^{\boldsymbol{\theta}}, \hat{\boldsymbol{\xi}})] \cdot [\partial_{\boldsymbol{\theta}} \boldsymbol{g}(\boldsymbol{m}_0^{\boldsymbol{\theta}}; \boldsymbol{\theta}); \dots; \partial_{\boldsymbol{\theta}} \boldsymbol{g}(\boldsymbol{m}_{T-1}^{\boldsymbol{\theta}}; \boldsymbol{\theta})] + \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}) \\
&= -\textstyle\sum_{t=1}^{T} \Big(\boldsymbol{p}_t^{\boldsymbol{\theta}} \cdot \partial_{\boldsymbol{\theta}} \boldsymbol{g}(\boldsymbol{m}_t^{\boldsymbol{\theta}}; \boldsymbol{\theta}) + \frac{1}{T} \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}) \Big) \\
&= -\textstyle\sum_{t=1}^{T} \partial_{\boldsymbol{\theta}} H(\boldsymbol{m}_t^{\boldsymbol{\theta}}, \boldsymbol{p}_{t+1}^{\boldsymbol{\theta}}; \boldsymbol{\theta}),
\end{aligned}$$

which completes the proof. □

Theorem 3.2.4 implies that performing gradient descent to minimize $\mathcal{J}$ in (3.34a) with back-propagation is equivalent to maximizing the total Hamiltonian in light of (3.37).

Proposed algorithm The numerical implementation of the proposed discrete NMF is summarized in Algorithm 1.

Algorithm 1 Discrete neural mean-field (NMF) algorithm for network inference and influence estimation

```
Input: $\mathcal{D} = \{\mathcal{C}_k : k \in [K]\}$ where $\mathcal{C}_k = \{\hat{\boldsymbol{x}}^{(k)}(t) \in \{0,1\}^n : t = 0,1,\dots,T\}$.
Initialization: Parameter $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{w})$.
for $k = 1, \dots,$ MaxItrations do
    Sample a mini-batch $\hat{\mathcal{D}} \subset \mathcal{D}$ of cascades.
    Compute $\{\boldsymbol{m}_t : t \in [T]\}$ using (3.30) with $\boldsymbol{\theta}$ and $\boldsymbol{m}_0 = [\boldsymbol{\chi}_{\mathcal{S}}; \boldsymbol{0}]$ for each $\mathcal{C} \in \hat{\mathcal{D}}$. (Forward pass)
    Compute $\hat{\nabla}_{\boldsymbol{\theta}} \mathcal{J} = \sum_{\mathcal{C} \in \hat{\mathcal{D}}} \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{x}, \hat{\boldsymbol{x}})$ with $\ell$ in (3.31).    (Backward pass)
    Update parameter $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \tau \hat{\nabla}_{\boldsymbol{\theta}} \mathcal{J}$.
end for
Output: Network parameter $\boldsymbol{\theta}$.
```

Solution of Problem 1 With the trained discrete NMF model, we could fully solve Problem 1:

- For influence estimation task, we can predict the influence $\{\boldsymbol{x}_t : t \in [T]\}$ of any new source set $\mathcal{S}$ by a forward pass of NMF dynamics (3.30) with input $\boldsymbol{x}_0 = \boldsymbol{\chi}_{\mathcal{S}}$ and the learned parameters $\boldsymbol{\theta}$.

- For network inference task, the trained parameter $\boldsymbol{A}$ contained in $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{w})$ is the estimation of the transmission matrix which also reveals the edge $\mathcal{E}$ of the diffusion network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ by its support.

3.2.5 Implementation details

Environment All experiments are conducted on a Linux workstation with Intel i9 8-Core Turbo 5GHz CPU, 64GB of memory, and an Nvidia RTX 2080Ti GPU. All the algorithms implemented in Python 3. All experiments are performed on the same machine.

Synthetic datasets Three types of network models [57] are used to generate synthetic networks for mimicking the structure of real-world diffusion networks: hierarchical (Hier) network [17], core-periphery (Core) network [59] and Random (Rand) network with parameter matrices [0.9,0.1;0.1,0.9], [0.9,0.5;0.5,0.3], and [0.5,0.5;0.5,0.5], respectively. On each network, we simulate the diffusion models with three type of distribution $p(t;\cdot)$: exponential distribution (Exp), Rayleigh distribution (Ray), and general Weibull distribution (Wbl). In particular, we draw the parameters α_{ji} from Unif[0.1,1] to simulate the heterogeneous interactions between nodes for exponential and Reyleigh distributions. We generate both of the shape and scale parameters of Weibull distribution from Unif[1,10] randomly. All networks and cascades are generated by SNAP [60]. There are some facts we need to pay attentions:

- Our theoretical results in Section 3.1 are based on the standard diffusion models with exponential distribution. However, we still conduct experiments on other distributions to test the performance of NMF empirically.

- For a general Weibull distribution, its transmission likelihood is

$$f(t_j|t_i;\lambda_{ij},k_{ij}) \begin{cases} \frac{k_{ij}}{\lambda_{ij}}\left(\frac{x}{\lambda_{ij}}\right)^{k_{ij}-1} e^{-\left((t_j-t_i)/\lambda_{ij}\right)^{k_{ij}}} & \text{if } t_j > t_i \\ 0 & \text{otherwise} \end{cases}$$

  where $k_{ij} > 0$ is the shape parameter and $\lambda_{ij} > 0$ is the scale parameter of the distribution over the edge $(i,j) \in \mathcal{E}$. Thus, k_{ij} and λ_{ij} both contribute to the transmission strength of the edge α_{ij}. However, there is no explicit relation of α_{ij} and k_{ij}, λ_{ij}. So the results of the experiment on network inference for Weibull distribution can not be evaluated which will be ignored.

- For each edge $(i,j) \in \mathcal{E}$, Exponential distribution is a Weibull distribution with parameters $\lambda_{ij} = \frac{1}{\alpha_{ij}}$ and $k_{ij} = 1$. Rayleigh distribution is also a special Weibull distribution with parameters $\lambda_{ij} = \frac{1}{\sqrt{2}\alpha_{ij}}$ and $k_{ij} = 2$.

Neural network on $\boldsymbol{\varepsilon}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\eta})$ In our NMF implementation on discrete and continuous models, the neural mean field dynamic $\boldsymbol{g}(\cdot; \boldsymbol{\theta})$ derived from Theorem 3.1.4 is learned as Algorithm 1, where $\boldsymbol{\varepsilon}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\eta})$ is always a three-layer fully connected network. Specifically, the input layer size of $\boldsymbol{\varepsilon}$ is $2n$, and both of the hidden and output layer sizes are n. We use Exponential Linear Unit (ELU) as the activation function. The output is truncated into $[0, 1]$. We use the ℓ_0-norm regularization approximated by log-sum introduced by [83].

Evaluation metrics To evaluate the performance on influence estimation, we calculate the ground truth $\boldsymbol{x}^*$ of node infection probability with given source set $\mathcal{S}$ in two steps:

1. Generate 10,000 cascades start from the source set $\mathcal{S}$.

2. For each node i and time t, estimate the ground truth of $x_i^*(t; \boldsymbol{\chi}_{\mathcal{S}})$ by the frequency of cascades with node i infected by t.

With the ground truth node infection probability $\boldsymbol{x}^*$, the Mean Absolute Error (MAE) of influence (Inf) and infection probability (Prob) of the estimated $\boldsymbol{x}$ are defined by $|\mathbf{1} \cdot (\boldsymbol{x}(t) - \boldsymbol{x}(t)^*)|$ and $\|\boldsymbol{x}(t) - \boldsymbol{x}(t)^*\|_1/n$ for every t, respectively. For all the experiments related on the influence estimation, we also use the scaled influence MAE $|\mathbf{1} \cdot (\boldsymbol{x}(t) - \boldsymbol{x}(t)^*)|/n$ as an evaluation metric.

To evaluate the quality of $\mathcal{E}$ and $\boldsymbol{A}$, we use four metrics: precision (Prc), recall (Rcl), accuracy (Acc), and correlation (Cor), defined as follows,

$$\begin{aligned} \mathrm{Prc}(\mathcal{E}, \mathcal{E}^*) &= \tfrac{|\mathcal{E} \cap \mathcal{E}^*|}{|\mathcal{E}^*|}, & \mathrm{Rcl}(\mathcal{E}, \mathcal{E}^*) &= \tfrac{|\mathcal{E} \cap \mathcal{E}^*|}{|\mathcal{E}|}, \\ \mathrm{Acc}(\mathcal{E}, \mathcal{E}^*) &= 1 - \tfrac{|\mathcal{E} - \mathcal{E}^*|}{|\mathcal{E}| + |\mathcal{E}^*|}, & \mathrm{Cor}(A, A^*) &= \tfrac{|\mathrm{tr}(A^\top A^*)|}{\|A\|_F \|A^*\|_F}, \end{aligned}$$

where $|\mathcal{E}|$ counts the number of nonzero entries in $\mathcal{E}$, and $\mathcal{E}^*$ and $\boldsymbol{A}^*$ are their true values, respectively. In Acc, the edge set $\mathcal{E}$ is also interpreted as a matrix, and $|\mathcal{E}|$ counts the number of nonzeros in $\mathcal{E}$. In Cor, $\|A\|_F^2 = \text{tr}(A^\top A)$ is the Frobenius norm of the matrix A. Prc is the ratio of edges in $\mathcal{E}^*$ that are recovered in $\mathcal{E}$. Rcl is the ratio of correctly recovered edges in $\mathcal{E}$. Acc indicates the ratio of the number of common edges shared by $\mathcal{E}$ and $\mathcal{E}^*$ against the total number of edges in them. Cor measures similarity between A and A^* by taking their values into consideration. All metrics are bounded between $[0, 1]$, and higher value indicates better result. To allow some noises on the prediction results appearing in a small range, we set the recovered adjacency matrix $\mathcal{E}$ to the binary indicator matrix $\boldsymbol{A}^\top \geq \epsilon$, i.e., $(\mathcal{E})_{i,j} = 1$ if $(\boldsymbol{A})_{ji} \geq 0.01$.

3.2.6 Comparison models

Discrete and continuous NMF models are both developed under the specific setting of CIC model to solve the multi-task learning problem defined in Problem 1. To the best of my knowledge, there is no any other methods could solve all the tasks in one model. Thus, we compare our NMF models with the state-of-the-art approach for each task separately.

InfluLearner [24] It is a state-of-the-art method that can estimate individual node infection probability directly from cascade data in the CIC setting as our method. InfluLearner draws a set of random binary features from certain distribution for each node j indicating the reachabilities of j by other nodes, and then uses a convex combination of random basis function to parameterize the conditional infection probability of the node given a source set over these binary vectors. To estimate the reachability distribution, InfluLearner calculates the mean frequency of node j being influenced by a source node s, average over all cascades in the training dataset with the source s.

It is worth noting that InfluLearner requires additionally the source identity for each infection to estimate the coverage functions. That is, InfluLearner also needs to know the original source node in the source set for each and every new infection occurred in the cascade in the training data. This additional information is provided in our simulated data in favor of InfluLearner. However, it is often unavailable in real-world applications such as epidemic spreads. The proposed NMF methods do not have such restriction.

Moreover, to quantify estimation error, we compute the MAE of node infection probability and influence at $t_l = l$ for $l = 1, \dots, 20$, and average each over the 50 test source sets. Since InfluLearner needs to learn the coverage function for a prescribed time t, we have to run it for each of the 20 time points one by one. In contrast, the proposed discrete and continuous NMF models are more advantageous since they can both directly estimate all of needed infection probabilities by one run, which is more computationally efficient. In particular, continuous NMF model can estimate the entire evolution of infection probabilities during $[0, T]$.

NetRate [35] It is a state-of-the-art algorithm that uncovers the network structure and transmission rates from cascade data. It is worth noting that NetRate requires the knowledge of the specific diffusion model (e.g., Exp or Ray), so that the transmission likelihood function can be explicitly expressed. Moreover, NetRate can only estimate $\boldsymbol{A}$ of diffusion networks, but not the influence. In contrast, NMF tackles both network inference and influence extimation simultaneously. In terms of computation efficiency, we observed that the implementation of NetRate provided in [35] runs very slowly for large networks. Therefore, we only perform comparisons on networks of size $n = 128$ in this experiment.

3.2.7 Experiment results

Training and testing data We first test NMF on a set of synthetic networks that mimic the structure of real-world diffusion network. For each type of network model, we generate 5 networks consisting of 128 nodes and 512 edges. We generate training data consists of K=10,000 cascades, which is formed by 10 sample cascades for each of 1,000 source sets (a source set is generated by randomly selecting 1 to 10 nodes from the network). Extra 100 source sets are generated for testing.

Experiment setting Among the few existing methods capable of learning infection probabilities of individual nodes directly from cascade data as ours, we adopt InfluLearner

[24] and a conditional LSTM (LSTM for short) [49] as baseline to compare. In our NMF implementation, we use a standard LSTM architecture and 3 dense layers for the RNN $\boldsymbol{\varepsilon}$ at each time t. Regularization terms using l_1-norm of all parameters are added to the loss function to promote their sparsity and robustness. Specifically, we use 0.001 to weight $\boldsymbol{A}$ and 0.0001 to all other trainable parameters, respectively. The NMF networks are trained and tested in TensorFlow [1] using Adam optimizer [53] with default parameters (lr=0.001, β_1=0.9, β_2=0.999, ϵ=1e-8). The LSTM model is trained and tested in the same setting as NMF except a fixed regularization weight 0.001 for all trainable parameters. We take $T = 20$ and train NMF and LSTM in 15000 steps. InfluLearner is trained in Matlab, and the number of features is set to 128. All experiments are performed on the same machine.

Comparison results on influence estimation To evaluate accuracy, we compute the mean absolute error (MAE) of node infection probability and influence over the 100 source sets for each time t. The results are given in Figure 3.2, which shows the mean (center line) and standard deviation (shade) of the three methods. NMF generally has lowest MAE, except at some early stages where InfluLearner is better. Note that InfluLearner requires and benefits from the knowledge of the original source node for each infection in the cascade (provided in our training data), which is often unavailable in practice and not needed in our method.

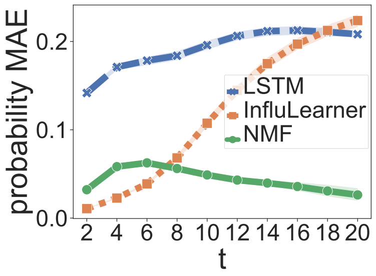
We also tested NMF on a real dataset [105] from Sina Weibo social platform consisting of more that 1.78 million users and 308 million following relationships among them. Following the setting in [24], we select the most popular tweet to generate diffusion cascades from the past 1,000 tweets of each user. Then we recreate the cascades by only keeping nodes of the top 1,000 frequency in the pooled node set over all cascades. For testing, we uniformly generate 100 source sets of size 10 and use $t = 1, 2, \ldots, 10$ as the time steps for observation. Finally, we test 100 source sets and compare our model NMF with the InfluLearner and LSTM. The MAE of all methods are shown in Figure 3.5a which shows that NMF significantly outperforms LSTM and is similar to InfluLearner. However, unlike InfluLearner that requires re-training for *every* t and is computationally expensive, NMF learns the evolution at all t in a single sweep of training and is tens of time faster.

Comparison results on network structure inference For comparison, we test NetRate [35] to the cascade data and learn $\boldsymbol{A}$ with Rayleigh distribution. Evaluation by four metrics are shown in Table 3.2, which indicates that NMF outperforms NetRate in all metrics. Note that NMF learns $\boldsymbol{A}$ along with the NMF dynamics for infection probability estimation in its training, whereas NetRate can only learn the matrix $\boldsymbol{A}$. We also show

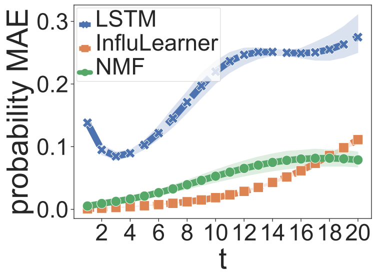
Table (3.2). Performance of structure inference using NetRate and the proposed discrete NMF on Random, Hierarchical, and Core-periphery networks with Rayleigh distribution as the diffusion time model on edges. Quality of the learned edge set $\mathcal{E}$ and distribution parameter $\boldsymbol{A}$ are measured by precision (Prc), recall (Rcl), accuracy (Acc), and correlation (Cor).

Network	Method	Prc	Rcl	Acc	Cor
Random	NetRate	0.481	0.399	0.434	0.465
	NMF	**0.858**	**0.954**	**0.903**	**0.950**
Hierarchical	NetRate	0.659	0.429	0.519	0.464
	NMF	**0.826**	**0.978**	**0.893**	**0.938**
Core-periphery	NetRate	0.150	0.220	0.178	0.143
	NMF	**0.709**	**0.865**	**0.779**	**0.931**

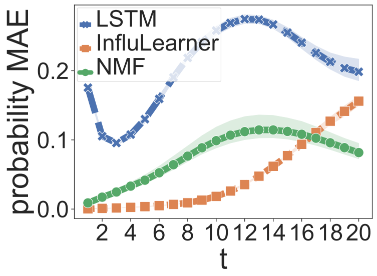
the visual appearance of $\boldsymbol{A}$ inferred by NMF in Figure 3.3. The ground truth $\boldsymbol{A}^*$ and $\boldsymbol{A}$ inferred by NetRate are also provided for comparison. As we can see, $\boldsymbol{A}$ inferred by NMF is much more faithful to $\boldsymbol{A}^*$ than that by NetRate. Note that NetRate requires knowledge of specific diffusion model type (Rayleigh in this test) whereas NMF does not. This result shows that NMF is versatile and robust when only cascade data are available.



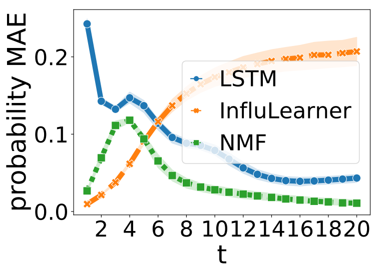
(a) Hier + Exp



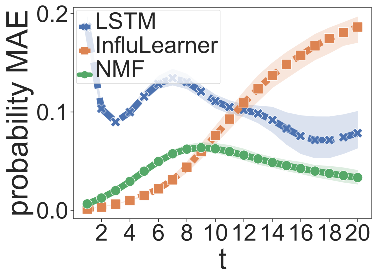
(b) Hier + Ray



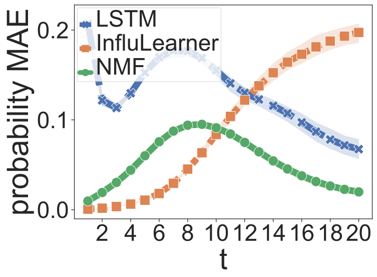
(c) Hier + Wbl



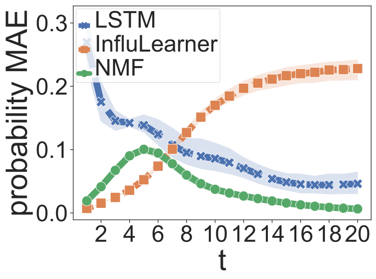
(d) Core + Exp



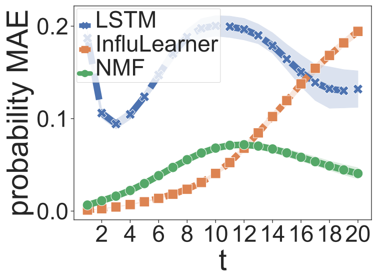
(e) Core + Ray



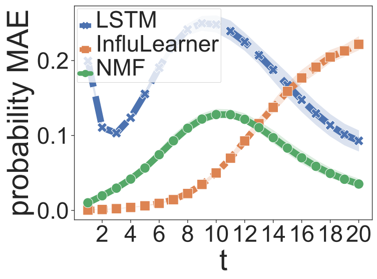
(f) Core + Wbl



(g) Rand + Exp

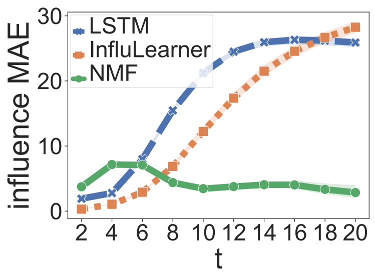


(h) Rand + Ray

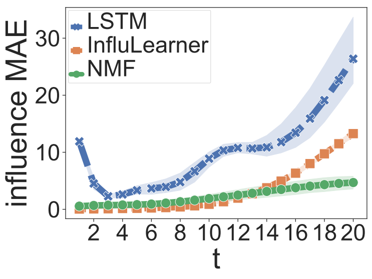


(i) Rand + Wbl

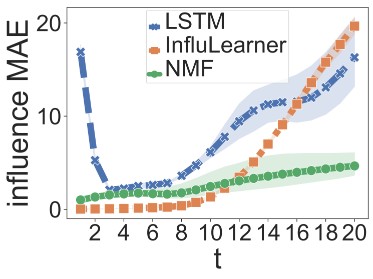
Figure (3.1). MAE of node infection probability by LSTM, InfluLearner, and discrete NMF on each of the 9 different combinations of Hierarchical (Hier), Core-periphery (Core) and Random (Rand) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 100 tests are shown.



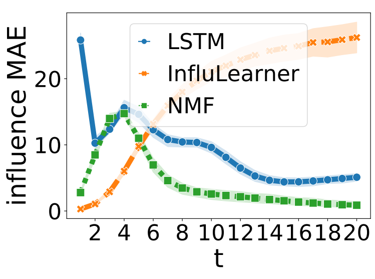
(a) Hier + Exp



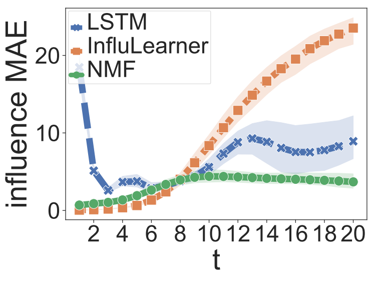
(b) Hier + Ray



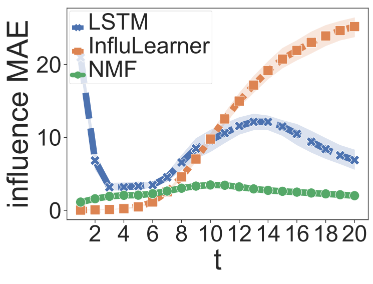
(c) Hier + Wbl



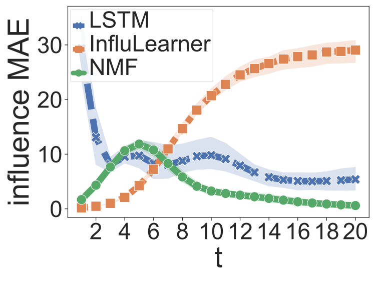
(d) Core + Exp



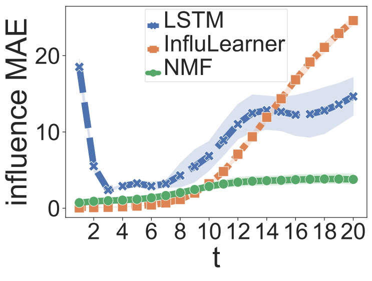
(e) Core + Ray



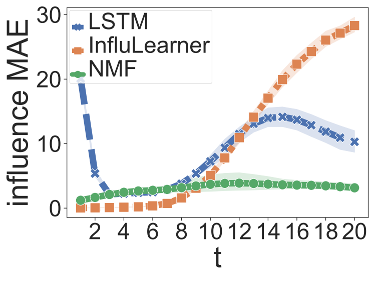
(f) Core + Wbl



(g) Rand + Exp

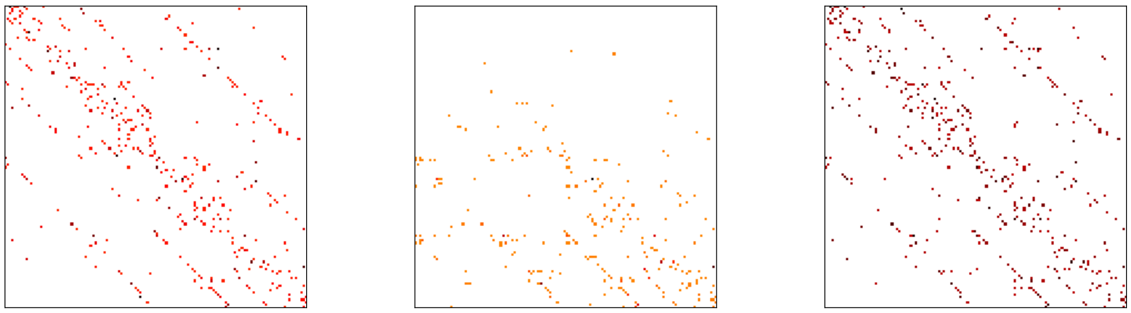


(h) Rand + Ray



(i) Rand + Wbl

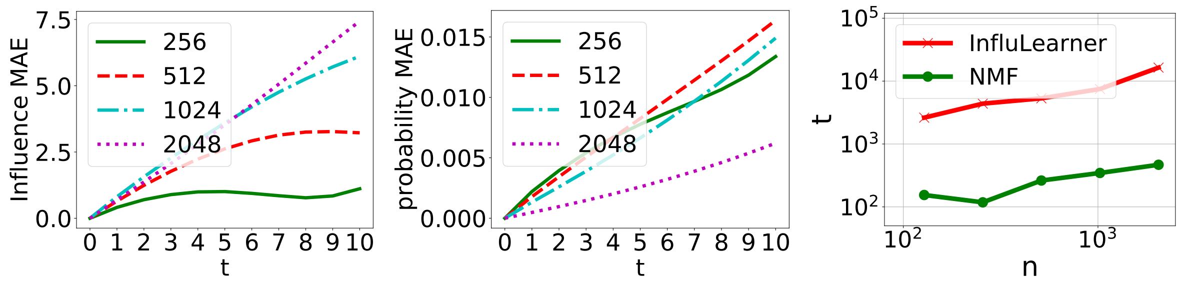
Figure (3.2). MAE of influence by LSTM, InfluLearner, and discrete NMF on each of the 9 different combinations of Hierarchical (Hier), Core-periphery (Core) and Random (Rand) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 100 tests are shown.



(a) True (b) NetRate (c) NMF

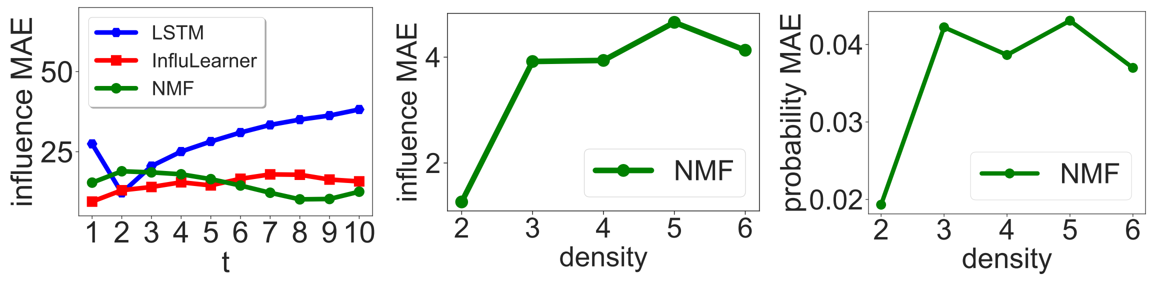
Figure (3.3). Ground truth $\boldsymbol{A}^*$ (left) and $\boldsymbol{A}$ inferred NetRate (middle) and discrete NMF (right) under the same color scale using cascaded data from a Hierarchical network with Rayleigh diffusion model.

Robustness and scalability We test the robustness of NMF for varying network density $|\mathcal{E}|/n$. The MAE of influence and infection probabilty by NMF on a hierarchical network with $n = 128$ are shown in Figure 3.5c and 3.5b, respectively. NMF remains accurate for denser networks, which can be notoriously difficult for other methods such as InfluLearner. We test NMF on networks of increasing sizes up to n=2,048 with $|\mathcal{E}| = 2n$ for each n using Hierarchical network and exponential diffusion model on cascade data containing 10,000 cascades. We also generate 100 extra cascades with 20%-validation and 80%-test. Figure 3.4 (a)–(b) shows the MAE of influence (Inf) and infection probability (Prob) estimated by NMF versus time for varying n, which indicate that the error remains low for large networks. We compare NMF to InfluLearner in terms of runtime for the influence estimation. For InfluLearner, we draw 200 features. For NMF, the batch size of training cascade data is set to 50 for the network with more than 2,048 nodes, and is 100 for smaller networks. The training is terminated when the average MAE of infection probability on validation data does not decrease for 20 epochs. Figure 3.4 (c) shows the comparison on runtime (in seconds) of training as we increase the network size n in InfluLearner and NMF. Note that the original implementation of InfluLearner [24] is in Matlab and the computational time increases drastically in network density, whereas our method retains similar runtime regardless of network density.



(a) Inf MAE vs t (b) Prob MAE vs t (c) Runtime vs # nodes

Figure (3.4). (a)–(b) MAE of influence (Inf) and infection probability (Prob) estimated by discrete NMF for Hierarchical networks with increasing network sizes from 256 to 2048. (c) runtime (in seconds) for training versus network sizes in log-log scale.



(a) Influ MAE vs t (b) Influ MAE vs density (c) Prob MAE vs density

Figure (3.5). (a) MAE of influence estimated by LSTM, InfluLearner on Weibo data; (b)–(c) MAE of influence and infection probability of discrete NMF for different network densities.

3.3 Continuous NMF algorithm

Based on Theorem 3.1.4, the continuous-time hidden state after an ε change in time can then be obtained.

Theorem 3.3.1. *The transformation of the system* (3.26) *with an* ε *change in time reduces to a continuous-depth residual neural network (ResNet) with hidden states formatted as* $\boldsymbol{x}(t+\varepsilon)$*:*

$$\boldsymbol{x}_{t+\varepsilon} = \boldsymbol{x}_t + \int_t^{t+\varepsilon} \boldsymbol{f}(\boldsymbol{x};\boldsymbol{A}) + \boldsymbol{\varepsilon}(\boldsymbol{x},\boldsymbol{h};\boldsymbol{\eta})ds \tag{3.43a}$$

$$\boldsymbol{h}_{t+\varepsilon} = \boldsymbol{h}_t + \int_t^{t+\varepsilon} \textstyle\sum_{l=1}^{L}(\boldsymbol{B}_l\boldsymbol{x} - \boldsymbol{C}_l\boldsymbol{h})ds \tag{3.43b}$$

with input $\boldsymbol{x}_0 = \boldsymbol{\chi}_{\mathcal{S}}$ *and* $\boldsymbol{h}_0 = \boldsymbol{0}$*.*

According to Theorem 3.3.1, by solving the system (3.43), we are able to solve Problem 1 by predicting the infection probability at arbitrary time $t \in [0,T]$ as the output of (3.43a), and uncovering the existence and strength of connection between edges by the trainable matrix $\boldsymbol{A}$ in (3.43a), which inspires us for a continuous-time setting of NMF for Problem 1. This setting enables the NMF model to estimate influence at any time $t \in [0,T]$, and also gives the possibility to use the most original cascade data which is not truncated by some time points in the training. However, the continuous-time setting is also a double-edged sword which leads to the following new challenges:

(C1) On loss function, how to design it for data in continuous-time setting which could keep the most of information of the original cascade data $\mathcal{D}$?

(C2) How to backward the continuous-depth neural network defined by (3.43)?

For (C1), we will introduce a specific loss function designed for the continuous-time NMF in subsection 3.3.2 by the special temporal point pattern shown in cascades. To overcome the challenge (C2), we employ the adjoint sensitivity method inspired by [10] to design the continuous backpropagation. More details are shown in the following subsections.

3.3.1 ODE system for continuous NMF

First of all, it is necessary to introduce a singe ordinary differential equation (ODE) system on $\boldsymbol{m} := [\boldsymbol{x};\boldsymbol{h}]$ derived from (3.26) or (3.43). As shown in Theorem 3.1.4, mean-field dynamics (3.24) reduces to a non-deday ODE system of $(\boldsymbol{x}, \boldsymbol{h})$ with (3.26a) and $\boldsymbol{h}' = \boldsymbol{B}\boldsymbol{x} - \boldsymbol{C}\boldsymbol{h}$ which implies one formulation of NNF in continuous-time setting as follows.

Definition 3.3.2 (Formulation of continuous NMF)**.** *Let $\boldsymbol{m}(t) = [\boldsymbol{x}(t); \boldsymbol{h}(t)] \in \mathbb{R}^{2n}$. For any $t \in [0, T]$, we consider the follow system for continuous NMF:*

$$\boldsymbol{m}'(t) = \boldsymbol{g}(\boldsymbol{m}(t); \boldsymbol{\theta}), \tag{3.44}$$

where $\boldsymbol{g}(\boldsymbol{m}; \boldsymbol{\theta}) = \begin{pmatrix} \boldsymbol{A}\boldsymbol{x} - \operatorname{diag}(\boldsymbol{x})\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\eta}) \\ \boldsymbol{B}\boldsymbol{x} - \boldsymbol{C}\boldsymbol{h} \end{pmatrix}$.

In Definition 3.3.2, $\boldsymbol{g}(\boldsymbol{m}; \boldsymbol{\theta})$ is the NMF dynamics derived in (3.26) or (3.43) with parameter $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{w}, \boldsymbol{\eta})$ and $\boldsymbol{w} = (\boldsymbol{B}, \boldsymbol{C})$. In particular, $\boldsymbol{\eta}$ stands for the network parameters of the deep neural network $\boldsymbol{\varepsilon}$ that wraps the augmented state $\boldsymbol{m}$ to approximate the MZ memory term (3.12).

In the following subsection, we establish a direction connection between mathematical optimal control and NODE, and provide a rigorous proof that NODE exactly evaluates the gradient of the target payoff function (likelihood function in our case) during optimization from the optimal control point of view. Compared to [10], our proof is based on calculus of variation which is more mathematically rigorous. Moreover, we show how to incorporate the running payoff (or loss) function at scattered observation times through a rigorous derivation, as needed in continuous-time NMF training.

3.3.2 Evolutionary point processes and loss function

In this subsection, we will introduce the loss function designed for continuous-time cascade data. Given a sample cascade with time log $\hat{\boldsymbol{\tau}} = \{t_i : i \in [n], 0 \le t_i \le T \text{ or } t_i = \infty\}$ from $\mathcal{D}$, where $[0, T]$ is the observation time window and t_i is the time node i is infected. Clearly, a cascade shows a temporal point pattern with the identities of infected nodes as marks. It can be defined as an evolutionary point process [44] since the propagation only depends on the preciously infected nodes. Recall from Section 3.1.2 that $\boldsymbol{X}(t)$ is the jump stochastic process describing the infection state of the nodes and $\boldsymbol{x}(t)$ is the infection probabilities. Therefore, $\boldsymbol{x}'(t)$ is essentially the (non-conditional) intensity of $\boldsymbol{X}(t)$. In other words, $\boldsymbol{X}(t)$ is identical to a non-homogeneous Poisson process with intensity function $\boldsymbol{x}'(t)$ for t almost everywhere. Due to the relation between the intensity function and the likelihood function of a point process [44], the log-likelihood function of the process is given by

Theorem 3.3.3. *Given a cascade $(\mathcal{S}, \boldsymbol{\tau})$ with $\boldsymbol{\tau} = \{t_i : i \in [n], 0 \le t_i \le T$ or $t_i = \infty\}$ from $\mathcal{D}$ on an observation interval $[0, T]$, the log-likelihood function is given by*

$$LL(\boldsymbol{x}) = \sum_{i\in[n] \ \ and \ t_i \in \boldsymbol{\tau}\cap(0,\infty)} \log x_i'(t_i) - \sum_{i=1}^{n} x_i(T)$$

where $\boldsymbol{x}(t; \boldsymbol{\chi}_{\mathcal{S}})$ is the node infection probability function.

Proof. Suppose that $X_i(t; \boldsymbol{\chi}_{\mathcal{S}})$ jumps from 0 to 1 for some node i and time t, i.e. $t_i = t$. Then let $f(t, i|\mathcal{H}_{t^-})$ be the conditional density function of (t, i) of the next jump given the history $\mathcal{H}_{t^-} = \{X_i(s; \boldsymbol{\chi}_{\mathcal{S}}) : s < t, i \in [n]\}$ and $F(t, i\,\big|\mathcal{H}_{t^-})$ be its corresponding cumulative distribution function. Denote $f(t, \cdot|\mathcal{H}_{t^-}) := \sum_{i=1}^{n} f(t, i|\mathcal{H}_{t^-})$ and $F(t, \cdot|\mathcal{H}_{t^-}) := \sum_{i=1}^{n} F(t, i|\mathcal{H}_{t^-})$. Define the conditional intensity functions by $\lambda^*(t, \cdot) = \frac{f(t,\cdot|\mathcal{H}_{t^-})}{1-F(t,\cdot|\mathcal{H}_{t^-})}$ and $\lambda^*(t, i) = \lambda^*(t, \cdot) f(i|t, \mathcal{H}_{t^-})$. Thus, $\lambda^*(t, i) = \frac{f(t,i|\mathcal{H}_{t^-})}{1-F(t,\cdot|\mathcal{H}_{t^-})}$. Assume that before t, the last jump happens at time $t^* > 0$. Then $\mathcal{H}_{t^-} = \mathcal{H}_{t^*}$. In particular, if $\mathcal{H}_{t^-} = \mathcal{H}_0 = \boldsymbol{\chi}_{\mathcal{S}}$, then set $t^* = 0$. In both cases, we have

$$\lambda^*(t, \cdot) = \frac{f(t, \cdot|\mathcal{H}_{t^*})}{1 - F(t, \cdot|\mathcal{H}_{t^*})} = -\frac{d}{dt} \log(1 - F(t, \cdot|\mathcal{H}_{t^*})).$$

Thus, $\int_{t^*}^{t} \lambda^*(s, \cdot) ds = -\log(1 - F(t, \cdot|\mathcal{H}_{t^*}))$ since $F(t^*, \cdot|\mathcal{H}_{t^*}) = 0$. Therefore, $F(t, \cdot|\mathcal{H}_{t^*}) = 1 - \exp\left(-\int_{t^*}^{t} \lambda^*(s, \cdot) ds\right)$ and so

$$f(t, \cdot|\mathcal{H}_{t^*}) = \lambda^*(t, \cdot) \exp\left(-\int_{t^*}^{t} \lambda^*(s, \cdot) ds\right).$$

Without loss of generality, we may assume that $t_i \in (0, \infty)$ when $i \in [n_0]$, and $t_i = 0$ or

∞ otherwise. Moreover, assume $t_0 := 0 < t_1 < t_2 < \cdots < t_{n_0}$. Then the likelihood function is

$$L = f(t_1, 1|\mathcal{H}_{t_0})f(t_2, 2|\mathcal{H}_{t_1}) \cdots f(t_{n_0}, n_0|\mathcal{H}_{t_{n_0-1}})(1 - F(T, \cdot|\mathcal{H}_{t_{n_0}}))$$

where the last term appears since there is no new node infected in the time interval $(t_{n_0}, T]$ when $t_{n_0} < T$.

Therefore, we get that

$$\begin{aligned}
L &= \left(\prod_{i=1}^{n_0} f(t_i, i|\mathcal{H}_{t_{i-1}})\right) \frac{f(T, \cdot|\mathcal{H}_{t_{n_0}})}{\lambda^*(T, \cdot)} \\
&= \left(\prod_{i=1}^{n_0} f(t_i, \cdot|\mathcal{H}_{t_{i-1}}) \cdot f(i|t_i, \mathcal{H}_{t_{i-1}})\right) \frac{f(T, \cdot|\mathcal{H}_{t_{n_0}})}{\lambda^*(T, \cdot)} \\
&= \left(\prod_{i=1}^{n_0} f(i|t_i, \mathcal{H}_{t_{i-1}})\lambda^*(t_i, \cdot) \exp\left(-\int_{t_{i-1}}^{t_i} \lambda^*(s, \cdot)ds\right)\right) \exp\left(-\int_{t_{n_0}}^{T} \lambda^*(s, \cdot)ds\right) \\
&= \left(\prod_{i=1}^{n_0} \lambda^*(t_i, i)\right) \exp\left(-\int_0^T \lambda^*(s, \cdot)ds\right) \\
&= \left(\prod_{i=1}^{n_0} \lambda^*(t_i, i)\right) \exp\left(-\int_0^T \sum_{i=1}^{n} \lambda^*(s, i)ds\right).
\end{aligned}$$

Since this is an inhomogeneous Poisson process by the assumption, $\lambda^*(t_i, i) = \lambda_i(t_i) = x_i'(t_i)$ by the proof of Theorem 3.1.2. Thus,

$$L = \prod_{i=1}^{n_0} x_i'(t_i) \exp\left(-\sum_{i=1}^{n} x_i(T)\right)$$

which establishes the log-likelihood we want to prove. □

According to Theorem 3.3.3, the negative log-likelihood function of $\boldsymbol{x}'(t)$ given the cascade $\mathcal{C} = (\mathcal{S}, \boldsymbol{\tau})$ can be easily obtained, and it is also the "loss function" ℓ we need to minimize:

$$\ell(\boldsymbol{x};\mathcal{C}) = \sum_{i\in[n] \text{ and } t_i\in\boldsymbol{\tau}\cap(0,\infty)} (-\log x_i'(t_i)) + \sum_{i=1}^{n} x_i(T). \tag{3.45}$$

It indicates that the loss is measured by the node states at the terminal time and take an addition for time $t \in [0, T]$ only if there is one node gets infected at that moment. To analysis the running loss of the process, we rewrite the summation in (3.47) as an integration over the continuous time interval $[0, T]$ by employing the Dirac delta function $\delta(\cdot)$:

Definition 3.3.4 (Loss function)**.** *Given a cascade* $(\mathcal{S}, \boldsymbol{\tau})$ *with* $\boldsymbol{\tau} = \{t_i : i \in [n], 0 \le t_i \le T \text{ or } t_i = \infty\}$ *from* $\mathcal{D}$ *on an observation interval* $[0, T]$*, the negative log-likelihood function of* $\boldsymbol{x}'(t)$ *is*

$$\ell(\boldsymbol{x};\mathcal{C}) = \int_0^T r(\boldsymbol{x}(t), \boldsymbol{\theta})\,\mathrm{d}t + \mathbf{1}^\top \boldsymbol{x}(T), \tag{3.46}$$

where the running loss function is defined by

$$r(\boldsymbol{x}(t), \boldsymbol{\theta}) = \sum_{i=1}^{n} -\delta(t - t_i)\log x_i'(t) = \sum_{i=1}^{n} -\delta(t - t_i)\log(\tilde{\boldsymbol{f}}(\boldsymbol{x}, \boldsymbol{h}; \boldsymbol{\theta}))_i, \tag{3.47}$$

and $\delta(\cdot)$ *is the Dirac delta function.*

The running loss takes into account the changes of $\boldsymbol{x}(t)$ at intermediate times during $[0, T]$. We can further add regularization or incorporate prior information to (3.46). More implemented suggestions are listed as follows.

- **Control on support.** If $\mathcal{E}$ is given, we know that $\boldsymbol{A}$ must be supported on $\mathcal{E}$, which serves as the constraint of $\boldsymbol{A}$.

- **Control on sparsity.** If we know the network has low density (sparse edges), then we can enforce a sparsity regularization such as $\|\boldsymbol{A}\|_1$ (the l_1 norm of the vectorized $\boldsymbol{A} \in \mathbb{R}^{n^2}$).

- **Learning of $\boldsymbol{A}$.** In general, $\boldsymbol{A}$ can be interpreted as the convolution to be learned from a graph convolution network (GCN) [54, 101]. The support and magnitude of

$\boldsymbol{A}$ implies the network structure and strength of interaction between pairs of nodes, respectively.

We will provide more details of our choice of regularization and its parameter setting in subsection 3.3.6.

3.3.3 Optimal control formulation of parameter learning

In this subsection, we will formulate the problem to minimize the loss function in Definition 3.3.4 as an optimal control problem which enables us to get theoretical support from the corresponding control theories.

To summarize, we give the following formulation on the control problem.

Definition 3.3.5 (Optimal control on $\boldsymbol{\theta}$ for continuous NMF)**.** *The optimal parameter $\boldsymbol{\theta}$ of* (3.24) *can be obtained by minimizing the loss function in* (3.46) *for the given cascade $\mathcal{C}$:*

$$\min_{\boldsymbol{\theta}} \quad \ell(\boldsymbol{\theta};\mathcal{C}) := \int_0^T r(\boldsymbol{x}(t),\boldsymbol{\theta})\,\mathrm{d}t + \mathbf{1}^\top \boldsymbol{x}(T), \tag{3.48a}$$

$$\text{s.t.} \quad \boldsymbol{m}'(t) = \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta}), \quad \boldsymbol{m}(0) = [\boldsymbol{\chi}_{\mathcal{S}_k};\mathbf{0}], \quad t\in[0,T], \tag{3.48b}$$

where $\boldsymbol{m}(t) = [\boldsymbol{x}(t);\boldsymbol{h}(t)] \in \mathbb{R}^{2n}$ and

$$\boldsymbol{g}(\boldsymbol{m};\boldsymbol{\theta}) = \begin{pmatrix} \boldsymbol{A}\boldsymbol{x} - \operatorname{diag}(\boldsymbol{x})\boldsymbol{A}\boldsymbol{x} + \boldsymbol{\varepsilon}(\boldsymbol{x},\boldsymbol{h};\boldsymbol{\eta}) \\ \boldsymbol{B}\boldsymbol{x} - \boldsymbol{C}\boldsymbol{h} \end{pmatrix}. \tag{3.49}$$

This is the parameter training problem given one cascade $\mathcal{C}$, and can be trivially extended to the case $\mathcal{D}$ which consists of K cascades. In what follows, we drop the notation $\mathcal{C}$ for conciseness.

Remark 3.3.6. *Solving such a system for the optimal parameter $\boldsymbol{\theta} = (\boldsymbol{A},\boldsymbol{w},\boldsymbol{\eta})$ has been cast as the so-called neural ODE (NODE) in [10]. Note that*

- *The so-called control variable $\boldsymbol{\theta}$ is constant and time-invariant in NODE [10] as well* *as in NMF. Therefore, it is considerably easier to handle than that in classical optimal control* $\boldsymbol{\theta}$*.*

Motivated by Remark 3.3.6, in subsection 3.3.4, we will revisit the neural ODE (NODE) [10]. In subsection 3.3.5, we will develop an algorithm for solving $\boldsymbol{\theta}$ in (3.48) inspired by NODE, which is easy to implement. Moreover, we will derive a rigorous proof of the relation between the gradient of the loss function and the solution of the augmented dynamics.

3.3.4 Revisiting neural ODE (NODE)

Consider a sequence of simple transformations to a hidden state

$$\boldsymbol{h}(t+1) = \boldsymbol{h}(t) + \boldsymbol{f}(\boldsymbol{h}(t), \boldsymbol{\theta}(t)). \tag{3.50}$$

where $t = 0, 1, \cdots, T$ and $\boldsymbol{h}(t)$ is the state vector at time t. Then (3.50) is the Euler solution at the discretized grid of n time steps with normalized size 1 on the ordinary differential equation (ODE) system with dynamic

$$\boldsymbol{h}'(t) = \boldsymbol{f}(\boldsymbol{h}(t), \boldsymbol{\theta}(t)) \tag{3.51}$$

and initial value h_0. In other word, we can see (3.50) as Euler discretization of a continuous transformation [45,66,88]. In the view of limit over the finite number of layers, Neural ODE parameterize the continuous dynamic of hidden units as form

$$\boldsymbol{h}'(t) = \boldsymbol{f}(\boldsymbol{h}(t), t, \boldsymbol{\theta}),$$

which could be specified by a neural network. More specifically, the main ideas of NODE is that

- In stead of iterative update the state vector to get $h(T)$ like residual or recurrent neural network, NODE first learns the ODE (3.51) by a neural network.

- Then solve the learned ODE model for $h(T)$ by using an efficient and accurate ODE solvers [28, 55, 87].

Since the dynamics are defined in continuous time, it does not require discretizing observation. Data could includes point process with temporal events happen at any time.

The main technical difficulty in training an NODE model is performing reverse-mode differentiation (backpropagation) through one ODE solver. The normal backward pass works for the NODE models, but it causes a high memory cost and bigger numerical error. To overcome this challenge, [10] also proposed a reverse-mode automatic differentiation of ODE solutions using the adjoint method [82]. The main theorem is that

Theorem 3.3.7. *[10] Let $\boldsymbol{x}(t)$ follow the differential equation*

$$\boldsymbol{x}'(t) = f(\boldsymbol{x}(t), t, \boldsymbol{\theta}),$$

where $\boldsymbol{\theta}$ are the parameters and let the loss function be $L(\boldsymbol{x}(t))$. If we define an adjoint state

$$\boldsymbol{a}(t) = \nabla_{\boldsymbol{x}} L(\boldsymbol{x}(t)),$$

then it follows the differential equation

$$\boldsymbol{a}'(t) = -\boldsymbol{a}(t) \nabla_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}, t, \boldsymbol{\theta}).$$

It leaves some rooms for us to improve:

- The proof of Theorem 3.3.7 is not based on the Pontryagin maximum principle (PMP) or any control theory. It works on

$$\boldsymbol{a}'(t) = \lim_{\varepsilon \to 0^+} \frac{\boldsymbol{a}(t+\varepsilon) - \boldsymbol{a}(t)}{\varepsilon}$$

by substituting $\boldsymbol{a}'(t)$ by

$$\boldsymbol{a}(t) = \boldsymbol{a}(t+\varepsilon) \frac{\mathrm{d}\boldsymbol{x}(t+\varepsilon)}{\mathrm{d}\boldsymbol{x}(t)}$$

and employing the first order approximation of $\boldsymbol{x}(t+\varepsilon)$:

$$\boldsymbol{x}(t+\varepsilon) = \int_{t}^{t+\varepsilon} \boldsymbol{f}(\boldsymbol{x}(t), t, \boldsymbol{\theta})\,\mathrm{d}t + \boldsymbol{x}(t) = \boldsymbol{x}(t) + \varepsilon \boldsymbol{f}(\boldsymbol{x}(t), t, \boldsymbol{\theta}) + O(\varepsilon^2).$$

- By checking the maximum principle, we can tell the adjoint method developed for NODE is only for an optimization problem on a loss function with independency on intermediate time points.

- When the loss depends on intermediate states, the reverse-mode derivative of NODE is broken into a sequence of separate solves over each consecutive pair of output times $[t_i, t_{i+1}]$. And the adjoint is adjusted by adding $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}(t_i))$. However, the theory of this part is missing.

In our research, we addressed all these issues for a completed theory system.

Many literature are inspired by the NODE framework on the continuous deep learning [104] in the research of neural network architecture design [3, 22, 43, 80, 81, 86] and generative modeling [42, 102].

3.3.5 Backpropagation in continuous NMF

As we can see, to find the optimal $\boldsymbol{\theta}$ of (3.48), the key is to compute $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$ for any $\boldsymbol{\theta}$. To this end, we recall that the *Hamiltonian* function associated with the control problem (3.48) is

$$H(\boldsymbol{m}(t), \boldsymbol{p}(t); \boldsymbol{\theta}) = \boldsymbol{p}(t) \cdot \boldsymbol{g}(\boldsymbol{m}(t); \boldsymbol{\theta}) + r(\boldsymbol{m}(t), \boldsymbol{\theta}), \tag{3.52}$$

where $\boldsymbol{p}(t) \in \mathbb{R}^{2n}$ is the co-state variable (also known as the adjoint variable) associated with $\boldsymbol{m}(t)$. Here, $\boldsymbol{p}(t)$ plays the role of Lagrange multiplier for the ODE constraint (3.48b).

Dynamics of co-state The standard optimal control theory states that the co-state $\boldsymbol{p}(t)$ follows the ODE backward in time as follows:

Definition 3.3.8 (Backward ODE on co-state $\boldsymbol{p}(t)$)**.**

$$\begin{cases} \boldsymbol{p}'(t) = -\nabla_{\boldsymbol{m}} \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})\boldsymbol{p}(t) - \nabla_{\boldsymbol{m}} r(\boldsymbol{m}(t),\boldsymbol{\theta}), \qquad T \geq t \geq 0, \\ \boldsymbol{p}(T) = [\mathbf{1};\mathbf{0}]. \end{cases} \tag{3.53}$$

The terminal condition $\boldsymbol{p}(T) = [\mathbf{1};\mathbf{0}]$ has this simple form because the "terminal loss" in (3.48) is given by $[\mathbf{1};\mathbf{0}] \cdot \boldsymbol{m}(T) = \mathbf{1} \cdot \boldsymbol{x}(T)$.

Optimal theory To show that $\nabla_{\boldsymbol{\theta}} \ell$ can be obtained by solving the ODE (3.48b) forward in time and an augmented ODE backward in time, we prove the following theorem.

Theorem 3.3.9. *The gradient $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$ of the loss function ℓ defined in* (3.48) *for any parameter $\boldsymbol{\theta}$ and cascade data $\mathcal{C}$ is given by*

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}) = \int_0^T \Big(\nabla_{\boldsymbol{\theta}} \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})\boldsymbol{p}(t) + \nabla_{\boldsymbol{\theta}} r(\boldsymbol{m}(t),\boldsymbol{\theta}) \Big) \,\mathrm{d}t. \tag{3.54}$$

Moreover, if $\boldsymbol{m}^*$ is the solution of (3.48b) *using the optimal solution $\boldsymbol{\theta}^*$ to* (3.48)*, and $\boldsymbol{p}^*$ is the co-state determined by* (3.53) *with $\boldsymbol{m}^*$ and $\boldsymbol{\theta}^*$, then*

$$\int_0^T \nabla_{\boldsymbol{\theta}} H(\boldsymbol{m}^*(t), \boldsymbol{p}^*(t); \boldsymbol{\theta}) \,\mathrm{d}t = \mathbf{0}.$$

Proof. Let $\boldsymbol{\zeta} \in \mathbb{R}^m$ and $\varepsilon \geq 0$ be arbitrary. Consider the variation of any control $\boldsymbol{\theta}$ given by $\boldsymbol{\theta}_\varepsilon := \boldsymbol{\theta} + \varepsilon\boldsymbol{\zeta}$ and denote $\boldsymbol{m}_\varepsilon(t)$ the state process following (3.48b) with $\boldsymbol{\theta}_\varepsilon$. Then we have

$$\boldsymbol{m}_\varepsilon(t) = \boldsymbol{m}(t) + \varepsilon \boldsymbol{y}(t) + o(\varepsilon), \qquad 0 \leq t \leq T,$$

where the first-order perturbation $\boldsymbol{y}(t)$ satisfies

$$\begin{cases} \boldsymbol{y}'(t) = \nabla_{\boldsymbol{m}} \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})\boldsymbol{y}(t) + \nabla_{\boldsymbol{\theta}} \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})\boldsymbol{\zeta}, \qquad 0 \leq t \leq T, \\ \boldsymbol{y}(0) = \mathbf{0}. \end{cases}$$

Therefore, the directional derivative of ℓ defined in (3.48a) at $\boldsymbol{\theta}$ along the direction $\boldsymbol{\zeta}$ is

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\ell(\boldsymbol{\theta}_\varepsilon)\Big|_{\varepsilon=0} = \int_0^T \Big(\nabla_{\boldsymbol{m}} r(\boldsymbol{m}_t, \boldsymbol{\theta})\boldsymbol{y}(t) + \nabla_{\boldsymbol{\theta}} r(\boldsymbol{m}(t), \boldsymbol{\theta})\boldsymbol{\zeta}\Big)\,\mathrm{d}t + \boldsymbol{p}(T)\boldsymbol{y}(T). \tag{3.55}$$

On the other hand, we have

$$\begin{aligned}(\boldsymbol{p}\cdot\boldsymbol{y})' = \boldsymbol{p}'\cdot\boldsymbol{y} + \boldsymbol{p}\cdot\boldsymbol{y}' &= -\Big(\nabla_{\boldsymbol{m}}\boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})\boldsymbol{p}(t) + \nabla_{\boldsymbol{m}} r(\boldsymbol{m}(t), \boldsymbol{\theta})\Big)\cdot\boldsymbol{y} \\ &\quad + \boldsymbol{p}\cdot\Big(\nabla_{\boldsymbol{m}}\boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})^\top\boldsymbol{y}(t) + \nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})^\top\boldsymbol{\zeta}\Big) \\ &= -\nabla_{\boldsymbol{m}} r(\boldsymbol{m}(t), \boldsymbol{\theta})^\top\boldsymbol{y}(t) + \boldsymbol{p}(t)^\top\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})\boldsymbol{\zeta}.\end{aligned}$$

Since $\boldsymbol{y}(0) = \mathbf{0}$, we know

$$\boldsymbol{p}(T)\cdot\boldsymbol{y}(T) = \int_0^T \Big(-\nabla_{\boldsymbol{m}} r(\boldsymbol{\theta}, \boldsymbol{m}(t))^\top\boldsymbol{y}(t) + \boldsymbol{p}(t)^\top\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})\boldsymbol{\zeta}\Big)\,\mathrm{d}t. \tag{3.56}$$

Substituting (3.56) into (3.55) yields

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\ell(\boldsymbol{\theta}_\varepsilon)\Big|_{\varepsilon=0} = \int_0^T \Big(\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})^\top\boldsymbol{p}(t) + \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}, \boldsymbol{m}(t))\Big)\,\mathrm{d}t\cdot\boldsymbol{\zeta}.$$

As $\boldsymbol{\zeta}$ is arbitrary, we know that the gradient $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta})$ is as claimed in (3.54).

Note that the integrand in (3.54) is

$$\Big(\nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}, \boldsymbol{m}(t)) + \nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta})^\top\boldsymbol{p}(t)\Big) = \nabla_{\boldsymbol{\theta}} H(\boldsymbol{m}(t), \boldsymbol{p}(t);\boldsymbol{\theta}).$$

Hence, at the optimal $\boldsymbol{\theta}^*$ of ℓ, we have

$$\frac{\mathrm{d}}{\mathrm{d}\varepsilon}\ell(\boldsymbol{\theta}^*_\varepsilon)\Big|_{\varepsilon=0} = \Big(\int_0^T \nabla_{\boldsymbol{\theta}} H(\boldsymbol{m}^*(t), \boldsymbol{p}^*(t);\boldsymbol{\theta}^*)\,\mathrm{d}t\Big)\cdot\boldsymbol{\zeta} \ge 0, \tag{3.57}$$

for all $\boldsymbol{\zeta}\in\mathbb{R}^m$, from which we readily deduce the identity regarding H at $\boldsymbol{\theta}^*$. $\square$

Backward ODE for the gradients The formula (3.54) in Theorem 3.3.9 suggests that we can compute $\nabla_{\boldsymbol{\theta}}\ell$ by tracking an auxiliary variable $\boldsymbol{q}$ that follows the following backward differential equation and terminal condition.

Definition 3.3.10 (Backward ODE for $\nabla_{\boldsymbol{\theta}}\ell$)**.**

$$\begin{cases} \boldsymbol{q}'(t) = -\nabla_{\boldsymbol{\theta}}\boldsymbol{g}(\boldsymbol{m}(t), \boldsymbol{\theta})^\top \boldsymbol{p}(t) - \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}, \boldsymbol{m}(t)), \qquad T \geq t \geq 0, \\ \boldsymbol{q}(T) = \boldsymbol{0}. \end{cases} \tag{3.58}$$

The gradients $\nabla_{\boldsymbol{\theta}}\ell$ is the initial value of the solution on the system (3.58). That is,

$$\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}) = \boldsymbol{q}(T) - \int_0^T \boldsymbol{q}'(t)\,\mathrm{d}t = \boldsymbol{q}(0).$$

Integration on running loss To solve the ODE system (3.58) for the gradients, we need to clarify one implementation issue with running loss r: the second term of the dynamic (3.58) could not be solved by any numerical ODE solver. Same issue appears for solving the backward ODE dynamic (3.53).

Suppose that the infection times in the cascade $\mathcal{C}$ can be sorted $0 < t^{(1)} < t^{(2)} < \cdots < t^{(m)} < t^{(m+1)} := T$. That is, there are m infections (excluding the infections at the source nodes) during the cascade $\mathcal{C}$. (Note that any two infection times coincide with probability 0 since the point process is simple.) For notation simplicity, suppose that at time $t^{(i)}$, the new infected node is i.

Theorem 3.3.11. *The integral of the running loss in* (3.58) *reduces to*

$$\int_0^T \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}, \boldsymbol{m})\,\mathrm{d}t = \sum_{i=0}^{m} \nabla_{\boldsymbol{\theta}} \left(-\log \boldsymbol{g}_i(\boldsymbol{m}(t^{(i)}); \boldsymbol{\theta}) \right), \tag{3.59}$$

where $\boldsymbol{g}_i(\boldsymbol{m}(t), \boldsymbol{\theta})$ is the ith component of $\boldsymbol{g}(\boldsymbol{m}(t), \boldsymbol{\theta})$.

Proof. Let $\boldsymbol{g}_i(\boldsymbol{m}(t), \boldsymbol{\theta})$ is the ith component of $\boldsymbol{g}(\boldsymbol{m}(t), \boldsymbol{\theta})$. Since $\boldsymbol{g}_i(\boldsymbol{m}(t), \boldsymbol{\theta})$ is only defined in $[0, T]$, we could suppose that $\boldsymbol{g}_i(\boldsymbol{m}(t), \boldsymbol{\theta}) = 0$ for any $t \notin [0, T]$. By employing the property of the Dirac delta function $\delta(\cdot)$, we obtain that

$$\begin{aligned}\int_0^T \nabla_{\boldsymbol{\theta}} r(\boldsymbol{\theta}, \boldsymbol{m})\,\mathrm{d}t &= \int_{i=0}^T \nabla_{\boldsymbol{\theta}} \sum_{i=0}^m \left(-\delta(t - t^{(i)}) \log \boldsymbol{g}_i(\boldsymbol{m}(t); \boldsymbol{\theta})\right) \mathrm{d}t \\ &= \sum_{i=0}^m \left(\nabla_{\boldsymbol{\theta}} \int_0^T \left(-\delta(t - t^{(i)}) \log \boldsymbol{g}_i(\boldsymbol{m}(t); \boldsymbol{\theta})\right) \mathrm{d}t\right) \\ &= \sum_{i=0}^m \left(\nabla_{\boldsymbol{\theta}} \int_0^\infty \left(-\delta(t - t^{(i)}) \log \boldsymbol{g}_i(\boldsymbol{m}(t); \boldsymbol{\theta})\right) \mathrm{d}t\right) \\ &= \sum_{i=0}^m \nabla_{\boldsymbol{\theta}} \left(-\log \boldsymbol{g}_i(\boldsymbol{m}(t^{(i)}); \boldsymbol{\theta})\right).\end{aligned}$$

This completes the proof. □

Similarly, we could also get the following theorem.

Theorem 3.3.12. *The integral of the running loss in* (3.53) *reduces to*

$$\int_0^T \nabla_{\boldsymbol{m}} r(\boldsymbol{\theta}, \boldsymbol{m})\,\mathrm{d}t = \sum_{i=0}^m \nabla_{\boldsymbol{m}} \left(-\log \boldsymbol{g}_i(\boldsymbol{m}(t^{(i)}); \boldsymbol{\theta})\right), \tag{3.60}$$

where $\boldsymbol{g}_i(\boldsymbol{m}(t), \boldsymbol{\theta})$ is the ith component of $\boldsymbol{g}(\boldsymbol{m}(t), \boldsymbol{\theta})$.

Calculation on the gradients Based on Theorem 3.3.11, we could compute $\boldsymbol{q}(0)$ by solving the ODE (3.58) backward in each time interval as

$$\boldsymbol{q}(t^{(i-1)}) = \boldsymbol{q}(t^{(i)}) - \int_{t^{(i)}}^{t^{(i-1)}} \nabla_{\boldsymbol{\theta}} \boldsymbol{g}(\boldsymbol{m}(t), \boldsymbol{\theta})^\top \boldsymbol{p}(t)\,\mathrm{d}t - \nabla_{\boldsymbol{\theta}} \log \boldsymbol{g}_{i-1}(\boldsymbol{m}(t^{(i-1)}); \boldsymbol{\theta}). \tag{3.61}$$

which indicates the needs of the knowledge on the co-state $\boldsymbol{p}(t)$ and state $\boldsymbol{m}(t)$ along the entire trajectory of (3.58). Fortunately, in a similar behavior on the backward ODE (3.53), we have

$$\boldsymbol{p}(t^{(i-1)}) = \boldsymbol{p}(t^{(i)}) - \int_{t^{(i)}}^{t^{(i-1)}} \nabla_{\boldsymbol{m}} \boldsymbol{g}(\boldsymbol{m}(t), \boldsymbol{\theta})^\top \boldsymbol{p}(t)\,\mathrm{d}t - \nabla_{\boldsymbol{m}} \log \boldsymbol{g}_{i-1}(\boldsymbol{m}(t^{(i-1)}); \boldsymbol{\theta}). \tag{3.62}$$

The ODE of $\boldsymbol{m}(t)$ remains the same as in (3.48b) since it does not involve the running loss r. The computation of $\boldsymbol{m}$ could also be put in a similar backward process in time as (3.61) and (3.62) starting from $\boldsymbol{m}(T)$ which can be calculated in a forward step as follows:

$$\boldsymbol{m}(t^{(i-1)}) = \boldsymbol{m}(t^{(i)}) - \int_{t^{(i)}}^{t^{(i-1)}} \boldsymbol{g}(\boldsymbol{m}(t), \boldsymbol{\theta})\,\mathrm{d}t. \tag{3.63}$$

To summarize, we can adopt the following approach to compute the gradients $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta})$ for any given $\boldsymbol{\theta}$:

1. Solve the ODE (3.48b) of $\boldsymbol{m}(t)$ forward in time from 0 to T.
2. Solve the ODE system (3.48b), (3.53), and (3.58) of $(\boldsymbol{m}(t), \boldsymbol{p}(t), \boldsymbol{q}(t))$ backward in time from T to 0 by employing the backward propagation in (3.63), (3.62), and (3.61) over the sequence of reversed time subintervals of $[0, T]$. In particular, the key points for implementation are:
   (1) For each reversed subinterval, the gradient parts in (3.62) and (3.61) can be calculated by simply calling an auto gradient tool built in the most of platforms.
   (2) The remaining parts of (3.62) and (3.61), and (3.63) can be calculated by calling an ODE solver.
   (3) To avoid of calling ODE solver three times for (2), We can combine them as a single call on a corresponding augmented dynamic system of (3.48b), (3.53), and (3.58).
4. Finally, we obtain $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}) = \boldsymbol{q}(0)$.

Proposed algorithm The complete training process is summarized in Algorithm 2, where mini-batches of cascades are used to compute the stochastic gradient in searching the (local) minimizer $\boldsymbol{\theta}$. We did not include the gradient of the regularization of $\boldsymbol{\theta}$, but its computation is standard and can be easily added to $\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta})$.

Algorithm 2 Continuous neural mean-field (NMF) dynamics

```
1: **Input:** $\mathcal{D} = \{\mathcal{C}_k = (\mathcal{S}_k, \boldsymbol{\tau}_k) : k \in [K]\}$.
2: **Initialization:** Network architecture $\boldsymbol{g}(\cdot;\boldsymbol{\theta})$ and parameter $\boldsymbol{\theta} = (\boldsymbol{A}, \boldsymbol{\eta}, \boldsymbol{w})$.
3: **for** $k = 1, \dots, \text{MaxIterations}$ **do**
4:     Sample a mini-batch of cascades $\hat{\mathcal{D}} \subset \mathcal{D}$.
5:     Compute $\boldsymbol{m}(t)$ in (3.48b) forward in time for each $\mathcal{C} \in \hat{\mathcal{D}}$.    (Forward pass)
6:     Compute $\sum_{\mathcal{C}\in\hat{\mathcal{D}}} \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta};\mathcal{C})$ using the **BackwardMode** below.    (Backward pass)
7:     Update parameter $\boldsymbol{\theta}$ using ADAM with stochastic gradient $\sum_{\mathcal{C}\in\hat{\mathcal{D}}} \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta};\mathcal{C})$.
8: **end for**
9: **Output:** Network parameter $\boldsymbol{\theta}$.
```

```
   **BackwardMode**
10: **Input:** Cascade $\mathcal{C} = (\mathcal{S}, \boldsymbol{\tau})$ with $\boldsymbol{\tau} : 0 = t^{(0)} < t^{(1)} < \cdots < t^{(m+1)} = T$ and $\boldsymbol{m}(T)$.
11: **Terminal augmented state:** $[\boldsymbol{m}(T); \boldsymbol{p}(T); \boldsymbol{q}(T)] = [\boldsymbol{m}(T); [\mathbf{1}; \mathbf{0}]; \mathbf{0}]$.
12: **for** $i = m+1, \dots, 1$ **do**
13:     Solve the ODE below backward in time $(t^{(i-1)}, t^{(i)}]$:
        $$\begin{pmatrix} \boldsymbol{m}'(t) \\ \boldsymbol{p}'(t) \\ \boldsymbol{q}'(t) \end{pmatrix} = \begin{pmatrix} \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta}) \\ -\nabla_{\boldsymbol{m}} \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta}) \boldsymbol{p}(t) \\ -\nabla_{\boldsymbol{\theta}} \boldsymbol{g}(\boldsymbol{m}(t);\boldsymbol{\theta}) \boldsymbol{p}(t) \end{pmatrix}$$
        with terminal condition $[\boldsymbol{m}(t^{(i)}); \boldsymbol{p}(t^{(i)}); \boldsymbol{q}(t^{(i)})]$.
14:     $\boldsymbol{p}(t^{(i-1)}) \leftarrow \boldsymbol{p}(t^{(i-1)}) - \nabla_{\boldsymbol{m}} \log \boldsymbol{g}_{i-1}(\boldsymbol{m}(t^{(i-1)});\boldsymbol{\theta})$.
15:     $\boldsymbol{q}(t^{(i-1)}) \leftarrow \boldsymbol{q}(t^{(i-1)}) - \nabla_{\boldsymbol{\theta}} \log \boldsymbol{g}_{i-1}(\boldsymbol{m}(t^{(i-1)});\boldsymbol{\theta})$.
16: **end for**
17: **Output:** $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta};\mathcal{C}) \leftarrow \boldsymbol{q}(0)$.
```

3.3.6 Experiment evaluation

In this subsection, we show that our approach is robust to the variation of the unknown underlying diffusion models, and it also significantly outperforms existing approaches a series of numerical experiments. The same basic setting in subsection 3.2.5 are employed here.

Training and testing synthetic data For each type of network, we randomly generate 5 networks of $(n, d) = (128, 4)$ and another 5 networks of $(n, d) = (1024, 4)$, where n is the total number of nodes on the network and d is the average out-degree per node. For each combination of network model and diffusion model, we randomly generate 900 source node sets of size varying between 1 and 10, and simulate 10 diffusion cascades for each source set for training. Thus the training data consists of K=9,000 cascades, all of which are truncated into time window $[0, T]$ with $T = 20$. We generate 100 additional source sets in a similar way, and then split them as 50%-validation and 50%-test. This setting on validation and test data will be used for all the experiments related to influence estimation.

Algorithm and parameter settings In the training of NMF, the batch size of cascade data is set to 300 and the number of epochs is 50. The coefficients of the regularization term on $\boldsymbol{A}$ and weight decay in Adam optimizer are set to (0.01,1) and (0.001,0) for network of size 128 and 1024, respectively. We use Runge-Kutta 4th order (rk4) method with 40 time steps to solve the ODEs numerically.

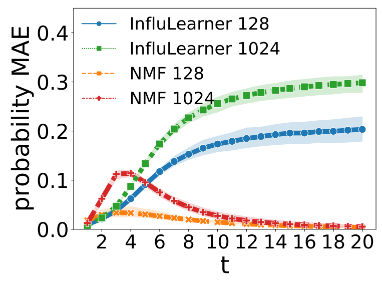
Comparison results on influence estimation We only compare the accuracy of continuous NMF with InfluLearner. In the test, we set the number of random features for InfluLearner to 200 as suggested in [24].

We show the numerical results of InfluLearner and NMF for influence estimation on the three aforementioned synthetic diffusion networks (i.e., Hier, Core, and Rand) in Figure 3.6 and Figure 3.7. For each of these three networks, we simulate three types of diffusion times (i.e., Exp, Ray, and Wbl). Therefore, we have 9 network/diffusion combinations in total. For each of these 9 combinations, we show the scaled influence MAE (top) and probability MAE (bottom) of InfluLearner and NMF on networks of size 128 and 1024 as explained above. In each plot of Figure 3.6 and Figure 3.7, we show the mean (center line) and standard deviation (shade) averaged over 5 instances. As we can observe in Figure 3.6 and Figure 3.7, the error of NMF is much smaller than that of InfluLearner for almost all times, except at some early stages and on Hierarchical network with Weibull distribution. This demonstrates that NMF is a much more accurate method in influence estimation.

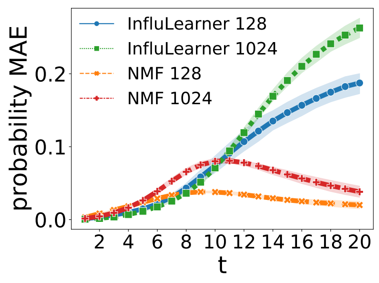
Comparison results on network structure inference We compared the estimated $\mathcal{E}$ and $\boldsymbol{A}$ using NetRate and NMF using the four criteria mentioned above in Table 3.3 for three types of networks (Random, Hierarchical, and Core-periphery) and two diffusion models (Exponential and Rayleigh). In all of these tests, NMF consistently outperforms NetRate in all accuracy metrics. We also draw $\boldsymbol{A}$ inferred by NetRate and NMF for a visual comparison in Figure 3.8. In Figure 3.8, we show the ground truth $\boldsymbol{A}^*$ (left), the matrix $\boldsymbol{A}$ inferred by NetRate (middle), and $\boldsymbol{A}$ learned by NMF (right). The values of α_{ij} are indicated by the color—the darker the red is, the higher the value of α_{ij}—and the white pixels represent where α_{ij} is zero. As we can see, $\boldsymbol{A}$ learned by NMF is much more faithful to $\boldsymbol{A}^*$ than that by NetRate. This result shows that NMF is very versatile and robust in learning network structure from cascade data.

Since NetRate code [35] was implemented in MATLAB and is executed on CPU in our experiment, the computation times of NetRate and NMF cannot be directly compared. However, we notice that NetRate takes approximately 10+ hours on average to infer each network structure $\boldsymbol{A}$ in Table 3.3, whereas NMF only requires about 300 seconds on average to return both more accurate $\boldsymbol{A}$ and an influence estimation mechanism.

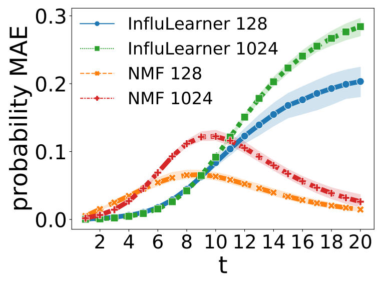
Scalability to network size and density In this test, we will demonstrate the robustness of NMF in influence estimation when the network size n and density d vary. Recall that d stands for the average out-degree per node. The larger and/or denser the network is, the more challenging the estimation becomes. In all the experiments, we use training data



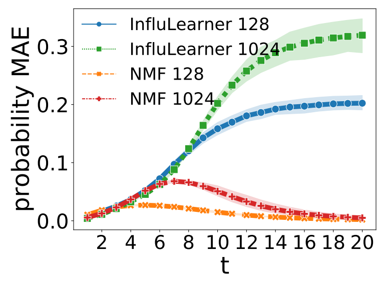
(a) Core + Exp



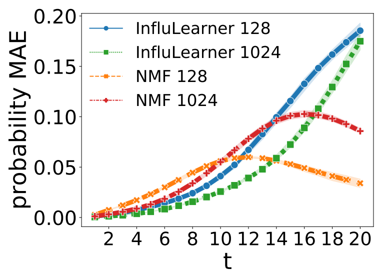
(b) Core + Ray



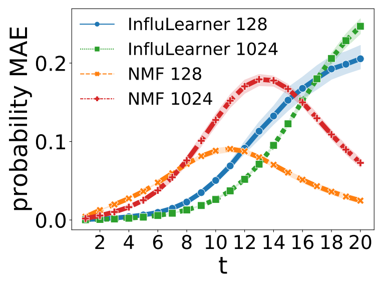
(c) Core + Wbl



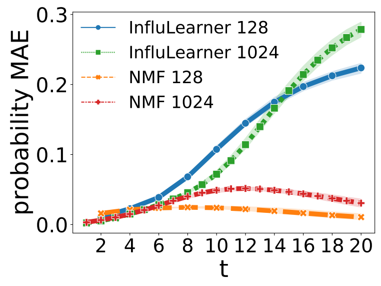
(d) Rand + Exp



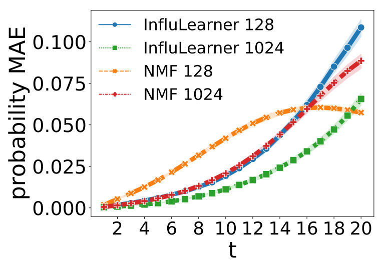
(e) Rand + Ray



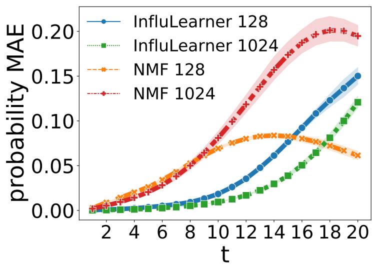
(f) Rand + Wbl



(g) Hier + Exp

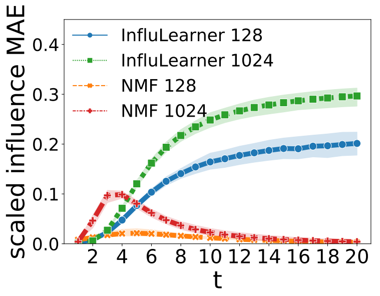


(h) Hier + Ray

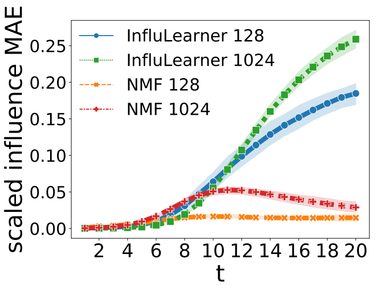


(i) Hier + Wbl

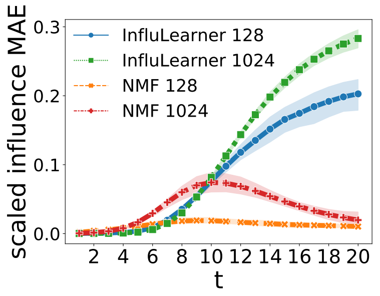
Figure (3.6). MAE of node infection probability (bottom) by InfluLearner [24] and continuous NMF on each of the 9 different combinations of Core-periphery (Core), Random (Rand) and Hierarchical (Hier) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 50 test source sets are shown. Each network has two configurations of (n, d): $(128, 4)$ and $(1024, 4)$, where n is the number of nodes in the diffusion network, and d is the average out-degree per node.



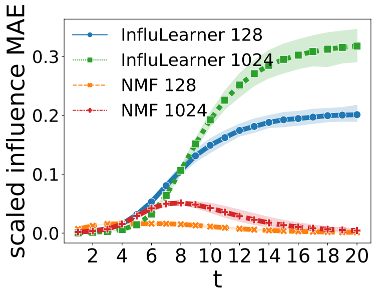
(a) Core + Exp



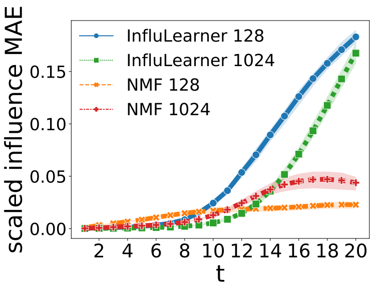
(b) Core + Ray



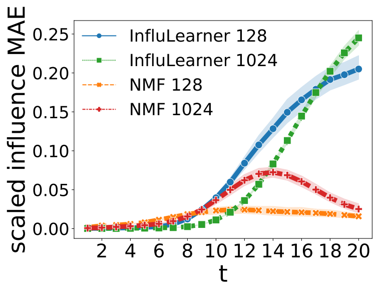
(c) Core + Wbl



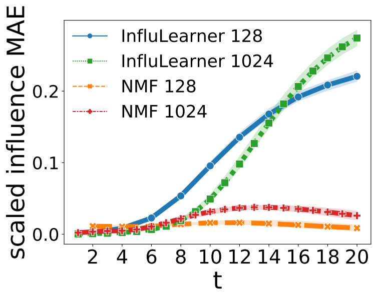
(d) Rand + Exp



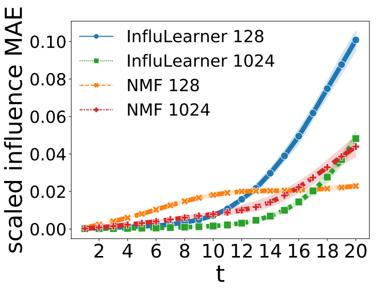
(e) Rand + Ray



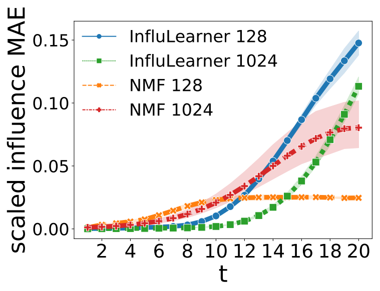
(f) Rand + Wbl



(g) Hier + Exp



(h) Hier + Ray

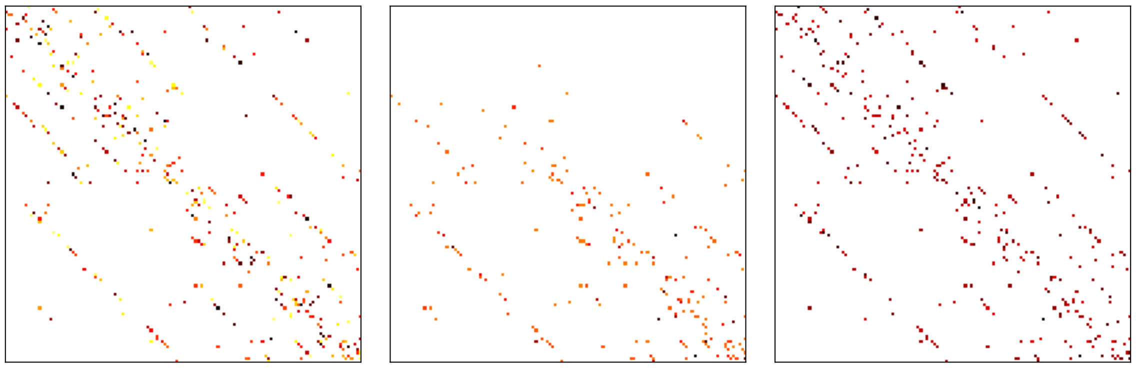


(i) Hier + Wbl

Figure (3.7). MAE of scaled influence by InfluLearner [24] and continuous NMF on each of the 9 different combinations of Core-periphery (Core), Random (Rand) and Hierarchical (Hier) networks, and exponential (Exp), Rayleigh (Ray) and Weibull (Wbl) diffusion models. Mean (centerline) and standard deviation (shade) over 50 test source sets are shown. Each network has two configurations of (n, d): $(128, 4)$ and $(1024, 4)$, where n is the number of nodes in the diffusion network, and d is the average out-degree per node.

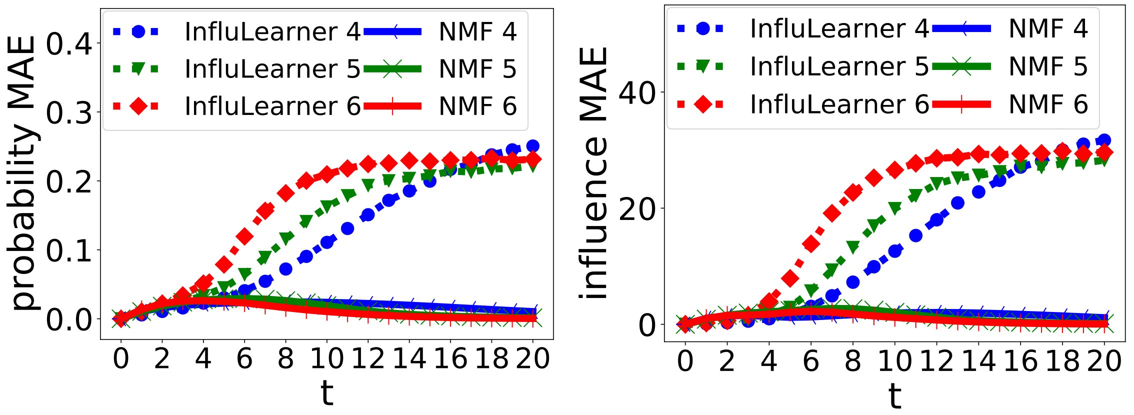
Table (3.3). Performance of network structure inference using NetRate [35] and the proposed continuous NMF on Random, Hierarchical, and Core-periphery networks consisting of 128 nodes and 512 edges with Exponential and Rayleigh as diffusion distribution on edges. Quality of the learned edge set $\mathcal{E}$ and distribution parameter $\boldsymbol{A}$ are measured by precision (Prc), recall (Rcl), accuracy (Acc), and correlation (Cor). Larger value indicates higher accuracy.

Diffusion	Network	Method	Prc	Rcl	Acc	Cor
Exponential	Random	NetRate	0.457	0.821	0.515	0.438
		NMF	**0.459**	**0.997**	**0.622**	**0.910**
	Hierarchical	NetRate	0.395	0.748	0.515	0.739
		NMF	**0.595**	**0.997**	**0.745**	**0.928**
	Core-periphery	NetRate	0.277	0.611	0.264	0.264
		NMF	**0.292**	**0.997**	**0.450**	**0.839**
Rayleigh	Random	NetRate	0.481	0.399	0.434	0.465
		NMF	**0.883**	**0.905**	**0.894**	**0.909**
	Hierarchical	NetRate	0.659	0.429	0.519	0.464
		NMF	**0.889**	**0.936**	**0.911**	**0.913**
	Core-periphery	NetRate	0.150	0.220	0.178	0.143
		NMF	**0.649**	**0.820**	**0.724**	**0.820**



(a) True (b) NetRate (c) NMF

Figure (3.8). Ground truth $\boldsymbol{A}^*$ (left) and $\boldsymbol{A}$ inferred by NetRate (middle) and continuous NMF (right) in same color scale using cascades from a Hierarchical network consisting of 128 nodes and 512 edges with exponential diffusion model. Darker pixel indicates larger value of an entry of $\boldsymbol{A}$.

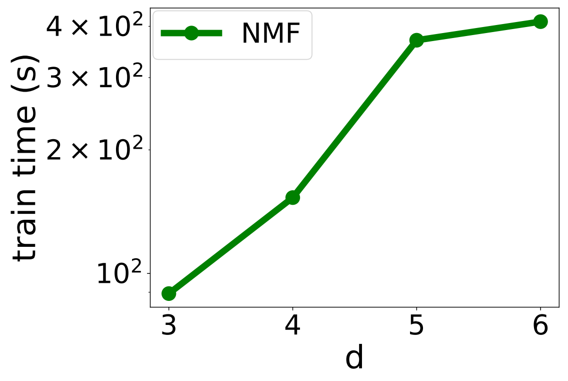


(a) Probability MAE (b) Influence MAE

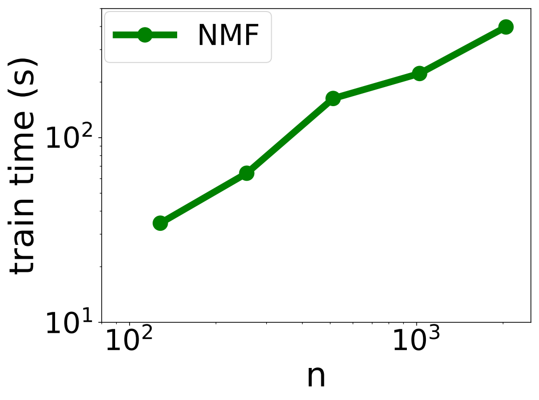
Figure (3.9). MAE of infection probability (a) and influence (b) obtained by InfluLearner [24] and continuous NMF on Hierarchical networks of size $n = 128$ and increasing d from 4 to 6.

consisting of 9,000 cascades generated from Hierarchical network and exponential diffusion model and set the batch size to 300.

Recall that we have showed in Figure 3.6 and Figure 3.7 that NMF consistently outperforms than InfluLearner when the network size is set to 128 and 1024. To show the scability



(a) Train time vs d



(b) Train time vs n

Figure (3.10). (a) Training time (in seconds) of continuous NMF versus density (average out-degree per node) d. (b) Training time (in seconds) versus network size n.

of NMF, we further test NMF on increasing network size n from 128 to 2048 (with density $d = 4$). To test the training time of NMF, we terminate the computation when the average MAE of infection probability on validation data over 20 timepoints $t_\ell = \ell$ ($\ell = 1, 2, \ldots, 20$) is below 0.07. Euler method with 40 steps is employed as the ODE solver and the learning rate of the Adam optimizer is set to 0.0001 for network with 2048 nodes. The training time of NMF is shown in Figure 3.10b, which demonstrate that NMF is scalable for large network size n.

We also test the performance of NMF for varying network density d. We compare the infection probability and influence MAE of InfluLearner and NMF for varying edge density d set to 4, 5, and 6 on a Hierarchical network on exponential diffusion model with 128 nodes. Figure 3.9a and Figure 3.9b show that the MAE of infection probability and influence estimation obtained by InfluLearner and NMF. These two plots show that NMF is very robust when the density of the network increases by consistently generating estimates of low MAE. Figure 3.10a shows the training time of NMF versus network density d while $n = 128$ is fixed. In this plot, the computation time is recorded when the training MAE at time t_h is below 0.04, where t_h is the time when on average half of the nodes on the network are infected as indicated by the ground truth. Here, rk4 method with 40 steps is employed as the ODE solver. Similarly, Figure 3.10b shows the training time versus network size n while $d = 4$ is fixed. From Figures 3.10a and 3.10b, we can see that the computational cost of NMF grows approximately quadratic in density d and linear in size n.

CHAPTER 4

INFLUENCE MAXIMIZATION WITH LEARNED CONTINUOUS NMF

The proposed neural mean-field dynamics also provide a new perspective for efficiently solving the challenging influence maximization problem. In this chapter, we develop a highly efficient algorithm for the very challenging influence maximization problem based on our continuous-time NMF framework.

4.1 Optimal control formulation

In this section, we show how the proposed NMF can be used to tackle the important but very challenging influence maximization problem stated as Problem 2.

4.1.1 On original influence maximization

Suppose we have trained a continuous NMF with parameters $\boldsymbol{\theta}$ in Algorithm 2, such that we can estimate $\boldsymbol{x}(t)$ for any $t \in [0, T]$ and any given source node set $\boldsymbol{\chi}_{\mathcal{S}}$. Then the goal of influence maximization is to identify $\boldsymbol{\chi}_{\mathcal{S}} \in \{0, 1\}^n$ such that its influence at the prescribed time T (or any other prescribed $t \in (0, T)$) is maximized. Namely, our goal is to solve the following optimization problem

Definition 4.1.1 (Optimal control on $\boldsymbol{\chi}_{\mathcal{S}}$ for influence maximization)**.** *The objective is to find the optimal control $\boldsymbol{\chi}_{\mathcal{S}}$ for the following constrained optimization problem:*

$$\max_{\boldsymbol{\chi}_{\mathcal{S}}} \quad \sigma(T; \boldsymbol{\chi}_{\mathcal{S}}) := \mathbf{1}_n^\top \boldsymbol{x}(T; \boldsymbol{\chi}_{\mathcal{S}}), \tag{4.1a}$$

$$\text{s.t.} \quad \boldsymbol{\chi}_{\mathcal{S}} \in \{0, 1\}^n, \quad \mathbf{1}_n^\top \boldsymbol{\chi}_{\mathcal{S}} = n_0, \tag{4.1b}$$

where $n_0 \in \mathbb{N}$ is the given budget size.

Note that $\boldsymbol{x}(T; \boldsymbol{\chi}_{\mathcal{S}})$ is the first n components of $\boldsymbol{m}(T)$ computed by forward NMF dynamics with initial value $\boldsymbol{m}(0) = [\boldsymbol{\chi}_{\mathcal{S}}; \mathbf{0}]$. However, (4.1) is an NP-hard combinatorial optimization problem [37].

4.1.2 On relaxed influence maximization

To overcome the computation complexity of (4.1), we propose to relax the binary-valued decision vector $\boldsymbol{\chi}_{\mathcal{S}}$ to $\boldsymbol{u} \in [0,1]^n$ in the continuous hypercube $[0,1]^n$ which reduces the following formulation.

Definition 4.1.2 (Optimal control on $\boldsymbol{u}$ for relaxed influence maximization)**.** *The objective is to find the optimal control* $\boldsymbol{u}$ *for the following constrained optimization problem:*

$$\min_{\boldsymbol{u}\in\mathcal{U}} \quad L(\boldsymbol{u}) := \mathcal{R}(\boldsymbol{u}) - \mathbf{1}_n^\top \boldsymbol{x}(T;\boldsymbol{u}), \tag{4.2a}$$

$$\text{s.t.} \quad \boldsymbol{u} \in \mathcal{U} := \{\boldsymbol{u} \in [0,1]^n : \mathbf{1}_n^\top \boldsymbol{u} = n_0\}, \tag{4.2b}$$

where $n_0 \in \mathbb{N}$ *is the given budget size and* $\mathcal{R}(\boldsymbol{u})$ *is a regularizer that encourages all components of* $\boldsymbol{u}$ *to take values close to either* 0 *or* 1.

Remark 4.1.3. *In our experiments, we simply set* $\mathcal{R}(\boldsymbol{u}) = \sum_{i=1}^n u_i(1-u_i)$.

4.2 Projected gradient descent (PGD)

We can employ the projected gradient descent (PGD) method to solve the relaxed influence maximization problem formulated in Definition 4.1.2 by updating $\boldsymbol{u}$ as follows.

$$\boldsymbol{u}_{l+1} = \Pi_{\mathcal{U}}(\boldsymbol{u}_l - \gamma_l \nabla_{\boldsymbol{u}} L(\boldsymbol{u}_l)) := \operatorname*{arg\,min}_{\boldsymbol{u}\in\mathcal{U}} \|\boldsymbol{u} - (\boldsymbol{u}_l - \gamma_l \nabla_{\boldsymbol{u}} L(\boldsymbol{u}_l))\|^2, \tag{4.3}$$

where l is the iteration counter of PGD, $\tau_l > 0$ is the step size, and $\Pi_{\mathcal{U}}$ denotes the orthogonal projection onto $\mathcal{U}$. If $\nabla_{\boldsymbol{u}} L(\boldsymbol{u}_l)$ is known, then (4.3) is a standard quadratic program (QP) and can be solved efficiently by off-the-shelf solvers. Hence, the only remaining question is to solve $\nabla_{\boldsymbol{u}} L(\boldsymbol{u}_l)$ which will be discussed in the next section.

4.3 Gradients calculation for relaxed IM

In this section, we employ the standard optimal control theory to compute $\nabla_{\boldsymbol{u}} L(\boldsymbol{u})$ for any given $\boldsymbol{u}$. The following theorem states that this quantity can be computed very efficiently using the proposed NMF dynamics.

Theorem 4.3.1. *Let $[\boldsymbol{m}(t); \boldsymbol{s}(t)]$ be the solution of the augmented NMF system:*

$$\begin{pmatrix} \boldsymbol{m}'(t) \\ \boldsymbol{s}'(t) \end{pmatrix} = \begin{pmatrix} \boldsymbol{g}(\boldsymbol{m}(t); \boldsymbol{\theta}) \\ \nabla_{\boldsymbol{x}} \boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{m}; \boldsymbol{\theta})^\top \boldsymbol{s}(t) \end{pmatrix} \tag{4.4}$$

with initial value $[\boldsymbol{m}(0); \boldsymbol{s}(0)] = [[\boldsymbol{u}; \boldsymbol{0}]; \boldsymbol{1}]$ forward in time $[0, T]$, where $\boldsymbol{g}_{\boldsymbol{x}}$ is the first n components of $\boldsymbol{g}$. Then

$$\nabla_{\boldsymbol{u}} L(\boldsymbol{u}) = \nabla_{\boldsymbol{u}} \mathcal{R}(\boldsymbol{u}) - \boldsymbol{s}(T).$$

Proof. Let $\boldsymbol{v} \in \mathbb{R}^n$ be arbitrary and consider the variation $\boldsymbol{u}_\epsilon := \boldsymbol{u} + \epsilon \boldsymbol{v} + o(\epsilon)$ of $\boldsymbol{u}$ with $\epsilon > 0$. Let $\boldsymbol{x}_\epsilon(t)$ be the $\boldsymbol{x}$-part of the solution $\boldsymbol{m}_\epsilon(t)$ to (3.48b) with initial $[\boldsymbol{u}_\epsilon; \boldsymbol{0}]$. Suppose $\boldsymbol{x}_\epsilon(t) = \boldsymbol{x}(t) + \epsilon \boldsymbol{w}(t) + o(\epsilon)$ for all $t \in [0, T]$ as $\epsilon \to 0$, then $\boldsymbol{w}(t)$ solves

$$\begin{cases} \boldsymbol{w}'(t) = \nabla_{\boldsymbol{x}} \boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{m}(t); \boldsymbol{\theta}) \boldsymbol{w}(t), \quad 0 \le t \le T, \\ \boldsymbol{w}(0) = \boldsymbol{v}. \end{cases} \tag{4.5}$$

Note that (4.5) is a linear ODE of $\boldsymbol{w}$ and thus has an analytic solution as follows:

$$\boldsymbol{w}(T) = e^{\int_0^T \nabla_{\boldsymbol{x}} \boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{m}(t); \boldsymbol{\theta}) \, \mathrm{d}t} \boldsymbol{v}.$$

Next, we compute the directional derivative of L defined in (4.2) at $\boldsymbol{u}$ along direction $\boldsymbol{v}$:

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\epsilon}L(\boldsymbol{u}_\epsilon)\Big|_{\epsilon=0} &= \frac{\mathrm{d}}{\mathrm{d}\epsilon}\Big(\mathcal{R}(\boldsymbol{u}_\epsilon) - \mathbf{1}\cdot\boldsymbol{x}_\epsilon(t)\Big)\Big|_{\epsilon=0} \\ &= \nabla_{\boldsymbol{u}}\mathcal{R}(\boldsymbol{u})\cdot\boldsymbol{v} - \mathbf{1}\cdot\boldsymbol{w}(T) \\ &= \Big(\nabla_{\boldsymbol{u}}\mathcal{R}(\boldsymbol{u}) - e^{\int_0^T \nabla_{\boldsymbol{x}}\boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{x}(t);\boldsymbol{\theta})^\top\,\mathrm{d}t}\mathbf{1}\Big)\cdot\boldsymbol{v}.\end{aligned}$$

As $\boldsymbol{v}$ is arbitrary, we know the gradient $\nabla_{\boldsymbol{u}}L(\boldsymbol{u})$ is

$$\nabla_{\boldsymbol{u}}L(\boldsymbol{u}) = \nabla_{\boldsymbol{u}}\mathcal{R}(\boldsymbol{u}) - e^{\int_0^T \nabla_{\boldsymbol{x}}\boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{x}(t);\boldsymbol{\theta})^\top\,\mathrm{d}t}\mathbf{1}. \tag{4.6}$$

It is clear that the second term on the right hand side of (4.6) is $\boldsymbol{s}(T)$ solved from

$$\begin{cases}\boldsymbol{s}'(t) = \nabla_{\boldsymbol{x}}\boldsymbol{g}_{\boldsymbol{x}}(\boldsymbol{x}(t);\boldsymbol{\theta})^\top\boldsymbol{s}(t), \quad 0\le t\le T, \\ \boldsymbol{s}(0) = \mathbf{1}.\end{cases} \tag{4.7}$$

This completes the proof. □

Theorem 4.3.1 implies that $\nabla_{\boldsymbol{u}}L(\boldsymbol{u})$ can be easily computed by solving NMF augmented by an auxiliary variable $\boldsymbol{s}(t)$ forward in time $[0,T]$ as in (4.4). Note that the computation complexity of (4.4) is linear in the network size n and standard numerical ODE integrators can quickly solve the ODE to high accuracy.

4.4 Proposed algorithm NMF-InfMax

We summarize the steps for solving (4.2) in Algorithm 3. Note that the output $\boldsymbol{u}$ may not be binary, and thus we can set the largest n_0 components of $\boldsymbol{u}$ to 1 and the rest to 0 as the final source set selection.

Algorithm 3 Influence maximization via neural mean-field dynamics (NMF-InfMax)

```
1: Input: Trained NMF with $\boldsymbol{g}(\cdot;\boldsymbol{\theta})$ from Algorithm 2, budget $n_0 \in \{1,\dots,n-1\}$
2: Initialization: $\boldsymbol{u} \in \mathcal{U}$.
3: for $l = 1,\dots,$ MaxIterations do
4:     Solve $[\boldsymbol{m}(T), \boldsymbol{s}(T)]$ from (4.4) forward in time with initial $[[\boldsymbol{u};\mathbf{0}];\mathbf{1}]$.   (Forward pass)
5:     Set $\hat{\boldsymbol{u}} \leftarrow \boldsymbol{u} - \gamma \nabla_{\boldsymbol{u}} L(\boldsymbol{u})$ where $\nabla_{\boldsymbol{u}} L(\boldsymbol{u}) = \boldsymbol{s}(T)$.
6:     Solve a QP: $\boldsymbol{u} \leftarrow \arg\min_{\boldsymbol{u}\in\mathcal{U}} \|\boldsymbol{u} - \hat{\boldsymbol{u}}\|^2$.
7: end for
8: Output: Source set selection $\boldsymbol{u}$.
```

4.5 Experiment evaluation

This part of the experiment is dedicated to performance evaluation in influence maximization. Specifically, we use the trained NMF to find the optimal source set with limited budget for maximal influence by following Algorithm 3 which is referred to as NMF-InfMax.

4.5.1 Comparison algorithms

For comparison purpose, we also test the following methods for influence maximization.

- IMINFECTOR [77]: IMINFECTOR represents the cascade data into two datasets consisting of seed-cascade length pairs and seed-influenced node pairs to approximate the influence spread and infection probability of each node by a regression model and a probability classifier, respectively. The outputs are used to reduce the number of candidate seeds and reformulate the computation of the influence spread in a greedy solution to influence maximization. Like our method, IMINFECTOR only uses cascade data as inputs, with embedding size 50 and sampling percentage 120, trained for 50 epochs with a learning rate of 0.1. The reduction percentage P is set to 100 to keep full information of cascades.

- IMM [93]: IMM is a reverse reachable (RR) sketch based method which applies the standard greedy algorithm for maximum coverage to derive a budget size node set that covers a large number of RR sets sampled from the given network. We consider the case when IMM return $(1 - 1/e - \varepsilon)$-approximate solution with $\epsilon = 0.1$ and parameter $\ell = 1$, following the experiments in [93].

- InfluMax [34,39]: InfluMax speed up the greedy influence maximization algorithm by exploiting submodularity. We incorporate it with the influence estimation algorithm ConTinEst [25]. For ContinEst, we draw 10,000 random samples, each of which has 5 random labels for each node.

Since IMM and InfluMax both require the knowledge of the transmission matrix $\boldsymbol{A}$, we apply NetRate method to learn $\boldsymbol{A}$ from cascade data first, then feed $\boldsymbol{A}$ to these two methods. We remark that NetRate and IMINFECTOR are both in favor of training data consisting of cascades with the source sets of size 1 (i.e., only one source node). In contrary, NMF-InfMax does not have this restriction and thus is more flexible. However, for comparison purpose, we only feed cascade data with single source node to all methods in this experiment.

4.5.2 Experiment setting

Diffusion network We again use three types of Kronecker graph models: Hierarchical (Hier), Core-periphery (Core) and Random (Rand) networks, and simulate the diffusion processes using exponential distribution with transmission rates randomly sampled from Unif[0.1,1]. For each type of network model, we generate two networks of size $n = 1,024$ with $d = 2$ and $d = 4$, respectively.

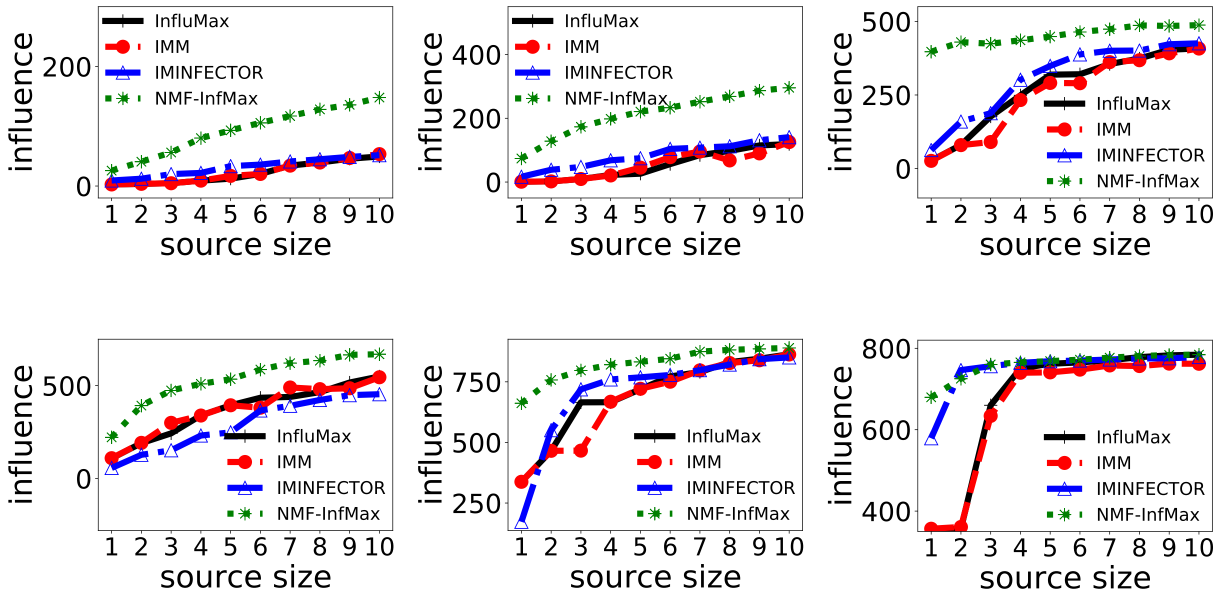
Parameters setting To train NMF, we set the batch size to 300 and the number of epochs to 50. The coefficients of the regularization term on $\boldsymbol{A}$ is set to 0.001, and the rk4 method with 40 time steps is employed as the ODE solver. To train NMF-InfMax, we set the step size to constant 0.01, and terminate PGD if either the iteration number reaches 500 or the computed influence does not change for 10 consecutive iterations.

Synthetic cascade data We sample 100 source nodes, and for each source node we simulate 10 cascades. Hence we have a total of K=1,000 cascades for training. In Figure 4.2a, we show the accuracy of NMF-InfMax when the number of training cascades increases from 1,000 to 5,000 for each fixed source set of size from 1 to 10. As we can observe, the accuracy increases significantly when the number of cascades grows from 1,000 to 2,000 but then improvements become insignificant. This suggests that 2,000 cascades is necessary to obtain more accurate influence maximization results for NMF-InfMax. However, due to the limited scalibilty of NetRate which performs extremely slowly when the number of cascades is over 1,000 and average out-degree is over 4. We also tested IMINFECTOR with larger training data set, but unlike our method, the accuracy of IMINFECTOR does not improve over 1,000. Hence we still only feed 1,000 cascades to all the compared methods despite that this choice is only in favor of three existing methods.

4.5.3 Comparison results

First of all, it is important to note that both InfluMax and IMM require the knowledge of diffusion model given by the shape and scale parameters of edges for the computation of NetRate and their own. Thus, they are more vulnerable to model mis-specification. In this test, we assume they know the ground truth diffusion model, so they can attain their highest accuracy. However, it is worth noting that the network inference by NetRate can be very expensive computationally. For example, it took NetRate up to 160 hours to infer the network structure from 1,000 cascades of a core-periphery network of $n = 1,024$ and $d = 4$ in Figure 4.1c. In contrast, the computational cost of IMINFECTOR is very low, but IMINFECTOR is more restrictive on data because it requires that the training cascades contain the nodes to be selected. This may not be feasible in practice. Moreover, the influence maximization results obtained by IMINFECTOR also appear to be worse than that by NMF, as shown below.

The influence maximization results obtained by the aforementioned algorithms and NMF are shown in Figure 4.1. As we can see, NMF-InfMax consistently returns more influential source sets with all budget n_0 for all varying network structure, density, and budget.

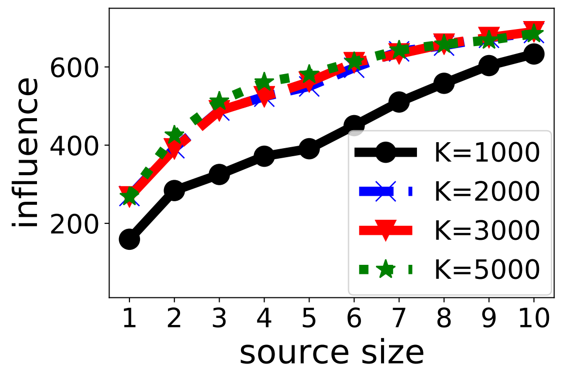


(a) Hierarchical (b) Random (c) Core-periphery

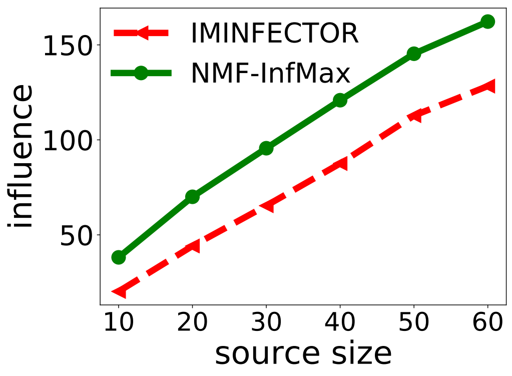
Figure (4.1). Influence of the source sets selected by the compared methods on three different types of networks: (a) Hierarchical, (b) Random, and (c) Core-periphery, with exponential diffusion model at $T = 10$ and varying source sizes n_0 from 1 to 10. Each network consists of 1024 nodes and 2048 edges (top) or 4096 edges(bottoms).

4.5.4 Real data

We extract diffusion cascades from the MemeTracker dataset [56] which includes 300 million blog posts and articles collected from 5,000 active media sites between March 2011 and February 2012. Following [24], we select the group of cascades with the keyword "apple and jobs" and then split them as 60%-train and 40%-validation for the influence maximization models. As the diffusion model of real-world cascade data is unknown, we only test IMINFECTOR and NMF-InfMax. We follow the setting in [24] to compute the influence of any selected source set: we uniformly sample one cascade from the data for each node in



(a) Varying training set size



(b) Influence vs n_0

Figure (4.2). (a) Influence generated the source sets selected by NMF-InfMax trained using increasing number of cascades on Hierarchical networks with 1,024 nodes and 4,096 edges. (b) Influence generated by the source sets selected by IMINFECTOR and NMF-InfMax on the MemeTracker dataset at $T = 10$ hours.

the set and take the union of all sampled cascades as the set of infected nodes. We repeat this process for 1,000 times and take the average as the true influence of the selected set. Figure 4.2b shows the result of influence maximization results. In Figure 4.2b, we set $T = 10$ and the source size $n_0 = 10, 20, \cdots, 60$, and plot the influence of the source sets selected by IMINFECTOR and NMF-InfMax. As we can see, NMF-InfMax consistently selected more influential combination of nodes that generate greater influence than those selected by IMINFECTOR do.

CHAPTER 5

SUMMARY AND DISCUSSION

5.1 Summary

We propose a comprehensive framework applied neural mean-field (NMF) dynamics in discrete and continuous setting for simultaneous influence estimation and network inference directly from cascade data on diffusion network. Based on our continuous NMF framework, we also develop a highly efficient algorithm for the influence maximization problem.

5.2 Future work

We expect that the proposed framework can be applied to many other optimization and control problems arising from diffusion network applications, such as optimal campaigning, propagation control, and source identification, which will also be investigated in our future work.

5.2.1 Implementation for application

The proposed models are implemented in the worst case which suppose the cascades are the only known information. However, in the applications, we could customize the models to fit to different significant data setting. More specifically, if the knowledge of $\mathcal{E}$ and $\boldsymbol{A}$ are allowed, then they can be integrated for further performance improvement.

For example, in the social network, the existence of followee-follower relations (edges) is easy to be tracked, then we could employ the network adjacent matrix as support of $\boldsymbol{A}$ in Algorithm 1 and Algorithm 2 for the corresponding problems.

On the other hand, while some node and edge attributes could be important to reflect the interaction strength in our intuition, we can take these features into account. For example, in the research on the citation dataset, the subject categories and abstract of papers (nodes) strongly suggests on the co-citing connections (edge) of papers since a researcher should cite papers on related topics. In this case, the paper features become very important on the learning of $\boldsymbol{A}$. As we mentioned, $\boldsymbol{A}$ can be interpreted as the convolution to be learned from a graph convolution network (GCN) [54, 101]. Thus we could employ some graph representation technique to embedding node features to the space of $\boldsymbol{A}$ which provide the knowledge on $\boldsymbol{A}$.

5.2.2 Dynamic diffusion network

The diffusion networks $G = (\mathcal{V}, \mathcal{E})$ discussed in this thesis are inherently static in the following aspects:

(A1) The node set $\mathcal{V}$ does not change.

(A2) The Edge set $\mathcal{E}$ does not change.

(A3) suppose that the transmission rate $\alpha_{i,j}$ is a constant function over time for any edge $(i, j) \in \mathcal{E}$.

(A4) Suppose that if a node is infected, then its status will not change. That is, an infected node is never recovered.

However, diffusion networks in the real world keep evolving. For example, in a social network, new users join, and new followee-follower relationship form at each seconds, which continuously affects the network structure. In the dynamic network for Covid-19, some infected people get recovered and some healthy people get fully vaccinated. Thus a recover rate and varying infection strengths between persons should be considered. The statement of one problem on dynamic diffusion network is

Problem 3. *Consider a directed network $G = (\mathcal{V}(t), \mathcal{E}(t), \omega(t))$ with node set $\mathcal{V}(t)$, edge set $\mathcal{E}(t)$, and node weight mapping $\omega(t)$ which indicate the node recover rates. Given the original node set $\mathcal{V}^0$ and edges set $\mathcal{E}^0$, some observed cascade data $\mathcal{D}$, and a diffusion model to describe the distribution $p(t; \alpha_{ij}(t))$ of the time t of node i impact on j where $\alpha_{ij}(t)$ is the transmission rate. The objective is that*

- *to find the solution of influence estimation and maximization problem, or*
- *for a new added node v at time t, to predict the infection status of v.*

5.2.3 Context-aware problems

In the real world, context-aware influence maximization problems are also discussed which consider the influence $\sigma(\mathcal{S})$ related to some contextual features, such as topic, time and location.

Problem 4 (Context-aware influence maximization)**.** *With the necessary given conditions, the objective is to find $\mathcal{S} \subseteq \mathcal{V}$ to maximize the number of infected node at time T with property P such that $|\mathcal{S}| = n_0$.*

Problem 5 (Extended context-aware influence maximization)**.** *Define $\mathcal{S}_i = \{v \in \mathcal{V} : v \text{ has property } P_i\}$ for $i = 1, 2, \cdots, k$. With the necessary given conditions, the objective is to find $\mathcal{S} \subseteq \mathcal{V}$ to maximize the number of infected node at time T such that $|\mathcal{S}_i| = n_i$ for $i \in [k]$ where n_i is the budget size of nodes with property P_i.*

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning, 2016.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. Ieee, 2017.

[3] Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. Deep equilibrium models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[4] Suman Banerjee, Mamata Jenamani, and Dilip Kumar Pratihar. A survey on influence maximization in a social network. *Knowledge and Information Systems*, 62(9):3417–3455, 2020.

[5] Dimitri P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[6] Ágnes Bodó, Gyula Y Katona, and Péter L Simon. SIS epidemic propagation on hypergraphs. *Bulletin of mathematical biology*, 78(4):713–735, 2016.

[7] Marián Boguná and Romualdo Pastor-Satorras. Epidemic spreading in correlated complex networks. *Physical Review E*, 66(4):047104, 2002.

[8] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. Maximizing

social influence in nearly optimal time. *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, Dec 2013.

[9] Qi Cao, Huawei Shen, Jinhua Gao, Bingzheng Wei, and Xueqi Cheng. Popularity prediction on social platforms with coupled graph neural networks. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20, pages 70–78, New York, NY, USA, 2020. Association for Computing Machinery.

[10] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In S Bengio, H Wallach, H Larochelle, K Grauman, N Cesa-Bianchi, and R Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc., 2018.

[11] Wei Chen, Chi Wang, and Yajun Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1029–1038, 2010.

[12] Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 199–208, 2009.

[13] Wei Chen, Yifei Yuan, and Li Zhang. Scalable influence maximization in social networks under the linear threshold model. In *2010 IEEE international conference on data mining*, pages 88–97. IEEE, 2010.

[14] Suqi Cheng, Huawei Shen, Junming Huang, Guoqing Zhang, and Xueqi Cheng. Staticgreedy: solving the scalability-accuracy dilemma in influence maximization. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 509–518, 2013.

[15] Alexandre J. Chorin, Ole H. Hald, and Raz Kupferman. Optimal prediction and

the mori–zwanzig representation of irreversible processes. *Proceedings of the National Academy of Sciences*, 97(7):2968–2973, 2000.

[16] Shui-Nee Chow, Xiaojing Ye, Hongyuan Zha, and Haomin Zhou. Influence prediction for continuous-time information propagation on networks. *Networks and Heterogenous Media*, 13(4):567–583, 2018.

[17] Aaron Clauset, Cristopher Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453(7191):98–101, May 2008.

[18] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *Journal of Computer and System Sciences*, 55(3):441–453, 1997.

[19] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 629–638, 2014.

[20] Edith Cohen, Daniel Delling, Thomas Pajor, and Renato F. Werneck. Sketch-based influence maximization and computation: Scaling up with guarantees. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 629–638, New York, NY, USA, 2014. Association for Computing Machinery.

[21] Gerard Cornuejols, Marshall L Fisher, and George L Nemhauser. Exceptional paper—location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management science*, 23(8):789–810, 1977.

[22] Miles Cranmer, Sam Greydanus, Stephan Hoyer, Peter Battaglia, David Spergel, and Shirley Ho. Lagrangian neural networks, 2020.

[23] X. Dong, D. Thanou, M. Rabbat, and P. Frossard. Learning graphs from data: A signal representation perspective. *IEEE Signal Processing Magazine*, 36(3):44–63, 2019.

[24] Nan Du, Yingyu Liang, Maria-Florina Balcan, and Le Song. Influence function learning in information diffusion networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–2016–II–2024. JMLR.org, 2014.

[25] Nan Du, Le Song, Manuel Gomez-Rodriguez, and Hongyuan Zha. Scalable influence estimation in continuous-time diffusion networks. In *Advances in Neural Information Processing Systems*, pages 3147–3155, 2013.

[26] Nan Du, Le Song, Ming Yuan, and Alex J. Smola. Learning networks of heterogeneous influence. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2780–2788. Curran Associates, Inc., 2012.

[27] Ahmed A Elnaggar, Mahmoud Gadallah, Mostafa Abdel Aziem, and Hesham El-Deeb. A survey of game tree searching methods. *International Journal of Computer Applications*, 1975:8887, 2014.

[28] Gerhard Wanner Ernst Hairer, Syvert P. Nørsett. Nonstiff problems. In *Solving Ordinary Differential Equations I*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag Berlin Heidelberg, 1993.

[29] Lawrence C Evans. An introduction to mathematical optimal control theory. *Lecture notes available at http://math. berkeley. edu/˜ evans/control. course. pdf*, 1983.

[30] Mehrdad Farajtabar, Xiaojing Ye, Sahar Harati, Le Song, and Hongyuan Zha. Multistage campaigning in social networks. In *Advances in Neural Information Processing Systems*, pages 4718–4726, 2016.

[31] Shanshan Feng, Gao Cong, Arijit Khan, Xiucheng Li, Yong Liu, and Yeow Meng Chee. Inf2vec: Latent representation model for social influence embedding. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 941–952. IEEE, 2018.

[32] Marshall L Fisher, George L Nemhauser, and Laurence A Wolsey. An analysis of approximations for maximizing submodular set functions—ii. In *Polyhedral combinatorics*, pages 73–87. Springer, 1978.

[33] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3):211–223, 2001.

[34] M Gomez Rodriguez and B Schölkopf. Influence maximization in continuous time diffusion networks. In *29th International Conference on Machine Learning (ICML 2012)*, pages 1–8. International Machine Learning Society, 2012.

[35] Manuel Gomez-Rodriguez, David Balduzzi, and Bernhard Schölkopf. Uncovering the temporal dynamics of diffusion networks. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pages 561–568, Madison, WI, USA, 2011. Omnipress.

[36] Manuel Gomez-Rodriguez, Jure Leskovec, and Andreas Krause. Inferring networks of diffusion and influence. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 5(4):21, 2012.

[37] Manuel Gomez-Rodriguez, Jure Leskovec, and Bernhard Schölkopf. Structure and dynamics of information pathways in online media. *CoRR*, abs/1212.1464, 2012.

[38] Manuel Gomez-Rodriguez and Bernhard Schölkopf. Submodular inference of diffusion networks from multiple trees. In *ICML*, 2012.

[39] Manuel Gomez-Rodriguez, Le Song, Nan Du, Hongyuan Zha, and Bernhard Schölkopf. Influence estimation and maximization in continuous-time diffusion networks. *ACM Transactions on Information Systems (TOIS)*, 34(2):1–33, 2016.

[40] Amit Goyal, Wei Lu, and Laks VS Lakshmanan. Celf++ optimizing the greedy al-

gorithm for influence maximization in social networks. In *Proceedings of the 20th international conference companion on World wide web*, pages 47–48, 2011.

[41] Mark Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420–1443, 1978.

[42] Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018.

[43] Samuel Greydanus, Misko Dzamba, and Jason Yosinski. Hamiltonian neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[44] Jakob Gulddahl Rasmussen. Lecture Notes: Temporal Point Processes and the Conditional Intensity Function. *arXiv e-prints*, page arXiv:1806.00221, June 2018.

[45] Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, Dec 2017.

[46] Qiang He, Xingwei Wang, Zhencheng Lei, Min Huang, Yuliang Cai, and Lianbo Ma. Tifim: A two-stage iterative framework for influence maximization in social networks. *Applied Mathematics and Computation*, 354:338–352, 2019.

[47] Shushan He, Hongyuan Zha, and Xiaojing Ye. Network diffusions via neural mean-field dynamics. In *Advances in Neural Information Processing Systems 33*, 2020.

[48] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[50] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[51] David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In *Automata, languages and programming*, pages 1127–1138. Springer, 2005.

[52] Masahiro Kimura and Kazumi Saito. Tractable models for information diffusion in social networks. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 259–271, 09 2006.

[53] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, Dec 2014.

[54] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[55] W. Kutta. Beitrag zur näherungsweisen Integration totaler Differentialgleichungen. *Zeit. Math. Phys.*, 46:435–53, 1901.

[56] Jure Leskovec, Lars Backstrom, and Jon Kleinberg. Meme-tracking and the dynamics of the news cycle. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 497–506, New York, NY, USA, 2009. Association for Computing Machinery.

[57] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.

[58] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 420–429, New York, NY, USA, 2007. Association for Computing Machinery.

[59] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 695–704, New York, NY, USA, 2008. Association for Computing Machinery.

[60] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1):1, 2016.

[61] C. K. Leung, A. Cuzzocrea, J. J. Mai, D. Deng, and F. Jiang. Personalized deepinf: Enhanced social influence prediction with deep learning and transfer learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2871–2880, 2019.

[62] Cheng Li, Jiaqi Ma, Xiaoxiao Guo, and Qiaozhu Mei. Deepcas: An end-to-end predictor of information cascades. In *Proceedings of the 26th international conference on World Wide Web*, pages 577–586, 2017.

[63] Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*, 18(1):5998–6026, 2017.

[64] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1852–1872, 2018.

[65] Yuxuan Liang, Zhongyuan Jiang, and Yu Zheng. Inferring traffic cascading patterns. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances*

in Geographic Information Systems, SIGSPATIAL '17, New York, NY, USA, 2017. Association for Computing Machinery.

[66] Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3276–3285. PMLR, 10–15 Jul 2018.

[67] Brendan Lucier, Joel Oren, and Yaron Singer. Influence at scale: Distributed computation of complex contagion in networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 735–744, New York, NY, USA, 2015. Association for Computing Machinery.

[68] Sahil Manchanda, AKASH MITTAL, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. Gcomb: Learning budget-constrained combinatorial algorithms over billion-sized graphs. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 20000–20011. Curran Associates, Inc., 2020.

[69] Gonzalo Mateos, Santiago Segarra, Antonio G. Marques, and Alejandro Ribeiro. Connecting the dots: Identifying network structure via graph signal processing. *IEEE Signal Processing Magazine*, 36(3):16–43, May 2019.

[70] Joel C Miller and Istvan Z Kiss. Epidemic spread in networks: Existing methods and current challenges. *Mathematical modelling of natural phenomena*, 9(2):4, 2014.

[71] Seth A. Myers and Jure Leskovec. On the convexity of latent social network inference. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, pages 1741–1749, Red Hook, NY, USA, 2010. Curran Associates Inc.

[72] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14(1):265–294, 1978.

[73] Mark Newman. *Networks: an introduction.* Oxford University Press, 2010.

[74] Hung T. Nguyen, Tri P. Nguyen, Tam N. Vu, and Thang N. Dinh. Outward influence and cascade size estimation in billion-scale networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1), June 2017.

[75] Hung T. Nguyen, My T. Thai, and Thang N. Dinh. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 695–710, New York, NY, USA, 2016. Association for Computing Machinery.

[76] Naoto Ohsaka, Takuya Akiba, Yuichi Yoshida, and Ken-ichi Kawarabayashi. Fast and accurate influence maximization on large networks with pruned monte-carlo simulations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[77] George Panagopoulos, Fragkiskos Malliaros, and M Vazirgiannis. Multi-task learning for influence estimation and maximization. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2020.

[78] George Panagopoulos, Fragkiskos D. Malliaros, and Michalis Vazirgianis. Influence maximization using influence and susceptibility embeddings. *Proceedings of the International AAAI Conference on Web and Social Media*, 14(1):511–521, May 2020.

[79] Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Reviews of modern physics*, 87(3):925, 2015.

[80] Michael Poli, Stefano Massaroli, Junyoung Park, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Graph neural ordinary differential equations, 2019.

[81] Michael Poli, Stefano Massaroli, Atsushi Yamashita, Hajime Asama, and Jinkyoo Park. Hypersolvers: Toward fast continuous-depth models, 2020.

[82] Lev Semenovich Pontryagin. *Mathematical theory of optimal processes*. CRC press, 1987.

[83] Chen Qiao, Yan Shi, Yu-Xian Diao, Vince D. Calhoun, and Yu-Ping Wang. Log-sum enhanced sparse deep neural network. *Neurocomputing*, 407:206–220, 2020.

[84] Jiezhong Qiu, Jian Tang, Hao Ma, Yuxiao Dong, Kuansan Wang, and Jie Tang. Deepinf: Social influence prediction with deep learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2110–2119, 2018.

[85] Yu Rong, Qiankun Zhu, and Hong Cheng. A model-free approach to infer the diffusion network from event cascade. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1653–1662, New York, NY, USA, 2016. Association for Computing Machinery.

[86] Yulia Rubanova, Ricky T. Q. Chen, and David K Duvenaud. Latent ordinary differential equations for irregularly-sampled time series. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[87] C. Runge. Ueber die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46:167–178, 1895.

[88] Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations, 2018.

[89] Faryad Darabi Sahneh and Caterina Scoglio. Epidemic spread in human networks. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 3008–3013. IEEE, 2011.

[90] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

[91] Thomas C Schelling. *Micromotives and macrobehavior*. WW Norton & Company, 2006.

[92] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[93] Youze Tang, Yanchen Shi, and Xiaokui Xiao. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1539–1554. ACM, 2015.

[94] Youze Tang, Xiaokui Xiao, and Yanchen Shi. Influence maximization: Near-optimal time complexity meets practical efficiency. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, pages 75–86, New York, NY, USA, 2014. Association for Computing Machinery.

[95] V.V. Vazirani. *Approximation Algorithms*. Springer Berlin Heidelberg, 2002.

[96] Maurice Vergeer, Liesbeth Hermans, and Steven Sams. Online social networks and micro-blogging in political campaigning the exploration of a new campaign tool and a new campaign style. *Party Politics*, 19(3):477–501, 2013.

[97] Liaoruo Wang, Stefano Ermon, and John E Hopcroft. Feature-enhanced probabilistic models for diffusion network inference. In *Machine Learning and Knowledge Discovery in Databases*, pages 499–514. Springer, 2012.

[98] Xiaoyang Wang, Ying Zhang, Wenjie Zhang, Xuemin Lin, and Chen Chen. Bring order into the samples: A novel scalable method for influence maximization. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):243–256, 2016.

[99] David Williams. *Probability with martingales*. Cambridge university press, 1991.

[100] Jennifer Wortman. Viral marketing and the diffusion of trends on social networks. *Technical Reports (CIS)*, page 880, 2008.

[101] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–21, 2020.

[102] Guandao Yang, Xun Huang, Zekun Hao, Ming-Yu Liu, Serge Belongie, and Bharath Hariharan. Pointflow: 3d point cloud generation with continuous normalizing flows. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019.

[103] Yaohua Zang, Gang Bao, Xiaojing Ye, Hongyuan Zha, and Haomin Zhou. A jump stochastic differential equation approach for influence prediction on heterogenous networks. *Communications in Mathematical Sciences*, 18(8):2341–2359, 2020.

[104] Huaguang Zhang, Zhanshan Wang, and Derong Liu. A comprehensive review of stability analysis of continuous-time recurrent neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 25(7):1229–1262, 2014.

[105] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. Social influence locality for modeling retweeting behaviors. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 2761–2767. AAAI Press, 2013.

[106] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6:1–23, 2019.

[107] Yaochen Zhu, Jiayi Xie, and Zhenzhong Chen. Predicting the popularity of micro-videos with multimodal variational encoder-decoder framework. *arXiv preprint arXiv:2003.12724*, 2020.