Information diffusion analysis in online social networks based on deep representation learning

Chen, X.

Information Diffusion Analysis in Online Social Networks based on Deep Representation Learning

Xueqin Chen

Information Diffusion Analysis in Online Social Networks based on Deep Representation Learning

Proefschrift

ter verkrijging van
de graad van doctor aan de Universiteit Leiden,
op gezag van rector magnificus prof.dr.ir. H. Bijl,
volgens besluit van het college voor promoties
te verdedigen op dinsdag 25 oktober 2022
klokke 15.00 uur

door

Xueqin Chen

geboren te Sichuan, China
in 1993

Promotores:
Prof. dr. M.M. Bonsangue
Prof. dr. F. Zhang (Univ. of Electronic Science and Technology of China)
Co-promotor:
Dr. F. Zhou (Univ. of Electronic Science and Technology of China)

Promotiecommissie:
Dr. E. Isufi (Delft University of Technology)
Prof. dr. A.A. Khokhar (Iowa State University)
Prof. dr. A. Plaat
Prof. dr. F. Verbeek
Dr. S. Verberne

ISBN 978-94-6469-057-6

The research in this thesis was performed at the Leiden Institute of Advanced Computer Science (LIACS, Leiden University) and the University of Electronic Science and Technology of China.

This work is financially supported by the Chinese Scholarship Council (CSC No. 201906070093)

Contents

Chapter 1

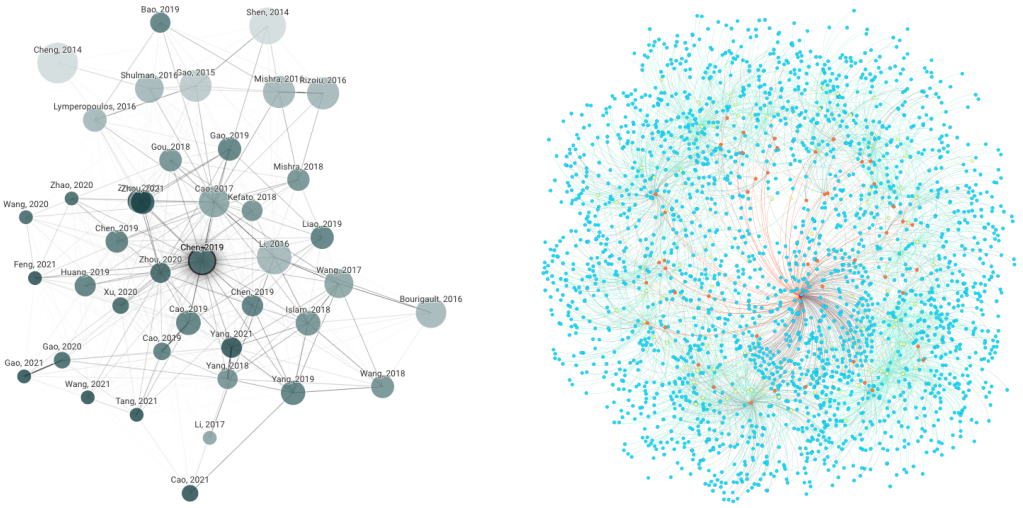
Introduction

1.1 Background

According to Pew Research[1], as of 2017 approximately 88% of American adults have either free or paid Internet access at home, and about 81% obtain news from online platforms (e.g., news websites/apps, social media, or both). We can see that with the rapid development of Internet technology, the way people share information has gradually democratized. Especially, the rise of large online social network (OSN) platforms, such as Twitter[2], Sina Weibo[3], Facebook[4], etc., have provided an environment for free information creation and distribution while substantially changing how people acquire and share information. In a nutshell, everyone can generate and share various contents and communicate on topics of mutual interest on these platforms without hindrance. Such activities facilitate the fast diffusion of information (both true and false) in various contexts, for example, the spread of rumors in News, the propagation of marketing campaigns, the diffusion of innovative technological achievements, and so on, which spur the phenomenon of information cascades. Information cascade is formed as information or innovative ideas propagated among users [1], which is ubiquitous and has been identified in various settings: e.g., paper citations [2], blogging space [3, 4], email forwarding [5, 6], as well as in social medias (e.g., Twitter [7, 8] and Sina Weibo[9]). Figure 1.1 shows the example for citation cascade and retweet cascade, respectively.

Understanding how information spreads through OSNs, as well as what elements drive the success of information diffusion, and making forecasts about the population size that information can affect, has significant implications for a wide range of real-world applications [11], such as marketing viral discrimination [12], user behavior prediction [13], media advertising [14], social recommendation [15], and fake

[1]https://www.journalism.org/2016/07/07/pathways-to-news/
[2]https://twitter.com/
[3]https://weibo.com/
[4]https://www.facebook.com/



(a) Citation cascade **(b)** Retweet cascade

Figure 1.1: Examples of cascade for paper citation and tweet retweet. (a) A citation cascade for CasCN [10], a screenshot from the website – Connected Paper (https://www.connectedpapers.com/). (b) A retweet casdcade from Weibo.

news detection [16], etc. Therefore, analyzing the diffusion of a given piece of information is urgently expected by academia and industry. Among various information diffusion analysis tasks, modeling and predicting the information cascade turns out to be of utmost importance since it enables controlling (or accelerating) information spreading in various scenarios, which is the first research problem we try to tackle in this thesis.

Furthermore, OSNs are a double-edged sword. On the one hand, OSNs brought enormous convenience to people's daily life. On the other, the proliferation of fake news, rumors and false information has had strong and negative societal and economic consequences. The explosive spread of false information can pose a threat to the credibility of legitimate online platforms and resources and has a serious negative impact on both individuals and society [17], with the potential consequences to destabilize nations, affect the fairness of competition [18], and shock the stock market [19]. Take the more recent event as an example. In the global effort to contain the COVID-19 pandemic, misinformation abounds and flourishes on the Internet [20, 21], and people have been led to believe that COVID-19 can be cured by ingesting fish tank cleaning products or that 5G networks generate radiation that triggers the virus. Such misinformation not only causes panic among citizens but could potentially undercut collective efforts to control the pandemic. Thus, detecting rumors on OSNs as early as possible is a necessary, urgent and socially beneficial task, which is the second research problem studied in this thesis.

In the past decade, researchers made enormous efforts and remarkable progress in trying to find effective solutions to the above two problems, i.e., information cas-

cade modeling and rumor detection. Specifically, prior works tried to solve these two tasks by extracting various hand-crafted features (e.g., content features [22, 23, 24], user features [25, 26], and social context features [27, 28, 29], etc.) and incorporating the power of machine learning models (e.g., random forest, naive Bayes, and support vector machines). However, these well-designed features required extensive domain knowledge and thus are hard to be generalized to new domains. Besides that, researchers also proposed point process-based approaches [2, 30] to do information cascade modeling, focusing on modeling the intensity function of the arrival process for each message independently. These methods demonstrate an enhanced interpretability but are still unable to fully leverage the information encoded in the cascade for a satisfactory prediction. And as for rumor detection, some researchers leveraged the inter-entity relations and constructed the so-called credibility network [31, 32] to find the truth against conflicting information, but the performance of these models heavily relies on the quality of the credibility network used.

Inspired by the recent success of deep representation learning in computer vision and natural language processing, recently, researchers began to employ deep learning techniques to develop models for information cascade modeling [8, 9] and rumor detection [33, 34]. These models aim to learn more powerful high-level feature representations directly from raw data via various deep learning techniques, which improve model performance and alleviate the heavy manual effort in conventional methods at the same time. However, the existing deep learning-based approaches in information cascade modeling and rumor detection still face some limitations, such as incomplete feature extraction, inefficient feature fusion, absence of fine-grained feature learning, and so on. The core work of this thesis is to eliminate these problems, focusing on the development of **deep learning-based models** to solve the problem of **information cascade modeling** and **rumor detection**.

1.2 Research Questions and Contributions

This thesis is structured in two parts: Part I will introduce deep learning-based models for information cascade modeling task. And Part II will focus on the topic of rumor detection.

Information cascades modeling is accomplished via specific prediction tasks, which are categorized into two levels: Micro-level and Macro-level. (1) At micro-level, local patterns of social influence are studied – e.g., inferring the action status of a user [35, 36]. The methods predict the likelihood of a user propagating a particular piece of information, or forecast when the next propagation might occur given a certain information cascade [36]. (2) At macro-level, typical studies include cascade size prediction [8, 9, 35, 37, 38] and outbreak prediction (above a certain threshold) [1, 7, 37, 39], both cascade size (popularity) prediction and outbreak prediction are aiming

to estimate the future size (popularity) of the diffusion cascade. Because micro-level tasks requires the complete diffusion network/social network, we need to know the historical interactions (retweet/social relationship (follower/following)) among all users, which is extremely large and hard to acquire. Therefore, in this thesis, we focus on **macro-level information cascade prediction**, and the research questions and corresponding contributions of Part I are listed as follows:

Research Question 1 (RQ 1) *Can we develop an effective deep learning-based model to capture structural and temporal features from the observed cascade graph for macro-level information cascade prediction?*

Earlier deep learning-based approaches either only extracted temporal features from the diffusion paths [9] but ignored the structural features or learned structural features from the global graph [8]. The key challenge of **RQ 1** is how to learn structural and temporal features at the same time when only given the observed cascade graph. Our key contribution to address this research question is the design of a novel graph-based neural network called **Recurrent Cascades Convolutional Network (CasCN)**. CasCN is introduced in Chapter 4 and based on the following publication[10], that to the best of our knowledge, is the first work to study this problem :

- **Chen, X.**, Zhou, F., Zhang, K., Trajcevski, G., Zhong, T., Zhang, F.: Information diffusion prediction via recurrent cascades convolution. In: 2019 IEEE 35th International Conference on Data Engineering. ICDE '19 (2019) 770–781

CasCN is designed based on the long short-term memory network [40] and the graph convolutional network [41], which captures structural-temporal features from a sequence of subgraphs – sampled from the observed cascade graph. CasCN also introduces a new way to calculate the Laplacian, allowing it to handle directed graphs. We demonstrate significant advantages of using CasCN compared with earlier state-of-the-art methods both in terms of feature learning and predictive accuracy.

Research Question 2 (RQ 2) *How can we improve upon earlier deep learning-based models that learn the latent representation for the observed cascade graph from a multi-scale perspective to predict the future size of this cascade?*

Existing works demonstrate the effectiveness to extract structural-temporal features for macro-level information cascade prediction. However, they still fail to take the higher-order and position information for each node into consideration in modeling the structure of information diffusion, which carries out our second research question **RQ 2**. To address RQ 2, we introduce a novel graph-based model – **Multi-scale Cascades model (MUCas)** in Chapter 5 based on the following publication [42]:

- **Chen, X.**, Zhang, F., Zhou, F., Bonsangue, M.: Multi-scale graph capsule with influence attention for information cascades prediction. International Journal of Intelligent Systems 37 (2022) 2584–2611

MUCas makes full use of the direction-scale, high-order-scale, position-scale, and dynamic-scale of cascades via a newly designed multi-scale graph capsule network (MUG-Caps) and the influence-attention mechanism. And the experiments on two real-worlds datasets demonstrate the superiority of the proposed model on macro-level information cascades prediction.

Having introduced deep learning-based models for macro-level information cascade prediction task in Part I of this thesis (Chapter 4 and 5). In Part II, we will explore the possibility to detect rumors by extracting various diffusion patterns. Besides that, we also investigate the importance of users in the diffusion of rumors by developing participant-level rumor detection models.

Research Question 3 (RQ 3) *Can we detect rumors at an early stage by learning various diffusion patterns from the information diffusion?*

Rumors are created by mimicking the real news, which aims to mislead the public, making it difficult to be detected by using textual and visual features. A recent empirical study [43] demonstrated that rumors and non-rumors show different diffusion patterns. The key challenge of **RQ 3** is how to develop an effective deep learning-based model to explore the full-scale diffusion patterns of rumors, i.e., from both macroscopic and microscopic perspectives. In Chapter 6, we design a novel diffusion-based rumor detection model to solve **RQ 3**, called **Macroscopic and Microscopic-aware Rumor Detection model (MMRD)**, this chapter is based on the following publication[44]:

- **Chen, X.**, Zhou, F., Zhang, F., Bonsangue, M.: Modeling microscopic and macroscopic information diffusion for rumor detection. International Journal of Intelligent Systems 36 (2021) 5449–5471

MMRD leverages graph neural networks to learn the macroscopic diffusion of rumor propagation and capture microscopic diffusion patterns using bidirectional recurrent neural networks while taking into account the user-time series. Moreover, it leverages knowledge distillation technique to create a more informative student model and further improve the model performance. Experiments conducted on two real-world data sets demonstrate that our method achieves significant accuracy improvements over the state-of-the-art baseline models on rumor detection.

Research Question 4 (RQ 4) *Can we improve the model performance by developing effective rumor detection models at the participant level?*

Users are the main contributor to rumor spreading in online social networks, modeling the rumor spreading at a more fine-grained participant-level rather than event-level. This hypothesis improve detection accuracy and is at the core of research

question **RQ 4**. To answer RQ 4, we proposed two participant-level models in Chapter 7, i.e., **Participant-level Rumor Detection model (PLRD)** and **User-aspect Multi-view Learning with Attention for Rumor Detection model (UMLARD)**. This chapter is based on the following publications [45, 46]:

- **Chen, X.**, Zhou, F., Zhang, F., Bonsangue, M.: Catch me if you can: A participant-level rumor detection framework via fine-grained user representation learning. Information Processing & Management 58 (2021) 102678
- **Chen, X.**, Zhou, F., Trajcevski, G., Bonsangue, M.: Multi-view Learning with Distinguishable Feature Fusion for Rumor Detection. Knowledge-Based Systems 240 (2022) 108085

PLRD aims to learn the users' social homophily, social influence, susceptibility, and temporal features for rumor detection, while UMLARD exploits different embedding methods to learn the view-specific high-level representations of a given post from the hierarchical diffusion process and user profiles. Both models are designed at participant-level, and experimental results show their superiority over state-of-the-art methods.

1.3 Thesis Outline

The overall organization of this thesis is as follows. In Chapter 1, we give a brief introduction to the thesis' background and its motivation, the main research questions and contributions, and the overview of this thesis. Chapter 2 provides a comprehensive literature review for existing methods in information cascade modeling and rumor detection, and focuses more on deep learning-based methods. In Chapter 3 presents some general definitions, which are throughout the whole dissertation and problem definitions of macro-level information cascades prediction and rumor detection. Besides that, Chapter 3 also briefly introduces some related technical supports.

The main content of this thesis related to the research questions is divided into two parts. Part I (Chapter 4 and Chapter 5) introduces two deep learning-based models for macro-level information cascades prediction. And Part II (Chapter 6 and Chapter 7) focuses on the task of rumor detection.

Specifically, in Chapter 4, we target **RQ 1** and propose the first graph-based neural network, which extracts structural-temporal features from a sequence of subgraphs and makes a prediction of the incremental size of the cascade. Chapter 5 address **RQ 2** and introduces a multi-scale graph capsule network to fully explore the direction-scale, high-order-scale, position-scale, and dynamic-scale information from the observed cascade graph.

Chapter 6 is related to **RQ 3** and introduces a novel deep learning model for rumor detection by exploring the microscopic and macroscopic diffusion patterns. In

Chapter 7, we target **RQ 4** and propose two participant-level model for rumor detection.

At last, Chapter 8 concludes the contributions of the thesis and discusses possible future work.

Chapter 2

Literature Review

In this chapter, we provide a brief review of the studies that are relevant to this thesis. We focus on summarizing the existing methods for two specific tasks: information cascade modeling and rumor detection. Existing methods for both tasks can be divided into two groups, i.e., conventional and deep learning-based methods. Since this thesis is inspired by deep learning, our literature review concentrate more on deep learning-based methods and only briefly describe conventional methods.

2.1 Information Cascades Modeling

As we mentioned in Section 1.2, in this thesis, we focus on modeling the information cascade through macro-level information cascade prediction tasks. The macro-level information cascade prediction tasks aim at modeling the cascade scale via estimating the future popularity of the diffusion cascade. The information cascade is a phenomenon caused by information transmission from one user to another based on social interactions (e.g., follower/following) in OSNs, which always used to describe the information diffusion and consists of the trajectories and structures of information diffusion, as well as the participants in information spreading [11].

2.1.1 Conventional methods

Conventional methods in macro-level information cascade prediction mainly fall into two categories: (1) point process-based methods, and (2) handcrafted feature-based methods.

Point process-based methods: Most individual activities in the social system can be described as a point process [47]. The point process-based methods regard message diffusion as the arrival process of users' retweet behavior. Specifically, these methods focus on modeling the intensity function in the arrival process for each message independently, it observes each event within the observation window and learns

the parameters through maximizing the probability of events occurring during a period of time. Typical point process methods include Poisson process [2, 30, 48] and Hawkes process [49, 50]. Both of these point processes are committed to describing the key factors in message diffusion: (1) self-influence–i.e., each retweet user will influence the trend of future retweets, (2) time-decay effect–i.e., the influence of a retweet user decays with the time elapsed, and (3) rich-get-richer phenomenon–i.e., a message shared by influential users will get more retweets. Shen et al. [2] upgraded the Poisson process to reinforced Poisson processes (RPP) to model stochastic popularity dynamics and then incorporate it into the Bayesian framework for external factors inference and parameter estimation. PETM [30] improved RPP by introducing a power-law temporal relaxation function, an exponential reinforcement function and time mapping process. Gao et al. [48] split the complete diffusion process into many subprocesses and used RPP to model the subprocesses, which makes the proposed model efficient when trained on a single tweet. Mishra et. al. [49] present a hybrid predictor which combines Hawkes self-exciting point process for modeling each cascade and leverages feature-driven method for estimating the content virality, memory decay, and user influence. Later work HIP (Hawkes Intensity Process) [50] extended the original Hawkes process, which can explain the complex popularity history of each video according to its type of content, diffusion network, and sensitivity to promotion. As previously stated, the point process-based methods learn the intensity function for each event within the observation window, and learn the parameters by maximizing the occurrence probability, which can capture the dynamic process of the message re-sharing behavior, hence, have good comprehensibility. However, these methods hold the hypothesis that historical events will always excite future events, which is obviously not true in real life. Furthermore, these methods are not directly supervised by popularity, so there is a gap between modeling and prediction, which has hampered model performance in information cascade prediction.

Handcrafted feature-based methods: Theses methods extracted various handcrafted features from raw data. They typically include content features [22, 38, 51, 52], user features [25, 39, 53], structural features [27, 37, 54] and temporal features [28, 55], and then feed these features to discriminative machine learning algorithms to perform the cascades prediction tasks. Tsur et al. [38] demonstrated that combining content features with other types of features, e.g., temporal and structural features will reduce the prediction error. Bakshy et al. [25] studied the features related to early adopters and found that user features are informative predictors. Recently, in spite of exploring informative features, Shulman et al. [56] compared the predictive power of models using different sets of features and found that temporal features are the most predictive, almost as accurate as using content and user features. Cheng et al. [1] cast the information cascade size prediction as a classification task and concluded both temporal and structural features are almost

equally useful in predicting cascade size with an accuracy of 0.622 and 0.620, respectively. Summarizing, the performance of feature-based methods heavily depends on the hand-craft features, but there is not a standard and systematic way to design these features.

2.1.2 Deep learning-based methods

With the rapid advancement of deep learning in computer vision and natural language processing, researchers have developed a number of deep models to solve the problem of macro-level information cascades modeling and prediction. The key idea of such deep learning-based models is to automatically extract various diffusion features from the input cascades by leveraging different kinds of neural networks.

DeepCas [8] first demonstrated the effectiveness of deep neural networks in modeling information cascades. It first transformed the cascade graph as a set of node sequences by random walk and then automatically learned the structural features of individual graphs using GRU [40] and the attention mechanism. Li et al. extended DeepCas to DCGT [57] by incorporating content features. DeepHawkes [9] extracted temporal features by modeling diffusion paths via GRU rather than the random walks in DeepCas, and proposed the non-parametric time-decay effect to further improve the prediction performance, which bridged the gap between deep representation learning and the conventional Hawkes process. Gou et al. [58] proposed LSTMIC, which first converted the retweeting time series into several viewpoints, and then employed a long short-term memory (LSTM) architecture and pooling mechanism to extract sequential temporal features for information outbreak prediction. NT-GP [59] extracted node sequences from the user's activity log using the time decay sampling method, and then uses gated recurrent units (GRUs) to learn the temporal features from the sampling sequences and predict the target event's future diffusion range. The latest work TempCas [60] introduced a heuristic method for sampling full critical paths and it was shown to be more powerful than random walks and diffusion paths. It uses BiGRU with attention pooling for path embedding while modeling the short-term outbreaks and the impact of historical short-term outbreaks with an attention convolutional neural network (CNN) and an LSTM. Chen et al. [10] proposed the first graph neural networks (GNNs) based model called CasCN. It learned the structural and temporal information from sub-cascade graphs via a combination of graph convolutional network (GCN) [61] and LSTM, which also took into account the diffusion direction and time-decay effect. Later, some works [62, 63, 64] were built upon the CasCN through changing the graph kernel or using different sampling methods. For example, Cascade2Vec [62] improved the convolutional kernel of CasCN with the idea from graph Isomorphism network (GIN) [65] and residual networks [66]. Xu et al. proposed CasGCN [63], which first represented cascade graph as an in-coming graph and an out-coming graph, and then applied GCN to learn the structural features from both in-coming and out-coming

cascade graphs. The temporal features are learned through the normalization of diffusion time. CasSeqGCN [64] assumed that each sampled sub-cascade graph has the same topology but with a different state vector. Huang et al. proposed a graph sequence attention network – GSAN [67], which captured bi-directional and long dependencies between sub-cascade graphs via a collaboration of graph transformer block and a sequence transformer block. Another work [68]–CoupledGNN learned the cascading effect in information diffusion via coupled GNNs, towards capturing the interpersonal influence and individual user behavior based on the global graph. VaCas [69] first used the unsupervised graph wavelet to learn the structural information for cascade graphs, and employed variational autoencoder (VAE) [70] to enhance the cascade representation learning. And MUCas [42] tried to learn complete structural features from a multi-scale perspective.

Furthermore, some existing works attempt to extract temporal and structural information by performing both micro-level and macro-level tasks concurrently using multi-task learning [71] or reinforcement learning [72]. Also, some works have emerged to solve the general problem inherent in deep cascade learning, such as catastrophic forgetting [73], long-tail data distribution [74], and model generalization [75, 76].

2.2 Rumor Detection

The problem of rumor (or fake news/information, misinformation) detection is an important research topic in recent social media studies and receives increased attention in various disciplines including politics [18], finance [19], marketing [12], healthcare [77], etc. “Rumor" is usually defined as a misleading story or misinterpretation of information, circulating among communities and pertaining to an object, event, or issue in public concern [43].

2.2.1 Conventional methods

Handcrafted feature-based methods: Most of earlier works extracted various hand-crafted features from raw data, which can be typically summarized as two types: (1) content features extracted from both text (e.g. characters, words, sentences and documents) and visual elements (e.g. images and videos), which can be further partitioned as lexical features [23, 29, 78], syntactic features [23, 79, 80], topic features [81], visual statistical features [24, 77], and visual content features [82]; and (2) social context features extracted from the user behavior and the diffusion network, which reflect the relationship among users and describe the diffusion process of a rumor, including user features [23, 26, 83], propagation features [29, 81, 84], and temporal features [29, 85]. After feature engineering, the selected features are used in discriminative machine learning algorithms (e.g., random forest, naive Bayes, and support vector machines) to classify the news or tweets.

Rumors aim to arouse much attention and stimulate the public mood. Therefore, their texts/images/videos tend to have certain patterns in contrast to truth. Zhao et al. [78] discovered two types of language patterns in rumors, i.e., inquiry and correction patterns, and detected the patterns of rumor messages through supervised feature selection on a set of labeled messages. Wu et al. [81] defined a set of topic features to summarize semantics and trained a Latent Dirichlet Allocation (LDA) model for detecting rumors on Weibo. Towards a more comprehensive understanding of the text on social media, existing works also come up with textual features derived from social media platforms, apart from general textual features, such as source links [77] and emotions [23]. As for visual content features, Jin et al. [82] found that images in rumors and non-rumors are visually distinctive on their distributions and propose five visual features to measure the rumors, i.e., visual clarity score, visual coherence score, visual similarity distribution histogram, visual diversity score, and visual clustering score. Social context features are derived from the social connection characteristics of social media. Rumors are usually created by a few users and spread by a large number of users. Therefore, user profiles are commonly used to measure the user's characteristics and credibility. For example, Castillo et al. [23] first identified the credibility of tweets on Twitter based on user features. Diffusion patterns, i.e., structural patterns and temporal patterns, are also shown to be effective for detecting rumors. Kwon et al. [29] extended the work of [23] by proposing 15 structural features extracted from the diffusion network and the user friendship network. In the work [85], the authors proposed a method for discretizing time and capturing the variation of temporal features associated with rumors.

However, the performance of feature-based methods heavily depends on the handcraft features, which lacks a standard and systematic way to design general features across platforms and to deal with different types of rumors. In fact, the conclusions of existing works usually contradict each other, primarily due to the differences between different types of datasets. For example, Yang et al. [86] designed a set of features (e.g., client-based features and location-based features) based on Weibo, whose users are mainly restricted to China. It is therefore difficult to use these features for detecting rumors spread on Twitter and Facebook due to the differences in languages, clients' and users' geographic distributions, etc.

Credibility Propagation-based methods: Inspired by the work of truth discovery that aims to find truth with conflicting information, this line of methods consists of two main steps, i.e., (1) credibility network construction and (2) credibility propagation. The underlying assumption of these methods is that the credibility of news is highly related to the reliability of relevant social media posts, and both homogeneous and heterogeneous credibility networks can be built for the propagation process. Homogeneous credibility networks consist of a single type of entity, such as posts and events. In contrast, heterogeneous credibility networks involve different

types of entities, such as posts, sub-events, and events. Gupta et al. [24] first introduced a PageRank-like credibility propagation algorithm by encoding users' credibility and tweets' implications on a user-tweet-event information network. Inspired by the idea of linking entities altogether and leveraging inter-entity connections for credibility propagation, Jin et al. [31] proposed a three-layer hierarchical credibility network, which includes news aspects and utilizes a graph optimization framework to infer event credibility. The work in [32] found that relations between messages on microblogs (i.e. support and oppose) are crucial for evaluating the truthfulness of news events, and built a homogeneous credibility network among tweets to guide the process of credibility evaluation. While comparing with direct classification on the individual entity, credibility propagation-based methods may leverage the inter-entity relations for robust detection results. However, the performance of these methods strongly relies on the constructed credibility network.

2.2.2 Deep learning-based methods

The deep learning-based models have shown improved performance over traditional methods due to their enhanced ability to automatically representation learning. Most existing deep learning-based methods are content-aware that mainly focused on extracting textual features [33, 87, 88, 89, 90] and visual features [91, 92] from news content, user comments, and images, etc. Ma et al. [33] proposed the first deep learning-based rumor detection model, which applies recurrent neural networks (RNN) to model rumors as varied length time series aimed to learn both textual and temporal features from raw data and thus detect rumors. Shu et al. [87] proposed a co-attention network to exploit both news content and user comments for rumor detection while discovering explainable sentences. Jin et al. [91] presented a model to extract the visual, textual, and social context features, which are fused by the attention mechanism. Moreover, researchers also employed other deep learning techniques, such as multi-task learning [93], transformer [94, 95, 96, 97], and knowledge enhancement [98], to learn more robust content-aware features for rumor detection. However, rumors are intentionally written by mimicking real news [99], which makes content-aware methods hard to further improve detection performance due to the lack of necessary domain knowledge.

Recently, a few works have tried to exploit diffusion patterns in news spreading for rumor detection, e.g., temporal features [44, 100, 101] and structural features [16, 34, 44, 102]. For example, Liu et al. [100] presented a time series classifier with RNN and CNN to predict whether a given news story is fake at an early stage, taking common user characteristics and propagation paths into consideration. Song et al. [101] proposed a temporal propagation-based model that can distinguish rumors from true news through modeling dynamic evolution patterns of news. As for the structural features, Ma et al. [34] presented a tree-structured RNN to catch the hidden representations from both propagation structures and text contents. Inspired

by the success of graph neural networks in information cascades modeling [10, 71], Bian et al. [16] proposed a graph convolutional network [61]-based model that can learn global structural relationships of rumor dispersion. He et al. [103] improved the work [16] by using event augmentation and contrastive learning. Similarly, Lu et al. [102] improved the work of [100] by calculating the similarity between users and used a graph-aware attention network for rumor detection.

Furthermore, researchers began to consider both structural and temporal features for rumor detection. We [44] introduced a hierarchical diffusion modeling model by extracting both temporal features and propagation structures from the microscopic diffusion and macroscopic diffusion jointly. In addition, some researchers realized that users play significant roles in rumor spreading. For example, we proposed PLRD [45], which extracted social homophily, influence, and susceptibility of users from the user interaction network for rumor detection. UMLARD [46] improved PLRD by considering the disentangled feature learning and introducing textual features. Dou et al. [104] proposed a user preference-aware rumor detection model to learn user endogenous preference and exogenous context from users' historical posts and reply network, respectively.

Chapter 3

Preliminaries

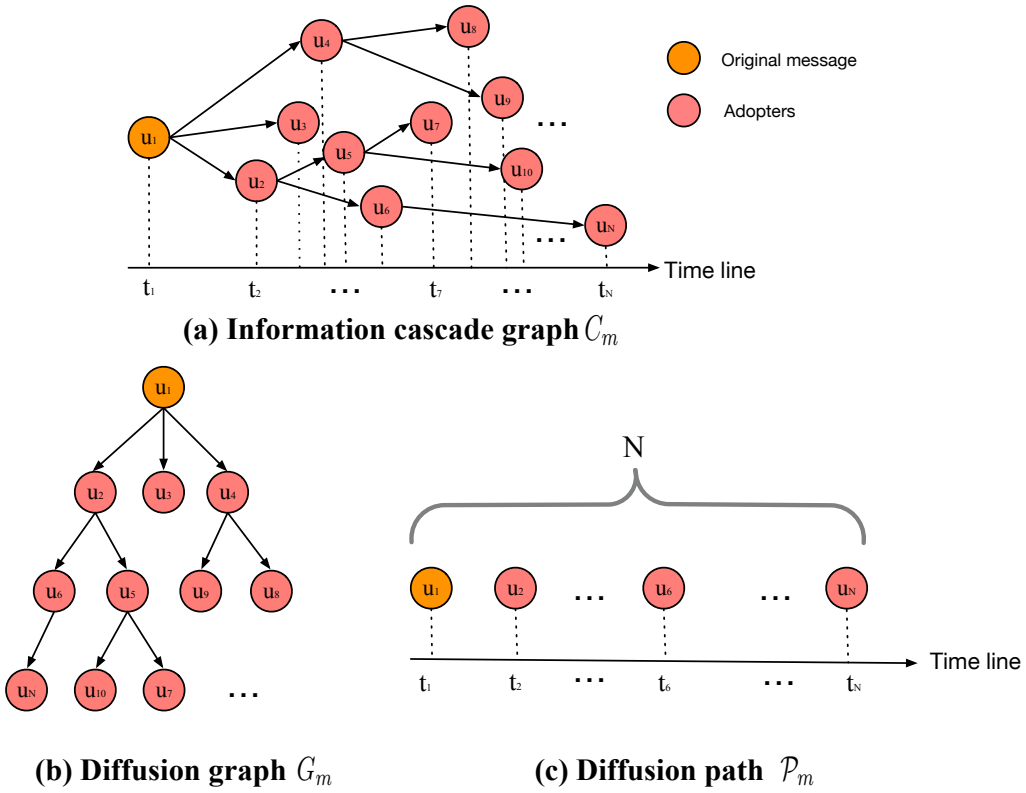
In this chapter, we introduce general definitions about information cascades, the basic formal problem statements forming the core of our research, and few technical concepts needed in the rest of the thesis. Table 3.1 summarizes the main notations used in this chapter.

Table 3.1: Main notations used throughout this chapter.

Symbol	Description
m	An information item, e.g., a tweet or a paper.
C_m	Information cascade graph regarding m.
G_m	Diffusion graph regarding m.
$\mathcal{P}_m$	Diffusion path regarding m.
U_m	Nodes set.
E_m	Edges set
T_m	Time mapping function.
t_o	Observation time.
t_p	Prediction time.
$C_m(t_o)$	Observed cascade graph regarding m.
$G_m(t_o)$	Observed diffusion graph regarding m.
$\mathcal{P}_m(t_o)$	Observed diffusion path regarding m.
$\mathbf{u}$	User vector.
$\mathbf{U}_m$	User characteristic matrix.
d_{user}	Dimension of user vector $\mathbf{u}$.
$S(t_p)$	Popularity label.

3.1 General Definitions

To begin with, we present some general definitions, which are used throughout the whole thesis. The chapter-specific definitions are presented in the corresponding chapter.



(a) Information cascade graph C_m

(b) Diffusion graph G_m

(c) Diffusion path $\mathcal{P}_m$

Figure 3.1: An example of information cascade.

Online social platforms, such as Twitter and Weibo, allow users to post and re-share various contents and communicate on topics of mutual interest. Such activities facilitate fast diffusion of information to reach a large number of users and spur the phenomenon of **information cascades**. Given an information item, such as a message m we denote by N the number of its corresponding adopters, i.e. those users who post or re-share the information in the online social platform, each associated with a user-specific adoption timestamp. According to the user interaction (e.g., retweet, citation, etc.), we can construct the information cascade graph for m.

Definition 1 ***Information Cascade Graph.*** *An* **evolving cascade graph** *for m is defined as a dynamic tree $C_m = \{U_m, E_m, T_m\}$, where U_m is a finite set of nodes, $E_m \subseteq U_m \times U_m$ is a set of edges such that (U_m, E_m) forms a tree, and T_m is function associating to each node in $u \in U_m$ a timestamp t_u.*

In the example of Twitter, a node $u \in U_m$ can represent a user who tweets or re-tweets the post m from some sources (e.g., other users) in the social networks. A directed edge $(u_1, u_2) \in E_m$ denotes the retweet relationship between the users u_1 and u_2. The timestamp $t_u = T(u)$ represents the time when the retweeting behavior occurs for u, that we tacitly assume to be temporally after the time of u's parent. Clearly, time ordering forbids having loops in the graphs. Note that in real life, the same user can be involved in multiple adoptions of the same message, but at different times. This would imply that the number N of users adopting the message m is typically larger than $|U_m|$. However, in this thesis, we only keep track of the earliest

adoption of an information item by a user even if the same user adopt it multiple times. This way we have that the size of U_m is exactly N, i.e., $|U_m| = N$.

In order to describe the diffusion process of information item m in a more fine-grained way, we further break down the information cascade into a diffusion graph and diffusion path.

Definition 2 ***Diffusion Graph.*** *The information cascade graph of m is denoted as $C_m = \{U_m, E_m, T_m\}$, we define the diffusion graph for the message m with as its underlying tree $G_m = \{U_m, E_m\}$.*

In a diffusion graph G_m thus, nodes denote users are involved in the diffusion process of the message m, and the edges represent the adoption (e.g., re-tweet) from the user at the target of the edge of the message published by the user on the source of the edge.

Definition 3 ***Diffusion Path.*** *A diffusion path of message m is defined as a multivariate time series $\mathcal{P}_m = \{(u_1, t_1), \cdots, (u_N, t_N)\}$, where $t_1 \leq t_2 \leq \ldots \leq t_N$.*

Here each pair (u_j, t_j) indicates that the user u_j adopts the message m at time t_j. We assume that the diffusion path $\mathcal{P}_i$ can be totally ordered, implying that when two timestamps for different user are the same, i.e. when $t_i = t_j$ for some $i \neq j$, then the order in the sequence is determined by the order of the nodes (for example based on the ordering of the user IDs which are assumed to be unique). The first user u_1 denotes the source user (i.e., the one who initially posted the message m at time t_1), whereas the rest of the users u_i for $2 \leq i \leq N$, are those participating in spreading the information.

From every information cascade graph C_m we can derive its diffusion path by taking the sets of all pairs (u, t_u) with u a node in C_m and t_u its associated timestamp.

The diffusion graph and the diffusion path derived by the information cascade graph at the top of Figure 3.1 are illustrated at bottom. Even though both the diffusion graph and the diffusion path are abstracted from the diffusion thread of messages, they are independent and different. Specifically, the diffusion graph reflects the direction of message passing between users, while the diffusion path reflects the time and sequential information of user engagement.

In this thesis, we only focus on a part of a information cascade graph, i.e., a partial cascade graph extracted before a specific observation time. The formal definition of observed cascade graph is given as follows:

Definition 4 ***Observed Cascade Graph.*** *Given an information cascade graph $C_m = \{U_m, E_m, T_m\}$ for a message m, and an observation time t_o, we define the observed cascade graph for m until time t_o to be the sub-graph $C_m(t_o)$ determined by all nodes u with time stamp $T(u) \leq t_o$.*

Similarly, the diffusion graph and diffusion path observed until time t_o are represented as $G_m(t_o)$ and $\mathcal{P}_m(t_o)$, respectively.

Users are the main participant in information spreading. From the online social platforms, each user is associated with a number of profiles such as the user id, the number of followers, whether a user is verified or not, etc.. In other words, user can be considered as a vector of different user profiles and all users of an information cascade graph can be assembled into a matrix.

Definition 5 ***User data.*** *Given an information cascade graph $C_m = \{U_m, E_m, T_m\}$ for a message m, assume that every node $u \in U_m$ is a vector $\boldsymbol{u} \in \mathbb{R}^{d_{user}}$, for some fixed number of user profiles d_{user}. We define the user characteristic matrix $\mathbf{U_m}$ for C_m to be a matrix $N \times d_{user}$, where each row is a user vector ordered as in the corresponding diffusion path.*

3.2 Problem Definition

Next we give a brief description of the problem of information cascades modeling and rumor detection.

3.2.1 Information Cascades Modeling

In this thesis, we do cascade modeling via a macro-level prediction task, i.e., cascade size prediction and outbreak prediction. Macro-level information cascades prediction is defined as follows:

Definition 6 ***Macro-level Information Cascades Prediction.*** *Given an observed cascade graph $C_m(t_o)$ of a message m before an observation time t_o, macro-level prediction task aims at learning a predictive function $f(\cdot)$ that can predict the popularity label $S(t_p)$ for m at a future prediction time t_p, i.e., $f(C_m(t_o), t_p)$ is the prediction of $S(t_p)$ for every $t_p > t_o$.*

In existing works, the macro-level prediction task can be defined as either a classification problem or a regression problem. And in this thesis, we cast the macro-level prediction task as a regression problem, estimating the cascade size (popularity) at a specific time. The popularity label $S(t_p)$ in Definition 6, is the exact popularity value (i.e., the number of retweets or citations) that message m will achieve in the future, so that the prediction will be fully correct when $f(C_m(t_o), t_p) = S(t_p)$.

3.2.2 Rumor Detection

Rumor is a type of misinformation. Specifically, a rumor is a message with a false statement, which is posted by users on an online social platform, and circulating from user to user, aiming to mislead them [105]. The task of rumor detection aims to learn a classifier to allocate different labels for given messages, which is formally defined as follows:

Definition 7 ***Rumor Detection.*** *Given a message m, the goal of rumor detection is to learn a classification function $f(\cdot)$ to classify m as a rumor or non-rumor.*

$$f(m) = \begin{cases} 1, & if\ m\ is\ rumor \\ 0, & otherwise \end{cases} \tag{3.1}$$

3.3 A Brief Recall of Neural Networks

In this thesis, we mainly focused on extracting structural and temporal information from the graph-structure data and time-series data. Therefore, in this section, we will briefly recall two specialized neural networks we use: recurrent neural network and graph neural network.

3.3.1 Recurrent Neural Networks

Recurrent neural network (RNN) [106] is a popular technique in modeling temporal dependencies for sequential data, such as sentences and multivariate time series. We assume given an input sequence of vectors, such as $(\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_n)$, with the subscript t representing the time steps in sequence. At time step t, a RNN takes input x_t, and updates the hidden state $\mathbf{h}_t$. This update in an ordinary RNN is defined as follows:

$$\mathbf{h}_t = \tanh\left(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}\right), \tag{3.2}$$

where $\mathbf{W}$ and $\mathbf{U}$ are weight matrices, $\mathbf{b}$ is the bias vectors, and $\tanh(\cdot)$ is the hyperbolic tangent non-linearity function. However, ordinary RNNs faced the limitation in modeling long-distance dependencies due to the problem of gradient vanishing and exploding with the increasing of the input sequence length [107]. To alleviate this problem, researchers introduced memory blocks into RNNs, and proposed the Long Short-Term Memory (LSTM) [40] and Gated Recurrent Units (GRU) [108]. The memory block in LSTM is composed of one memory cell and three different gates, i.e., input gate, forget gate, and output gate, which are responsible for writing, reading, and resetting on the memory cell, respectively. The cell remembers values over arbitrary time steps. The mathematical expressions defining a general LSTM cell update are as follows:

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

where $\mathbf{f}_t$, $\mathbf{i}_t$ and $\mathbf{o}_t$ are the forget gate, the input gate, and the output gate, respectively. Specifically, the input gate $\mathbf{i}_t$ controls the degree of the new memory $\widetilde{\mathbf{c}}_t$ that is added to the memory cell. The forget gate $\mathbf{f}_t$ determines how much of the existing memory can be forgotten. By forgetting part of the memory $\mathbf{c}_{t-1}$ from the last step and adding the new memory $\widetilde{\mathbf{c}}_t$, the memory $\mathbf{c}_t$ is updated. Also, the output gate $\mathbf{o}_t$ is filtered $\mathbf{c}_t$ to generate the hidden state $\mathbf{h}_t$ for the next step. $\mathbf{W}_*$ and $\mathbf{U}_*$ are the weight matrices and $\mathbf{b}_*$ are the biases for $*$ being f, i, o or c, respectively. Here $\sigma(\cdot)$ is the logistic sigmoid function and $\odot$ is the element-wise vector product.

The GRU architecture simplifies the computations and parameters of a LSTM by reducing the type of gate and removing the cell state. The GRU cell updates can be mathematically expressed as follows:

$$\begin{aligned}
\mathbf{r}_t &= \sigma\left(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r\right) \\
\mathbf{z}_t &= \sigma\left(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z\right) \\
\widetilde{\mathbf{h}}_t &= \tanh\left(\mathbf{W}_h \mathbf{x}_t + \mathbf{U}_h (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{b}_h\right) \\
\mathbf{h}_t &= (1 - \mathbf{z}_t) \odot \mathbf{h}_{t-1} + \mathbf{z}_t \odot \widetilde{\mathbf{h}}_t
\end{aligned} \tag{3.4}$$

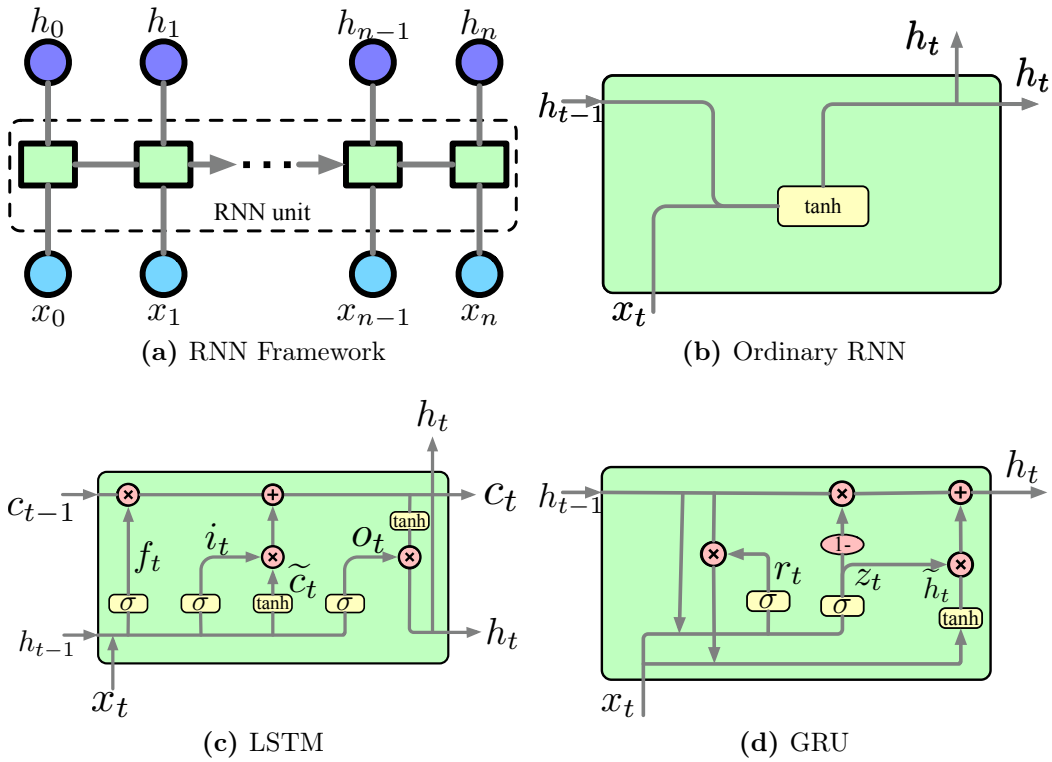
where $\mathbf{r}_t$ and $\mathbf{z}_t$ are reset gate and update gate, respectively. The reset gate $\mathbf{r}_t$ decides how the input $\mathbf{x}_t$ and previous memory $\mathbf{h}_{t-1}$ are combined, and the update gate $\mathbf{z}_t$ controls how much information from the previous memory and candidate hidden state should be forgotten and added in the current step, respectively. Figure 3.2a shows the general framework of RNNs, and the unit's structure for different types of RNN, i.e., ordinary RNN unit, LSTM unit, and GRU unit, are shown in Figure 3.2b–3.2d.

3.3.2 Graph Neural Networks

As we depicted in Definition 1, information cascade is represented as a graph structure. To effectively model this non-Euclidean graph structure data, we use graph neural networks (GNNs). GNNs are meant to learn better structural representations on graphs by combining node feature information with graph structure via feature propagation and aggregation. The core idea of GNNs is the neighborhood aggregation strategy, also called the message passing strategy. GNNs iteratively update a node's representation by aggregating its neighbors' representations. After k iterations[1] of aggregation, the representation of a node contains the structural information within its k–hop neighborhood. Formally, the abstract general formulation of k–th iteration/layer of a GNN is defined as follows:

$$\begin{aligned}
\mathbf{m}_v^{(k)} &= \text{AGGREGATE}(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}_{(v)}\}), \\
\mathbf{h}_v^{(k)} &= \text{UPDATE}(\mathbf{h}_v^{(k-1)}, \mathbf{m}_v^{(k)}),
\end{aligned} \tag{3.5}$$

[1]Also known as "layers" in GNNs.



(a) RNN Framework **(b)** Ordinary RNN

(c) LSTM **(d)** GRU

Figure 3.2: (a) General framework of RNNs; (b) Ordinary RNN unit; (c) LSTM unit, f_t, i_t and o_t are the forget, input and output gates, respectively, c_t and $\widetilde{c}_t$ denote the memory cell and the new memory cell content; (d) GRU unit, r_t and z_t are the reset and update gates, and h_t and $\widetilde{h}_t$ are the activation and the candidate activation.

where $\mathbf{h}_v^{(k)}$ is the learned feature vector of node v at the k–th iteration/layer. We initiate the $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, where $\mathbf{x}_v$ is the vector of features of node v. $\mathcal{N}_{(v)}$ is a set of nodes adjacent to v, i.e. the immediate neighborhoods of node v. The choice of the "AGGREGATE" and "UPDATE" functions is various [61, 65, 109]. Assuming the final iteration is K, and its corresponding node representation is $\mathbf{h}_v^{(K)}$, we can calculate the representation for the entire graph G as:

$$\mathbf{h}_G = \text{READOUT}(\{\mathbf{h}_v^{(K)}, \forall v \in G\}). \tag{3.6}$$

where "READOUT" can be a summation or a more complicated graph-level pooling function [110, 111, 112].

By far, we abstracted the GNNs as a series of message-passing iterations utilizing "AGGREGATE" and "UPDATE" functions. As a concrete example, let us recall the original GNN model proposed by Scarselli et al. [113]. The original GNN message passing is defined as:

$$\begin{aligned} \mathbf{m}_v^{(k)} &= \sum_{u \in \mathcal{N}_{(v)}} \mathbf{h}_u^{k-1}, \\ \mathbf{h}_v^{(k)} &= \sigma(\mathbf{W}_1^{(k)} \mathbf{h}_v^{(k-1)} + \mathbf{W}_2^{(k)} \mathbf{m}_v^{(k)}), \end{aligned} \tag{3.7}$$

where "AGGREGATE" is just a summation and "UPDATE" is defined in terms of the trainable parameter matrices $\mathbf{W}_1^{(k)}$ and $\mathbf{W}_2^{(k)}$ and an activation function σ , such as $\tanh(\cdot)$ or $\mathrm{ReLU}(\cdot)$. Note that, this definition is at node-level. We can also write it at graph-level as:

$$\mathbf{H}^{(k)} = \sigma(\mathbf{H}^{(k-1)}\mathbf{W}_1 + \mathbf{A}\mathbf{H}^{(k-1)}\mathbf{W}_2), \tag{3.8}$$

where $\mathbf{H}^{(k)} = \{\mathbf{h}_v^k | v \in G\}$ is the matrix of node representations at layer k, and each row of $\mathbf{H}^{(k)}$ corresponding to a node, $\mathbf{A}$ is the adjacency matrix, $\mathbf{H}^{(0)} = \mathbf{X}$ ($\mathbf{X} = \{\mathbf{x}_v | v \in G\}$ is the initial node feature matrix).

Most of the existing GNNs always add self-loops to the input graph and omit the update step to simplify the neural message-passing strategy, which simplifies as:

$$\mathbf{h}_v^{(k)} = \mathrm{AGGREGATE}(\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}_{(v)} \cup v\}), \tag{3.9}$$

where "AGGREGATE" is applied to the set $\mathcal{N}_{(v)} \cup v$, i.e., v's neighbors as well as itself. Adding self-loops is equivalent to sharing parameters between the two trainable matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, so that the Eq. (3.8) can be further simplified to:

$$\mathbf{H}^{(k)} = \sigma((\mathbf{A} + \mathbf{I})\mathbf{H}^{(k-1)}\mathbf{W}), \tag{3.10}$$

where $\mathbf{W} = \mathbf{W}_1 = \mathbf{W}_2$, and $\mathbf{I}$ is the identity matrix.

3.3.2.1 Graph Convolutional Networks

Next we turn to a brief description of a specific type of GNNs: Graph Convolutional Networks. In particular we concentrate on the Chebyshev spectral graph convolutional (ChebConv) [41] and vanilla graph convolutional network (GCN) [61] becuase they will be used throughout this thesis.

Chebyshev spectral graph convolutional: ChebConv [41] defines a spectral formulation in the Fourier domain for the convolution operator on graphs $*\mathcal{G}$. More specifically, let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be a graph signal, $\mathbf{A} \in \mathbb{R}^{n \times n}$ the corresponding adjacency matrix and $\mathbf{D} \in \mathbb{R}^{n \times n}$ the diagonal degree matrix with $\mathbf{D}_{ii} = \sum_j \mathbf{A}_{ij}$. The spectral convolutions on graphs are defined as:

$$g_\theta * \mathcal{G}\mathbf{X} = g_\theta(\mathbf{L})\mathbf{X} = g_\theta\left(\mathbf{U}\Lambda\mathbf{U}^T\right)\mathbf{X} = \mathbf{U}g_\theta(\Lambda)\mathbf{U}^T\mathbf{X} \tag{3.11}$$

where $g_\theta = \mathrm{diag}(\theta)$ is a filter parameterized by $\theta \in \mathbb{R}^n$ in the Fourier domain, and can be regarded as a function the eigenvalues of the Laplacian matrix $\mathbf{L}$, i.e., $g_\theta(\Lambda)$. $\mathbf{U} = [\mathbf{u}_0, \cdots, \mathbf{u}_{n-1}] \in \mathbb{R}^{n \times n}$ and $\Lambda = [\lambda_0, \cdots, \lambda_{n-1}] \in \mathbb{R}^{n \times n}$ are the matrix of eigenvectors and the diagonal matrix of eigenvalues of the normalized graph Laplacian $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}} = \mathbf{U}\Lambda\mathbf{U}^T \in \mathbb{R}^{n \times n}$, respectively. $\mathbf{U}^T\mathbf{X}$ is the graph Fourier transform of graph signal $\mathbf{X}$. However, computing the eigen-decomposition of $\mathbf{L}$

in the first place might be prohibitively expensive for large graphs and the complexity of multiplication with $\mathbf{U}$ is $\mathcal{O}(n^2)$. Defferrard et al. [41] approximate $g_\theta(\Lambda)$ with a truncated expansion in terms of Chebyshev polynomials $T_k(x)$ up to K–th order:

$$g_{\theta'}(\Lambda) \approx \sum_{k=0}^{K} \theta'_k T_k\left(\widetilde{\Lambda}\right) \tag{3.12}$$

with $\widetilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - \mathbf{I}$ (λ_{max} denotes the largest eigenvalue of $\mathbf{L}$), identity matrix $\mathbf{I} \in \mathbb{R}^{n\times n}$ and a vector of Chebyshev coefficients θ'_k. The Chebyshev polynomials of order K are recursively defined as $T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$, with $T_0(x) = 1$ and $T_1(x) = x$. The graph filtering operation can now be written as:

$$g_{\theta'} * \mathcal{G}\mathbf{X} \approx \sum_{k=0}^{K} \theta'_k T_k\left(\widetilde{\mathbf{L}}\right)\mathbf{X} \tag{3.13}$$

where $\widetilde{\mathbf{L}} = \frac{2}{\lambda_{max}}\mathbf{L} - \mathbf{I}$. Note that as Eq.(3.13) is now an order K polynomial of the Laplacian, the complexity is reduced to $\mathcal{O}(K|\mathcal{E}|)$. We refer the readers to [41] for details and an in-depth discussion.

Graph convolutional network: GCN [61] can be regarded as a simplification of ChebConv [41]. Specifically, GCN assumes $k = 1$ and $\lambda_{max} = 2$ in ChebConv, so that the calculation of Eq. (3.13) simplifies to:

$$g_{\theta'} * \mathcal{G}\mathbf{X} \approx \theta'_0\mathbf{X} + \theta'_1(\mathbf{L} - \mathbf{I})\mathbf{X} = \theta'_0\mathbf{X} - \theta'_1\mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X} \tag{3.14}$$

where the two parameters θ'_0 and θ'_1 can be shared over the whole graph. Eq. (3.14) can be further simplified as:

$$g_{\theta'} * \mathcal{G}\mathbf{X} \approx \theta'(\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X} \tag{3.15}$$

in the case when $\theta' = \theta'_0 = \theta'_1$. Due to the range of eigenvalues of $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$ falling into $[0, 2]$, Eq. (3.15) can be numerical instable by exploding gradients and vanishing gradients. To overcome these problems, GCN applies a renormalization trick to rescale the range of the eigenvalues to $[0, 1]$. More concretely, GCN uses $\widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ to approximate $\mathbf{I} + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}$, where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and $\widetilde{\mathbf{D}}_{ii} = \sum_j \widetilde{\mathbf{A}}_{ij}$. Finally, the GCN layer is defined as:

$$\mathbf{H} = \hat{\mathbf{A}}\mathbf{X}\mathbf{W} \tag{3.16}$$

where $\hat{\mathbf{A}} = \widetilde{\mathbf{D}}^{-\frac{1}{2}}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-\frac{1}{2}}$ can be calculated at the pre-processing step, $\mathbf{W} \in \mathbb{R}^{d\times f}$ is the weight matrix (i.e., the parameters θ' in Eq. (3.15)), and f is the number of filters or the dimension of the learned representation $\mathbf{H}$. The complexity of a GCN layer is $\mathcal{O}(|\mathcal{E}|fd)$. In the GCN-based model, it captures the information of the kth-order neighborhood of a node through successively employing the GCN layers with an activation function.

Consider a two-layer GCN model as an example. The final output $\mathbf{H}$ is calculated as:

$$\mathbf{H} = \sigma_2(\hat{\mathbf{A}}(\sigma_1(\hat{\mathbf{A}}\mathbf{X}\mathbf{W}^{(0)}))\mathbf{W}^{(1)}) \tag{3.17}$$

where σ_1 and σ_2 are the activation functions. In a semi-supervised node classification task, the functions σ_1 and σ_2 can be $\mathrm{ReLU}(\cdot)$ and $\mathrm{softmax}(\cdot)$, respectively. For more details on GCN, we refer the readers to [61].

Part I

Information Cascades Modeling

Chapter 4

Learning Structural-temporal Features for Cascades Modeling

4.1 Chapter Overview

As described in Chapter 2, the plethora of approaches proposed to tackle the cascade prediction problem fall into three main categories:

1. *handcrafted feature-based approaches* – mostly focusing on identifying and incorporating complicated hand-crafted features, e.g., structural [27, 37, 54], content [22, 38, 51, 52], temporal [28, 55], etc. Their performance strongly depends on extracted features requiring extensive domain knowledge, which is hard to be generalized to new domains;

2. *point process-based approaches* – typically relying on Hawkes point process [9, 30, 49], which models the intensity function of the arrival process for each message independently, enabling knowledge regarding the popularity dynamics of information – but with less desirable predictive power; and

3. *deep learning-based methods*, especially Recurrent Neural Networks (RNN) based approaches [8, 9, 35, 36] – which automatically learn temporal characteristics but fall short in the intrinsic structural information of cascades, essential for cascade prediction [1].

Challenges and Our Approach: Effective and efficient prediction of the size of cascades has several challenges: (1) lack of knowledge of complete network structure through which the cascades propagate [114]. This impedes many global structure based approaches since obtaining a complete graph or further embedding into it is hard, if not impossible. (2) efficient representation of cascades – difficult due to their varying size (from very few to millions [1]), making the random walk based cascade sampling methods biased and ill-suited. (3) modeling diffusion dynamics of information cascades not only requires locally structural characteristics (e.g., community size and activity degree of users) but also needs some temporal characteristics – e.g.,

information within the first few hours plays crucial role in determining the cascades' size.

To address above challenges, we propose a novel framework called *CasCN* (Recurrent **Cas**cades **C**onvolutional **N**etworks) which, while relying on existing paradigms, incorporates both structural and temporal characteristics for predicting the future size of a given cascade. Specifically, *CasCN* samples sub-cascade graphs rather than a set of random-walk sequences from a cascade, and learns the local structures of each sub-cascades by graph convolutional operations. The convoluted spatial structures are then fed into a recurrent neural network for training and capturing the evolving process of a cascade structure. Our main contributions and advantages of *CasCN* are:

• *Use of less information*: We rely solely on the structural and temporal information of cascades, avoiding massive and complex feature engineering, and our model is more generalizable to new domains. In addition, *CasCN* leverages deep learning to learn latent semantics of cascades in an end-to-end manner.

• *Representation of a cascade graph*: We sample a cascade graph as a sequence of sub-cascade graphs and use an adjacency matrix to represent each sub-cascade graph. This fully preserves the structural dynamics of cascades as well as the topological structure at each diffusion time, while eliminating the intensive computational cost when working with large graphs.

• *Additional impacting factors*: *CasCN* takes into account two additional crucial factors for estimating cascade size – the directionality of cascade graphs and the time of re-tweeting (e.g., decay effects).

• *Multi-cascade convolutional networks*: We propose a holistic approach, with variants capturing *temporal*, *structural*, and *directional* patterns in multiple sub-graphs, aware of temporal evolution of dynamic graphs – making our methodology readily applicable to other spatio-temporal data prediction tasks.

• *Evaluations on real-world datasets*: We conduct extensive evaluations on several publicly available benchmark datasets, demonstrating that *CasCN* significantly outperforms the state-of-the-art baselines.

This chapter is based on the following publication[10]:

- **Chen, X.**, Zhou, F., Zhang, K., Trajcevski, G., Zhong, T., Zhang, F.: Information diffusion prediction via recurrent cascades convolution. In: 2019 IEEE 35th International Conference on Data Engineering. ICDE '19 (2019) 770–781

4.2 Problem Statement

Recall (see Definition 4 in chapter 3) that the observed cascade graph of a post m_i before time t_o is denoted as $C_i(t_o)$. In this chapter, we use $g_i^{t_j} = (U_i^{t_j}, E_i^{t_j}, t_j)$ as a short-

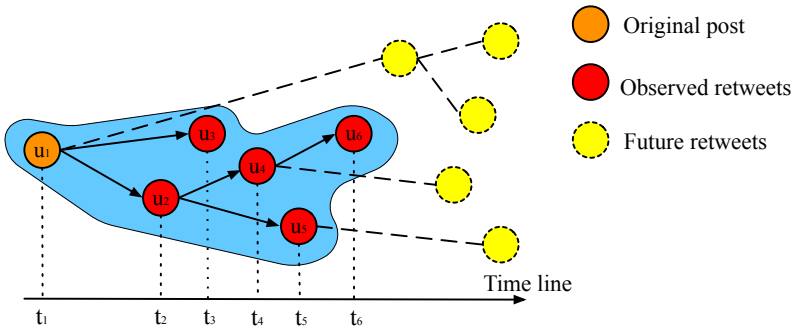


Figure 4.1: The observed cascade graph of a post m_i. Node u_0 initiates the original message m_i.

hand for the snapshot of $C_i(t_o)$ reflecting the diffusion status of a post m_i at time t_j ($t_j \leq t_o$). For example: a node $u_x \in U_i^{t_j}$ represents a user who (for the first time) re-tweets the post m_i from some other users in Twitter (or a paper in a citation network). An edge $\left(u_x, u_y\right) \in E_i^{t_j}$ represents a re-tweet (or a citation) of u_x from u_y ; and t_j is the time-instant when the last re-tweeting (or the citation) behavior occurs in the current snapshot. Figure 4.1 illustrates how the cascade graph can be represented as $g_i^{t_1} = \left(\left(u_1\right), \varnothing, t_1\right), \cdots, g_i^{t_6} = \left(\left(u_1, u_2, \cdots, u_5, u_6\right), \left[\left(u_1, u_2\right), \left(u_1, u_3\right), \cdots, \left(u_4, u_6\right)\right], t_6\right)$.

As depicted above, from observed cascade graph $C_i(t_o)$, we can get different snapshots $g_i^{t_j}$, so that $C_i(t_o)$ can be further represented as a sequence of sub-cascade graphs $G_i^{t_o} = \left\{g_i^{t_j} | t_j \leq t_o\right\}$. In this chapter, our task is to predict the increment size ΔS_i regarding the post m_i for a fixed time interval Δt ($\Delta t = t_p - t_o$), i.e., $\Delta S_i = \left|U_i^{t_o+\Delta t}\right| - \left|U_i^{t_o}\right|$.

Definition 8 *The cascade size predictor is a function $f\left(\cdot\right)$ that is to be learned, mapping $G_i^{t_o} = \left\{g_i^{t_1}, \cdots, g_i^{t_j}, \cdots; t_j \leq t_o\right\}$ to ΔS_i for the time interval Δt.*

Table 4.1: Main notations used throughout this chapter.

Symbol	Description
$g_i^{t_j}, \mathbf{a}_i^{t_j}$	A sub-cascade graph of $C_i(t_o)$ at diffusion time t_j and it's adjacency matrix.
$G_i^{t_o}, \mathbf{A}_i^{t_o}$	A sequence of sub-cascade graphs of $C_i(t_o)$ and the corresponding adjacency matrices.
Δt	The fixed time interval.
ΔS_i	The increment size of m_i after Δt.
$\mathbf{P}_c$	Transition matrix of a cascade.
ϕ, Φ	Stationary transition distribution and diagonalized ϕ
Δ_c	Laplacian of a cascade.
λ_{max}	The largest eigenvalue of Laplacian.
K	Maximum steps from the central node, i.e., K^{th}-order neighborhood or Chebyshev coefficients.

4.3 CasCN: Information diffusion prediction via recurrent cascades convolution

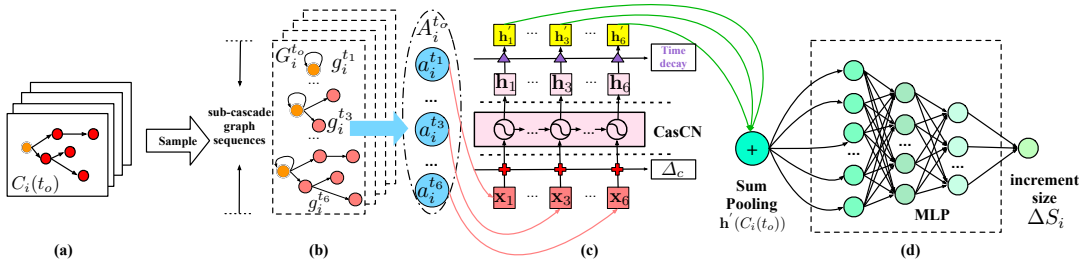


Figure 4.2: Overview of *CasCN*: (a) The input is a cascade graph $C_i(t_o)$ for a given time window t_o of a certain post m_i. (b) We obtain a sequence of sub-cascade graphs from $C_i(t_o)$, and use an adjacency matrix $\mathbf{a}_i^{t_j}$ to represent an instance $g_i^{t_j}$ of the sub-cascade graph. We refer to $\mathbf{A}_i^{t_o} = \{\mathbf{a}_i^{t_1}, \mathbf{a}_i^{t_2}, \cdots\}$ as signals. (c) We feed the signals and the Laplacian matrix Δ_c of cascade $C_i(t_o)$ into *CasCN*. The output $\mathbf{h}_t$ of *CasCN* is transformed to a new representation $\mathbf{h}_t^{'}$ by multiplying it by a time decay factor. All $\mathbf{h}_t^{'}$'s will be assembled via a sum pooling to form the final $C_i(t_o)$ representation: $\mathbf{h}^{'}(C_i(t_o))$. (d) Finally, we use a MLP to predict the increment size of cascade (ΔS_i) for a fixed time interval Δt.

Our deep learning framework *CasCN* takes the observed cascade graph $C_i(t_o)$ as an input and predicts the increment size ΔS_i regarding certain information (e.g., a post) m_i. *CasCN* leverages LSTM and GCN to fully extract temporal and structural information from the cascade graph. After an overview of *CasCN*, we focus on the details in the respective sub-sections.

CasCN is an end-to-end type of framework consisting of three basic components, depicted in Figure 4.2: (1) Cascade graph sampling: it dynamically samples a sequence of sub-cascade graphs from the original cascade graph, and then represents sub-cascade graphs as a sequence of adjacency matrices; (2) Structural and temporal modeling: it feeds the adjacency matrix sequences and the structural information of cascade graphs (i.e., the Laplacian matrices of cascade graphs) within an observation window into a neural network. It combines recurrent neural networks and graph convolutional networks with a time decay function to learn the representation of cascades; (3) Prediction network: a Multi-Layer Perceptron (MLP) is used to predict the increment cascade size based on the representation learned from the previous steps.

4.3.1 Cascade Graph as Sub-cascade Graph Sequences

Given a post m_i, the first step in *CasCN* is to initialize the representation of its corresponding cascading graph $C_i(t_o)$. Existing methods typically treat the graph in two ways: either sampling the graph as a bag of nodes, which ignores both local and global structural information, or denoting the graph as a set of paths. For example, DeepCas [8] samples a set of paths from each cascade. The sampling

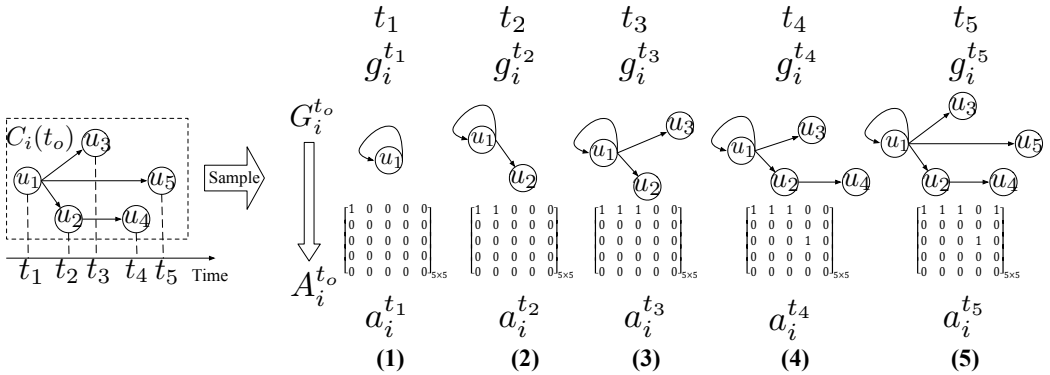


Figure 4.3: Illustration of sampling and representation of sub-cascade graph sequence.

process could be generalized as performing a random walk over a cascade graph similar to DeepWalk which, however, fails to consider the dynamics of cascades – one of the most important factor in information diffusion. DeepHawkes [9] transforms the cascade graph into a set of diffusion paths according to the diffusion time, each of which depicts the process of information propagation between users within the observation time; however, this method ignores the structural information of cascade graphs.

Our approach samples the cascade graph $C_i(t)$ to obtain a sequence of sub-cascade graphs $G_i^{t_o}$ which is used to represent cascades within the observation time t_o. $G_i^{t_o}$ is denoted as:

$$G_i^{t_o} = \left\{ g_i^{t_1}, \cdots, g_i^{t_j}, \cdots \right\}, t_j \leq t_o. \tag{4.1}$$

The sampling process is illustrated in Figure 4.3, where each sub-cascade graph is represented by an adjacency matrix: the rows correspond to the alphabetical order of nodes' labels (top to bottom) and the columns correspond to edges, as illustrated above each instance of the adjacency matrix. The first sub-cascade graph in $G_i^{t_o}$ only contains one single node because it is the generator of the post m_i, so we add a self-connection for this initiator. Thus, $G_i^{t_o}$ is represented with a sequence of adjacency matrices $\mathbf{A}_i^{t_o} = \left\{ \mathbf{a}_i^{t_1}, \cdots, \mathbf{a}_i^{t_j}, \cdots \right\}$.

4.3.2 Laplacian Transformation of Cascades

Classical GCN methods cannot be applied for cascades modeling since they focus on *fixed* and *undirected* graphs which, in turn, cannot consider the temporal information of cascade evolution. In contrast, cascade graphs in our problem are dynamic directed trees (DATs, which belong to dynamic directed graphs). As mentioned above, the graph Laplacian for an undirected graph is a symmetric difference operator $\mathbf{L} = \mathbf{D} - \mathbf{W}$, where $\mathbf{D}$ is the degree matrix and $\mathbf{W}$ is the weight matrix of the graph, which cannot be adapted in DAGs.

Recently, Li et al. [115] propose DCRNN to model the traffic flow as a diffusion process on a *fixed* DAG (a *directed* sensor graph), and define a diffusion convolution as:

$$y = g_\theta * \mathcal{G}x = \sum_{k=0}^{K} \left(\theta_{k,1} \left(\mathbf{D}_O^{-1}\mathbf{W} \right)^K + \theta_{k,2} \left(\mathbf{D}_I^{-1}\mathbf{W}^T \right)^K \right) \mathbf{X}$$

where $\mathbf{D}^{-1}\mathbf{W}$ is the random walk matrix used to replace Laplacian $\widetilde{\mathbf{L}}$ in Eq. (3.13) This operation is actually a diffusion process convolution proposed by Atwood and Towsley [116] where the diffusion process is modeled as Markov process and may converge to a stationary distribution $\mathcal{P} \in \mathbb{R}^{n \times n}$ after many steps, and the i^{th} row $\mathcal{P}_{i,:} \in \mathbb{R}^n$ represents the likelihood of diffusion from node v_i.

In our settings, various cascades are different DATs, all of which require incorporating special structure and direction information rather than a *single* and *fixed* sensor network in [115]. To overcome this challenge, we introduce Laplacian of cascade Δ_c, called *CasLaplacian*, for modeling the convolution operation over a single cascade signal $\mathbf{X}$ as:

$$y = g_\theta * \mathcal{G}\mathbf{X} = \sum_{k=0}^{K} \theta_k T_k \left(\widetilde{\Delta}_c \right) \mathbf{X} \tag{4.2}$$

where $\widetilde{\Delta}_c = \frac{2}{\lambda_{max}} \Delta_c - \mathbf{I}$ is a scaled Laplacian.

Now we introduce the way of computing Laplacian of cascade Δ_c, which can capture special structural and directional characteristics of different cascades. For a *directed* graph, we define the normalized *directed* Laplacian as:

$$\mathcal{L} = \mathbf{I} - \frac{\Phi^{\frac{1}{2}}\mathbf{P}\Phi^{-\frac{1}{2}} + \Phi^{-\frac{1}{2}}\mathbf{P}^T\Phi^{\frac{1}{2}}}{2}, \tag{4.3}$$

(cf. [117]) where $\mathbf{P}$ is a transition probability matrix, Φ is a diagonal matrix with entries $\Phi(v, v) = \phi(v)$, and $\phi = [\phi_i]_{1 \le i \le n}$ is the *column* vector of the stationary probabilities distribution of $\mathbf{P}$.

However, such a symmetrical $\mathcal{L}$ can not capture the unique characteristic of the random walk on the different cascades. For example, given a cascade with transition probability matrix $\mathbf{P}_c$, there exist cascades which have the same stationary distribution matrix $\mathcal{P}_c$, such that all these cascades have the same Laplacian matrix. To solve this problem, we relied on *Diplacian* [118] which computes Laplacian of DAGs as:

$$\Gamma = \Phi^{\frac{1}{2}} (\mathbf{I} - \mathbf{P}) \Phi^{-\frac{1}{2}} \tag{4.4}$$

where the transition probability matrix $\mathbf{P}$ is defined as $\mathbf{P} = \mathbf{D}^{-1}\mathbf{W}$ with the hypothesis that the graph is strongly connected (SCGs) [118]. In contrast, our cascade graphs are not SCGs. Thus, we define a transition probability matrix $\mathbf{P}_c$ of given cascade graph as:

$$\mathbf{P}_c = (1 - \alpha) \frac{\mathbf{E}}{n} + \alpha \left(\mathbf{D}^{-1}\mathbf{W} \right), \tag{4.5}$$

where $\mathbf{E} \in \mathbb{R}^{n \times n}$ is an all–one matrix and $\alpha \in (0, 1)$ is an initial probability, used to restrict the state transition matrix $\mathbf{D}^{-1}\mathbf{W}$ to be a strongly connected matrix without 0 anywhere. Then the transition matrix $\mathbf{P}_c$ is irreducible, and has a unique stationary probability distribution $\{\phi_i | \phi_i > 0, 1 \leq i \leq n\}$. The stationary distribution vector $\{\phi_i\}$ can be obtained by solving an eigenvalue problem $\phi^T \mathbf{P}_c = \phi^T$ subject to a normalized equation $\phi^T e = 1$, where $e \in \mathbb{R}^n$ is an all-one vector.

Finally, we can compute *CasLaplacian* as:

$$\Delta_c = \Phi^{\frac{1}{2}} \left(\mathbf{I} - \mathbf{P}_c\right) \Phi^{-\frac{1}{2}}. \tag{4.6}$$

Relationship with GCN: We now explain the relationship between our directed CasLaplacian and the normalized one in GCN, as well as the rationale behind CasLaplacian.

A random walk on *undirected* graph G is a Markov chain defined on G with the transition probability matrix $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$, and there exists a unique stationary distribution $\{\phi_1, \phi_2, \ldots, \phi_n\}$. Let $\phi = [\phi_i]_{1 \leq i \leq n}$ be the *column* vector of the stationary probabilities, where $\phi^T \mathbf{P} = \phi^T$. Note that, as for undirected graph, the normalized Laplacian $\mathbf{L}$ can be transformed as:

$$\mathbf{L} = \mathbf{D}^{-\frac{1}{2}} \left(\mathbf{D} - \mathbf{A}\right) \mathbf{D}^{-\frac{1}{2}} = \mathbf{D}^{\frac{1}{2}} \left(\mathbf{I} - \mathbf{P}\right) \mathbf{D}^{-\frac{1}{2}}. \tag{4.7}$$

Also, the stationary probabilities ϕ of an undirected graph can be calculated as

$$\phi_i = \frac{d_i}{\sum_k d_k} = \frac{d_i}{d}, i = 1, 2, \cdots, n, \tag{4.8}$$

and $\Phi^{\frac{1}{2}} = \sqrt{\text{diag}(\phi)}$, where $\phi = [\phi_1, \phi_2, \cdots, \phi_n]^T$, which can be used to approximate the degree matrix $\mathbf{D}$:

$$\mathbf{L} = \mathbf{D}^{\frac{1}{2}} \left(\mathbf{I} - \mathbf{P}\right) \mathbf{D}^{-\frac{1}{2}} \approx \Phi^{\frac{1}{2}} \left(\mathbf{I} - \mathbf{P}\right) \Phi^{-\frac{1}{2}}. \tag{4.9}$$

Algorithm 1 formalizes the process of constructing the Laplacian of cascades.

4.3.3 Structural and Temporal Modeling

We represent and model the cascade graph in a structural-temporal way. After obtaining the adjacency representation of sub-cascade graph sequence $\mathbf{A}_i^{t_o}$ and the Laplacian matrix Δ_c for each cascade graph, *CasCN* turns to learn the structural and temporal patterns via the combination of classical LSTM and GCN.

We leverage the RNNs to model the temporal dependence of diffusion – in particular, using the Long Short-Term Memory (LSTM) [40], which is a stable and powerful variant of RNNs. We replace the multiplications by dense matrices W with graph

Algorithm 1: Laplacian of cascade.

Input: A cascade graph C, initial probability α.
Output: *CasLaplacian*–Laplacian of cascade Δ_c.

1: Compute degree matrix $\mathbf{D}$ and weighted adjacency matrix $\mathbf{W}$ of a cascade graph C.
2: Compute transition probability matrix $\mathbf{P}_c$ of cascade graph according to Eq. (4.5) .
3: Solve the eigenvalue problem $\phi^T \mathbf{P}_c = \phi^T$ subject to a normalized equation $\phi^T e = 1$ to get $\{\phi_i\}$.
4: $\Phi = \mathrm{diag}(\phi)$.
5: Compute *CasLaplacian* Δ_c according to Eq. (4.6).

convolutions to incorporate the structural information:

$$\begin{aligned}
\mathbf{i}_t &= \sigma\left(\mathbf{W}_i * \mathcal{G}\mathbf{X}_t + \mathbf{U}_i * \mathcal{G}\mathbf{h}_{t-1} + \mathbf{V}_i \odot \mathbf{c}_{t-1} + \mathbf{b}_i\right) \\
\mathbf{f}_t &= \sigma\left(\mathbf{W}_f * \mathcal{G}\mathbf{X}_t + \mathbf{U}_f * \mathcal{G}\mathbf{h}_{t-1} + \mathbf{V}_f \odot \mathbf{c}_{t-1} + \mathbf{b}_f\right) \\
\mathbf{o}_t &= \sigma\left(\mathbf{W}_o * \mathcal{G}\mathbf{X}_t + \mathbf{U}_o * \mathcal{G}\mathbf{h}_{t-1} + \mathbf{V}_o \odot \mathbf{c}_t + \mathbf{b}_o\right)
\end{aligned} \tag{4.10}$$

where $*\mathcal{G}$ denotes the graph convolution defined in Eq.(4.2), signal $\mathbf{X}_t$ is the cascade graph sequences $\mathbf{A}_i^{t_o} \in \mathbb{R}^{d_T \times n \times n}$, d_T denotes the number of diffusion time steps of post m_i. We leverage $\mathbf{W}_i * \mathcal{G}\mathbf{X}_t$ to mean a graph convolution of signal $\mathbf{X}_t$ with $d_h \times n$ filters which are functions of the cascade Laplacian Δ_c parametrized by K Chebyshev coefficients. $\sigma(\cdot)$ is the logistic sigmoid function and $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t, \mathbf{b}_*$ are respectively the input gate, forget gate, output gate and bias vector. The matrices $\mathbf{W} \in \mathbb{R}^{K \times n \times d_h}$, $\mathbf{U} \in \mathbb{R}^{K \times d_h \times d_h}$ and $\mathbf{V} \in \mathbb{R}^{n \times d_h}$ are the different gate parameters, and n denotes the number of nodes in a cascade graph, and d_h is the size of cell states.

In particular, the memory cell $\mathbf{c}_t$ is updated by replacing the existing memory unit with a new cell $\mathbf{c}_t$ as:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tanh\left(\mathbf{W}_c * \mathcal{G}\mathbf{X}_t + \mathbf{U}_c * \mathcal{G}\mathbf{h}_{t-1} + \mathbf{b}_c\right) \tag{4.11}$$

The hidden state is then updated by

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{c}_t\right) \tag{4.12}$$

where $\tanh(\cdot)$ refer to the hyperbolic tangent function, and $\odot$ is the entry-wise product.

4.3.4 Cascades Size Prediction

Previous works [30, 49] have shown the existence of time decay effect – i.e., that the influence of a node on other nodes will decrease over time. Various time decay functions have been defined: (1) power-low functions $\phi^p(\tau) = (\tau + c)^{-(1+\theta)}$; (2)

exponential functions $\phi^e(\tau) = e^{-\theta\tau}$; (3) Rayleigh functions $\phi^r(\tau) = e^{-\frac{1}{2}\theta\tau^2}$, where $\tau = t_j - t_i$ and $t_i \leq t_j$. In practice, the choice of such function varies for different scenarios, e.g., exponential functions are suitable for financial data while Rayleigh functions perform better for epidemiology and power law functions are more applicable in geophysics and social networks [119, 120].

However, all the above time-decay functions have the limitation of parametric assumption which is greatly influenced by assumed prior distribution (and intuition). In this chapter, we employ a non-parametric way to define the time decay function. More specifically, we assume that the time window of the observed cascade is $[t_1, t_o]$, and then split the time window into l disjoint time intervals $\{[t_1, t_2), [t_2, t_3), \cdots, [t_{l-1}, t_l = t_o]\}$ to approximate the continuous time window by discrete time intervals. It not only allocates each diffusion time a corresponding interval, but also allows us to learn the discrete variable of time decay effect $\lambda = \{\lambda_m, m \in (1, 2, \cdots, l)\}$. Therefore, we define a function to compute the corresponding time interval m of time decay effect for a re-tweet at time t_j:

$$m = \lfloor \frac{(t_j - t_1)}{\lceil t_o/l \rceil} \rfloor \tag{4.13}$$

Where t_0 is the time of original post, l is the number of time intervals, $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ are the floor and ceiling operations.

For a cascade graph $C_i(t_o)$ regarding post m_i within the observation time window $[t_1, t_o]$, we get the hidden states $\{\mathbf{h}_1, \cdots, \mathbf{h}_t, \cdots, \mathbf{h}_{d_T}\}$ and we multiply a time decay effect λ_m for each hidden state to obtain $\left\{\mathbf{h}_1^{'}, \cdots, \mathbf{h}_t^{'}, \cdots, \mathbf{h}_{d_T}^{'}\right\}$ by:

$$\mathbf{h}_t^{'} = \lambda_m \mathbf{h}_t \tag{4.14}$$

summed up to get the representation vector for the cascade graph $C_i(t)$:

$$\mathbf{h}^{'}\left(C_i(t_o)\right) = \sum_{t=1}^{d_T} \mathbf{h}_t^{'} \tag{4.15}$$

The last component of *CasCN* is a multi-layer perceptron (MLP) with one final output unit. Given the representation $\mathbf{h}^{'}\left(C_i(t_o)\right)$, we calculate the increment size ΔS_i as:

$$\Delta S_i = f\left(C_i(t_o)\right) = \mathrm{MLP}\left(\mathbf{h}^{'}\left(C_i(t_o)\right)\right) \tag{4.16}$$

Our ultimate task is to predict the increment size for a fixed time interval, which can be done by minimizing the following loss function:

$$\ell\left(\Delta S_i, \Delta\widetilde{S}_i\right) = \frac{1}{P}\sum_{i=1}^{P}\left(\log \Delta S_i - \log \Delta\widetilde{S}_i\right)^2 \tag{4.17}$$

where P is the number of posts, ΔS_i is the predicted incremental size for post m_i, and $\Delta\widetilde{S}_i$ is the ground truth.

The process of training *CasCN* is shown in Algorithm 2.

Algorithm 2: Learning with *CasCN*.

Input: sequences of adjacency matrices of cascade graphs $\mathbf{A} = \{\mathbf{A}_1^{t_o}, \mathbf{A}_2^{t_o}, \cdots\}$ within an observation time t_o; Laplacian sequence for cascade graphs $\Delta = \{\Delta_c^1, \Delta_c^2, \cdots\}$, batch size b.

Output: Increment sizes $\Delta S = \{\Delta S_1, \Delta S_2, \cdots\}$ of cascades.

repeat
  b = 1, 2, ...
  for adjacency matrix sequence $\mathbf{A}_i^{t_o}$ and corespoding Laplacian Δ_c^i in batch b **do**
    Compute the Structural and Temporal information $\mathbf{h}_t$ of cascade $C_i(t_o)$ according to Eq. (4.10) - Eq. (4.12).
    Multiply each hidden state $\mathbf{h}_t$ with time decay effect λ_m to get $\mathbf{h}_t^{'}$, according to Eq. (4.14).
    $\mathbf{h}^{'}\left(C_i(t_o)\right) \leftarrow \text{Aggregate}\Big(\Big\{\mathbf{h}_t^{'}, t \in [1, d_T]\Big\}\Big)$
    Feed $\mathbf{h}^{'}\left(C_i(t_o)\right)$ into MLP to compute increment size ΔS_i of cascade, according to Eq. (4.16)
    Use Adaptive moment estimation (Adam) to optimize the objective function in Eq. (4.17) and update parameters in Eq. (4.10), (4.11), (4.13)
  end for
until convergence;

4.4 Evaluation

In this section, we compare the performance of our proposed model *CasCN* with several state-of-the-art approaches that we use as baselines, and a few variants of *CasCN* itself, for cascade size prediction using two real-world datasets. For reproducability of our results, supplemental materials, implementation details and instructions are available online at https://github.com/ChenNed/CasCN.

4.4.1 Datasets

We evaluate the effectiveness and generalizability of *CasCN* on two scenarios of information cascade prediction, and compare with previous works such as DeepCas and DeepHawkes – using publicly available datasets. The first one is to forecast the size of re-tweet cascades on Sina Weibo and the second one is to predict the citation count of papers in Citation dataset HEP-PH. The statistics of the datasets as shown in Table 4.2.

Table 4.2: Statistics of datasets

	Data sets	Sina Weibo			HEP-PH		
posts-papers	All	119,313			34,546		
edges	All	8,466,858			421,578		
t_o		1hour	2hours	3hours	3years	5years	7years
cascades	train	25,145	29,515	31,780	3,458	3,467	3,478
	val	5,386	6,324	6,810	837	839	848
	test	5,386	6,324	6,810	837	839	848
Avg. nodes	train	28.58	29.30	29.48	5.27	5.27	5.27
	val	28.71	29.47	29.69	4.32	4.93	4.27
	test	29.11	29.77	30.21	4.91	4.27	4.28
Avg. edges	train	27.78	28.54	28.74	4.27	4.27	4.27
	val	27.91	28.70	28.94	3.31	3.93	3.95
	test	28.32	29.01	29.48	3.91	3.27	3.28

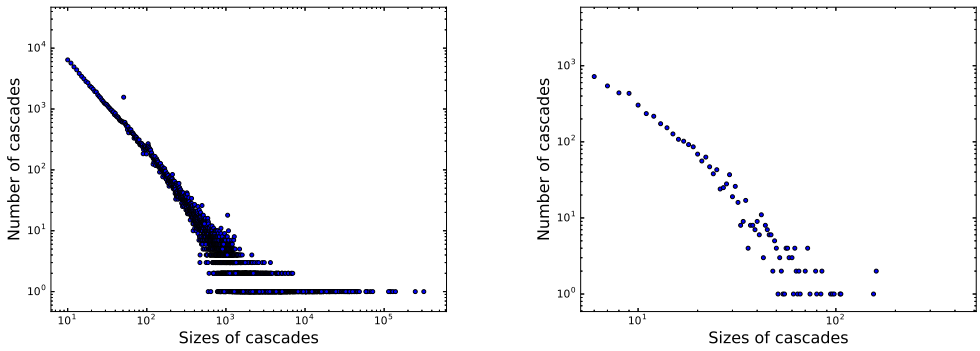
• **Sina Weibo**[1]: The first dataset is Sina Weibo, a popular Chinese microblog platform [2], provided in [9] – which collects all original posts generated on June 1^{st}, 2016, and tracks all re-tweets of each post within the next 24 hours. It includes $119,313$ posts in total. Figure 4.5a shows that the popularity of cascades saturates after 24 hours since publishing. Figure 4.4a shows the distribution of cascade size (the number of re-tweets of each post). We follow similar experimental setup as in DeepHawkes [9] – i.e., the length t_o of the observation time window being $t_o = 1$ hour, 2 hours and 3 hours, and the cascades with the publication time before 8 am and after 6 pm being filtered out. Finally, we sort the cascades in terms of their publication time after preprocessing and choose the first 70% of cascades for training and the rest for validation and testing via even split.

• **HEP-PH**[3]: HEP-PH dataset is from the e-print arXiv and covers papers in the period from January 1993 to April 2003 (124 months). If a paper i cites paper j, the graph contains a directed edge from i to j. The data was originally released as a part of 2003 KDD Cup [121]. For the observation window, we choose $t_o = 3$, 4 and 5 years corresponding to the year that the popularity reaches about 50%, 60% and 70% of the final size, as shown in Figure 4.5b. Then, we pick up 70% of cascades for training and the rest for validation and testing via even split.

4.4.2 Baselines

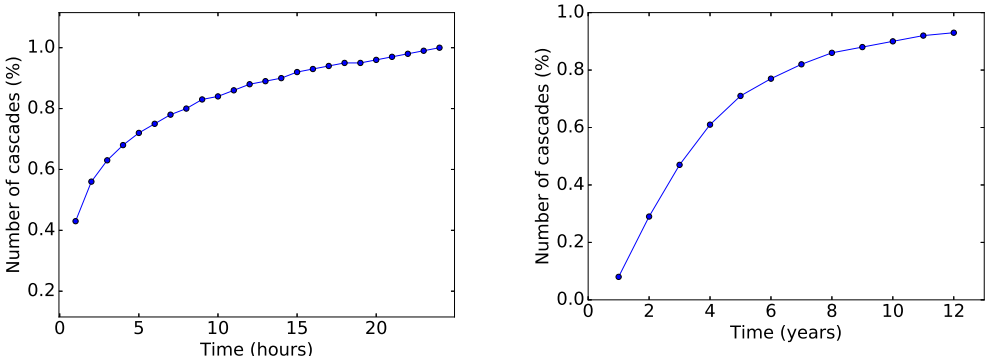
We have already seen that existing relevant methods for information cascade prediction are mainly falling into three categories: (1) Handcrafted feature-based approaches, (2) Point process-based approaches, and (3) Deep learning-based approaches. Therefore, we select several methods in each group as baselines. As

[1]https://github.com/CaoQi92/DeepHawkes
[2]http://www.weibo.com
[3]http://snap.stanford.edu/data/cit-HepPh.html



(a) Weibo dataset **(b)** HEP-PH

Figure 4.4: Distribution of cascades size, the X axis is the size of cascades, and the Y axis is the number of cascades corresponding to the different sizes.



(a) Weibo dataset **(b)** HEP-PH

Figure 4.5: Percentage distribution between time and the number of cascades

deep learning methods, we select three representative methods: DeepCas [8], DeepHawkes [9] and Topo-LSTM [35]. Note that DeepHawkes is also regarded as a successful implementation of Hawkes process – i.e., point process-based approaches. Furthermore, we include a network representation method to enrich our experiment – Node2Vec. We also introduce a baseline named LIS [13] from diffusion model-based approaches, which used to model cascades dynamics. The baselines and their implementation details are as follows:

Feature-based: Recent studies [1, 39, 49, 56] show that structural features, temporal features, and other features (e.g., content features and user features) are informative for information cascade prediction. In our implementations, we include all features mentioned above that could be generalized across datasets. These features include:

Structural features: We count the number of leaf nodes, the average degree (both in-degree and out-degree), average and max length of retweet path of cascades as measures of structural features.

Temporal features: We extract the time elapsed since the initial post for each retweet, the cumulative growth and incremental growth every 10 minutes for Sina Weibo and every 31 days for HEP-PH, for the reason that the time in Sina Weibo can be accurate to minutes, and the unit in HEP-PH is a day.

Other features: We use node ids as *node identity* feature.

After extracting all the cascade features, we use two models, i.e., **Feature-linear** and **Feature-deep**, to perform information cascade prediction. The label (incremental size of cascade) has been logarithmically transformed before feeding into models, so that the baseline of feature-based methods optimizes the same loss function as CasCN.

- **Feature-linear**: We feed the features into a linear regression model with L_2 regularization. The details of the L_2-coefficient setting can be found in Section 4.4.5.
- **Feature-deep**: For fairness of comparison of the performance of the feature-based approaches with CasCN, we propose a strong baseline denoted as *Feature-deep*, which also uses a MLP model to predict the incremental size of cascade with hand-craft feature vectors.

LIS [13]: LIS is a diffusion model-based approach. This method models the cascade dynamics by learning two low-dimensional latent vectors for messages from observed cascades to capture their influence and susceptibility, respectively.

Node2Vec [122]: Node2Vec is selected as a representative of node embedding methods, and can be replaced with any other embedding methods, e.g., DeepWalk [123] and LINE [124]. We conduct random walks from cascade graphs and generate embedding vectors for each node. Next, the embeddings of all nodes in a cascade graph are fed into MLP to make predictions.

DeepCas [8]: The first deep learning architecture for information cascades prediction, which represents a cascade graph as a set of random walk paths and piped through bi-directional GRU neural network with an attention mechanism to predict the size of the cascade. It mainly utilizes the information of structure and node identities for prediction.

DeepHawkes [9]: DeepHawkes model integrates the high prediction power of end-to-end deep learning into interpretable factors of Hawkes process for popularity prediction. The marriage between deep learning technique and a well-established interpretable process for modeling cascade dynamics bridges the gap between prediction and understanding of information cascades. This method belongs to both point process-based approaches and deep learning-based approaches.

Topo-LSTM [35]: A novel topological recurrent neural network which is a directed acyclic graph-structured (DAG-structured) RNN takes dynamic DAGs as inputs and

generates a topology-aware embedding for each node in the DAGs as outputs. The original application of Topo-LSTM is to predict node activations. We replace the logistic classifier in Topo-LSTM with a diffusion size regressor to predict the size of cascades.

4.4.3 Variants of CasCN

In addition to the comparison with existing baselines, we also derive a few variants of CasCN:

CasCN-GL: CasCN-GL replaces the structural-temporal modeling component of CasCN with the combination of GCN and LSTM for modeling structural and temporal patterns, respectively.

CasCN-GRU: This method replaces the LSTM of CasCN with GRU. Similar to LSTM, CasCN with GRU models structural-temporal information using extra gating units, but without separated memory cells. Formally, we update the state of $\mathbf{h}_t$ by a linear interpolation between the last state $\mathbf{h}_{t-1}$ and the candidate state $\widetilde{\mathbf{h}}_t$.

CasCN-Path: In CasCN-Path, we use random walks to represent a cascade graph (shown in Figure 4.6) rather than sub-cascade graphs used in CasCN. Therefore, we first embed users into a 50-dimensional space to represent the latent (re-tweeting) relationships among users in a cascade graph. Next, we use random walks to sample sufficient number of sequences for all cascade graphs. Finally, we feed them to CasCN and predict the size of information cascades.

CasCN-Undierected: In CasCN-Undierected, we regard the cascade graphs as undirected graphs and calculate the normalized Laplacian according to $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-\frac{1}{2}}\mathbf{W}\mathbf{D}^{-\frac{1}{2}}$.

CasCN-Time: In CasCN-Time, we do not consider the time decay effect of re-tweeting.

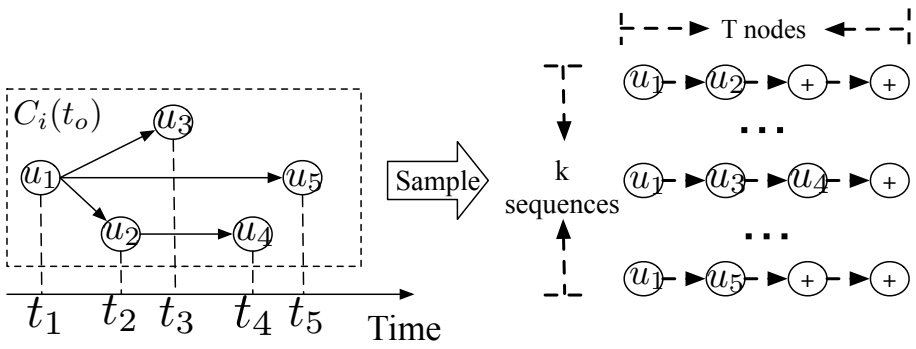


Figure 4.6: Sampling the cascade graph as random walks.

4.4.4 Evaluation Metric

Following the existing works, we choose standard evaluation metrics – *MSLE* (mean square log-transformed error) in our experiments [8, 9, 35]. Note that the smaller *MSLE*, the better the prediction performance. Specifically, *MSLE* is the metric for evaluating the linking accuracy, defined as:

$$MSLE = \frac{1}{P}\sum_{i=1}^{P}\left(\log \Delta S_i - \log \Delta \widetilde{S}_i\right) \tag{4.18}$$

where P is the total number of posts, ΔS_i is the predicted incremental size for post m_i, and $\Delta \widetilde{S}_i$ is the ground truth.

4.4.5 Hyper-parameters

Models mentioned above involve several hyper-parameters. For example, L_2 coefficient in **Feature-linear** are chosen from $\{1, 0.5, 0.1, 0.05, ..., 10^{-8}\}$. For **Feature-deep**, parameters are similar to deep learning-based approaches.

For **LIS**, we follow the work in [13], the maximum number of epochs M is 1×10^5. We use random values to initialize regularization parameters γ_I and γ_S.

For **Node2Vec**, we follow the work in [122], i.e., parameters p and q are selected from $\{0.25, 0.50, 1, 2, 4\}$, the length of walk is chosen from $\{10, 25, 50, 75, 100\}$, and the number of walks per node varies from $\{5, 10, 15, 20\}$.

For **DeepCas**, **DeepHawkes** and **Topo-LSTM**, we follow the setting of DeepCas [8], where the embedding dimensionality of users is 50, the hidden layer of each GRU has 32 units and the hidden dimensions of the two-layer MLP are 32 and 16, respectively. The learning rate for user embeddings is 5×10^{-4} and the learning rate for other variables is 5×10^{-3}. The batch size for each iteration is 32 and the training process will stop when the loss of validation set does not decline for 10 consecutive iterations. The time interval of DeepHawkes is set to 10 minutes for Sina Weibo and 2 months for HEP-PH. For ***CasCN***, the basic parameters (e.g., learning rate and batch size, etc.) are the same as above deep learning-based approaches, except that we choose the support $K = 2$ of GCN and calculate the max eigenvalue λ_{max} of the cascade Laplacian.

4.4.6 Overall performance

Table 4.3 summarizes the performance comparison among *CasCN* and baselines on both Sina Weibo dataset and HEP-PH dataset. In general, the proposed *CasCN* model performs relatively well on information cascade prediction for both datasets (post re-tweeting and paper citing). It outperforms traditional approaches, e.g.,

Table 4.3: Overall performance comparison of information cascades prediction among different approaches. M: model; t_o: observation time.

Datasets	Weibo Dataset			HEP-PH		
Metric	MSLE					
M \ t_o	1 hour	2 hours	3 hours	3 years	5 years	7 years
Features-deep	3.680	3.361	3.296	1.893	1.623	1.619
Features-linear	3.501	3.435	3.324	1.715	1.522	1.471
LIS	3.731	3.621	3.457	2.144	1.798	1.787
Node2Vec	3.795	3.523	3.513	2.479	2.157	2.096
DeepCas	2.958	2.689	2.647	1.765	1.538	1.462
Topo-LSTM	2.772	2.643	2.423	1.684	1.653	1.573
Deep-Hawkes	2.441	2.287	2.252	1.581	1.470	1.233
CasCN	**2.242**	**2.036**	**1.910**	**1.353**	**1.164**	**0.851**

Table 4.4: Performance comparison between CasCN and its variants. M: model; t_o: observation time.

Datasets	Weibo Dataset			HEP-PH		
Metric	MSLE					
M \ t_o	1 hour	2 hours	3 hours	3 years	5 years	7 years
CasCN	**2.242**	**2.036**	**1.916**	1.35	**1.164**	**0.851**
CasCN-GRU	2.288	2.052	1.965	**1.347**	1.166	0.874
CasCN-Path	2.557	2.483	2.404	1.664	1.437	1.332
CasCN-GL	2.312	2.028	1.942	1.364	1.357	1.302
CasCN-Undierected	2.309	2.132	1.978	1.562	1.425	1.118
CasCN-Time	2.652	2.547	2.363	1.732	1.512	1.451

feature-based and point process-based approaches, and it is superior to the state-of-the-art deep learning methods, with a statistically significant drop of MSLE.

The performance gap between these Feature-deep and Feature-linear is quite small meaning that if we have a set of representative features of information cascades, deep learning does not always perform better than traditional predicting methods. However, as discussed earlier, the performance of such methods heavily depends on hand-crafted features which are difficult to select for different scenarios in practice.

For embedding methods, Node2Vec [122] does not perform well. Through the comparison with DeepCas [8], it proves that only taking the node embeddings as the graph representation is not enough and is not comparable with representing the graph as a set of random paths.

DeepCas, as the first proposed end-to-end deep learning method for cascade size prediction, exhibits better performance than feature-based approaches and diffusion

model-based approaches. But it still way worse than other deep learning based approaches, because of failing to consider temporal information and the topological structure of cascade graphs. The latest method, Topo-LSTM, also lacks time feature, so that it performs a little bit worse than DeepHawkes and our model. DeepHawkes, while successful in modeling cascades in a deep generative way, it does not perform the best due to its weak ability to learn structural information.

Finally, our proposed *CasCN* model, which purely relies on and fully explores structural and temporal information, significantly outperforms all baselines. For example, it achieves excellent prediction results with MSLE = 1.916 when observing for 3 hours in Sina Weibo and MSLE = 0.851 when observing for 7 years in HEP-PH, respectively. It reduces the prediction error by 15.2% and 30.9% comparing to the second best DeepHawkes.

When comparing methods with different observation time t_o, we clearly see a general pattern that the larger the t_o, the easier to make a good prediction. It is mainly because of the fact that longer t_o reveals more information for prediction.

4.4.7 Ablation study

To investigate and demonstrate the effectiveness of each component of our model (e.g., to understand the effect of the sampling part of *CasCN*), we present five variants of *CasCN*, where all are built upon the original *CasCN* model with some components changed. Their details can be found in Section 4.4.3.

The experimental results are shown in Table 4.4, from which we can see that our original *CasCN* leads to a certain reduction of prediction error when compared with other variants. From the comparison to CasCN-Undierected and CasCN-Time, we find that directionality and time decay effect are proved to be indispensable for cascade size prediction. Similarly, CasCN-Path brings a remarkable decrease of the prediction performance, which tells the necessity and effectiveness of sampling in *CasCN*. This indicates that the way to sample cascade graph as sub-cascade graph sequence can better reflect the dynamics of the cascade structure and the topological structure of each diffusion time.

In summary, sub-graphs sampling and structural-temporal modeling are critical components in *CasCN*, both of which essentially improve the performance of information cascade prediction as presented in the results.

4.4.8 Parameter analysis in CasCN

We now turn to investigating whether the parameters of *CasCN* have impact on the performance of cascade size prediction. The results are summarized in Table 4.5. We consider two vital parameters of graph convolutional kernel, i.e., the Chebyshev coefficient K and the largest eigenvalue λ_{max} of the Laplacian matrix Δ_c. For the

Table 4.5: Analysis of parameter impact on performance.

Dataset	Weibo Dataset		
Metric	MSLE		
Parameter \ t_o	1 hour	2 hours	3 hours
K=1	2.284	2.061	1.932
K=2	**2.242**	**2.036**	**1.910**
K=3	2.312	2.078	1.939
$\lambda_{max} \approx 2$	2.418	2.217	2.046
λ_{max} = real	**2.242**	**2.036**	**1.910**

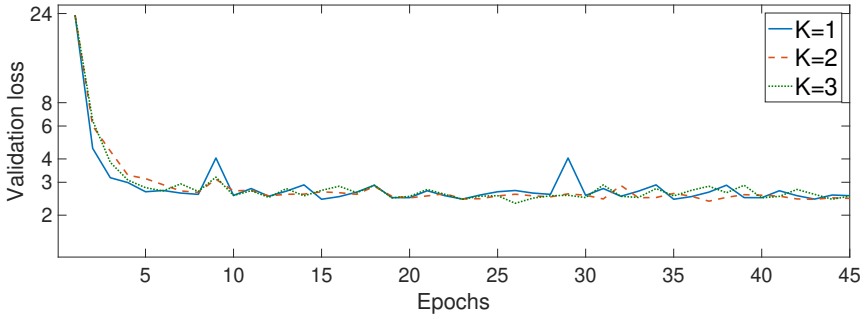
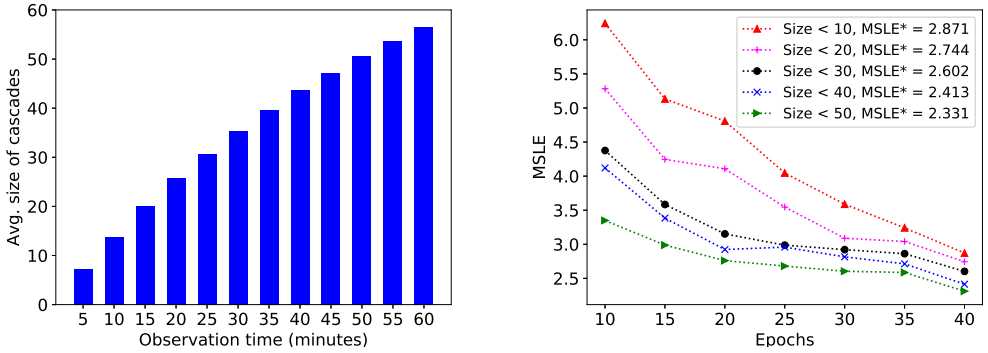


Figure 4.7: Loss of CasCN on the validation set with varying K



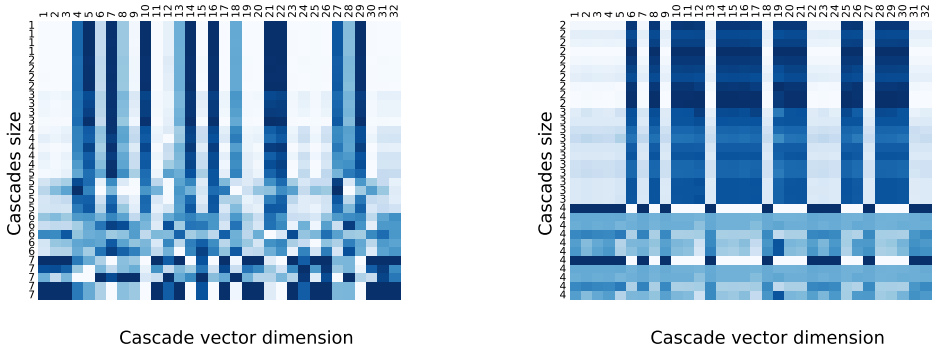
(a) Avg. cascade size distribution. **(b)** Results.

Figure 4.8: Impact of smaller-size observations.

Chebyshev coefficients K we selected from $\{1, 2, 3\}$. To obtain λ_{max}, we have two ways: the first is to approximate it as $\lambda_{max} \approx 2$, and the second is to compute the exact value of λ_{max} for each cascade graph. Table 4.5 shows that *CasCN* with $K = 2$ can achieve better performance than K=1 and 3. And in Figure 4.7, the validation loss in each epoch steadily declines. There is no evidence showing that a larger or smaller K is better than a median one. Further, bigger K will increase the computational cost. For the value of λ_{max}, precise values can lead to better prediction results at a higher computational cost.

We also investigated the performance of *CasCN* when the observed cascades are small – e.g., the size of the cascades is within 10, 20. Figure 4.8a gives the statistics of Weibo dataset illustrating the average cascade size increasing with time. Figure 4.8b shows that the MSLE results for various size decrease with training epochs. Apparently, the larger the size of the observed cascade, the lower the MSLE value *CasCN* achieves.

4.4.9 Discussions on feature learning

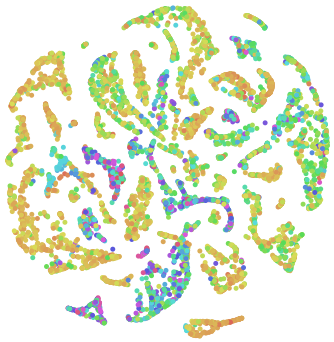


(a) Heatmap of Weibo **(b)** Heatmap of HEP-PH

Figure 4.9: Feature visualization. Figure (a) and (b) are learned representations by *CasCN*.

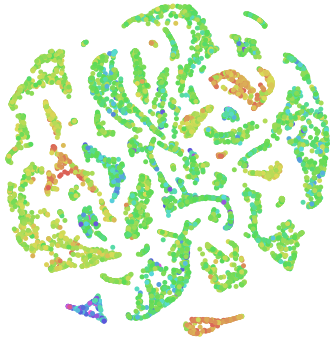
Finally, we discuss and demonstrate the capability of *CasCN* on feature learning in a visual way. We use the latent representation of each cascade graph $C_i(t_o) : \mathbf{h}^{'}(C_i(t_o))$ to plot a heatmap (as shown in Figure 4.9). The value in each dimension corresponds to some implicit or explicit features related to predicting the cascade size. Figure 4.9 tells us that the latent representation varies with cascade size. And surprisingly, there exists a clear pattern separation between outbreak (larger cascades) and non-outbreak (smaller) cascades, which indicates that *CasCN* is able to learn a good latent representation of cascades with different sizes and thus can be applied for outbreaking prediction.

Next, we try to understand/interpret the importance of some hand-crafted features in cascade size prediction. First, we use t-SNE [125] to project the vector representation summarized in $\mathbf{h}^{'}(C_i(t_o))$ for the cascade graph $C_i(t_o)$ to one data point in a 2-D space. Note that cascade graphs with similar vector representations are placed closely. Second, we color each data point (transformed from a cascade graph) using different strategies, such as based on the value of a certain feature f (e.g., number of leaf nodes, mean time, etc.), or the true increment size (the ground-truth label). The distribution of colors suggests a connection between the learned representations and network properties. That is, if a colored plot based on a certain feature f is well correlated with that of the true increment size, this feature is positively useful

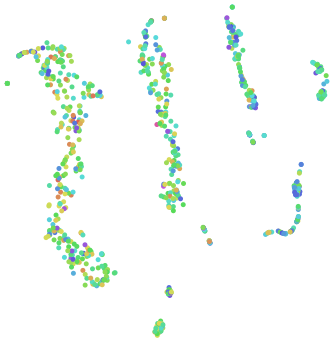


(a) f: Leaf nodes plot for Weibo

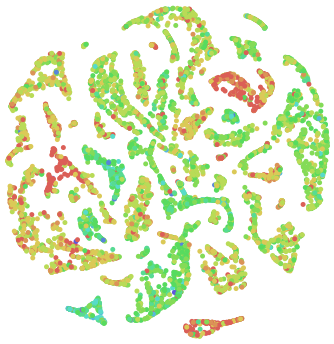
(b) f: Leaf nodes plot for HEP-PH



(c) f: Mean time plot for Weibo



(d) f: Mean time plot for HEP-PH



(e) Increment size plot for Weibo

(f) Increment size plot for HEP-PH

Figure 4.10: Feature visualization. In Figure (a) - (f), we layout the cascade graphs as data points in the test set to a 2-D space using t-SNE. Every layout is colored using hand-crafted network properties or the ground-truth (captioned "f: feature").

for cascade size prediction. Take the Weibo dataset as an example: Figure4.10a and 4.10c have similar color distributions with the true increment size 4.10e – we believe that leaf nodes and mean time are two important features for cascade size prediction.

4.5 Summary

In this chapter we proposed a novel deep learning based framework – CasCN. It is an end-to-end modeling framework for cascade growth prediction that does not rely heavily on feature engineering and can be easily generalized; enabling the information cascade prediction by exploiting both structural and temporal information. The CasCN model can learn a better latent representation for cascade graph with less information, using structural and temporal information of cascades within a deep learning framework. Incorporating the directionality of cascades and time decay effect further improves the prediction performance. Our experiments conducted on two scenarios, i.e., forecasting the size of re-tweeting cascades in Sina Weibo and predicting the citation of papers in HEP-PH, demonstrate that CasCN outperforms various state-of-the-art methods.

Chapter 5

Extracting Multi-scale Information for Cascades Modeling

5.1 Chapter Overview

Following the successes of deep learning methods in many fields, recently researchers have developed various neural network-based models to extract diverse features from the cascade graph that can be used for cascade prediction. For example, researchers have leveraged RNNs and attention mechanisms to automatically learn the cascade's temporal characteristics in a sequential learning manner by sampling the cascades as random walks or diffusion paths [8, 9, 35, 36]. These approaches, however, fail to capture topological structure features and the dynamic changes of information diffusion. Later studies [10, 69] introduce graph embedding methods to handle the structural diffusion learning problem, and have achieved promising results in cascade prediction. Despite the significant progress made by recent deep learning-based approaches, currently they are still confronted with several limitations:

(L1) Lack of efficient ways to sample cascade graphs. Most of the existing studies try to decompose a cascade graph into a bag of nodes [8] or denote it as a set of diffusion paths [9]. These methods either ignore the structural information or fail to capture the time-evolving structure of the cascade. CasCN [10] breaks down the original cascade graph into a sequence of sub-graphs based on timestamps, which may introduce bias and increase computation cost because there is a large number of timestamps in the diffusion process.

(L2) Incomplete structural feature extraction. Structural features are demonstrated as one of the most powerful features in information cascade prediction [10, 69]. Existing works not only capture nodes' first-order information but also take the edges' directional information into consideration. However, they still fail to capture long-distance message passing between nodes and nodes' position information in the cascade graph.

(L3) Absence of feature-level fusion. After obtaining different features from the cascade graph, e.g., temporal and structural features, most of the current works try to directly concatenate them and then fed them into a fully connected layer to make predictions [8, 9, 10]. However, different features play different roles in information cascades prediction, which necessitates a more fine-grained feature fusion that would facilitate predictions.

To overcome the limitations mentioned above, we first define multi-scale information for cascade graph, including (1) direction-scale, representing the propagating direction of the information between nodes; (2) high-order-scale, which is the higher-order interactions between the nodes; (3) position-scale, which means the sequential/position information of each node (i.e., the emerging time of each node in the diffusion); and (4) dynamic-scale, which is the dynamic information captured from the evolving sub-graph sequence. Then, we propose **MU**lti-scale **Cas**cades model (MUCas) – a novel framework for modeling the information cascades and predicting the increment size of information items. MUCas first employs time interval-aware sub-cascade graph sampling method, which decomposes the observed cascade graph into a sequence of sub-cascade graphs based on *disjoint* time intervals. And then it uses a multi-scale graph capsule network and an influence attention to learn and fuse the multi-scale information to form a unique cascade representation for popularity prediction.

We make the following contributions:

- **Efficient sampling method (L1).** We propose a novel cascade sampling method to sample sub-cascade graphs based on disjoint time-intervals. This method can significantly decrease the number of required sub-cascade graphs and eliminate the bias in processing the dynamic-scale in cascade modeling.
- **Multi-scale information learning (L2).** We design a multi-scale graph network for modeling sub-cascade graphs that can capture direction-scale, high-order-scale, and position-scale features of information diffusion jointly. Simultaneously, we design a neural function to learn the influence-attention between dynamic-scale sub-cascade graphs.
- **Hybrid feature aggregation (L3).** We propose a capsule-based hybrid aggregation layer, which selectively aggregates the learned multi-scale features in a more fine-grained way, i.e., from order-level and node-level to graph-level.
- **Comprehensive evaluations.** We conducted extensive experiments on two benchmark datasets. The results demonstrate that MUCas can significantly improve the prediction accuracy on cascade size prediction compared to the state-of-the-art baselines.

This chapter is based on the following publication:

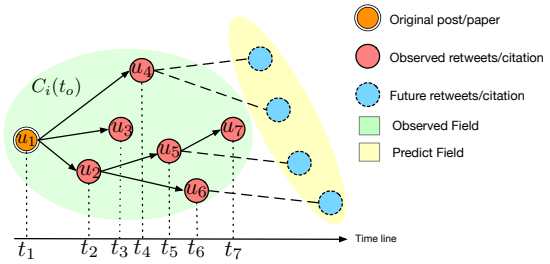


Figure 5.1: A Toy example of cascade graph.

- **Chen, X.**, Zhang, F., Zhou, F., Bonsangue, M.: Multi-scale graph capsule with influence attention for information cascades prediction. International Journal of Intelligent Systems 37 (2022) 2584–2611

5.2 Preliminaries

As illustrated in Figure 5.1, we can find that (1) the cascade graph is dynamic, in other words, cascade graph is evolving over time, i.e., new nodes will join in the diffusion process with time elapsed that leads the cascade size growth. For example, at t_7, u_7 joins in the diffusion process, and make the cascade size increase to 7; (2) the message between two nodes, e.g., u_1 and u_2, can be only passed from u_1 to u_2, i.e., the message passing in cascade graph is directed; (3) nodes are infected in chronological order, and the sequential information can be regarded as the position of each node in the cascade graph; (4) nodes with high influence will indirectly infect the long-distance nodes, and the long-distance dependency is the higher-order information of each node in a cascade graph, e.g., u_1 to u_6.

In this work, we define the aforementioned aspects as the multi-scale information, including (1) **dynamic-scale**, (2) **direction-scale**, (3) **position-scale**, and (4) **high-order-scale** of cascades.

Definition 9 ***Cascade Size Prediction*** – *Given the observed cascade graph $C_i(t_o)$ of post m_i, the goal of cascade size prediction is predicting m_i's incremental size ΔS_i, which is defined as $\Delta S_i = |U^{t_p}| - |U^{t_o}|$, where t_p and t_o are the prediction time and the observation time, respectively; and $|U^t|$ denotes the number of nodes, in terms of the size of cascade graph at time t.*

5.3 MUCas: Multi-Scale Graph Capsule with Influence Attention for Information Cascades Prediction

In this section, we present the proposed MUCas model, as well as its implementation details and computational complexity.

Table 5.1: Main notations used throughout this chapter.

Symbol	Description
$g_i^{T_j}, \mathbf{A}_i^{T_j}$	A sub-cascade graph of $C_i(t_o)$ at time interval T_j and it's adjacency matrix.
$G_i^{T_{iv}}, \mathbf{A}_i^{T_{iv}}$	A sequence of time interval-aware sub-cascade graphs of $C_i(t_o)$ and the corresponding adjacency matrices.
$\mathbf{PE}(u)$	Positional encoding.
$\mathbf{S}$, $\mathbf{H}_l$	A set of node embedding matrix, and a node embedding matrix.
$\mathbf{h}_m$ $\mathbf{n}_m$ $\mathbf{g}_i$	Order-level capsule. Node-level capsule. Graph-level capsule.
$\boldsymbol{\lambda}_j$, α_j	Time decay effect, influence attention.

Overview. Figure 5.2 illustrates the overall framework and the main components of **MUCas**, which consists of four major parts: (1) a time interval-aware sampling layer to generate sub-cascade graphs from observed cascade graph; (2) MUG-Caps to learn the direction-scale, position-scale and high-order-scale information from sub-cascade graphs; (3) influence attention for dynamic-scale learning; and (4) a prediction layer for cascade size prediction.

5.3.1 Time interval-aware Sub-cascade Sampling

Taking the observed cascade graph $C_i(t_o)$ of a given post m_i as input, the existing works try to decompose the observed cascade graph into a bag of nodes [8] or denote it as a set of diffusion paths [9]. Such methods either ignore both local and global structural information or fail to consider the dynamic information. Recently, some works such as CasCN [10] and VaCas [69] use a *Time-aware* sampling method to decompose $C_i(t_o)$ into a sequence of consecutive sub-cascade graphs based on the diffusion timestamp, which has been proved to be an efficient way to treat the observed cascade graph. However, the *Time-aware* sampling method still faces some challenges: 1) the difference between the fine-grained sub-graphs is trivial, which will introduce biases in dynamic modeling; and 2) too many sub-graphs would significantly increase computation cost. Figure 5.3a shows a toy example of *Time-aware* sampling method. Compared with the previous time step, each sub-graph only contains one more node (e.g., t_1 vs. t_2). Finally, it would generate z sub-graphs in total, where z is the number of varying time-stamps in the propagation process, resulting a huge number of sub-graphs within a short time. However, the difference between consecutive graphs are too trivial to be distinguished, which may confuse the model to learn discriminative features of information propagation.

In order to address the aforementioned challenges, we propose a new *Time interval-aware* sampling method, as shown in Figure 5.3b. This sampling method breaks down the observed cascade graph $C_i(t_o)$ into l discrete sub-cascade graphs $G_i^{T_{iv}} =$

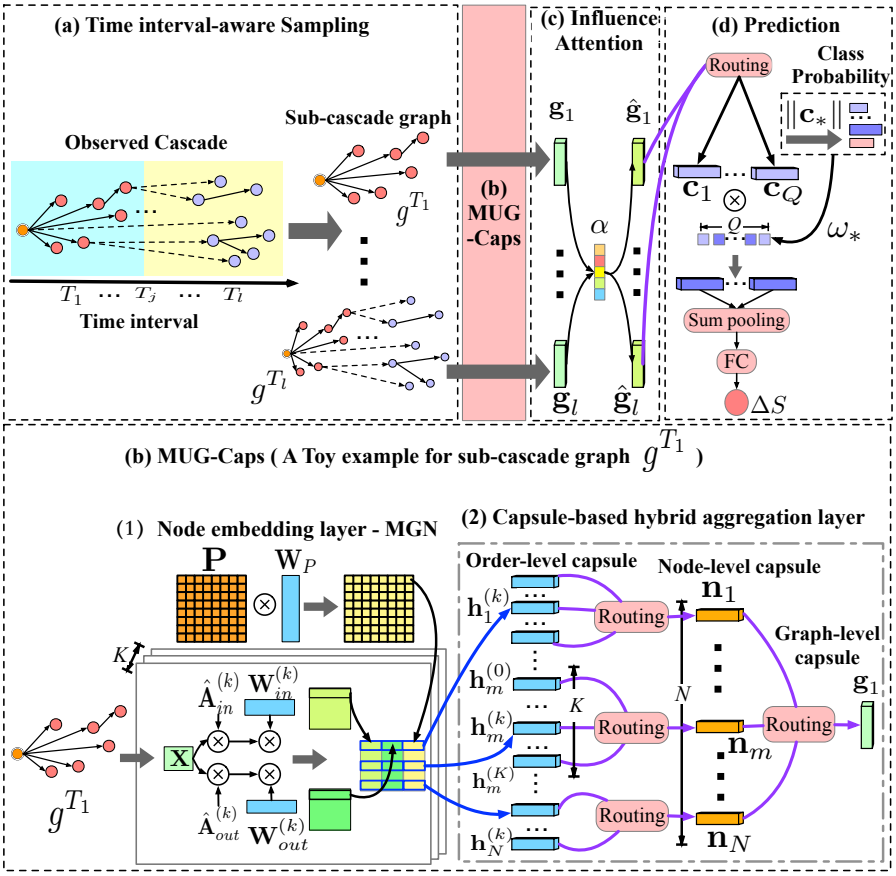


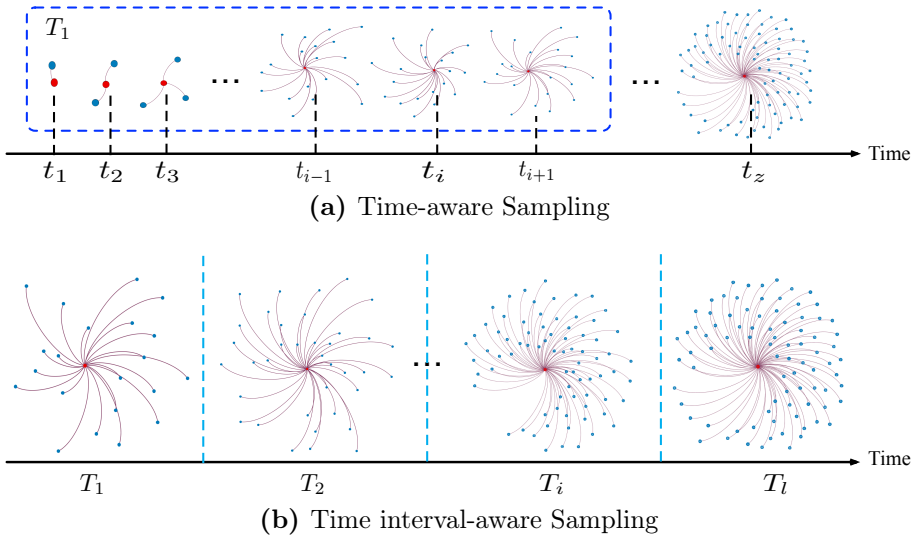
Figure 5.2: Overview of **MUCas**: (a) A time interval-aware sub-cascade graphs sampling layer; (b) MUG-Caps layer; (c) Influence attention layer; and (d) The prediction layer.

$\left\{g_i^{T_1}, \cdots, g_i^{T_j}, \cdots, g_i^{T_l}, |j \in [1, l]\right\}$. Specifically, we first split the observation time window t_o into l disjoint time intervals. Then, we sample sub-cascade graphs based on these intervals. Each sub-cascade graph in $G^{T_{iv}}$ is represented by an adjacency matrix. Thus, $G_i^{T_{iv}}$ is further represented as a sequence of adjacency matrices $\mathbf{A}_i^{T_{iv}} = \left\{\mathbf{A}_i^{T_1}, \mathbf{A}_i^{T_2}, \cdots, \mathbf{A}_i^{T_l}\right\}$.

Since the proposed sampling rule transforms the observed cascade graph into a fixed number of sub-graphs, it is possible that no new retweet/citation occurs in one of the intervals. To address this issue, we use the sub-graph of the interval before the empty one as padding to ensure that the final length of $G^{T_{iv}}$ equals l. Algorithm 3 formalizes the process of the *Time interval-aware* sampling method.

5.3.2 Multi-scale Cascade Representation Learning

After generating l discrete sub-cascade graphs, MUCas turns to learn high-level representation of these sub-cascade graphs, which contains the multi-scale information of cascade graphs. Inspired by the recent success of graph neural networks [41, 61, 126] and capsule network [127] in handling graph structured data,



(a) Time-aware Sampling

(b) Time interval-aware Sampling

Figure 5.3: Illustration of sampling sub-cascade graph sequence: time-aware sampling vs. time interval-aware sampling. z is the number of timestamps; l is the number of time intervals, and l is fixed and set manually.

we propose a MUlti-scale Graph Capsule Network (MUG-Caps) to learn the latent representation for cascade graph from $G^{T_{iv}}$. MUG-Caps is composed of two main parts, i.e., (1) the *node embedding layer* and (2) the *capsule-based hybrid aggregation layer*.

Node Embedding Layer: We propose a Multi-scale Graph Network (MGN) as the node embedding module, which learns node representations at the sub-graph level simultaneously from direction-scale, position-scale, and high-order-scale. The implementation of MGN is based on the graph convolutional networks (GCN) [61]. Original GCN proposes graph convolution approximations in the spectral domain based on graph Fourier transform, which is computationally efficient and achieves competitive performance in many tasks [128, 129]. However, it still faces some limitations in cascade modeling:

(1) GCN focuses on *static* and *undirected* graphs, whereas cascade graphs are *dynamic* and *directed* graphs.

(2) GCN updates a node's representation by aggregating its first-order neighbors and itself, failing to capture each node's infected order, i.e., the node's position information.

(3) GCN aggregates the high-order information for a node through stacking multiply graph convolutional layers. As demonstrated by many later improved works [126, 130], deeper GCN could not improve the performance and even performs worse in graph representation learning.

Our MGN addresses these limitations through revising the convolution kernel of

Algorithm 3: The algorithm for transforming cascade graph into a fixed-length sub-graph sequence: **Time interval-aware sampling**.

Input: Observed cascade graph $C_i(t_o)$, time window t_o, and time interval number l.

Output: A fixed-length sub-graph sequence $G_i^{T_{iv}} = \left\{g_i^{T_1}, \cdots, g_i^{T_l}\right\}$

```
for n = 1,2,...,l do
    for Set of nodes U_i(t_j), set of edges E_i(t_j) and corresponding timestamp t_j in C_i(t_o) do
        Compute the time interval index m = ⌊(t_j − t_1)/⌈t_o/l⌉⌋ + 1.
        if m ≤ n then
            Add U_i(t_j) and E_i(t_j) into g_i^{T_n}.
        end if
    end for
end for
```

GCN, which is defined as:

$$\mathbf{H} = g_\theta * \mathbf{X} = \sigma[\mathop{\Vert}_{k\in\mathcal{O}}, \mathop{\Vert}_{\phi\in\{in,out\}} (\hat{\mathbf{A}}_\phi^{(k)}\mathbf{X}\mathbf{W}_\phi^{(k)}), \mathbf{P}\mathbf{W}_P], \tag{5.1}$$

where $\mathop{\Vert}_{k\in\mathcal{O}}$ and $\mathop{\Vert}_{\phi\in\{in,out\}}$ represent the order-level concatenation and direction-level concatenation, respectively; $[\cdot]$ is a tiling concatenate operation; σ is an element-wise activation such as ReLU; $\hat{\mathbf{A}}_\phi^{(k)}$ denotes the normalized adjacency matrix $\hat{\mathbf{A}}_\phi \in \mathbb{R}^{N\times N}$ multiplied by itself k times; N is the number of nodes in current sub-cascade graph; $\mathbf{X} \in \mathbb{R}^{N\times F}$ is the input graph signal – F is the dimension number; and $\mathcal{O}$ is a set of integer adjacency powers – the value of $\mathcal{O}$ is from 0 to the max-order K of the current sub-cascade graph. $\phi \in \{in, out\}$ represents the in- and out-directions of the adjacency matrix, respectively. $\mathbf{A} = \mathbf{A}_{in} = (\mathbf{A}_{out})^T$. The asymmetric normalized adjacency matrix $\hat{A}_\phi$ of each direction can be calculated as:

$$\begin{aligned} \hat{\mathbf{A}}_\phi &= (\bar{\mathbf{D}}_\phi)^{-1}\bar{\mathbf{A}}_\phi, \\ \bar{\mathbf{A}}_\phi &= \mathbf{A}_\phi + \mathbf{I}_N, \end{aligned} \tag{5.2}$$

where $\mathbf{I}_N$ is the identity matrix, and $(\bar{\mathbf{D}}_\phi)_{ii} = \sum_j (\bar{\mathbf{A}}_\phi)_{ij}$ is the diagonal degree matrix.

In MGN, initially $\mathbf{X} = \mathbf{A}$. Further, $\mathbf{P} \in \mathbb{R}^{N\times F_p}$ is a position embedding matrix for current sub-cascade graph, and F_p is an adjustable dimension. Specifically, we initialize the position embedding matrix $\mathbf{P} = \{\mathbf{p}_1, \cdots, \mathbf{p}_u, \cdots, \mathbf{p}_N\}$ through positional encoding $\mathbf{PE}(u)$ [131] as:

$$\begin{aligned} \mathbf{PE}(u)_{2d} &= \sin(u/10000^{2d/d_p}), \\ \mathbf{PE}(u)_{2d+1} &= \cos(u/10000^{2d/d_p}), \end{aligned} \tag{5.3}$$

where $1 \leq d \leq F_p/2$ denotes the dimension index in $\mathbf{p}_u$. The details of this formula are referred to [131].

In Eq.(5.1), $\mathbf{W}_{\phi}^{(k)} \in \mathbb{R}^{F \times F_d}$ is the weight matrix for each direction on different order, and $\mathbf{W}_P \in \mathbb{R}^{d_p \times F_p}$ is another weight matrix used to transform position embeddings. The output node embedding matrix is denoted as $\mathbf{H} \in \mathbb{R}^{N \times K \times (2F_d + F_p)}$ and $\mathbf{H} = \{\mathbf{h}_1, \cdots, \mathbf{h}_m, \cdots, \mathbf{h}_N\}$, respectively. When implementing MGN, there is no need to calculate $\hat{\mathbf{A}}_{\phi}^{k}$ for each order. Instead, we calculate $\hat{\mathbf{A}}_{\phi}^{k}\mathbf{X}$ via right-to-left multiplication. For example, when $k = 2$, $\hat{\mathbf{A}}_{\phi}^{2}\mathbf{X}$ is calculated as $\hat{\mathbf{A}}_{\phi}(\hat{\mathbf{A}}_{\phi}(\mathbf{IX}))$, where $\mathbf{I}$ is the identity matrix. MGN can be regarded as a single layer using multiple times during actual training. The calculation of MGN is outlined in Algorithm 4.

The rationale behind MGN: MGN handles the **direction-scale** of cascade through modeling the incoming and outgoing relations of the cascades, i.e. $\hat{\mathbf{A}}_{\phi}^{(k)}\mathbf{X}\mathbf{W}_{\phi}^{(k)}$ in Eq. (5.1). Moreover, because the directed graph is asymmetric, we use the asymmetric normalization $\hat{\mathbf{A}}_{\phi} = (\hat{\mathbf{D}}_{\phi})^{-1}\bar{\mathbf{A}}_{\phi}$ to replace the symmetric normalization $\hat{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$ used in vanilla GCN. MGN utilizes the adjacency matrix's multiple powers to mix the feature representations of higher-order neighbors in one graph convolutional layer via transitive closure, which is used to handle the **high-order-scale**. In our implementation, the sub-cascade graph is an acyclic directed graph, and the value of $\mathbf{A}_{ij}^{(k)}$ can be either 0 or 1. When $\mathbf{A}_{ij}^{(k)} = 1$, there is a path between v_i to v_j. For arbitrary power p and q, $\mathbf{A}_{ij}^{p}\mathbf{A}_{ij}^{q} = 0$ will always be held, which eliminates the problem of layer output – imposing the lower-order information on higher-order relations and increase the feature correlations [132]. As for the **position-scale**, MGN adds position embeddings to the convolution kernel, which enriches each node feature with its corresponding position information.

Capsule-based Hybrid Aggregation Layer From the node embedding layer, we obtain a set of node embedding matrix for each sub-cascade graph, denoted as $\mathbf{S} = \{\mathbf{H}_1, \mathbf{H}_2, \cdots, \mathbf{H}_l\}$, where $\mathbf{S} \in \mathbb{R}^{l \times N \times K \times (2F_d + F_p)}$. Inspired by the work of capsule networks [127, 133], we design a capsule-based hybrid aggregation layer to aggregate the learned node features from order-level, node-level, and graph-level through dynamic routing, respectively.

The general procedure of dynamic routing is shown in Algorithm 5: (1) Lower-level capsules $\mathbf{U} \in \mathbb{R}^{|\mathbf{U}| \times F_U}$ are linearly transformed through shared matrix $\mathbf{W} \in \mathbb{R}^{|\mathbf{U}| \times |\mathbf{S}| \times F_U \times F_S}$, where $|\mathbf{U}|$ and F_U are the number of lower-level capsules and the dimensions, respectively. Here we introduce $\mathbf{W}$ that not only guarantees the feature representation ability of the center vector after clustering, but also being able to identify the order of input features. The result of this step is a set of votes $\hat{\mathbf{V}} \in \mathbb{R}^{|\mathbf{U}| \times |\mathbf{S}| \times F_S}$ (cf. line 1 in Algorithm 5), where $|\mathbf{S}|$ and F_S are the number of upper-level capsule and the dimensions, respectively. (2) Upper-level capsules $\mathbf{S} \in \mathbb{R}^{|\mathbf{S}| \times F_S}$ are computed based on the votes via line 3–7 in Algorithm 5, where $c_{ij} \in \mathbb{R}^{|\mathbf{U}| \times |\mathbf{S}| \times 1}$

Algorithm 4: Calculation of multi-scale graph network.

Input: Feature matrix $\mathbf{X}$, normalized adjacency matrix $\hat{\mathbf{A}}_\phi$ for both in- and out-directions, a set of order powers $\mathcal{O}$ and its max-value $K = \max(\mathcal{O})$, and positional embedding matrix $\mathbf{P}$.

Parameters: $\{\mathbf{W}_\phi^{(k)}\}_{k\in\mathcal{O}}$ and $\mathbf{W}_P$.

$\hat{\mathbf{P}} := \mathbf{P}\mathbf{W}_P$
$\mathbf{B}_{in}, \mathbf{B}_{out} := \mathbf{X}$
for k in $\mathcal{O}$ **do**
  if $k = 0$ **then**
    $\mathbf{B}_{in} := \mathbf{I}\mathbf{B}_{in}$
    $\mathbf{B}_{out} := \mathbf{I}\mathbf{B}_{out}$
  else
    $\mathbf{B}_{in} := \hat{\mathbf{A}}_{in}\mathbf{B}_{in}$
    $\mathbf{B}_{out} := \hat{\mathbf{A}}_{out}\mathbf{B}_{out}$
  end if
  $\mathbf{H}^{(k)} := \mathrm{CONCAT}(\mathbf{B}_{in}\mathbf{W}_{in}^{(k)}, \mathbf{B}_{out}\mathbf{W}_{out}^{(k)}, \hat{\mathbf{P}})$
end for
$\mathbf{H} := \mathrm{CONCAT}(\mathbf{H}^{(0)}, \cdots, \mathbf{H}^{(k)}, \cdots, \mathbf{H}^{(K)})$
return $\sigma(\mathbf{H})$

Algorithm 5: Dynamic routing mechanism in MUCas.

Input: Lower-level capsules $\mathbf{U}$, iteration number τ.

Output: Upper-level capsules $\mathbf{S}$

for all lower-level capsules i: $\hat{\mathbf{v}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i$
$\mathbf{b}_{ij} \leftarrow 0$
for τ iterations **do**
  for all lower-level capsules i: $c_{ij} \leftarrow \mathrm{softmax}(\mathbf{b}_{ij})$
  for all upper-level capsules j: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{v}}_{j|i} + \mathbf{b}_j$
  for all upper-level capsules j: $\widetilde{\mathbf{s}}_j \leftarrow \mathrm{Squash}(\mathbf{s}_j)$
  for all lower-level capsules i to all upper-level capsules j: $\mathbf{b}_{ij} \leftarrow \mathbf{b}_{ij} + \hat{\mathbf{v}}_{j|i} \cdot \widetilde{\mathbf{s}}_j$
end for

is the coupling co-efficiency that helps weight the votes, and the non-linear "squash" function is denoted as $\text{Squash}(x) = \frac{\|x\|^2}{0.5+\|x\|^2}\frac{x}{\|x\|}$.

We add biases to the calculation of $\mathbf{s}_j$ at line 5, which can solve a critical problem in capsule networks – indistinguishableness between the positive inputs and negative inputs [134].

In this layer, we defined three levels of capsules, i.e., order-level capsule, node-level capsule, and graph-level capsule, whose specific definitions are as follows:

Order-level capsule is represented as $\mathbf{h}_m \in \mathbb{R}^{K\times(2F_d+F_p)}$, focusing on specific node embedding in the sub-cascade graph. It aims to aggregate the higher-order information for each node into one node-level capsule $\mathbf{n}_m \in \mathbb{R}^{1\times F_n}$, where F_n is the dimension of node-level capsule.

Node-level capsule: As for a specific sub-cascade graph g^{T_j}, it has a set of node-level capsules $\mathbf{N}_j = \{\mathbf{n}_1, \cdots, \mathbf{n}_m, \cdots, \mathbf{n}_N\}$, $\mathbf{N}_j \in \mathbb{R}^{N\times F_n}$, used $\mathbf{N}_j$ to generate the graph-level capsule $\mathbf{g}_j \in \mathbb{R}^{1\times F_g}$ via dynamic routing, where F_g is the dimension of graph-level capsule.

Graph-level capsule: We have a set of graph-level capsules $\mathbf{G}$, each of which corresponds to a specific sub-cascade graph, i.e., $\mathbf{G} = \{\mathbf{g}_1, \cdots, \mathbf{g}_j, \cdots, \mathbf{g}_l\}$, $\mathbf{G} \in \mathbb{R}^{l\times F_g}$. Note that each graph-level capsule contains the properties of the cascade from different time intervals.

How does MUG-Caps work? Above we presented the details of the two components of MUG-Caps, i.e., MGN and the capsule-based hybrid aggregation layer. Now we turn to explain how the two components collaborate. MUG-Caps, as shown in Figure 5.2, takes a subgraph as input, which is first fed into an MGN layer to learn the embedding for each node that contains node direction-scale information, higher-order-scale information, and position-scale information as detailed in Section 5.2. After this layer, each node is represented as K vectors, with each vector matched to a different order level, referred to as the order-level capsules. Next, these order-level capsules are fed into the hybrid aggregation layer. Then dynamic routing is employed to aggregate order-level capsules to form a node-level capsule for each node. Finally, the dynamic routing aggregates these node-level capsules to build a graph-level capsule for the subgraph. The concrete calculation of MUG-Caps is shown in steps 3 to 11 in Algorithm 6.

5.3.3 Sub-graph Level Influence Attention

Previous works have demonstrated that user influence will decay significantly with time [9, 10]. In our work, we aim to learn such influence changes (dynamic-scale) at the sub-graph level, i.e., we assume that the sub-cascade graph's influence decays as the interval index increases. Inspired by self-attention mechanism [135], we employ a neural function to learn the influence attention. First, we represent the time-interval

as a one-hot vector $\mathbf{t}^j \in \mathbb{R}^l$, and then map $\mathbf{t}^j$ to $\boldsymbol{\lambda}_j$ through a fully-connected layer with sigmoid function. Here the $\boldsymbol{\lambda}_j$ is used to describe the time decay effect.

$$\boldsymbol{\lambda}_j = \text{sigmoid}(\mathbf{W}_t \mathbf{t}^j + \mathbf{b}_t), \tag{5.4}$$

where $\mathbf{W}_t \in \mathbb{R}^{F_g \times l}$ and $\mathbf{b}_t \in \mathbb{R}^{F_g}$. According to the time decay effect vector $\boldsymbol{\lambda}_j$ and graph-level capsules $\mathbf{G}$, we define the influence attention as:

$$\alpha_j = \frac{\exp(\langle \mathbf{w}, \boldsymbol{\lambda}_j \odot \mathbf{g}_j \rangle)}{\sum_{i=1}^{l} \exp(\langle \mathbf{w}, \boldsymbol{\lambda}_i \odot \mathbf{g}_i \rangle)} \tag{5.5}$$

where $\mathbf{w} \in \mathbb{R}^{F_g}$. Given the influence attention α_j, we calculate the graph-level capsule as:

$$\hat{\mathbf{g}}_j = \alpha_j \mathbf{g}_j \tag{5.6}$$

5.3.4 Information Cascade Prediction

Though our work focus on information popularity prediction, unlike existing works, we add an auxiliary task – an extra classification task i.e., whether a cascade would go viral, as a supplementary for the cascade size prediction. That is, we predict whether a cascade can break out a certain threshold value. This step is also implemented using dynamic routing over $\hat{\mathbf{G}} = \{\hat{\mathbf{g}}_j | j \in [1, l]\}$ to generate class capsules $\mathbf{C} \in \mathbb{R}^{Q \times F_c}$, where Q is the number of class, and F_c is the dimension of class capsules. The norm of class capsule $\|\mathbf{c}_q\|$ represents the probability belonging to class q. And, we use a margin loss to calculate the classification loss:

$$\ell_1 = \sum_q \left\{ \mu_q \text{max}(0, m^+ - \|\mathbf{c}_q\|)^2 + \xi(1 - \mu_q)\text{max}(0, \|\mathbf{c}_q\| - m^-)^2 \right\} \tag{5.7}$$

where $m^+ = 0.9$, $m^- = 0.1$, and $\mu_q = 1$ iff the cascade belongs to class q. Here ξ is used to stop initial learning from reducing the length of all class capsules, especially when Q is large.

Subsequently, we use a weighted sum operation on $\mathbf{C}$ to obtain the representation for a cascade $\hat{\mathbf{C}}$. The weight is calculated through $\|\mathbf{c}_*\|$:

$$\begin{aligned} \omega_q &= \frac{\exp(\|\mathbf{c}_q\|)}{\sum_{*=1}^{Q} \exp(\|\mathbf{c}_*\|)}, \\ \hat{\mathbf{C}} &= \sum_q \omega_q \mathbf{c}_q \end{aligned} \tag{5.8}$$

We use a fully-connected layer to predict the increment size $\Delta S = \text{FC}\left(\hat{\mathbf{C}}\right)$. The loss function is:

$$\ell_2 = \left(\log \Delta S - \log \Delta \widetilde{S}\right)^2 \tag{5.9}$$

where $\Delta\widetilde{S}$ is the ground truth. Finally, the overall loss function for a batch is

$$\mathcal{L} = \frac{1}{b}\sum_{i=1}^{b}(\beta\ell_1^i + (1-\beta)\ell_2^i) \tag{5.10}$$

where b is the number of cascades in a batch, and β is used to balance the ℓ_1 and ℓ_2 losses.

Algorithm 6: Learning with *MUCas*.

Input: Time-interval aware sub-cascade graphs for n cascade graphs $\mathbf{A}^{T_{iv}} = \{\mathbf{A}_1^{T_{iv}}, \cdots, \mathbf{A}_i^{T_{iv}}, \cdots, \mathbf{A}_n^{T_{iv}}, \}$; Max order K; Batch size b.
Output: Incremental size $\Delta S = \{\Delta S_1, \cdots\}$ of cascades.

repeat
  $b = 1, 2, ...$
  for adjacency matrix sequence $\mathbf{A}_i^{T_{iv}}$ in batch b **do**
    for $\mathbf{A}_i^{T_j} \in \mathbf{A}_i^{T_{iv}}$ **do**
      Compute the node embeddings $\mathbf{H}_j$ for j^{th} sub-cascade graph according to Eq. (5.1).
      for node embedding $\mathbf{h}_m \in \mathbf{H}_j$ **do**
        Use $\mathbf{h}_m$ to compute the node-level capsule $\mathbf{n}_m$ according to Algorithm 5.
      end for
      $\mathbf{N}_j \leftarrow \{\mathbf{n}_1, \cdots, \mathbf{n}_N\}$
      Use $\mathbf{N}_j$ to generate the graph-level capsule $\mathbf{g}_j$ according to Algorithm 5.
    end for
    $\mathbf{G}_i \leftarrow \{\mathbf{g}_1, \cdots, \mathbf{g}_l\}$
    Compute influence attention α_j according to Eq. (5.4), (5.5).
    Compute newly graph-level capsules $\hat{\mathbf{G}}_i$ according to Eq. (5.6).
    /*Extra classification task*/
    Use $\hat{\mathbf{G}}_i$ to compute class capsules $\mathbf{C}_i$ according to Algorithm 5.
    /*Popularity prediction task*/
    Calculate cascade representation $\mathbf{C}_i^{'}$ according to Eq. (5.8).
    Feed $\mathbf{C}_i^{'}$ into fully-connected layer to compute incremental size ΔS_i of cascade.
    Use Adaptive moment estimation (Adam) to optimize the objective function in Eq.(5.10) and update all the parameters.
  end for
until convergence;

5.3.5 Complexity Analysis

We finish this section with a discussion of the computational complexity of the main components in MUCas, i.e., the cost of (1) node embedding layer, and (2) capsule-based hybrid aggregation layer.

Node embedding layer: We use MGN to learn node embeddings from the sub-cascade graph, whose calculation is shown in Eq. (5.1). As for **direction-scale**, MGN models both the incoming and outgoing relations of the cascades via $\hat{\mathbf{A}}_{\phi}^{(k)}\mathbf{X}\mathbf{W}_{\phi}^{(k)}$. Recall that the dimensions of $\mathbf{X}$, $\mathbf{W}_{\phi}^{(k)}$, and $\mathbf{W}_p$ are F, F_d, and F_p, respectively. Besides, the max-order is K, and the normalized adjacency matrix $\hat{\mathbf{A}}_{\phi}$ in our implementation is a sparse matrix with $|\mathcal{E}|$ nonzero elements – $|\mathcal{E}|$ is the number of edges in current graph. In addition, the number of nodes is denoted as N. According to existing works [61, 126], for a single direction at each order, the calculation is conducted via sparse-dense matrix multiplications, and the computational complexity is $\mathcal{O}(|\mathcal{E}| \times F \times F_d)$. Taking the high-order-scale into consideration, the computational complexity is then $\mathcal{O}(K \times |\mathcal{E}| \times F \times F_d)$. As for **positional-scale**, the calculation of $\mathbf{W}_p\mathbf{P}$ is completed via matrix multiplication, which requires $\mathcal{O}(N \times d_p \times F_p)$ computations. Therefore, the total computational time cost of evaluating Eq. (5.1) is then $\mathcal{O}(2 \times K \times |\mathcal{E}| \times F \times F_d + N \times d_p \times F_p)$.

Capsule-based hybrid aggregation layer: This layer is implemented by executing dynamic routing between different levels capsules. Specifically, we adopt a dynamic routing mechanism for τ iterations over $|\mathbf{U}|$ lower-level capsules and generate $|\mathbf{S}|$ upper-level capsules. This learning process requires $\mathcal{O}(\tau \times |\mathbf{U}| \times |\mathbf{S}|)$ computations [136]. From the MGN, we get the order-level capsule $\mathbf{H} = \left\{\mathbf{h}_1, \cdots, \mathbf{h}_m, \cdots, \mathbf{h}_N | \mathbf{h}_m \in \mathbb{R}^{K \times (2F_d + F_p)}\right\}$. For each node m, it generates the node-level capsule from order-level capsules, whose computational complexity is $\mathcal{O}(\tau \times K \times 1)$. Because there are N nodes, the total time of generating node-level capsules for all nodes is $\mathcal{O}(N \times \tau \times K \times 1)$. Similarly, generating the graph-level capsule from node-level capsules can be done within $\mathcal{O}(\tau \times N \times 1)$ time. The overall computational time is therefore $\mathcal{O}(N \times \tau \times K \times 1 + \tau \times N \times 1)$.

5.4 Evaluation

In this section, we compare the performance of our proposed model *MUCas* with the state-of-the-art approaches, and several variants of *MUCas*, on information popularity prediction. In particular, we provide the quantitative results to answer the following research questions:

- **Q1**: How does MUCas perform on cascade size prediction compared with the state-of-the-art baselines?
- **Q2**: How do different scales of information modeled in MUCas contribute to the overall performance?
- **Q3**: How do the key hyper-parameters affect the performance of MUCas?

Table 5.2: Statistics of datasets

Dataset	Weibo	APS
# Ori. Cascades	119,313	636,294
# Nodes	5,918,473	636,294
# Edges	12,204,245	3,425,508
Avg. popularity	173	13
Train (0.5 Hour/ 3 Years)	19,472	24,636
Val (0.5 Hour/ 3 Years)	4,173	5,279
Test (0.5 Hour/ 3 Years)	4,172	5,278
Train (1 Hour/ 5 Years)	19,124	33,408
Val (1 Hour/ 5 Years)	4,098	7,159
Test (1 Hour/ 5 Years)	4,097	7,158

5.4.1 Datasets

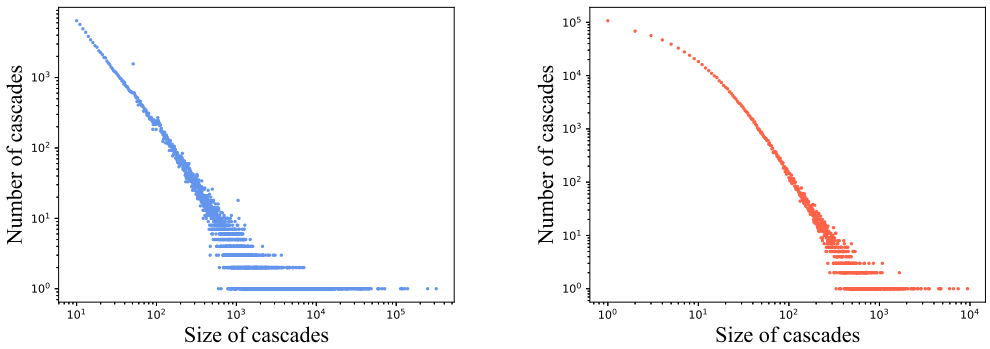
We evaluate the effectiveness and generalizability of *MUCas* on two scenarios. The first one is to predict the size of retweet cascades in Sina Weibo and the second one is to forecast the citation count of papers in citation dataset APS. The statistics of the datasets used in this work has shown in Table 5.2.

Sina Weibo: The description of the Weibo dataset is given in Section 4.4.1. Figure 5.4a plots the distribution of the cascade size (the number of re-tweets of each post), which, obviously, follows an power-law distribution and reflects the Pareto principle (80/20 rule). Figure 5.5a shows the distribution of depth over all cascades, which roughly follows an exponential distribution, indicating that the majority of cascades have a shallow depth, i.e., most of them are less than 5. The depth of a cascade is the length of the longest path, which also equals to the max-order of the cascade. Due to the effect of diurnal rhythm in Weibo [9], in our experiments, the cascades with the publication time before 8 am and after 6 pm were filtered out, leaving each post at least 6 hours to obtain retweets. As shown in Figure 5.6a, on average, a message receives about 70% retweets within 5 hours.

APS: APS [1] is provided by American Physical Society (APS), which consists of pairs of APS articles that cite each other for the corpus of Physical Review Letters, Physical Review, and Reviews of Modern Physics from 1893 to 2018. The papers from 1893 to 1997 are selected as observations so that each of the papers is allowed to develop for at least 20 years. In the citation scenario, the size of a cascade is the citation count. Figure 5.4b shows the distribution of cascade size in the APS dataset, which exhibits a power-law distribution. Figure 5.5b shows the distribution of the depth over all cascades in APS, which has a similar trend with Weibo. As shown in Figure 5.6b, on average, the citation reaches around 50% of the final size within 5 years.

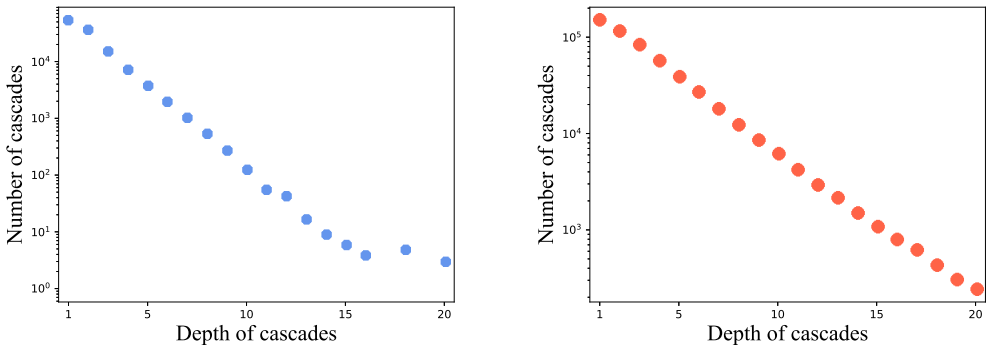
[1] http://journals.aps.org

Settings: In our experiments, we use the same dataset settings as in previous works [9, 69]. The observation time window in Weibo data is set to $t_o = 0.5$ and $t_o = 1$ hour. For the observation window of the APS dataset, we choose $t_o = 3$ and 5 years. For both Weibo and APS datasets, we filter out cascades whose observed size $S_{obs} < 10$. And for those cascades whose $S_{obs} > 100$, we only track the first 100 retweets. In addition, we randomly split each dataset into a training set (70%), a validation set (15%), and a testing set (15%) following existing works [9, 69].



(a) Weibo dataset **(b)** APS dataset

Figure 5.4: Cascade size distributions of Weibo (left), and APS (right) in log-log scales.



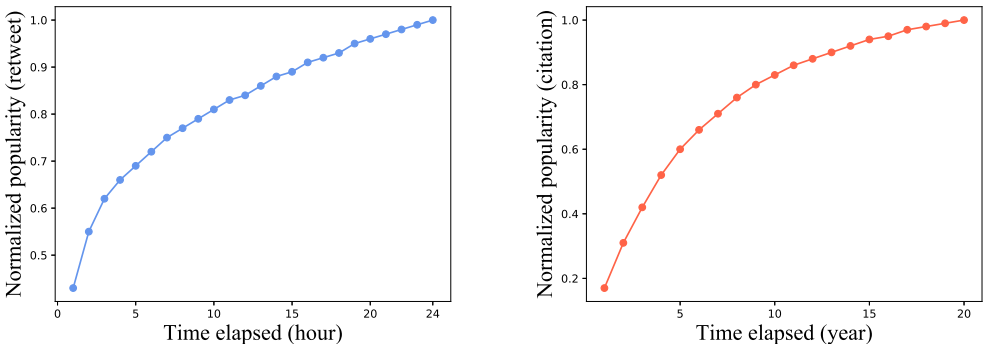
(a) Weibo dataset **(b)** APS dataset

Figure 5.5: Distribution of depth (max-order) of cascades.

5.4.2 Baselines

To validate MUCas's performance in cascade prediction, we select following state-of-the-art baselines for comparison:

- **Feature-Linear**, **Feature-Deep**, **DeepCas** [8] and **DeepHawkes** [9] are already introduced in Section 4.4.2, for brevity, we will not repeat them here.



(a) Weibo dataset **(b)** APS dataset

Figure 5.6: Normalized popularity distribution of cascades: retweets (citations) vs. time.

- **CasCN**: CasCN (in Chapter 4) is the first graph convolution network (GCN)-based framework exploiting both temporal and structural information for cascade size prediction. It decomposes a cascade graph into a sequence of sub-cascade graphs based on propagation time, while learning the local structure and its evolving process of cascade structure through the combination of graph convolutions and LSTM.

- **VaCas** [69]: VaCas is the first Bayesian learning-based approach that uses pre-trained node embeddings of the cascade as input and leverages a hierarchical variational information diffusion model to learn the posterior of cascade distribution with variational inference.

- **Cascade2vec** [62]: Cascade2vec is an improvement of CasCN, which proposes a new graph convolutional kernal – graph perception network (GPN) to replace the original GCN in CasCN. It also introduces the attention mechanism to learn different importance of each sub-graph.

5.4.3 Evaluation Metric

Following previous studies [8, 9, 35], we use mean square logarithmic error (MSLE) and symmetric mean absolute percentage error (SMAPE) for prediction performance evaluation. In addition, we also report the coefficient of determination (R^2) of

different models. The formulation of all evaluation metric is defined as:

$$\begin{aligned} \text{MSLE} &= \frac{1}{n}\sum_{i=1}^{n}\left(\log \Delta S_i - \log \Delta \widetilde{S}_i\right)^2 \\ \text{SMAPE} &= \frac{1}{n}\sum_{i=1}^{n}\frac{|\log \Delta S_i - \log \Delta \widetilde{S}_i|}{(|\log \Delta S_i| + |\log \Delta \widetilde{S}_i|)/2} \\ \text{R}^2 &= 1 - \frac{\sum_{i=1}^{n}(\log \Delta S_i - \log \Delta \widetilde{S}_i)^2}{\sum_{i=1}^{n}(\log \Delta \widetilde{S}_i - \frac{1}{n}\sum_{i=1}^{n}\log \Delta \widetilde{S}_i)^2} \end{aligned} \tag{5.11}$$

where n is the total number of posts, ΔS_i is the predicted incremental size for post m_i, and $\Delta \widetilde{S}_i$ is the ground truth. Note that, the value of MSLE and SMAPE the smaller the better, in contrast, the value of R^2 the bigger the better.

5.4.4 Experimental Settings and Parameter Tuning

Parameter settings: Models mentioned above are optimized to the best performance which involves several key hyper-parameters. The L_2 coefficient of Feature-Linear is chosen from $10^{\{0,-1,-2\cdots,-8\}}$. The node embedding size for DeepCas, DeepHawkes, CasCN, and Cascade2vec is set to 50. The hidden layer of each GRU has 32 units, and the hidden dimensions of the two-layer fully-connected layers for all deep learning-based methods are 32 and 16, respectively. The learning rate for node embeddings in DeepCas and DeepHawkes is 5×10^{-4} and the learning rate for other methods are 5×10^{-3}. The batch size is set as 64. All other hyper-parameters are set to the same values as used in the original papers.

As for our MUCas, the basic parameters (e.g., the learning rate is 5×10^{-4} and batch size is 64, etc.) are the same as above deep learning-based approaches, except that the max-order K and iteration number τ are chosen from 1 to 5. The embedding size of positional embedding is chosen from $\{30, 50, 100, 150, 200\}$, the hidden size for MGN is 60, and the hidden size for node-level capsule, graph-level capsule and the class capsule is 30, 8 and 16, respectively. In addition, the number of time intervals is set to 6. For the auxiliary classification task, the number of classes Q is equal to 2. Specifically, we label a cascade as 1 if its increased size is twice or more than its observed size, 0 otherwise. All methods, including ours, are tuned to the best performance with early stopping when validation errors has not declined for 10 consecutive epochs.

Experimental environment: The experiments are conducted on a sever with Intel E5-2680 v4 2.40GHz, one NVIDIA GeForce GTX 3090, and 256GB memory.

5.4.5 Performance Comparison (Q1)

The overall performance of MUCas as well as the state-of-the-art baselines are shown in Table 5.3 and Table 5.4, from which we have the following important observa-

Table 5.3: Performance comparison between baselines and MUCas on Weibo datasets. A paired t-test is performed and $*$ indicates a statistical significance $p < 0.001$ as compared to the best baseline method.

Model	Weibo					
	0.5 Hour			1 Hour		
	MSLE	SMAPE	R^2	MSLE	SMAPE	R^2
Feature-Linear	2.959	0.331	0.351	2.710	0.356	0.461
Feature-Deep	2.815	0.311	0.379	2.646	0.353	0.465
DeepCas	2.914	0.330	0.352	2.747	0.358	0.459
DeepHawkes	2.891	0.321	0.379	2.632	0.352	0.468
CasCN	2.804	0.311	0.381	2.601	0.350	0.468
Cascade2vec	2.752	0.308	0.384	2.589	0.348	0.479
VaCas	2.586	0.291	0.504	2.359	0.333	0.518
MUCas	**2.081**$*$	**0.271**$*$	**0.621**$*$	**1.882**$*$	**0.308**$*$	**0.647**$*$
(Improvements)	19.53%	6.87%	23.21%	20.22%	7.51%	24.90%

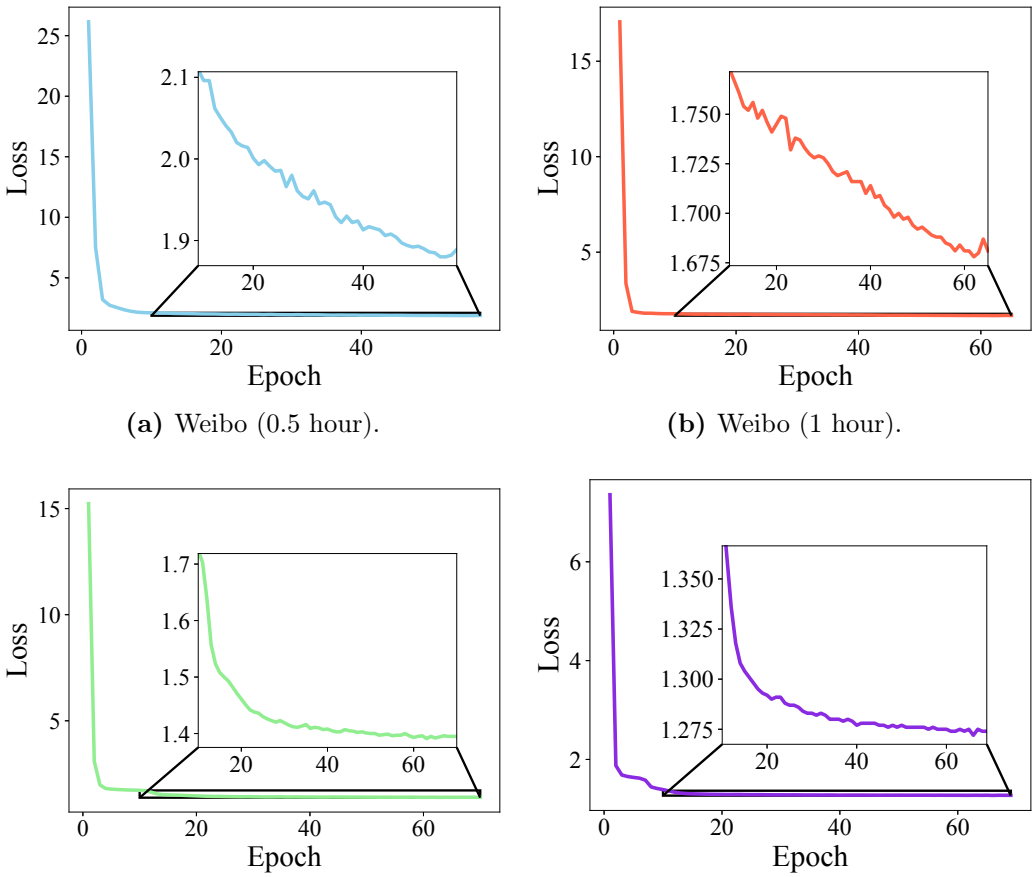
Table 5.4: Performance comparison between baselines and MUCas on APS datasets. A paired t-test is performed and $*$ indicates a statistical significance $p < 0.01$ as compared to the best baseline method.

Model	APS					
	3 Years			5 Years		
	MSLE	SMAPE	R^2	MSLE	SMAPE	R^2
Feature-Linear	2.100	0.289	0.126	2.087	0.358	0.311
Feature-Deep	1.996	0.358	0.221	1.874	0.352	0.322
DeepCas	2.033	0.361	0.213	1.944	0.365	0.318
DeepHawkes	1.831	0.344	0.241	1.588	0.337	0.363
CasCN	1.818	0.274	0.244	1.574	0.337	0.367
Cascade2vec	1.783	0.272	0.258	1.560	0.336	0.373
VaCas	1.723	0.268	0.283	1.507	0.335	0.394
MUCas	**1.557**$*$	**0.263**$*$	**0.355**$*$	**1.439**$*$	**0.333**$*$	**0.426**$*$
(Improvements)	9.63%	1.87%	25.44%	4.51%	0.59%	8.12%

tions.

(**O1**) MUCas outperforms the baselines by a large margin, e.g., as for the MSLE, it reduces the prediction error up to 19.53%, 20.22% on the Weibo dataset and 9.63% and 4.51% on the ASP dataset when compared to the best baseline – VaCas, when t_o is set to 0.5, 1 hour and 3, 5 years on Weibo and APS, respectively. We plot the training process of MUCas on the Weibo and APS dataset and show the results in Figure 5.7. Clearly, the training loss of MUCas consistently decreases and converges to a lower value.

(**O2**) The gap between handcrafted feature-based methods and most deep learning-based baselines are quite small. In some cases the handcrafted feature-based methods even beat some deep learning-based methods. Comparing the Feature-Deep with DeepCas, for example, we can observe that a fully connected layer is enough



(a) Weibo (0.5 hour). **(b)** Weibo (1 hour).

(c) APS (3 years). **(d)** APS (5 years).

Figure 5.7: Convergence of MUCas on Weibo and APS datasets.

to achieve competitive results than complicated neural networks (DeepCas) if we have a set of well-designed hand crafted features. However, obtaining such features requiring extensive domain knowledge, which is hard to be generalized to new domains.

(**O3**) DeepCas – the first deep-learning based approach for cascade size prediction – performs the worst among the deep learning baselines, because it simply learns the cascade representation based on sampled random walks but ignores temporal and topological information. DeepHawkes, while being successful in modeling temporal information for cascades in a generative learning manner, does not perform well due to its weak ability to learn structural information.

(**O4**) The rest of the baselines, i.e., CasCN, Cascade2vec and VaCas, generate competitive results because they explore structural and temporal information at the same time. When comparing CasCN with Cascade2vec, the performance of Cascade2vec is slightly better, due to the modified convolutional kernel in Cascade2vec, which indeed improves the ability of learning structural features. Besides, VaCas employed VAE [70] to solve the uncertainty problem in structural representation

learning and therefore achieves higher accuracy compared to other baselines. Our MUCas not only combines the advantages of CasCN and VaCas, but also takes into account the high-order-scale and position-scale of a cascade graph, leading to a significant improvement in prediction performance.

(**O5**) When examining the methods with different observation window t_o, we can observe a general trend, i.e., the larger the t_o, the accurate the predictions. This is intuitive because longer observation time reveals more temporal and structural knowledge regarding information diffusion that helps cascade size prediction.

5.4.6 Ablation Study (Q2)

To better investigate the contribution of each scale of information modeled in MUCas, we derive the following variants of MUCas:

-Direction: In "*-Direction*", we do not consider the directional relation in cascades, i.e., regarding the cascade graphs as undirected graphs. We replace the MGN to a vanilla GCN [61] and calculate the normalized adjacency matrix according to $\hat{\mathbf{A}} = \hat{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\hat{\mathbf{D}}^{-\frac{1}{2}}$.

-Order: In "*-Order*", we only focus on 1^{st}-order neighbors in the cascade graphs, i.e., setting the max-order number K to 1.

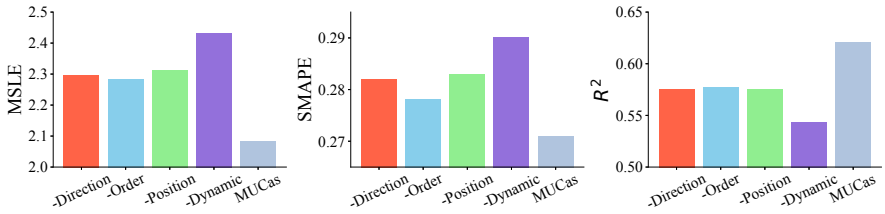
-Position: In "*-Position*", we ignore the node's relative position in cascade graph, i.e., removing the computation of position information $\mathbf{PW}_p$ in MGN.

-Dynamic: In "*-Dynamic*", we remove the sub-graph level influence attention component, and use $\mathbf{G}$ directly.

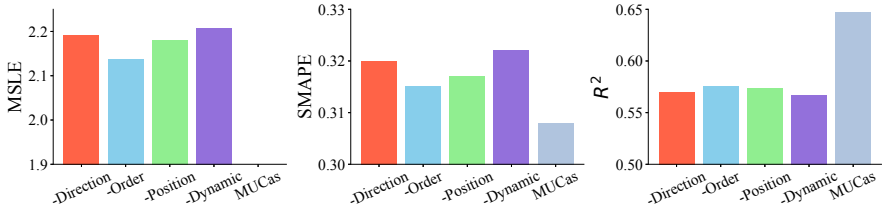
Figure 5.8 shows the performance comparisons among MUCas and its variants, which illustrates that: **(i)** the original MUCas achieves the best performance compared with other variants, demonstrating the motivation of our work, i.e., considering the four different scale information for cascade modeling. **(ii)** From the comparison between "-Direction" and "-Position", we find that effectively modeling the directional relation and node's relative position in the cascade graph will improve the prediction performance. **(iii)** Removing "-Order" and "-Dynamic" bring a remarkable decrease of the prediction performance, which implies that: (a) nodes with different orders play different importance in prediction task, and (b) the influence decreases as the cascade graph evolves.

In order to quantify the effectiveness of different levels of capsules, i.e., order-level, node-level, and graph-level capsule, we test the model performance by designing several single capsule-based variants of MUCas, including:

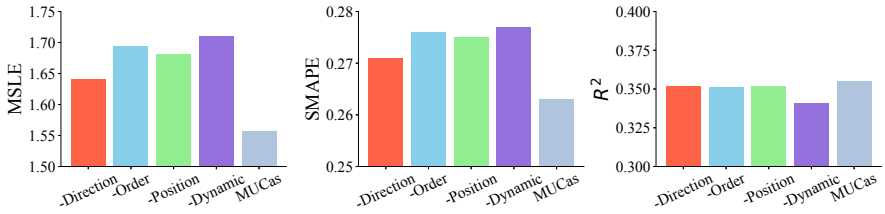
Order-level: In "*Order-level*", we apply sum-pooling to aggregate the order-level capsules $\mathbf{h}_*$ for each node and form node-level capsule $\mathbf{n}_*$. Finally, the sum-pooling



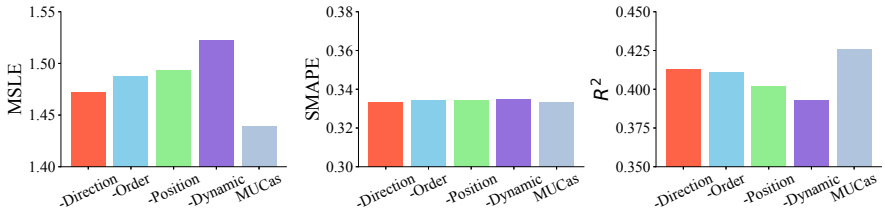
(a) Weibo – 0.5 hour



(b) Weibo – 1 hour



(c) APS – 3 years



(d) APS – 5 years

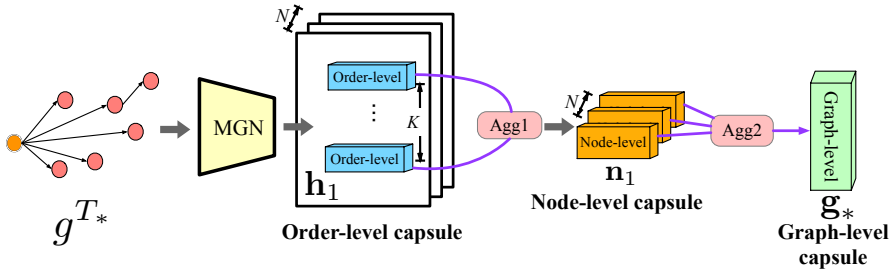
Figure 5.8: Ablation study of MUCas on two datasets.

operation is employed again to aggregate node-level capsules to form the graph-level capsule $\mathbf{g}_*$.

Node-level: In "*Node-level*", we apply dynamic routing to aggregate order-level capsules $\mathbf{h}_*$ to form node-level capsule $\mathbf{n}_*$. Subsequently, sum-pooling is used to aggregate node-level capsules to form the graph-level features $\mathbf{g}_*$.

Graph-level: In "*Graph-level*", we apply sum-pooling to generate node-level capsule $\mathbf{n}_*$ from $\mathbf{h}_*$, and then employ dynamic routing to aggregate node-level capsules to form the graph-level capsule $\mathbf{g}_*$.

Figure 5.9 illustrates the differences between three single capsule-based variants and MUCas. The experimental results are shown in Figure 5.10, where we can find that: (i) Compared to all single capsule-based variants, the original MUCas performs the best; (ii) Even keeping only one single-level capsule, the model performance is



Methods	Agg1	Agg2
MUCas	Dynamic routing	Dynamic routing
Order-level	Sum-pooling	Sum-pooling
Node-level	Dynamic routing	Sum-pooling
Graph-level	Sum-pooling	Dynamic routing

Figure 5.9: Illustration of single capsule-based variants of MUCas.

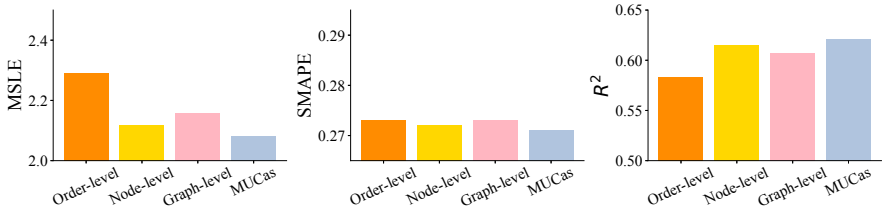
still superior to all the baselines in Table 5.3 and Table 5.4; and (iii) Compared to original MUCas, the "order-level" variant performs the worst, while the performance of the other two variants (i.e., node-level and graph-level) do not drop a lot. This result demonstrates that (1) the three-level capsules are indispensable, and (2) the dynamic routing is efficient in aggregating features.

5.4.7 Hyper-parameter Sensitivity (Q3)

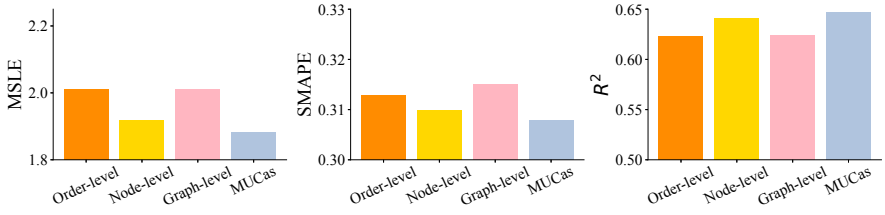
We study several important hyper-parameters that may influence the prediction performance of our model. Here we select the Weibo data set for the experiments and omit APS as the results are expected to be similar. The impact of the choice of hyper-parameters is shown in Figure 5.11.

- *Impact of max-order* K: we try different values of max-order K from 1 to 5, and the prediction results of MUCas are shown in Figure 5.11a and Figure 5.11b. When the observation time is 0.5 hour, our MUCas achieves the best performance if $K = 3$. But for 1 hour observation, the optimal value of K is 4. The reason is that with the increase of observation time, the information are more likely to propagated to more nodes with deeper depth.

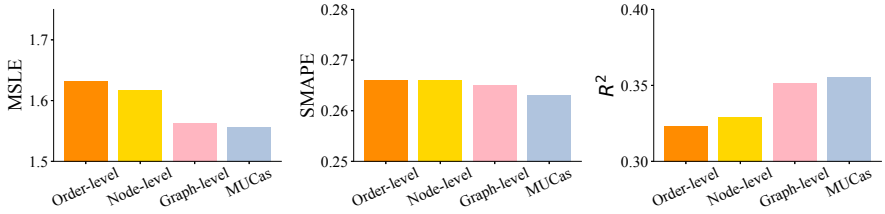
- *Impact of embedding size of positional embedding* F_p: we change the dimensions F_p of positional embedding $\mathbf{P}$ within $\{30, 50, 100, 150, 200\}$. Compare the results on 0.5 hour with the results on 1 hour (Figure 5.11c and Figure 5.11d), we can see that larger embedding size sometimes may degrade the performance, and the proper embedding size should fall into the scope of $[50, 150]$. We hypothesize the reason is that most cascade graphs are small, though a few of them may diffuse to a large number of nodes, i.e., the 80/20 rule as shown in Figure 5.4a and 5.4b. Therefore, a larger embedding dimensions would incur overfitting issue for those smaller cascade graphs.



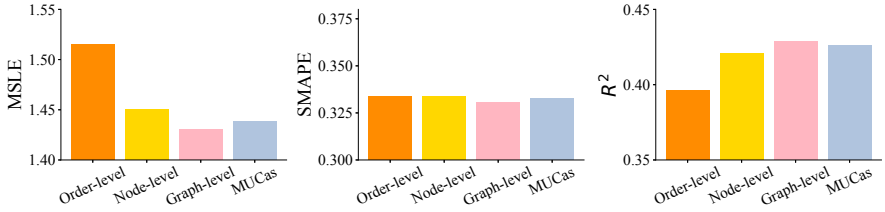
(a) Weibo – 0.5 hour



(b) Weibo – 1 hour



(c) APS – 3 years

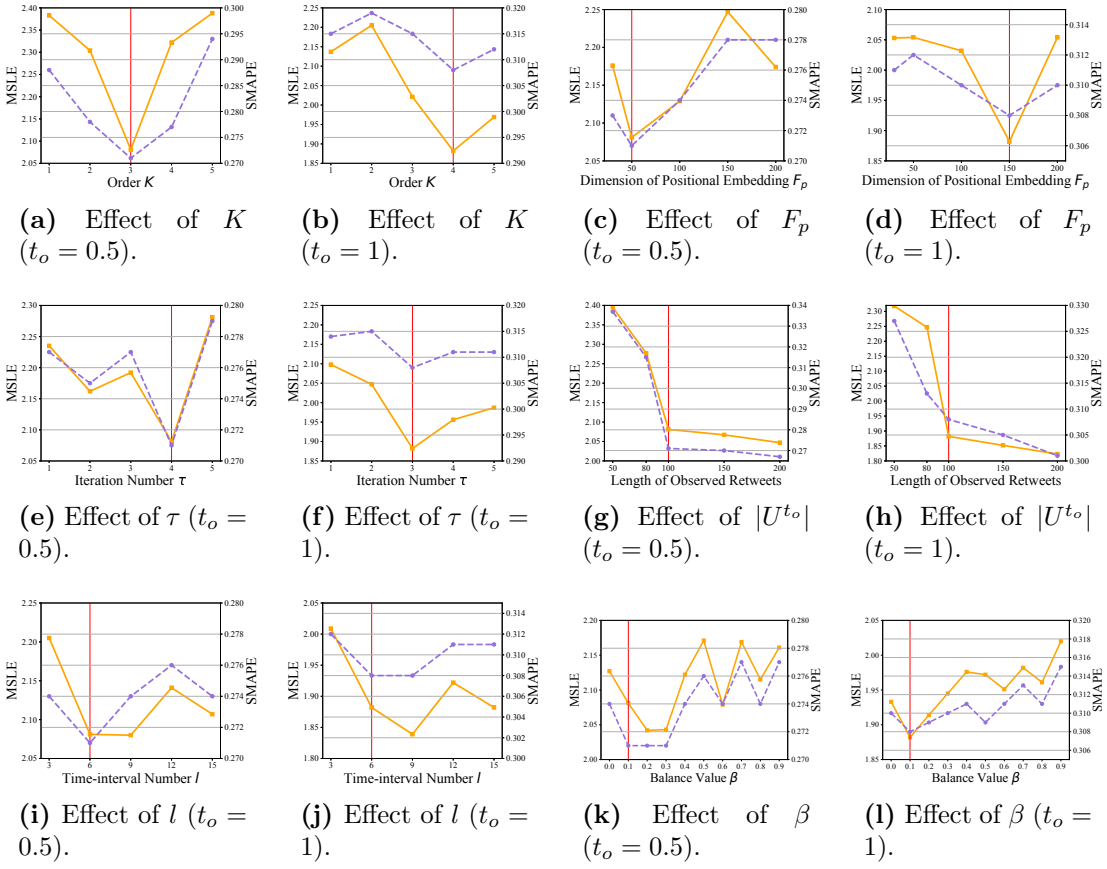


(d) APS – 5 years

Figure 5.10: Capsule study of MUCas on two datasets.

- *Impact of iteration number* τ: we try different values of the iteration number τ in dynamic routing, specifically, by increasing the value from 1 to 5. The results are shown in Figure 5.11e and Figure 5.11f, which suggests that increasing the number of iterations would improve the performance first, but deteriorate the model soon. This result raises an open issue in capsule network learning. That is, the appropriate iterations requires careful tuning that makes the capsule network unstable. This issue should be addressed for robust representation learning, which is beyond the scope of this work and left as future work.

- *Impact of the length of observed retweets* $|U^{t_o}|$: we track the first 50, 80, 100, 150, and 200 retweets and report the performance of MUCas based on these observed retweets. The experimental results show in Figure 5.11g and Figure 5.11h, where we can observe the improved performance with the increase of observation retweets, which is a natural result of including more training



(a) Effect of K ($t_o = 0.5$). **(b)** Effect of K ($t_o = 1$). **(c)** Effect of F_p ($t_o = 0.5$). **(d)** Effect of F_p ($t_o = 1$).

(e) Effect of τ ($t_o = 0.5$). **(f)** Effect of τ ($t_o = 1$). **(g)** Effect of $|U^{t_o}|$ ($t_o = 0.5$). **(h)** Effect of $|U^{t_o}|$ ($t_o = 1$).

(i) Effect of l ($t_o = 0.5$). **(j)** Effect of l ($t_o = 1$). **(k)** Effect of β ($t_o = 0.5$). **(l)** Effect of β ($t_o = 1$).

Figure 5.11: Impact of the important hyper-parameters on MUCas. Vertical lines are settings we used in previous experiments. Orange solid lines represent the results of MSLE, and the purple dashed lines denote the results of SMAPE.

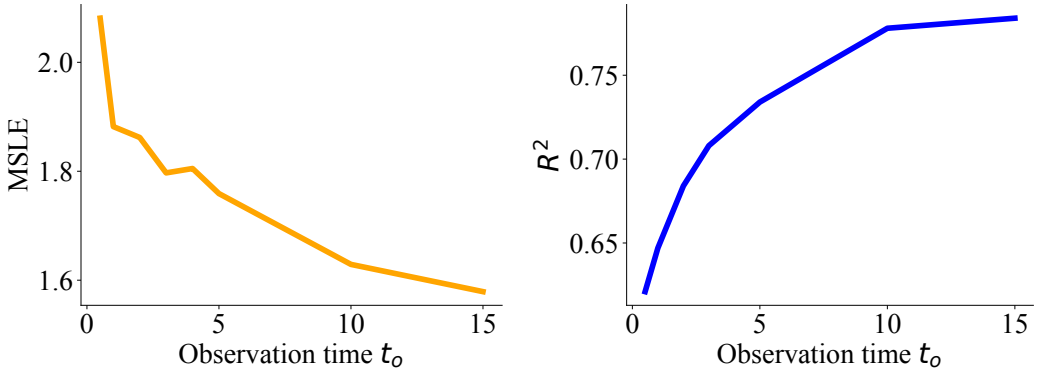
data.

- *Impact of the number of time intervals* l: when sampling sub-cascade graphs from the observed cascade, we set different numbers of time interval, i.e., $l = [3, 6, 9, 12, 15]$. The results are shown in Figure 5.11i and 5.11j, which reveal that: (1) the fine-grained sampling does not always perform better. For example, when the observation time is 0.5 hour, the performance of MUCas at $l = 6$ is much better than the results when $l = 9, 12$ and 15. This finding also proves our hypothesis in Section 5.3.1 that the differences between fine-grained sub-graphs become trivial as l grows, which will introduce biases in dynamic modeling. Besides, increasing the number of sub-graphs would significantly increase the computation cost. (2) The choice of the number of time interval heavily depends on the distribution of dataset. When the observation window is 0.5 hour and 1 hour, the model performance achieves the best at $l = 6$ and $l = 9$, respectively.

- *Impact of the balance value* β: we change the hyper-parameter β in loss function (Eq. (5.10)) from 0 to 0.9, and report the results in Figure 5.11k and

Figure 5.11l. We can see that a smaller value of β always achieves better performance. Furthermore, $\beta = 0.5$ can be regarded as a watershed, with a value less than 0.5 being far more suitable for model selection than a value greater than 0.5. Setting $\beta = 0$, which can be regarded as removing the auxiliary task from the original model. And from the results of $\beta = 0$, we can find that adding an auxiliary task indeed helps us to improve the model performance.

- In order to further support the finding (**O5**) in Section 5.4.5, we conduct an extra experiment to assess the model sensitivity varied with the observation time. Specifically, we consider the propagation in the first 15 hours because most of the messages in the Weibo dataset will decay after this time [11]. The results are shown in Figure 5.12, where we can clearly see that the longer the observations, the better the performance of the model.



(a) MSLE **(b)** R^2

Figure 5.12: The influence of time window in Weibo dataset.

5.4.8 Model Parameters and Computation Cost

Table 5.5: Model parameters and computation time measured by seconds when t_o = 0.5 hour and 3 years for Weibo and APS, respectively. "∼" means "approximatively".

Methods	Parameters	Time cost per epoch (in seconds)	
		Weibo (t_o = 0.5 hour)	APS (t_o = 3 years)
DeepCas	∼250M	∼150s	∼170s
DeepHawkes	∼250M	∼128s	∼158s
CasCN	∼278K	∼320s	∼400s
Cascade2Vec	∼368K	∼50s	∼75s
VaCas	∼2M	∼83s	∼98s
MUCas	∼495K	∼104s	∼105s

We compute the time cost of training MUCas and the baselines, as well as the required parameters. The results are reported in Table 5.5. First, the memory required for DeepCas and DeepHawkes is much higher because they need the embeddings of

all users in the social network, which required $|U| \times F_u$ parameters – $|U|$ represents the total number of users and F_u denotes the embedding size. Besides, the training time for each epoch of MUCas is around $104s$ and $105s$ in Weibo and APS dataset when $t_o = 0.5$ hour and $t_o = 3$ years, respectively. In contrast, Cascade2Vec is the fastest model that only needs 50s and 70s for Weibo and APS datasets, respectively, but the quality of the model is much lower.

5.5 Summary

In this chapter, we proposed a novel cascade prediction model – MUCas, which capture the multi-scale features regarding information diffusion comprehensively and make good predictions. Specifically, MUCas consists of four components: (1) a time interval-aware sampling layer used to generate sub-cascade graphs from the observed cascade graph, (2) MUG-Caps extracts the direction-scale, position-scale, and high-order-scale information from sub-cascade graphs, (3) an attention layer applied to learn dynamic-scale, and (4) a prediction layer to make predictions. We conducted extensive experiments based on two real-world datasets, i.e., Weibo and APS. The experimental results demonstrate that our method achieves state-of-the-art performance on information cascade size prediction of tweet propagation in social networks and scientific papers' impact.

Part II

Rumor Detection

Chapter 6

Modeling Hierarchical Diffusion for Rumor Detection

6.1 Chapter Overview

Conventional methods for rumor detection broadly fall into two groups: (1) *hand-crafted feature-based approaches* — mostly identifying and incorporating complicated hand-crafted features for rumor detection, including lexical features [23, 29], syntactic features [23, 79], visual features [24, 77], user profile related features [26, 83], and social relationship features [137, 138]. Their performance highly depends on the effectiveness of extracted features, which require extensive domain knowledge. (2) *credibility propagation-based approaches* [24, 31, 32], which aims to find the truth against conflicting information. These approaches usually leverage the inter-entity relations but heavily rely on the constructed credibility network for high rumor identification accuracy. Recent studies inspired by the successes of deep learning methods in many fields have developed various neural network-based models to learn several feature representations for rumor detection in an end-to-end way [16, 33, 34, 102]. Although these methods have shown performance improvements over the previous methods, they still face several critical limitations. First, most of the existing methods still require a large volume of textual data or a rich collection of users' comments as input [16, 34, 102]. In addition, previous works focused on either microscopic diffusion patterns that emphasize users' personal retweeting behavior or macroscopic diffusion structures depicting the full rumor in-network diffusion paths [16, 100].

To overcome the limitations mentioned above, in this chapter, we propose MMRD (**M**acroscopic and **M**icroscopic-aware **R**umor **D**etection), a novel deep learning-based framework for rumor detection. MMRD models the rumor diffusion from both macroscopic and microscopic perspectives through newly designed encoding components MacroE and MicroE and enhancing the diffusion representations through the cross-learning mechanism. We design a fusion gate to selectively aggregate learned

macroscopic and microscopic knowledge and introduce the attention mechanism to merge row-level information to form a unique rumor representation. The rumor prediction is generated based on the learned rumor representation. Moreover, a knowledge distillation technique is applied to further improve the model's detection performance. Our main contributions are summarized as follows:

- Firstly, we propose a new model to learn the representation of rumor through modeling the macroscopic and microscopic diffusion. The model is flexible and can be easily integrated into any existing approaches.
- Secondly, we design two encoding components for macroscopic and microscopic diffusion modeling, respectively, as well as the mechanism to control the information aggregation.
- Thirdly, MMRD employs a powerful technique–knowledge distillation to transfer knowledge from a teacher model to a student model, which further improves the model performance since the student capture more knowledge than the teacher.
- Finally, we conduct extensive evaluations on two benchmark datasets. The experimental results demonstrate that our model significantly outperforms existing baseline methods on rumor detection.

This chapter is based on the following publication[44]:

- **Chen, X.**, Zhou, F., Zhang, F., Bonsangue, M.: Modeling microscopic and macroscopic information diffusion for rumor detection. International Journal of Intelligent Systems36(2021) 5449–5471

6.2 Problem statement

We first borrow the definitions of macro-level and micro-level diffusion prediction from the field of information cascades modeling [71] to define macroscopic diffusion and microscopic rumor diffusion, and then give the formalized definition of rumor detection, which are formally defined as follows.

In information cascades modeling, the macro-level diffusion prediction aims at predicting the eventual size of a given cascade. Similarly, the macroscopic diffusion in our work refers to the evolution of the network scale, representing both the change of edges and nodes.

Definition 10 ***Macroscopic diffusion.*** *We denote the macroscopic diffusion as a diffusion graph $G = \{U, E\}$ (see Definition 2), where U is the user set comprising N users, and $E = \{(u_i, u_j)|u_i, u_j \in U\}$ represents a set of edges connecting pairs of users when u_j retweets u_i.*

Similar to micro-level diffusion prediction that aims to predict the next infected user, we define the user infected process as microscopic diffusion, i.e., who will engage in the information spreading and when this event (retweeting) occurs.

Definition 11 ***Microscopic diffusion.*** *We represent the microscopic diffusion as a user-time series* $\mathcal{P} = \{(u_1, t_1), \cdots, (u_j, t_j), \cdots, (u_N, t_N)\}$*, where* (u_1, t_1) *denotes user* u_1 *created source tweet at time* t_1*, and the rest of* (u_j, t_j) *tuples denote user* u_j *retweet the source tweet at time* t_j*. Here, all users are in chronological order according to their timestamps.* $\mathcal{P}$ *is also known as diffusion path in Definition 3*

Recall the definition of observed diffusion graph $G(t_o)$ and diffusion path $\mathcal{P}(t_o)$ in Definition 4, where t_o is the observation window, we now give a formal definition of rumor detection in this work:

Definition 12 ***Rumor detection.*** *Given a tweet* $m = \{G_m(t_o), \mathcal{P}_m(t_o)\}$ *within an observation window* t_o*, the goal of rumor detection is to learn a classification function* $f(m)$ *to classify* m *as a rumor or non-rumor.*

Others definitions, such as user vector, used in next sections, are formally given in Definition 5.

Table 6.1: Main notations used throughout this chapter.

Symbol	Description
$\mathbf{MacroE}(\cdot)$	Macroscopic diffusion encoding component.
$\mathbf{MicroE}(\cdot)$	Microscopic diffusion encoding component.
$\mathbf{H}_{Macro}$	The macroscopic diffusion representation learned by MacroE.
F^{Macro}	Dimension of macroscopic diffusion representation.
$\mathbf{H}_{Micro}$	The microscopic diffusion representation learned by MicroE.
F^{Macro}	Dimension of microscopic diffusion representation.
$\mathbf{H}_{Fuse}$	Importance-aware diffusion representation.
$\mathbf{H}_{Rumor}$	Rumor representation.
$\mathcal{K}$	A set of order powers.
K	Max order (i.e., max-value of $\mathcal{K}$).
$\hat{\mathbf{y}}, \mathbf{y}$	Prediction and the ground truth.

6.3 MMRD: Modeling Microscopic and Macroscopic Information Diffusion for Rumor Detection

The overall framework of the proposed model MMRD is shown in Figure 6.1. In particular, it consists of the following main components: (a) the input layer, which takes the observed diffusion graph $G(t_o)$ and diffusion path $\mathcal{P}(t_o)$ as inputs; (b) the normal training process, so-called the training process of a teacher model, the teacher model consists of (1) the macroscopic and microscopic diffusion encoding layer, including MacroE, MicroE, and cross-learning mechanism, (2) the fusion gate,

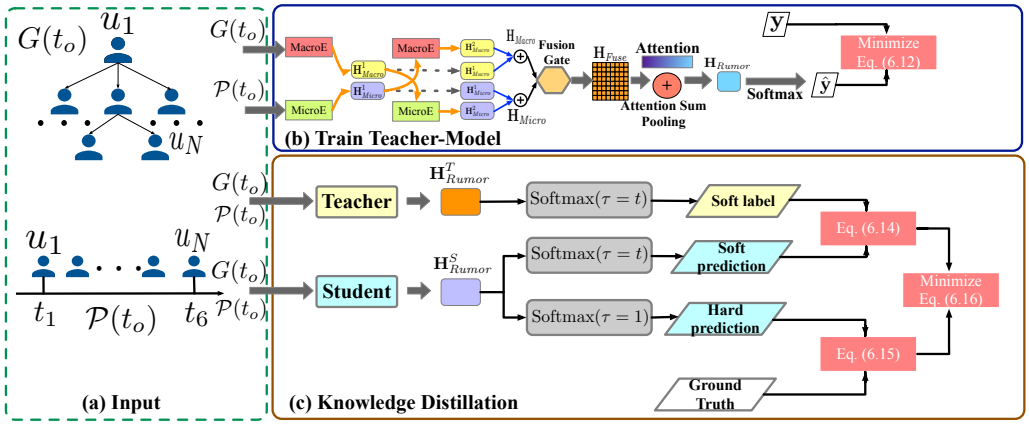
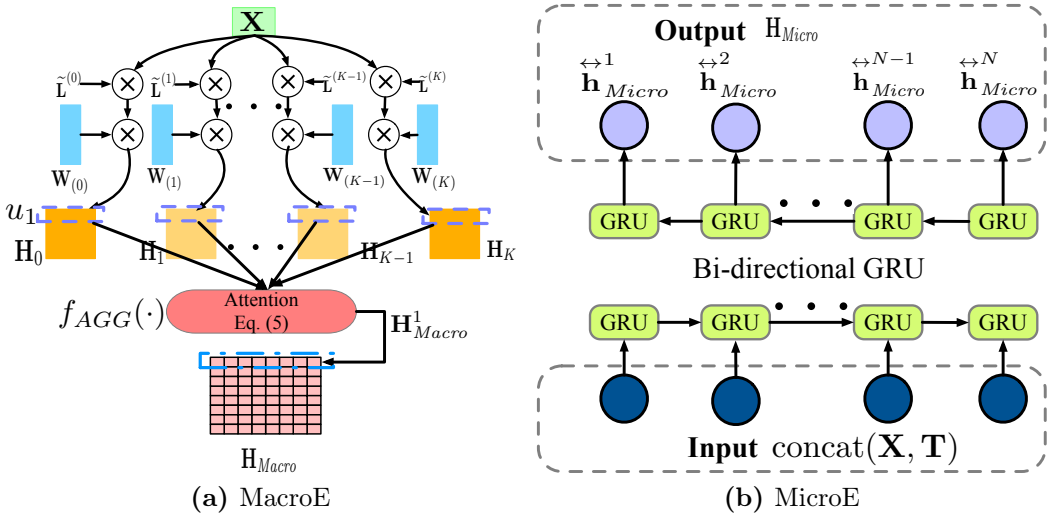


Figure 6.1: Overview of **MMRD**: (a) inputs of MMRD; (b) normal training process of MMRD; (c) the process of train MMRD with knowledge distillation.

and (3) the rumor detection layer; (c) knowledge distillation phase, which is used to further improve the model performance. With this model in mind, we first introduce two essential structural encoding components – **MacroE** and **MicroE**, and then discuss how to generate the unique rumor representation based on the two modules that will preserve both macroscopic and microscopic diffusion properties. Finally, we introduce how to use the knowledge distillation technique to develop a powerful student model for rumor detection.



(a) MacroE **(b)** MicroE

Figure 6.2: Illustration of diffusion encoding components.

6.3.1 Macroscopic Diffusion Encoding Component

The macroscopic diffusion of a tweet m reflects its diffusion scale. In our work, we cast the macroscopic diffusion modeling as learning the latent structural patterns from its diffusion graph $G_m(t_o)$. Inspired by the recent success of graph neural networks in processing the graph structural data, e.g., graph convolutional network (GCN) [41, 61] and graph attention network (GAT) [109], we implement the

macroscopic diffusion encoding component (**MacroE**) based on vanilla GCN [61]. The vanilla GCN is a multi-layer structure that contains several convolution layers, which is defined as:

$$\begin{aligned} \mathbf{H}^{(j+1)} &= \sigma\left(\widetilde{\mathbf{A}}\mathbf{H}^{(j)}\mathbf{W}^{(j)}\right) \\ \widetilde{\mathbf{A}} &= \mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I}_N)\mathbf{D}^{-\frac{1}{2}} \end{aligned} \tag{6.1}$$

where $\mathbf{H}^{(j)} \in \mathbb{R}^{N\times d_j}$ and $\mathbf{H}^{(j+1)} \in \mathbb{R}^{N\times d_{j+1}}$ are the input and output for layer j, $\mathbf{W}^{(j)} \in \mathbb{R}^{d_j\times d_{j+1}}$ is a trainable weight matrix and $\sigma(\cdot)$ is an activation function (e.g., Relu). $\widetilde{\mathbf{A}}$ is a symmetrically normalized adjacency matrix with self-connections, and $\mathbf{D}$ is a diagonal degree matrix. The adjacency matrix $\mathbf{A}$ and degree matrix $\mathbf{D}$ are expressed as the following:

$$\begin{aligned} \mathbf{A}_{ij} &= \begin{cases} 1 & \text{if } (u_i,u_j)\in E \text{ and } i\neq j, \\ 0 & \text{otherwise.} \end{cases} \\ \mathbf{D}_{ii} &= \sum_j \mathbf{A}_{ij} \end{aligned} \tag{6.2}$$

The initial input of the first GCN layer $\mathbf{H}^{(0)} = \mathbf{X}$, which is formed by user vectors, i.e., $\mathbf{X} = \{\mathbf{u}_1, \cdots, \mathbf{u}_N\} \in \mathbb{R}^{N\times F^{user}}$. Even the vanilla GCN shows powerful ability in graph embedding, it still faces some limitations: (1) it focuses on undirected graphs rather than the directed graph [10]; and (2) the nodes receive latent representations only from their immediate neighbors, cannot be summarized as higher-order adjacency information [111, 130].

To overcome the aforementioned limitations of GCN in modeling the directed graph and learning higher-order interactions, in this work, we reference the work of CasCN [10] (ref. Chapter 4) and MixHop [126], and extend the vanilla GCN. Finally, we propose a directed multi-hop graph convolutional network with attention aggregation as the MacroE (Figure 6.2a). The convolutional kernel of MacroE is defined as:

$$\begin{aligned} \mathbf{H}_{Macro} &= f_{AGG}\left[\sigma(\widetilde{\mathbf{L}}^{(k)}\mathbf{X}\mathbf{W}_{(k)})_{k\in\mathcal{K}}\right] \\ &= \sigma(f_{AGG}\left[\widetilde{\mathbf{L}}^{(0)}\mathbf{X}\mathbf{W}_{(0)} \vdots \widetilde{\mathbf{L}}^{(1)}\mathbf{X}\mathbf{W}_{(1)} \vdots \cdots \vdots \widetilde{\mathbf{L}}^{(K)}\mathbf{X}\mathbf{W}_{(K)}\right]) \\ &= \sigma(f_{AGG}[\mathbf{H}_{(0)} \vdots \mathbf{H}_{(1)} \vdots \cdots \vdots \mathbf{H}_{(K)}]) \end{aligned} \tag{6.3}$$

where $\mathbf{X} \in \mathbb{R}^{N\times F}$ is the input feature matrix and $\mathbf{H}_{Macro} = \{\mathbf{h}^1_{Macro}, \cdots, \mathbf{h}^N_{Macro}\} \in \mathbb{R}^{N\times F^{Macro}}$ is the output of MacroE. In order to capture the directional information from the diffusion graph, we replace the $\widetilde{\mathbf{A}}$ with $\widetilde{\mathbf{L}}$ – normalized Laplacian for directed

graph. The calculation of $\widetilde{\mathbf{L}}$ is defined as:

$$\begin{aligned}\mathbf{P} &= (1-\alpha)\frac{\mathbf{E}}{N} + \alpha\left(\mathbf{D}^{-1}\mathbf{A}\right),\\ \mathbf{L} &= \Phi^{\frac{1}{2}}\left(\mathbf{I}-\mathbf{P}\right)\Phi^{-\frac{1}{2}},\\ \widetilde{\mathbf{L}} &= \frac{2}{\lambda_{max}}\mathbf{L}-\mathbf{I}\end{aligned} \tag{6.4}$$

where $\mathbf{P}$ is a transition probability matrix, $\mathbf{E}\in\mathbb{R}^{N\times N}$ is an all–one matrix. $\alpha\in(0,1)$ is an initial probability used to restrict the state transition matrix $\mathbf{D}^{-1}\mathbf{A}$ a strongly connected matrix [10]. Φ is a diagonal matrix with entries $\Phi(v,v)=\phi(v)-\phi(v)$ is the column vector of $\mathbf{P}$ [117], and λ_{max} denotes the largest eigenvalue of $\mathbf{L}$.

$\mathcal{K}$ is a set of integer order powers – the value of $\mathcal{K}$ is from 0 to K, and $\mathbf{W}_k\in\mathbb{R}^{F\times F^{Macro}}$ is the weight matrix for k-hops. $\widetilde{\mathbf{L}}^{(k)}$ denotes the normalized Laplacian matrix $\widetilde{\mathbf{L}}$ multiplied by itself k times, and its value represents the probability connecting path from vertex u_i to vertex u_j in k-hops. Specifically, $\widetilde{\mathbf{L}}^{(0)}=\mathbf{I}$ is an identity matrix. Through the Laplacian matrix's multiple powers, MacroE mixes the feature representation of higher-order neighbors in one graph convolutional layer.

$f_{AGG}(\cdot)$ is an aggregation function, which is used to fuse the latent representation from different orders. In most of the existing works [112, 126], $f_{AGG}(\cdot)$ is similar to the pooling methods in CNNs, which can be a mean-pooling function, max-pooling function, or sum-pooling function. However, the distance of the message passing for each node is different, i.e., different nodes have different max-orders. In this work, we implement the aggregation function via the order attention mechanism at the node-level. As for each node u_j in the diffusion graph $G_m(t_o)$, it has a set of latent representations $\mathbf{h}^{u_j}_{(0)},\cdots,\mathbf{h}^{u_j}_{(K)}$ from K-orders. Then the order attention of u_j is calculated as:

$$\begin{aligned}a^{u_j}_{(k)} &= \frac{\exp(\langle\mathbf{w}_{u_j},\tanh(\mathbf{W}_{u_j}\mathbf{h}^{u_j}_{(k)}+\mathbf{b}_{u_j}\rangle))}{\sum_{*=1}^{K}\exp(\langle\mathbf{w}_{u_j},\tanh(\mathbf{W}_{u_j}\mathbf{h}^{u_j}_{(*)}+\mathbf{b}_{u_j}\rangle))},\\ \mathbf{h}^{u_j}_{Macro} &= \sum_{k=1}^{K}a^{u_j}_{(k)}\mathbf{h}^{u_j}_{(k)}\end{aligned} \tag{6.5}$$

where $\mathbf{W}_{u_j}\in\mathbb{R}^{F^{Macro}\times d}$, $\mathbf{b}_{u_j}\in\mathbb{R}^d$ and $\mathbf{w}_{u_j}\in\mathbb{R}^d$. So that, the aggregation function f_{AGG} is formulated as $f_{AGG}=\{\mathbf{h}^{u_j}_{Macro}=\text{Attention}(\mathbf{h}^{u_j}_{(0)},\cdots,\mathbf{h}^{u_j}_{(K)}),|j\in\{1,\cdots,N\}\}$. The calculation process of MacroE is outlined in Algorithm 7.

MacroE vs. GCN: As depicted above, the convolutional kernel in MacroE for one single order is similar to a single layer of GCN, i.e., $\widetilde{\mathbf{L}}^{(k)}\mathbf{X}\mathbf{W}_{(k)}$ and $\widetilde{\mathbf{A}}\mathbf{H}^{(j)}\mathbf{W}^{(j)}$, respectively. The main differences between our MacroE and GCN are: (1) we use normalized directed Laplacian $\widetilde{\mathbf{L}}$ to replace the symmetrically normalized adjacency matrix $\widetilde{\mathbf{A}}$ in GCN, which introduces the directional information of edges into the

Algorithm 7: Calculation of MacroE (Eq. (6.5)).

Input: Feature matrix $\mathbf{X}$, normalized laplacian matrix $\widetilde{\mathbf{L}}$, a set of order powers $\mathcal{K}$ and its max-value $K = \max(\mathcal{K})$.
Parameters: $\{\mathbf{W}_{(k)}\}_{k\in\mathcal{K}}$.
1: $\mathbf{B} := \mathbf{X}$
2: **for** $k = 0$ to K **do**
3: **if** $k = 0$ **then**
4: $\mathbf{B} := \mathbf{IB}$
5: **else**
6: $\mathbf{B} := \widetilde{\mathbf{L}}\mathbf{B}$
7: **end if**
8: $\mathbf{H}_{(k)} := \mathbf{B}\mathbf{W}_{(k)}$
9: **end for**
/*From step 1 to 9, complete the calculation of $(\widetilde{\mathbf{L}}^{(k)}\mathbf{X}\mathbf{W}_{(k)})_{k\in\mathcal{K}}$ in Eq. (6.3)*/
10: $\mathbf{H}_{Macro} := f_{AGG}([\mathbf{H}_{(0)}, \cdots, \mathbf{H}_{(k)}, \cdots, \mathbf{H}_{(K)}])$ via Eq. (6.5).
11: **return** $\sigma(\mathbf{H})$

convolution rather than only considering the link information between nodes; and (2) our MacroE can learn high-order information for each node by using one single layer; however, GCN relies on multi-layers and may introduce the over-smoothing issue in learning node feature representations [112].

6.3.2 Microscopic Diffusion Encoding Component

The microscopic diffusion encoding component (**MicroE**, shown in Figure 6.2b) aims to capture temporal patterns from the user engagement time series $\mathcal{P}_m(t_o)$. Inspired by the success of RNNs in sequential modeling, we employ a Bi-directional GRU (Bi-GRU) [108] as the encoding component, where the hidden states are used to memorize the diffusion history. At each step t_j, Bi-GRU takes the feature vector and previous hidden state as inputs and computes the updated hidden state as:

$$\overset{\leftrightarrow}{\mathbf{h}}^{j} = \text{Bi-GRU}(\mathbf{x}^j, \mathbf{h}^{j-1}), \overset{\leftrightarrow}{\mathbf{h}}^{j} \in \mathbb{R}^{F^{Micro}} \tag{6.6}$$

Then, the output of MicroE module is a sequence of hidden states $\mathbf{H}_{Micro} = \{\overset{\leftrightarrow}{\mathbf{h}}^{j}_{Micro} | j \in \{1, \cdots, N\}\} \in \mathbb{R}^{N\times F^{Micro}}$

6.3.3 Macroscopic and Microscopic Cross-learning

After introducing the necessary encoding components, we go to describe how to apply them to learn the latent representations from macroscopic and microscopic diffusion, summarized into two steps. In the first step, we train MacroE and MicroE separately. MacroE takes the feature matrix $\mathbf{X}$, the normalized Laplacian matrix $\widetilde{\mathbf{L}}$ and max-order number K as inputs. As for MicroE, we first represent the infected

timestamp of each user into one-hot vector $\mathbf{t}_j \in \mathbb{R}^{d_{time}}$, and then concatenate the timestamp vector matrix $\mathbf{T} = \{\mathbf{t}_1, \cdots, \mathbf{t}_N\}$ with $\mathbf{X}$ to form the input $\hat{\mathbf{X}}$ for MicroE. Specifically, assume that, the time window is $[t_1, t_o]$, and we first split the time window into l disjoint time intervals, and then compute the corresponding time interval for each retweet user u_j as $t^j_{int} = \left\lfloor \frac{t_j - t_1}{t_o/l} \right\rfloor$, where t_1 is the timestamp for the source post user, and t_j is timestamp for user u_j. Finally, each user's timestamp is falling into corresponding time intervals and each interval is related to a one-hot embedding, thus, for u_j its timestamp embedding equals to the related time-interval embedding. Note that, in our work, the initial feature matrix $\mathbf{X}$ is extracted from users' profiles. Figure 6.3 shows a toy example of the model inputs. The outputs of first step are $\mathbf{H}^1_{Macro}$ and $\mathbf{H}^1_{Micro}$, respectively:

$$\begin{aligned} \mathbf{H}^1_{Macro} &= \text{MacroE}(\mathbf{X}, \widetilde{\mathbf{L}}, K) \\ \mathbf{H}^1_{Micro} &= \text{MicroE}(\text{concat}(\mathbf{X}, \mathbf{T})) \end{aligned} \tag{6.7}$$

In the second step, we train MacroE and MicroE in a cross-learning manner. Specifically, we use $\mathbf{H}^1_{Macro}$ to train a new MicroE, and vice versa. The outputs of second step are $\mathbf{H}^2_{Macro}$ and $\mathbf{H}^2_{Micro}$:

$$\begin{aligned} \mathbf{H}^2_{Macro} &= \text{MacroE}(\mathbf{H}^1_{Micro}, \widetilde{\mathbf{L}}, K) \\ \mathbf{H}^2_{Micro} &= \text{MicroE}(\mathbf{H}^1_{Macro}) \end{aligned} \tag{6.8}$$

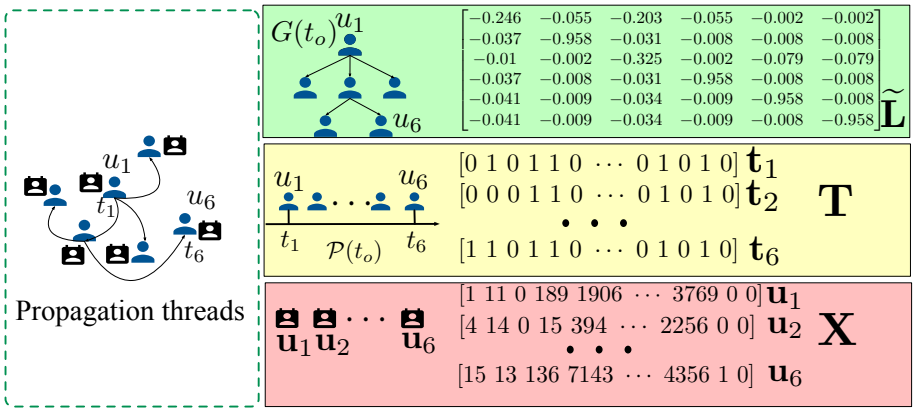


Figure 6.3: A toy example of the model inputs.

6.3.4 Feature fusion via hybrid aggregation layer

We concatenate $\mathbf{H}^1_{Macro}$ with $\mathbf{H}^2_{Macro}$ to form $\mathbf{H}_{Macro} \in \mathbb{R}^{N \times 2F^{Macro}}$, and concatenate $\mathbf{H}^1_{Micro}$ with $\mathbf{H}^2_{Micro}$ to form $\mathbf{H}_{Micro} \in \mathbb{R}^{N \times 2F^{Micro}}$. Thus, for each tweet m_i, we have a macroscopic representation $\mathbf{H}_{Macro}$ and a microscopic representation $\mathbf{H}_{Micro}$. In most of the existing works, after getting $\mathbf{H}_{Macro}$ and $\mathbf{H}_{Micro}$, they will concatenate them directly, however, this operation ignores the different dependence on the two different representations. In our work, in order to effectively aggregate the learned representations, inspired by the gate mechanism [108] and attention mechanism [131], we

design (1) a fusion gate to fuse $\mathbf{H}_{Macro}$ and $\mathbf{H}_{Micro}$ to form $\mathbf{H}_{Fuse}$, and (2) row-level attention to aggregate features to merge a unique representation $\mathbf{H}_{Rumor}$.

To selectively integrate the important information of two representations, we employ a concise and effective fusion gating mechanism that produces an importance-aware diffusion representation $\mathbf{H}_{\text{Fuse}}$ as follows:

$$\begin{aligned} \mathbf{G} &= \text{sigmoid}(\mathbf{W}^1_{gate}\mathbf{H}_{Macro} + \mathbf{W}^2_{gate}\mathbf{H}_{Micro} + \mathbf{b}_{gate}) \\ \mathbf{H}_{Fuse} &= \mathbf{G} \odot \mathbf{H}_{Macro} + (1 - \mathbf{G}) \odot \mathbf{H}_{Micro} \end{aligned} \tag{6.9}$$

where $\mathbf{W}^1_{gate}, \mathbf{W}^2_{gate} \in \mathbb{R}^{2F' \times 2F'}$, and $\mathbf{b}_{gate} \in \mathbb{R}^{2F'}$. Note that, here $F' = F^{Macro} = F^{Micro}$. $\mathbf{G}$ is used to drop trivial parts of macroscopic representation and add important information from microscopic representation. The rationale behind this design is that the representation fusion $\mathbf{H}_{Fuse} = \{\mathbf{h}^1_{\text{Fuse}}, \cdots, \mathbf{h}^N_{\text{Fuse}}\} \in \mathbb{R}^{N \times 2F'}$ would be aware of the different importance of macroscopic and microscopic diffusion.

Then, we merge the row-level information of $\mathbf{H}_{Fuse}$ to form an unique representation $\mathbf{H}_{Rumor}$ for tweet m through attention sum-pooling operation:

$$\begin{aligned} a_j &= \frac{\exp(\langle \mathbf{w}, \tanh(\mathbf{W}_a \mathbf{h}^j_{\text{Fuse}} + \mathbf{b}_a \rangle))}{\sum_{*=1}^{N} \exp(\langle \mathbf{w}, \tanh(\mathbf{W}_a \mathbf{h}^*_{\text{Fuse}} + \mathbf{b}_a \rangle))}, \\ \mathbf{H}_{Rumor} &= \sum_{j=1}^{N} a_j \mathbf{h}^j_{Fuse} \end{aligned} \tag{6.10}$$

where $\mathbf{W}_a \in \mathbb{R}^{2F' \times d}$, $\mathbf{b}_a \in \mathbb{R}^d$ and $\mathbf{w} \in \mathbb{R}^d$.

6.3.5 Rumor detection and optimization

Subsequently, $\mathbf{H}_{Rumor}$ is used to generate the corresponding binary prediction vector $\hat{\mathbf{y}} = [\hat{y}_0, \hat{y}_1]$, where $\hat{y}_0, \hat{y}_1$ indicate that the prediction probabilities of the label being 0 and 1, respectively, via a fully connected layer and the Softmax function:

$$\hat{\mathbf{y}} = \text{Softmax}\left(\text{FC}\left(\mathbf{H}_{Rumor}\right)\right). \tag{6.11}$$

In our implementation, we train all the model parameters by minimizing the *cross-entropy* between $\hat{\mathbf{y}}$ and $\mathbf{y}$:

$$\mathcal{L} = -\frac{1}{|B|} \sum_{i=1}^{|B|} \sum_{c=0}^{1} y_{i,c} \log \hat{y}_{i,c} + \lambda \left\| \Theta \right\|_2^2, \tag{6.12}$$

where $|B|$ is the batch size, $y_{i,c}$ and $\hat{y}_{i,c}$ are the ground truth and predicted results for the i-th sample. That is, if the sample belongs to c-th class, $\hat{y}_{i,c}$ is 1; otherwise it is 0. $\|\Theta\|_2^2$ is the L_2 regularizer over all the model parameters Θ, and λ is the trade-off coefficient. The optimization can be solved by stochastic gradient

descent–based optimization approaches, such as Adam [139] and RAdam [140]. The above computation process of our MMRD model is outlined in Algorithm 8.

Algorithm 8: Training process of MMRD.

Input: A set of tweets $M = \{m_i\}_{i=1}^{|M|}$, each tweet $m_i = \{G_i(t_o), \mathcal{P}_i(t_o)\}$, the max-order number K.
Output: MMRD-optimized parameters Θ.

initialize Θ
while Θ has not converged **do**
  for each tweets batch B **do**
    for each tweet m_i in tweets batch B **do**
      /* (1st) Train MacroE and MicroE separately. */
      1st macroscopic diffusion encoding: $\mathbf{H}^1_{Macro} \leftarrow \mathbf{X}, \widetilde{\mathbf{L}}, K$ via Eq.(6.3);
      1st microscopic diffusion encoding: $\mathbf{H}^1_{Micro} \leftarrow \hat{\mathbf{X}}$ via Eq.(6.6);
      /* (2nd) Cross-learning MacroE and MicroE */
      2nd macroscopic diffusion encoding: $\mathbf{H}^2_{Macro} \leftarrow \mathbf{H}^1_{Micro}, \widetilde{\mathbf{L}}, K$ via Eq.(6.3);
      2nd microscopic diffusion encoding: $\mathbf{H}^2_{Micro} \leftarrow \mathbf{H}^1_{Macro}$ via Eq.(6.6);
      /* Concatenate operation */
      $\mathbf{H}_{Macro} = \text{concat}(\mathbf{H}^1_{Macro}, \mathbf{H}^2_{Macro})$
      $\mathbf{H}_{Micro} = \text{concat}(\mathbf{H}^1_{Micro}, \mathbf{H}^2_{Micro})$
      macroscopic and microscopic representation fusion:
      $\mathbf{H}_{Fuse} \leftarrow \mathbf{H}_{Macro}, \mathbf{H}_{Micro}$ via Eq.(6.9);
      Attention sum-polling: $\mathbf{H}_{Rumor} \leftarrow \mathbf{H}_{Fuse}$ via Eq.(6.10);
      Estimate the probability $\hat{\mathbf{y}}$ via Eq.(6.11);
    end for
    $\mathcal{L} \leftarrow$ Eq.(6.12);
    $\Theta \leftarrow RAdam(\mathcal{L})$
  end for
end while

6.3.6 Rumor detection with knowledge distilling

In order to further improve the model performance on rumor detection task, we inspired by knowledge distillation technique [141]. The knowledge distillation technique, which involves capturing the "dark knowledge" from a teacher model to guide the learning of a student network, has emerged as an essential technique for model improvement. We first train a teacher model via Algorithm 8, and then transfer the knowledge from the teacher model to a student model. Here in our work, the student model has the same model architecture as the teacher model (self-distillation [142, 143]). Before introducing the concrete training procedure of MMRD with knowledge distillation, we first give the definition of the softmax with temperature:

$$q_i = \text{softmax}(\mathbf{H}, \tau) = \frac{exp(\mathbf{H}/\tau)}{\sum_j exp(\mathbf{H}/\tau)} \tag{6.13}$$

where τ is a temperature that is normally set to 1. We use a higher value for temperature τ to produce a softer probability distribution over the class, which brings the advantage that the information carried by the negative label will be relatively amplified, and the model training will pay more attention to the negative label.

The concrete training procedure of the knowledge distillation is listed in Algorithm 9, and Figure 6.1–(c) gives a visualization of Algorithm 9. The objective function of the knowledge distillation is a weighted average of two different objective functions. The first loss function is the cross-entropy with the soft targets and it is computed using the same high temperature $\tau = t$ in the softmax of the student model as was used for generating the soft targets from the teacher model.

$$\mathcal{L}_{soft} = -\sum_{i=1}^{|B|} \bar{\mathbf{y}}_i^T \log \bar{\mathbf{y}}_i^S \tag{6.14}$$

where $\bar{\mathbf{y}}_i^T = \text{softmax}(\text{FC}(\mathbf{H}_{Rumor}^T), \tau = t)$ is soft output from teacher model, and $\bar{\mathbf{y}}_i^S = \text{softmax}(\text{FC}(\mathbf{H}_{Rumor}^S), \tau = t)$ is soft output from student model.

The second loss function is the cross-entropy with the ground truth. This is computed using exactly the same logits in softmax of the student model but at a temperature of 1.

$$\mathcal{L}_{hard} = -\sum_{i=1}^{|B|} \mathbf{y}_i \log \hat{y}_i^S \tag{6.15}$$

where $\mathbf{y}_i$ is the ground truth and $\hat{y}_i^S = \text{softmax}(\text{FC}(\mathbf{H}_{Rumor}^S), \tau = 1)$ is the hard output of student model. Finally, the objective function of knowledge distillation is:

$$\mathcal{L}_{KD} = (1 - \beta)\mathcal{L}_{soft} + \beta\mathcal{L}_{hard} \tag{6.16}$$

where β is the balance weight, which always been a considerably lower value since the amplitude of the gradients produced by the scale of the soft output as $1/\tau^2$. This ensures that the relative contributions of the hard and soft targets remain roughly unchanged [141].

6.4 Evaluating MMRD

In this section present the findings from our experimental evaluations. We compare the performance of our MMRD with the state-of-art baselines on rumor detection, and we also investigate the effects of different components by comparing several variants of MMRD. Specifically, we aim at providing empirical evaluations to answer the following research questions:

- **Q1** How does MMRD perform compared with the state-of-the-art baselines on rumor detection?

Algorithm 9: Training procedure of MMRD with knowledge distillation.

Input: A set of tweets $M = \{m_i\}_{i=1}^{|M|}$, each tweet $m_i = \{G_i(t_o), \mathcal{P}_i(t_o)\}$, the max-order number K, temperature τ.

Output: **Student**-optimized parameters Θ.

Pre-train a **Teacher** model via Alogrithm 8.
initialize Θ in **Student** model.
while Θ has not converged **do**
  for each tweets batch B **do**
    for each tweet m_i in tweets batch B **do**
      $\mathbf{H}_{Rumor}^{T} \leftarrow$**Teacher**
      /* Train **Student** model via step 1 to step 9 in Alogrithm 8.*/
      $\mathbf{H}_{Rumor}^{S} \leftarrow$ **Student**
      /* Soft outputs from **Teacher***/
      $\bar{\mathbf{y}}_i^T = \text{softmax}(\text{FC}(\mathbf{H}_{Rumor}^{T}), \tau = t)$
      /* Soft outputs from **Student***/
      $\bar{\mathbf{y}}_i^S = \text{softmax}(\text{FC}(\mathbf{H}_{Rumor}^{S}), \tau = t)$
      /* Hard outputs from **Student** */
      $\hat{y}_i^S = \text{softmax}(\text{FC}(\mathbf{H}_{Rumor}^{S}), \tau = 1)$
      calculate loss $\mathcal{L}_{KD}$ via Eq. 6.16
    end for
    $\Theta \leftarrow \text{RAdam}(\mathcal{L}_{KD})$
  end for
end while

- **Q2** How does each component of MMRD contribute to the performance?
- **Q3** Can MMRD detect rumor at an early stage?

6.4.1 Datasets

We conducted our experiments on two real-world datasets: Twitter15, Twitter16. Both Twitter15 and Twitter16 datasets were collected by Ma et al. [33]. Each dataset contains a collection of source tweets with its corresponding propagation threads. The original datasets were constructed for multi-class classification, and we removed the tweets labeled as "unverified" or "true rumor" since they were beyond our research interest, and only keep "non-rumor" and "false-rumor" labels as ground truth in both datasets. We built the macroscopic diffusion graph and microscopic diffusion path for each source tweet from its propagation threads. The statistics of the datasets are presented in Table 6.2. The user profiles were crawled via Twitter API based on the provided user IDs, and for a fair comparison, we follow PPC_RNN+CNN [100] that extracts eight types of characteristics, including, (1) length of a user name; (2) created time of a user account; (3) length of description; (4) followers count; (5) friends count; (6) statuses count; (7) whether the user is verified; and (8) whether the geographical information is enabled.

Table 6.2: Statistics of the datasets.

Statistic	Twitter15	Twitter16
# source tweets	739	404
# non-rumor	370	199
# rumor	369	205
# users	306,402	168,659
Max. # retweets	2,990	999
Min. # retweets	97	100
Avg. # retweets	493	479
Avg. # time length	743 Hours	167 Hours

6.4.2 Baselines

We compare our model with a series of state-of-the-art baseline approaches for rumor detection:

- **DTC** [23]: A decision tree-based classification model that combines manually engineered characteristics of tweets to compute the information credibility.
- **SVM-TS** [85]: A linear SVM-based time series model, which can capture the variation of a broad spectrum of social context information over time by converting the continuous-time stream into fixed time intervals.

- **SVM-RBF** [86]: A SVM-based model that uses radius basis function (RBF) as the kernel, and leverages the handcrafted features of posts for rumor detection.
- **GRU** [33]: An RNN-based model, which learns temporal patterns and content features from user comments for rumor detection.
- **TD-RvNN** [34]: A top-down tree-structured RNN model that explores the importance of propagation structure for rumor detection.
- **PPC_RNN+CNN** [100]: A model combines RNN and CNN for early rumor detection, which learns the rumor representations through the characteristics of users.
- **Bi-GCN** [16]: A GCN-based model exploring rumor dissemination through bi-directional propagation structures and text contents for rumor detection.
- **GCAN** [102]: A state-of-the-art co-attention network for rumor detection, which learns the rumor representation based on the tweets content and the corresponding retweet users.

6.4.3 Parameter Settings and Evaluation Metrics

We implement DTC with Weka[1], SVM-TS and SVM-RBF with scikit–learn[2], and other deep learning-based baselines and our MMRD with Tensorflow[3]. The hyperparameters of baselines are the same as the settings described in the original papers.

Note that, in our work, MMRD only takes the user profiles and timestamps as inputs, and ignores the content features, such as source tweet text and comments, for a fair comparison, we implement some variants for the baselines by changing the initial inputs. Specifically, as for TD-RvNN and Bi-GCN, we use user profile features to replace the comment features, and the variants of these two baselines are denoted as TD-RvNN$_{(\text{User})}$ and Bi-GCN$_{(\text{User})}$, respectively. As for GCAN, we remove the source tweet features in the original inputs which termed as GCAN-Text.

The main hyperparameters in our MMRD are tuned as follows. The batch size is 32. The output dimension of MacroE $F^{Macro} = 64$, and the hidden sizes of both the forward GRU and backward GRU units are $F^{Micro} = 32$. The max-order number K is 3. The number of time intervals l is 100 and the embedding size for each timestamp vector d_{time} is 50. The learning rate for both the teacher training phase and knowledge distillation is 0.001 and the balance weight β in distillation is 0.3. The temperature τ in knowledge distillation is 2.5. The training process is iterated upon for 500 epochs but would be stopped earlier if the validation loss does not

[1]https://www.cs.waikato.ac.nz/ml/weka/
[2]https://scikit-learn.org/
[3]https://www.tensorflow.org/

decrease after 10 epochs. And we randomly choose 70% data for training and the rest of 10% and 20% for validation and testing. In this work, we measured the detection deadline by the number of retweets, i.e., the first k-th retweets. In the overall performance, the baselines and our MMRD consider the first 40-th retweets. We choose accuracy (ACC), precision (Pre), recall (Rec) and F-score (F1) as the evaluation protocols to measure the models' performance in this work.

Table 6.3: Overall performance comparison of rumor detection on Twitter15. The best method is shown in **bold**, and the second best one is underlined.

	Twitter15			
Method	Acc	Pre	Rec	F1
DTC	0.495	0.494	0.481	0.495
SVM-TS	0.519	0.519	0.518	0.519
SVM-RBF	0.535	0.552	0.521	0.536
GRU	0.580	0.544	0.545	0.544
TD-RvNN	0.628	0.594	0.616	0.604
PPC_RNN+CNN	0.691	0.674	0.686	0.679
Bi-GCN	0.748	0.731	0.759	0.745
GCAN	0.835	0.825	0.829	0.825
TD-RvNN$_{(User)}$	0.678	0.671	0.674	0.672
Bi-GCN$_{(User)}$	0.820	0.846	0.824	0.834
GCAN-Text	0.683	0.705	0.652	0.678
MMRD	**0.922**	**0.922**	**0.923**	**0.922**
Improvement	**10.41%**	**11.76%**	**11.34%**	**11.76%**

Table 6.4: Overall performance comparison of rumor detection on Twitter16. The best method is shown in **bold**, and the second best one is underlined.

	Twitter16			
Method	Acc	Pre	Rec	F1
DTC	0.561	0.575	0.537	0.562
SVM-TS	0.693	0.692	0.691	0.692
SVM-RBF	0.711	0.724	0.709	0.716
GRU	0.554	0.514	0.516	0.515
TD-RvNN	0.633	0.619	0.610	0.614
PPC_RNN+CNN	0.655	0.632	0.651	0.641
Bi-GCN	0.711	0.709	0.710	0.716
GCAN	0.823	0.803	0.841	0.822
TD-RvNN$_{(User)}$	0.661	0.632	0.641	0.636
Bi-GCN$_{(User)}$	0.814	0.815	0.816	0.816
GCAN-Text	0.664	0.716	0.579	0.648
MMRD	**0.876**	**0.877**	**0.874**	**0.875**
Improvement	**6.44%**	**9.22%**	**3.92%**	**6.45%**

6.4.4 Overall performance (Q1)

The overall performance is shown in Table 6.3 and 6.4, from which we can find that our MMRD model consistently outperforms all baselines on both Twitter15 and Twitter16 datasets. In addition to the overall superiority of our model, we have the following observations.

First, compared to the deep learning-based methods, feature-based methods such as DTC, SVM-TS, and SVM-RBF are not competitive because their performance heavily depends on the hand-crafted features. However, designing effective features is time-consuming and requires extensive field-specific knowledge. Furthermore, the performance gain of SVM-TS over DTC lies in its capability of considering time information. On the other hand, SVM-RBF performs slightly better than SVM-TS, suggesting that the kernel-based SVM is better than linear SVM but is still limited to the quality of hand-crafted features.

Second, among all the deep learning-based baselines, GRU, as the early deep learning-based work for rumor detection, performs the worst, primarily because it only relies on temporal-linguistics of the repost sequence but ignores other informative signals such as diffusion structures and user profiles. In addition, both TD-RvNN and Bi-GCN explore the dissemination of rumors on the basis of GRU and learn textual information from replies (i.e., the retweets with comments). However, their performance is not competitive when there are few comments or replies. Bi-GCN generally performs well than TD-RvNN, demonstrating that GCN is a powerful graph learning model compared with tree structure RNN. PPC_RNN+CNN performs relatively well than GRU and TD-RvNN, implying that user-profile information is more informative than text information in rumor detection, the reason is that compared with the replies, in reality, there exist more retweets without any comments, however, the user information of such retweets is acquirable. The same observations can find when compare Bi-GCN with its variant Bi-GCN$_{(\text{User})}$ and TD-RvNN with TD-RvNN$_{(\text{User})}$.

On the other hand, GCAN takes both text information and user-profile information as input and indeed outperforms other baselines. By comparing GCAN with its variant GCAN-Text, we can find that the performance of GCAN still heavily relies on text information. This is because it models the structural information from the user similarity matrix rather than the retweet network, which may be insufficient in capturing user interactions, and due to the two datasets were existed for a long time, when we try to crawl the user profiles for all users in the datasets, we find that some user accounts do not exist anymore, and it causes difficulties in constructing user similarity graph. Besides that, compare GCAN-Text with Bi-GCN$_{(\text{User})}$, the results of Bi-GCN$_{(\text{User})}$ far exceed GCAN-Text, this observation illustrates the diffusion graph is more powerful than user similarity graph in detecting rumor when ignoring the textual features. To further illustrate that our MMRD indeed significantly outperforms the baselines, we conduct a McNemar's test [144] between our MMRD

and the best baseline (GCAN) based on the prediction results on the testing set, and the p-values are 0.001 and 0.013 on Twitter15 and Twitter16, respectively. As $p < 0.05$ on both Twitter15 and Twitter16, we can conclude that MMRD significantly outperforms GCAN.

Our MMRD, in contrast, learns rumor representation from macroscopic and microscopic diffusion without any textual information, suggesting the possibility of detecting rumors by completely exploiting rumors' diffusion patterns. However, the performance of MMRD can be further improved by taking into account other information such as textual information.

6.4.5 Ablation study (Q2)

In order to answer the RQ2, we conduct several ablation studies from the following perspectives: (1) we first propose five variants of MMRD and compare their performance on both Twitter15 and Twitter16; then, (2) we compare the performance of MMRD in without knowledge distillation and cross distillation settings; finally, (3) we pick up two special parameters to test the model's accuracy change brings by them when changing their value.

6.4.5.1 Variants comparison

We conducted an ablation study to explore each component's effect in MMRD by removing a particular component from the original MMRD. Towards that, we derive the following variants of MMRD:

- **-AGG_Atten**: In "-AGG_Atten", we use sum-pooling function to replace the attention aggregation function f_{AGG} in MacroE.
- **-Gate**: In "-Gate", we remove the fusion gate from the MMRD, i.e., concatenate $\mathbf{H}_{Macro}$ and $\mathbf{H}_{Micro}$ directly ($\mathbf{H}_{Fuse} = \text{concat}(\mathbf{H}_{Macro}, \mathbf{H}_{Micro})$).
- **-Atten**: In "-Atten", we replace the attention sum-pooling with normal sum-pooling, that is $\mathbf{H}_{Rumor} = \sum_{j=1}^{N} \mathbf{h}_{Fuse}^{j}$.
- **-GCN**: In "-GCN", we replace the convolution kernel in MacroE with a vanilla GCN layer.
- **-Time**: In "-Time", we ignore the timestamp information, i.e., the input feature of the first MicroE are user profile features.

The results of the ablation study are summarized in Table 6.5, where we can observe that:

(1) The accuracy of "-Atten" remarkably decreases compared with other variants, which indicates that user-level attention sum-pooling can learn the importance of each user in rumor diffusion since it allocates different significance to each row (that correlated to a specific user) of $\mathbf{H}_{Fuse}$. The visualization of the attention weights is

Table 6.5: Performance comparison between MMRD and its variants.

	Twitter15				Twitter16			
Method	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
-AGG_Atten	0.854	0.855	0.855	0.855	0.826	0.827	0.824	0.825
-Gate	0.875	0.875	0.874	0.875	0.845	0.845	0.844	0.844
-Atten	0.831	0.835	0.829	0.832	0.795	0.799	0.769	0.784
-GCN	0.851	0.851	0.851	0.851	0.807	0.807	0.806	0.807
-Time	0.878	0.878	0.879	0.878	0.845	0.863	0.839	0.851
MMRD	**0.922**	**0.922**	**0.923**	**0.922**	**0.876**	**0.877**	**0.874**	**0.875**

depicted in Figure 6.4, which further proves the effectiveness. From Figure 6.4, we also find that the later users are more critical in rumor spreading, which confirms the hypothesis that rumors can spread deeper than non-rumors [43]. (2) Using the fusion gate to control the dependency on macroscopic diffusion and microscopic diffusion will improve the model performance as achieved by the "-Gate". (3) The results of "-GCN" demonstrate that multi-hop and directional information are essential for macroscopic diffusion modeling, and the performance of "-AGG_Atten" worse than MMRD, which further demonstrates that as for each node, their order-dependency is different. (4) As for "-Time", it shows the importance of the timestamp information in capturing microscopic diffusion.

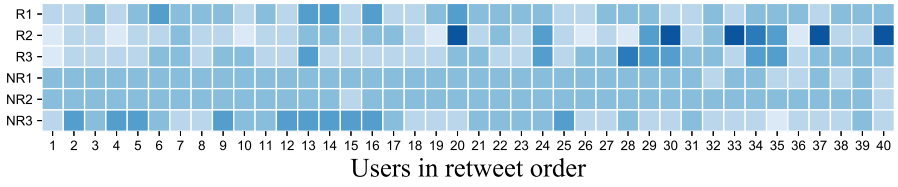
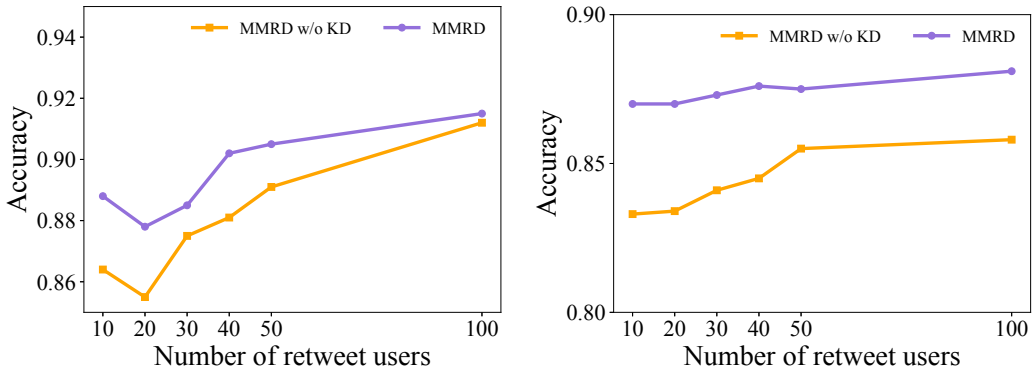


Figure 6.4: Visualization of attention weights in attention sum-pooling, which randomly choose 3 rumors and 3 non-rumors from Twitter15. Dark colors refer to a higher value.

6.4.5.2 Performance on knowledge distillation

In our work, one of the most important components is the use of knowledge distillation to enhance model performance. In order to test the performance of knowledge distillation (for briefly, simplify as KD), in this section, we conduct experiments on removing KD and using cross KD, respectively.

Figure 6.5 shows the results when removing the KD, we find that, after removing KD, although the model still can achieve better performance compared with the baselines in Table 6.3 and 6.4, it can be further improved by using KD to transfer knowledge from a teacher model to a student model. Besides that, the effect of KD is more remarkable on the Twitter16 dataset, it yields a large performance interval between "MMRD" and "MMRD w/o KD".



(a) Twitter15 **(b)** Twitter16

Figure 6.5: The effectiveness of knowledge distillation. The number of observed retweet users per source tweet varies from 10 to 100, and we plot the corresponding detection accuracy of MMRD with and without knowledge distillation. "MMRD w/o KD" denotes MMRD without knowledge distillation.

Figure 6.6 shows the comparison between different strategies of cross KD. Specifically, we train the teacher model and student model based on different datasets and then test the student's performance on both Twitter15 and Twitter16 datasets. For example, "T15/S16" means we first train a Teacher model "T15" on Twitter15 dataset and then distill the model on Twitter16 to get a student model "S16", and finally use the "S16" model to perform rumor detection on Twitter15 and Twitter16, respectively. From Figure 6.6, we observe that the performance of MMRD is much better when both the Teacher model and Student model train on the same dataset, this is because of some dataset-specific reasons, such as diffusion scale, the number of non-exist users, user-specific feature (e.g., create time), etc.

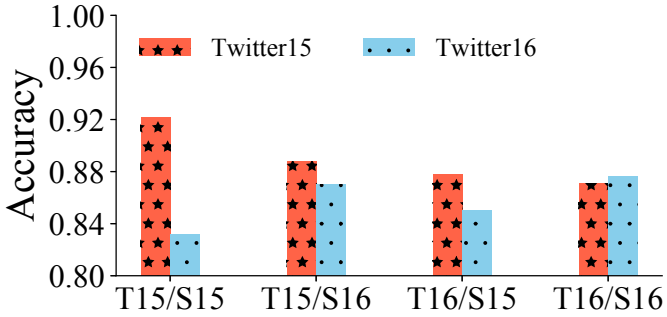
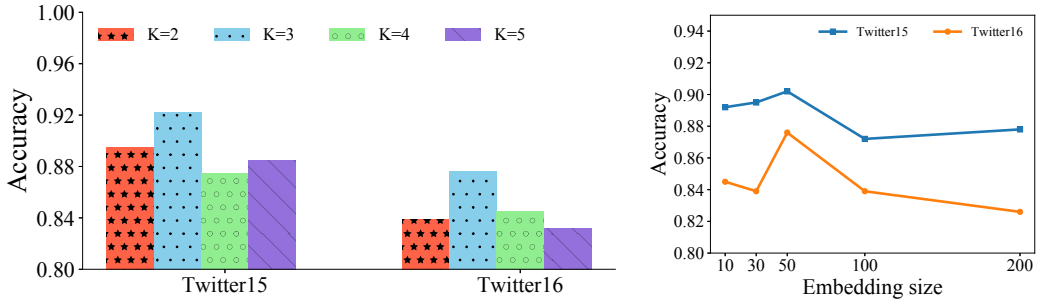


Figure 6.6: Cross knowledge distillation. The number of observed retweet users per source tweet sets to 40. Each bar represents the detection accuracy and the labels of the x-axis denote the datasets used when training the teacher model and student model. E.g., "T15" and "T16" denote that we train the teacher model on Twitter15, and Twitter16, respectively; "S15" and "S16" means that we learn the student model via distilling knowledge on Twitter15 and Twitter16, respectively.

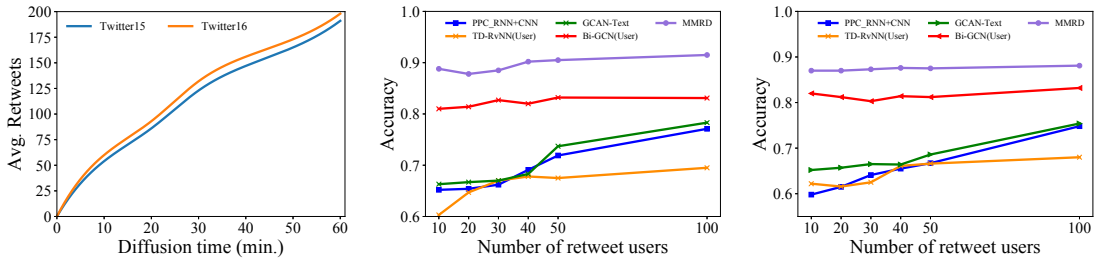
6.4.5.3 Parameter analysis

From all parameters in MMRD, we choose two special parameters to conduct parameter analysis experiments – the value of max-order K and the time embedding size d_{time}. The results shows in Figure 6.7. From both Figure 6.7a and Figure 6.7b, we find that by blindly increase the number of K and d_{time}, the model accuracy not improve, instead, decreased. And when set $K = 3$ and $d_{time} = 50$, the model achieves the best performance. Moreover, the embedding size d_{time} with small values achieve better performance than large values. And Figure 6.7a also demonstrates that take the higher-order of node interaction into consideration is useful when modeling macroscopic diffusion of tweets.



(a) Vary K **(b)** Vary embedding size

Figure 6.7: Results of parameter analysis on Twitter15 and Twitter16 when the number of observed retweet users per source tweet sets to 40. (a) Performance on different max-order value K, ranging from 2 to 5. (b) Performance on different embedding size d_{time} of timestamp vector $\mathbf{T}$.



(a) Average propagation speed **(b)** Twitter15 **(c)** Twitter16

Figure 6.8: Evaluations on early rumor detection. (a) The average propagation speed of tweets calculated based on Twitter15 and Twitter16 datasets. (b) and (c) plot the detection accuracy when the number of observed retweet users per source tweet are in the range of $[10, 20, 30, 40, 50, 100]$ on Twitter15 and Twitter16, respectively.

6.4.6 Early detection (Q3)

Detecting rumors as early as possible is crucial for public opinion control. Figure 6.8a shows the average propagation speed of messages on twitter calculated based on

Twitter15 and Twitter16. We find that within 60 minutes, both Twitter15 and Twitter16 have a diffusion speed near 190 retweets. And when the time is extremely small, i.e., within 30 minutes, the average retweets of both two datasets are close to 100. So, to investigate the performance of models on identifying rumors at an early stage, here, we consider the number of observed retweet users per source tweet from the list $[10, 20, 30, 40, 50, 100]$. Besides, for a fair comparision, we choose user profile-based mtheods, i.e., "TD-RvNN$_{(\text{User})}$", "PPC_RNN+CNN", "Bi-GCN$_{(\text{User})}$" and "GCAN-Text" as contrast methods. Figure 6.8b and Figure 6.8c show the performance comparison of early-stage detection between our MMRD and selected baselines. We can see that MMRD performs better, especially when there are only a few observations, i.e., MMRD achieves almost 89% and 87% accuracy on Twitter15 and Twitter16, respectively, even with only 10 retweet user observations.

6.5 Summary

This paper proposed a novel rumor detection model named MMRD, which can effectively and efficiently summarize a unique representation for each rumor propagation through capturing the dissemination patterns from both macroscopic and microscopic diffusion levels. Simultaneously, MMRD leverages the knowledge distillation technique to transfer knowledge from a pretraining teacher model to a student model which further improves the model detection performance. The experimental results based on two real-world Twitter data sets demonstrate that our method achieves state-of-the-art performance on rumor detection and also effective in detecting rumors at an early stage. Besides that, MMRD detects rumor via learning its spreading process, which can help us to develop rumor spreading models.

Chapter 7

Participant-level Rumor detection based on Information Diffusion Analysis

Users are the main contributor to the rumor spreading in Online social networks (OSNs) [26], which go through the whole life cycle of news diffusion. In this chapter, we propose two independent and complementary participant-level models for rumor detection, i.e., PLRD (Participant-Level Rumor Detection model) and UMLARD (User-aspect Multi-view Learning with Attention for Rumor Detection mlodel). And both PLRD and UMLARD are implemented according the following two design considerations.

First, we considered the life cycle of real and fake news on social media, which plays a crucial role in information diffusion. When the news is produced by the content creator, it starts its journey on the social media platform. Once people are exposed to the news, they become the content consumers. According to the confirmation bias theory, people tend to favor, interpret and share information in a way that confirms or strengthens their prior beliefs or ideologies [145]. As a result, if a news item confirms the consumers' prior ideology, they may share it within their social networks in the role of content distributor. Since fake news is intentionally written to mislead readers into believing and propagating false information (e.g., 5G networks trigger COVID-19), it is plausible that fake news is more easily distributed among its believers than real news, which is neutral in its beliefs and ideology. This idea is supported by prior studies, which have noted that false information tends to spread significantly faster, further, deeper, and more broadly than real information [43]. Therefore, considering all participants, including content creators and content distributors, in the news diffusion chain may improve the overall rumor detection performance of our approach.

Another design consideration is the lack of effective methods to represent all participants in the news diffusion chain. Prior predictive studies have simply used aggre-

gated statistics, such as the total number of content distributors (retweets) and the average time of information distribution, to quantify the diffusion process. This inevitably results in information loss and suboptimal performance. Other examples of such aggregated statistics include network-level attributes (e.g., density) to represent diffusion networks, the final hidden representation from recurrent neural networks to model the temporal spreading sequence, and overall descriptive statistics of user characteristics (e.g., mean user tenure) to describe users in the diffusion [23]. While such data may be helpful in modeling, they are not quite specific enough to provide a clear picture of by whom, when, why, and how news is diffused. Therefore, the key question motivating our study is how to design an effective predictive method that represents all-participant patterns throughout the whole diffusion process.

7.1 PLRD: A Participant-Level Rumor Detection Framework via Fine-grained User Representation Learning

7.1.1 Section Overview

In this section, we propose a novel framework based on deep representation learning for rumor detection, named PLRD (Participant-Level Rumor Detection). In view of theories on propagation and social influence, PLRD incorporates multi-scale features of all users[1] enrolled in the diffusion process to predict a given post's credibility (e.g., classify it as rumor or non-rumor). Specifically, PLRD first employs sparse matrix factorization to embed the global graph (i.e., user-interaction graph constructed on all propagation threads), which can efficiently learn the social homophily for users. Then, it uses a multi-hop graph convolutional layer, and a bi-directional GRU to learn fine-grained user representations (i.e., the user influence, user susceptibility, and user temporal information). To understand the different importance of users' multi-scale representations, a feature-level attention layer was designed to explain which types of features are essential in rumor propagation. Moreover, to capture the uncertainty in learned features, PLRD introduces a variational autoencoder. Finally, PLRD employs a user-level attention layer to allocate different importance to users and aggregate them to form a unique rumor representation. The rumor prediction is generated based on the learned unique representation.

Our design science work on rumor detection makes two main contributions to the literature in this field. First, our design is rooted in social influence and propagation theory, from which we derive various constructs in our model. PLRD detects rumors at a very fine-grained participant level. It is very different from prior works that also combine information from various sources (e.g., reply networks, diffusion sequence,

[1] In our work, the user influence, user susceptibility and user temporal information are collectively referred to as multi-scale information of users

and attributes of spreading users), but still heavily depend on the text features. Our approach comprehensively exploits user-profiles and propagation threads and shows a strong ability to detect rumors without using any text information. Second, we make a methodological contribution by proposing an approach to learn fine-grained user representation via deep representation learning that effectively captures all participant information in a diffusion chain. This information includes user influence, user susceptibility, and user temporal information. Experimental results using real-world datasets confirm the effectiveness of our approach over prior rumor detection methods. Our approach has direct implications for social media platforms that are vulnerable to rumor spreading since it can be deployed to identify original users who initiate rumors and those who spread rumors. Overall, the proposed rumor-detection model can help improve the user experience and benefit society by helping individuals obtain healthy and genuine information.

This section is based on the following publication [45]:

- **Chen, X.**, Zhou, F., Zhang, F., Bonsangue, M.: Catch me if you can: A participant-level rumor detection framework via fine-grained user representation learning. Information Processing & Management, 58(2021) 102678

7.1.2 Birds of a feather flock together: the perspective of all participants

Rumor detection has long been a subject of interdisciplinary research. Various theories have been proposed and validated. In this section, we discuss several major theories that guide us to derive relevant constructs in our model for better rumor detection.

7.1.2.1 Theory

Users play major roles in the dissemination of rumors or fake news. A set of user-based and propagation-based theories have been developed to study how a rumor spreads, how users engage with a rumor, and the role users play in rumor creation, propagation, or intervention. For example, in the echo chamber effect, individuals tend to believe information is correct after repeated exposures [146]. Confirmation bias theory tells us that individuals tend to trust information that confirms their preexisting beliefs or hypotheses, which they perceive to surpass that of others [147]. People choose to interact with those who share similar opinions and avoid those with whom they profoundly disagree. Both indicate that people may react to and process information differently based on information type (e.g., rumor vs. non-rumor). On the other hand, homophily theory says that individuals in homophilic relationships share common characteristics (beliefs, demographics, etc.) that make communication easier [148]. Meanwhile, social identity theory shows that individuals do something primarily because others are doing it and to conform in order to be

liked and accepted by others. Such social influence and homophilic atmospheres also exist in and are commonly seen in online social networks [149].

In sum, all the above considerations suggest that user attributes likely have an impact on rumor detection, such as user influence, susceptibility, and temporal, etc. This assumption is also empirically confirmed by our data exploration (see below) and computational experiments (see the Evaluation section below). We therefore hypothesize that:

Hypothesis 1. *Combining various information at a fine-grained all-participant level in a diffusion chain will improve rumor discovery performance.*

Hypothesis 2. *Deep representation learning-based methods will improve rumor discovery performance compared to using shallow machine learning methods.*

7.1.2.2 Data

In our work, we conduct experiments on four standard real-word testbeds: Twitter15, Twitter16, Science and RumourEval19, all of which were collected from Twitter[1], one of the most popular social media platforms in the U.S. The descriptive statistics of all datasets are shown in Table 7.1.

Twitter15/16[2] were released by [84]. More details can be found in Section 6.4.1. In Twitter15 and Twitter16, we keep the tweets labeled as "non-rumor" or "false rumor" (relabeled as "rumor" in our work), since the others were beyond our research interest.

Science[3] is collected and studied by [43]. It includes complete retweet cascades linked to rumors that were verified and published by fact-checking websites. In the original data, each tweet cascade is related to a specific label, i.e., "TRUE", "FALSE" or "MIXED". In our work, we keep only the tweets labeled as "TRUE" or "FALSE" (relabeled as "non-rumor" and "rumor" in our work, respectively). To the best of our knowledge, the Science dataset is the most credible dataset among all the existing Twitter-based rumor detection datasets as it overcomes issues of partiality or bias because of the sampling restriction characteristic. In this chapter, we use the Science dataset to provide model-free evidence to support the **Hypothesis 1**.

RumourEval19[4] [150] is an extensive dataset from RumourEval17 [151], which is augmented with new Twitter test data and Reddit material. Here, we keep Twitter data but ignore the Reddit material, to finally obtain 381 Twitter conversation threads. Each thread consists of a claim and a tree of comments, and is related to a specific label, i.e., "true", "false" or "unverified". We filter out unverified tweets and finally get 271 Twitter conversation threads (relabeled as "non-rumor" and "rumor",

[1]https://twitter.com/

[2]https://www.dropbox.com/s/46r50ctrfa0ur1o/rumdect.zip

[3]Researchers interested in gaining access to Science dataset should contact [43] directly.

[4]https://competitions.codalab.org/competitions/19938

respectively). Because the RumourEval19 contains rich textual information rather than diffusion patterns, in the experiments part, we use RumourEval19 to test our model performance when including textual features.

Table 7.1: Statistics of the datasets.

Statistic	Science	Twitter15	Twitter16	RumourEval19
# source tweets	16,901	739	404	271
# users	3,603,265	306,402	168,659	4,774
# edges of global graph	3,586,360	336,486	182,493	4,503
# non-rumors	14,442	370	199	167
# rumors	2,459	369	205	104
Max. # retweets	46,895	2,990	999	155
Min. # retweets	5	97	100	3
Avg. # retweets	213	493	479	18
Avg. # time length	749 Hours	743 Hours	167 Hours	37 Hour

For each tweet, we construct the diffusion graph and the global graph from the propagation threads (see Section 3.1 for recalling the formal definitions). In Twitter15/16, no user information is provided due to constraints set out in Twitter's terms of service. We crawl all the related user profiles via *Twitter API*[1] based on the provided user IDs. We follow the work of Liu et al. [100] and select 8 general user characteristics for experiments, which are summarized in Table 7.2. As for Science, all data, such as ids, were anonymized, so we directly use the user characteristics provided in this dataset, which are listed in Table 7.3. Here, the concrete definitions of each characteristic can be found in the supplementary of [43]. The RumourEval19 provided JSON files for each source tweet and its corresponding replies, and each file contains the complete information of the tweet and the users.

Table 7.2: Summary of user characteristics for Twiter15/16 and RumourEval19.

No.	Characteristic	Data Type
1	LENGTH OF USER NAME	Integer
2	USER COUNT CREATED TIME	Integer
3	LENGTH OF DESCRIPTION	Integer
4	FOLLOWERS COUNT	Integer
5	FRIENDS COUNT	Integer
6	STATUSES COUNT	Integer
7	IS VERIFIED	Binary
8	IS GEO ENABLED	Binary

7.1.2.3 Model-free evidence

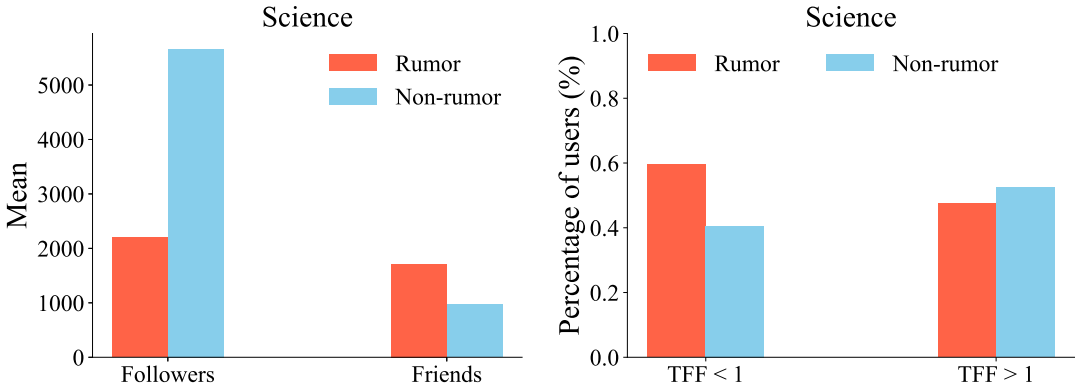
Following from our earlier **Hypothesis 1** that utilizing the information of all users who participated in a diffusion chain might improve rumor detection, we first check

[1]https://dev.twitter.com/rest/public

Table 7.3: Summary of user characteristics for Science.

No.	Characteristic	Data Type
1	USER COUNT CREATED TIME	Integer
2	FOLLOWERS COUNT	Integer
3	FRIENDS COUNT	Integer
4	ENGAGEMENT	Float
5	IS VERIFIED	Binary

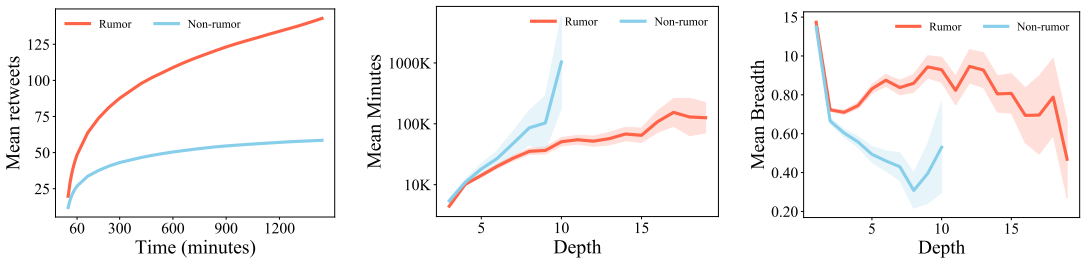
for any patterns or differences among involved participants in terms of their overall attributes across the rumors and non-rumors via analyzing the Science dataset.



(a) Follower/friends distribution **(b)** TFF distribution

Figure 7.1: Social influence/susceptibility analysis on Science.

- **Social influence/susceptibility: followers/friends.** Social influence may affect the speed and the depth of diffusion. On Twitter, a user's social influence can be measured by the size of their social circle in relation to two factors: the number of friends (i.e., users the specific user is following) and the number of followers (i.e., users following the specific user). We calculate the average followers/friends across all participants and find that this average in the rumors group is different from that in the non-rumors group (see Figure 7.1a). To compute the social influence of all participants, we define a new metric, *TFF* (the follower-friend-ratio [152]), which combines followers and friends: $TFF = \frac{\#\text{followers}}{\#\text{friends}}$. Users with $TFF < 1$ are less influential but with higher susceptibility, since they have fewer followers than friends, and extreme cases are fake users. In contrast, users with $TFF > 1$ are more influential, e.g., celebrity accounts. Figure 7.1b shows that a higher percentage of users with $TFF < 1$ are involved in rumors, while more influential users participate in non-rumors.
- **Structural and temporal impact of diffusion.** Many prior studies have demonstrated that falsehood diffuses significantly farther, faster, deeper, and



(a) Propagation speed **(b)** Depth-time **(c)** Depth-breadth

Figure 7.2: Structural and temporal analysis on Science.

more broadly than the truth in all categories of information [43]. This being the case, we expect to see that the spreading pattern, in terms of network structures and the temporal sequence of retweeting, should vary based on rumor type. To this goal, we analyze the propagation speed, depth-time distribution and depth-breadth distribution, respectively, which are shown in Figure. 7.2. In the diffusion graph, the depth of a node is the number of hops from the node to the source node, and the breadth at a specific depth of a graph is the total number of nodes at this depth level. From Figure. 7.2a, we can find that at the same time-scale, rumors can infect more users than non-rumors, which demonstrates that rumors spreading faster than non-rumors. In Figure. 7.2b, we measured the average time (in minutes) rumor or non-rumor tweets took to reach different depths, where we observe that (1) rumors can reach deeper users than non-rumors, and (2) rumors require less time to reach the same depth with non-rumors. In Figure. 7.2c, we plot the average breadth at every depth for rumors and non-rumors, which indicates that rumors spread broader than non-rumors, especially at a deeper level. These observations substantiate our belief that incorporating such structural and temporal information into the model may improve rumor detection performance.

7.1.3 Methodology

In this section, we first present a preliminary overview of rumor detection in our context and describe the overall framework of the proposed PLRD method, followed by details of each component in the model. As discussed above, two challenges need to be addressed when designing an effective rumor detection system: (1) how to incorporate fine-grained all-participant information in one model – not simply aggregating or concatenating – to capture rumor spreading patterns; and (2) how to effectively learn latent representations of users' propagation activities in a diffusion chain to capture fine-grained user representations. To answer these two questions, we first formally define our problem and describe its context.

Table 7.4: Main notations used throughout this chapter.

Symbol	Description
$\mathcal{G}, N$	Global graph, and the number of nodes of global graph.
G_i, G_i^T	Diffusion graph and inverse diffusion graph of m_i.
$\mathbf{E}^{\mathcal{G}}, \mathbf{e}_*^{\mathcal{G}}$	The user social homophily embedding matrix and social homophily vector for user u_*.
$\mathbf{T}_i$	Time-embedding of m_i.
$\mathbf{H}_i^{inf}$	The representations of the user influence.
$\mathbf{H}_i^{sus}$	The representations of the user susceptibility.
$\mathbf{H}_i^{temp}$	The representations of the user temporal information.
$\bar{\mathbf{U}}_i$	The representation after feature-level attention.
$\mathbf{U}_i^z$	The representation after VAE.
$\mathbf{U}_i^F$	The representation after concatenate operation.
$\mathbf{R}_i$	the final representation of tweet m_i.
$\hat{\mathbf{Y}}/\hat{\mathbf{y}}_*$	The predicted label.
$\mathbf{Y}/\mathbf{y}_*$	The ground truth.

7.1.3.1 Preliminaries and Problem Statement

In this section, we give the necessary background and formally define the rumor detection problem. We list the main mathematical notations used throughout the paper in Table 7.4.

In this study, we formalize our rumor detection problem as a supervised binary-class classification task. Suppose the input of the task is from a rumor detection dataset (e.g., Twitter) consisting of a set of posts (e.g., tweets) denoted as $M = \{m_i, i \in [1, |M|]\}$. Each m_i corresponds to its own diffusion process and all participating users, so that m_i can be represented by $\{C_i, \mathbf{U_i}\}$, where C_i and $\mathbf{U_i}$ are the cascade graph and the user characteristics matrix (see Definition 1 and Definition 5), respectively. The cascade graph C_i can be further broken down into a diffusion graph G_i and diffusion path $\mathcal{P}_i$ (see Definition 2 and Definition 3). In addition, we construct a global graph $\mathcal{G}$ based on all tweets M, to represent the social homophily among all users. See their formal definitions below.

Definition 13 ***Global Graph*** $\mathcal{G}$**.** *The global graph $\mathcal{G} = \{U, E\}$ is a collection of nodes and edges, which is constructed based on all posts in the dataset. $U = \bigcup_{i=1}^{|M|} U_i$ is a user set contains all users in dataset, and E is the edge set. An edge between u_i and u_j refers to these two users share the same tweet (or discuss the same topic).*

Definition 14 ***Rumor Detection.*** *Given a tweet $m_i = \{C_i(t_o), \boldsymbol{U}_i(t_o)\}$ within an observation window t_o (in our work, t_o is the total number of retweets), and the global graph $\mathcal{G}$, the goal of rumor detection is to learn a function $f(\hat{y}_i|m_i, \mathcal{G})$ to classify the source tweet m_i into one of the rumor categories, where the predicted result $\hat{y}_i$ represents either non-rumor or rumor.*

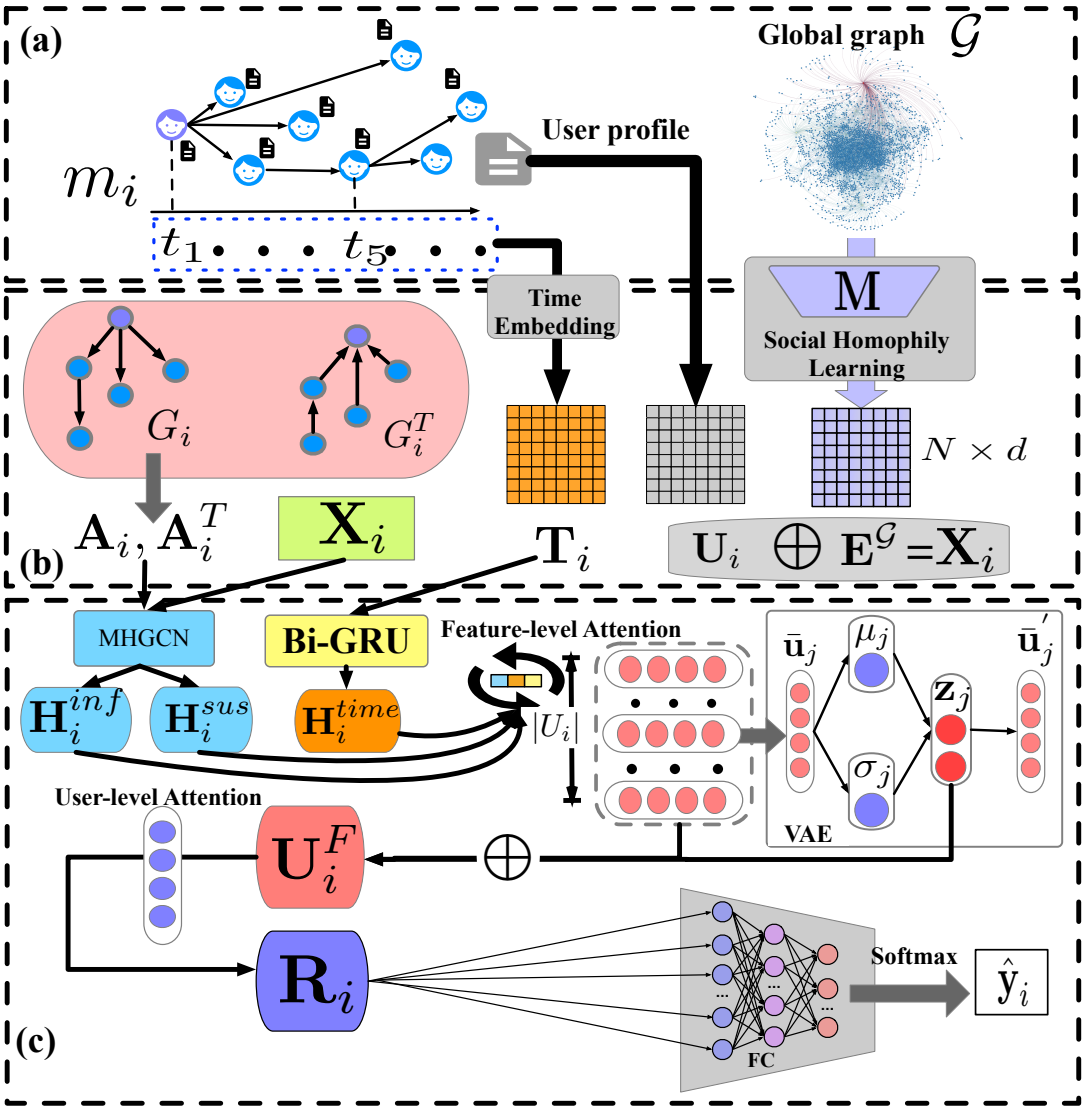


Figure 7.3: Overview of **PLRD**: (a) input of **PLRD**; (b) preprocessing layer; (c) fine-grained user representation learning and rumor detection layer.

7.1.3.2 Overall framework of PLRD

In this section, we describe our proposed PLRD rumor detection system. It consists of the following components (see Figure 7.3): (a) inputs, including (1) the propagation threads of tweet m_i, and (2) a global graph constructed on all tweets propagation threads; (b) the preprocessing layer, which consists of (1) constructing a diffusion graph and inverse diffusion graph based on propagation threads, (2) constructing a user characteristic matrix based on user profiles, (3) pre-training retweet time-stamps via positional encoding, and (4) pre-training global graph via randomized truncated singular value decomposition-based sparse matrix factorization; and (c) the fine-grained user representation learning layer and the rumor detection layer, in which we learn user influence and susceptibility via a multi-hop graph convolution layer, model user temporal feature via bi-directional GRU, then aggregate and enhance the learned multi-scale user representations through a feature-level attention

layer and a variational autoencoder. Finally, after a user-level attention layer, we feed the unique rumor representation into a rumor classifier. Specifically, we use several fully connected feedforward layers and a softmax output layer to generate a rumor prediction. Below, we explain each of the above components in detail.

7.1.3.3 Social homophily learning from global graph

As mentioned in section 7.1.3.1, we construct the global graph based on all the retweet threads in the dataset. Our goal is to capture social homophily for all users. The social homophily among users specifies that users with similar interests are more likely to closely connected [153]. We assume that the users who discuss the same post in social communities share homophilous relationships in our work. Then, we try to model social homophily in an unsupervised manner, which encourages users with shared social neighborhoods to have similar latent representation.

More specifically, we cast the problem of learning social homophily as the task of graph embedding. The global graph $\mathcal{G}$ always contains tens of thousands of nodes, and to model such a large graph effectively is a tough challenge in the field of graph representation learning [154, 155]. Inspired by the success of sparse matrix factorization (SMF) in large-sized graph representation learning, in our work, we use a randomized tSVD-based SMF to learn social homophily from the global graph. Here, tSVD is truncated singular value decomposition, which can prevent the problem of infeasible computation of factorization for a large-sized matrix [156, 157]. Specifically, given global graph $\mathcal{G}$, we can obtain the adjacency matrix $\mathbf{A}^{\mathcal{G}} \in \mathbb{R}^{N\times N}$ and diagonal degree matrix $\mathbf{D}^{\mathcal{G}} \in \mathbb{R}^{N\times N}$, N is the number of nodes in global graph. Each entry $\mathbf{A}^{\mathcal{G}}_{i,j}$ of $\mathbf{A}^{\mathcal{G}}$ equals to 1 when u_j and u_j share the same post or $i = j$, otherwise $\mathbf{A}^{\mathcal{G}}_{i,j} = 0$. And $\mathbf{D}^{\mathcal{G}}_{i,i} = \sum_j^N \mathbf{A}^{\mathcal{G}}_{i,j}$. To learn the embedding of $\mathcal{G}$ via randomized tSVD-based SMF, we first define a proximity matrix $\mathbf{M}^{\mathcal{G}}$ as:

$$\mathbf{M}^{\mathcal{G}}_{i,j} = \begin{cases} \ln p^{\mathcal{G}}_{i,j} - \ln(\lambda \mathcal{N}^{\mathcal{G}}_{E,j}), & (u_i, u_j) \in E \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

where $p^{\mathcal{G}}_{i,j} = \mathbf{A}^{\mathcal{G}}_{i,j}/\mathbf{D}^{\mathcal{G}}_{i,i}$ indicates the weight of (u_i, u_j) in E. $\mathcal{N}^{\mathcal{G}}_{E,j}$ are the negative samples connected with user u_j, which can be defined as $\mathcal{N}^{\mathcal{G}}_{E,j} \propto (\sum_{i:(i,j)\in E} p_{i,j})^{3/4}$ ($y \propto x$ means that y and x are in a directly proportional relation) [158]. The goal of the global graph embedding is transformed to factorize the matrix $\mathbf{M}^{\mathcal{G}}$. Specifically, the step for the approximate matrix factorization of $\mathbf{M}$[1] are as follows: (1) we first look for a matrix $\mathbf{Q} \in \mathbb{R}^{N\times d}$ with d orthonormal columns that let $\mathbf{M} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{M}$; (2) suppose we found such matrix $\mathbf{Q}$, we define $\hat{\mathbf{M}} = \mathbf{Q}^T\mathbf{M} \in \mathbb{R}^{d\times N}$, which is quite smaller compare with the original matrix $\mathbf{M}$. Then we have $\hat{\mathbf{M}} = \mathbf{S}\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$, where $\mathbf{\Sigma}$ is the diagonal matrix with top-d singular values, and $\mathbf{S}, \mathbf{V} \in \mathbb{R}^{N\times d}$ are orthonormal matrices with d selected singular values; (3) finally, the factorization of $\mathbf{M}$ is approximate to $\mathbf{M} \approx \mathbf{Q}\mathbf{Q}^T\mathbf{M} = (\mathbf{Q}\mathbf{S})\mathbf{\Sigma}\mathbf{V}^{\mathbf{T}}$ and the calculation of the output

[1] for brevity, we ignore the superscript $\mathcal{G}$.

embeddings for the global graph is $\mathbf{E}^{\mathcal{G}} = \mathbf{Q}\mathbf{S}\mathbf{\Sigma}^{\mathbf{1/2}}$ and $\mathbf{E}^{\mathcal{G}} = \{\mathbf{e}_j^{\mathcal{G}} | j \in [1, N]\} \in \mathbb{R}^{N \times d}$. That is, for each user u_j in the global graph, was allocated a relative latent embedding, i.e., $\mathbf{e}_j^{\mathcal{G}}$, and users with similar preferences and behavior (i.e., interested in the same posts) will have similar embeddings.

7.1.3.4 Users' influence and susceptibility learning

The role of each participant in the diffusion process of m_i has two types, i.e., sender and receiver [159]. In previous diffusion research [160, 161], the role of the sender reflects a user's influence, that is, the ability to spread information to other users. On the opposite, the receiver role reflects the susceptibility of a user, i.e., the ability of a user to be infected by possible senders. In our work, we learn the influence and susceptibility for each user from the diffusion graph G_i of m_i. However, in the original diffusion graph G_i, the information passed from a sender to a receiver, so that modeling G_i can acquire influence for each user, is not efficient for susceptibility learning. To overcome this problem, we introduce an inverse diffusion graph G_i^T, which changes the direction of information propagation, i.e., from receiver to sender.

Inspired by the recent success of deep learning technologies in graph representation learning, such as graph convolutional network (GCN) [41, 61], and graph attention network (GAT) [109], in order to model the higher-order relationships among participants, we propose a multi-hop graph convolution layer (MHGCN) to extract user influence and susceptibility from G_i and G_i^T, respectively. The convolution kernel of MHGCN is defined as:

$$\mathbf{H} = g_\theta * \mathbf{X} = \sigma(\mathop{\Big\|}_{k \in \mathcal{O}} (\hat{\mathbf{A}}^{(k)} \mathbf{X} \mathbf{W}^{(k)})) \tag{7.2}$$

where $\mathop{\Big\|}_{k \in \mathcal{O}}$ represents the order-level concatenate, and σ is a non-linear activation function such as ReLU. $\hat{\mathbf{A}}^{(k)}$ denotes the normalized adjacency matrix $\hat{\mathbf{A}} \in \mathbb{R}^{|U| \times |U|}$ multiplied by itself k times, $|U|$ is the number of nodes in graph, and $\mathcal{O}$ is a set of integer adjacency powers from 0 to K, K is the max-order. The calculation of normalized adjacency matrix is denoted as $\hat{\mathbf{A}} = \bar{\mathbf{D}}^{-1}\bar{\mathbf{A}}$, where $\bar{\mathbf{A}} = \mathbf{A} + \mathbf{I}$, and $\mathbf{I}$ is diagonal identity matrix. $\mathbf{X} \in \mathbb{R}^{N \times d_X}$ is the input graph signal, d_X is the dimension number. $\mathbf{W}^{(k)} \in \mathbb{R}^{d_X \times F}$ is the weight matrix for different order. Given the diffusion graph G_i and its inverse diffusion graph G_i^T, we have:

$$\begin{aligned} \mathbf{X}_i &= \text{Concat}(\mathbf{E}_i, \mathbf{U}_i), \\ \mathbf{H}_i^{inf} &= \sigma(\mathop{\Big\|}_{k \in \mathcal{O}} (\hat{\mathbf{A}}_i^{(k)} \mathbf{X}_i \mathbf{W}_1^{(k)})), \\ \mathbf{H}_i^{sus} &= \sigma(\mathop{\Big\|}_{k \in \mathcal{O}} ((\hat{\mathbf{A}}_i^{T})^{(k)} \mathbf{X}_i \mathbf{W}_2^{(k)})). \end{aligned} \tag{7.3}$$

$\mathbf{E}_i = \{\mathbf{e}_j^{\mathcal{G}} | u_j \in U_i\}$, where $\mathbf{e}_j^{\mathcal{G}}$ is looked up through global graph embedding matrix $\mathbf{E}^{\mathcal{G}}$ by given user id, and $\mathbf{U}_i$ is the user characteristics matrix. $\mathbf{H}_i^{inf}, \mathbf{H}_i^{sus} \in \mathbb{R}^{|U_i| \times F}$

are user influence and user susceptibility, respectively.

7.1.3.5 Users' temporal learning

We learn temporal information for users from the diffusion path $\mathcal{P}_i$. Specifically, by modeling the timestamp information we can extract the dynamic and temporal information for each participant user, which has been demonstrated to help rumor detection [162].

Assume that, the time window is $[t_1, t_o]$, we first split the time window into l disjoint time intervals, and compute the corresponding time interval for each retweet user u_j as $\text{pos} = \left\lfloor \frac{t_j - t_1}{t_o/l} \right\rfloor$, where t_0 is the timestamp for the source post user. Then, we use positional encoding introduced in the Transformer [131] to allocate initial embedding for each time interval.

$$\begin{aligned} TP(\text{pos})_{2d} &= \sin \frac{\text{pos}}{10000^{2d/d_{time}}}, \\ TP(\text{pos})_{2d+1} &= \cos \frac{\text{pos}}{10000^{2d/d_{time}}}. \end{aligned} \tag{7.4}$$

where $\text{pos} \in [0, l)$ denotes the time interval each user fall into, d refers to dimension, and d_{time} is the total dimensions of the time interval embedding. So that, for a given tweet m_i, we construct an initial time-embedding matrix denoted as $\mathbf{T}_i \in \mathbb{R}^{|U_i| \times d_{time}}$.

After that, we feed $\mathbf{T}_i$ into a Bi-directional GRU (Bi-GRU) [108] to learn the temporal information $\mathbf{H}_i^{time}$ for users.

$$\mathbf{H}_i^{time} = \text{Bi-GRU}(\mathbf{T}_i), \mathbf{H}_i^{time} \in \mathbb{R}^{|U_i| \times F} \tag{7.5}$$

7.1.3.6 Feature-level aggregation attention

After obtaining the multi-scale latent representation of users, i.e., $\mathbf{H}_i^{inf}$, $\mathbf{H}_i^{sus}$ and $\mathbf{H}_i^{time}$, we propose an attention-based method to capture the different importance among three types of representations. Let $\hat{\mathbf{u}}_j = \left[\mathbf{h}_j^{inf}, \mathbf{h}_j^{sus}, \mathbf{h}_j^{time}\right] \in \mathbb{R}^{3 \times F}$ denote the learned feature set for user u_j. The attention $\mathbf{a}_j$ for $\hat{\mathbf{u}}_j$ is calculated as:

$$\begin{aligned} \hat{\mathbf{u}}_j^{'} &= \tanh\left(\hat{\mathbf{u}}_j \cdot \mathbf{w}_j\right), \\ \mathbf{a}_j &= \text{softmax}\left(\hat{\mathbf{u}}_j^{'} \cdot \mathbf{w}_j^{'}\right), \end{aligned} \tag{7.6}$$

where $\mathbf{w}_j \in \mathbb{R}^{F \times F}$ and $\mathbf{w}_j^{'} \in \mathbb{R}^{F \times 1}$ are weight matrices, $\mathbf{a}_j \in \mathbb{R}^{3 \times 1}$ is the learned attention. Then the aggregated feature vector for user u_j is $\bar{\mathbf{u}}_j = \hat{\mathbf{u}}_j \cdot \mathbf{a}_j$, where $\bar{\mathbf{u}}_j \in \mathbb{R}^F$. Finally, we get the fused user feature vector matrix as $\bar{\mathbf{U}}_i = \{\bar{\mathbf{u}}_j | j \in [1, |U_i|]\}$.

7.1.3.7 VAE-based uncertainty learning

In most of the existing works, the learned $\bar{\mathbf{U}}_i$ can be directly fed into a classification layer to predict the label of m_i. In our work, motivated by the ability of variational autoencoders (VAE) [163] in coping with randomness and uncertainty, we employ VAE to capture the uncertainty in the learned user features. Let $f_{\text{Dec}}(\cdot)$, $f_{\text{Dec}}(\cdot)$ and $f_{\text{NN}}(\cdot)$ denote the encoder, decoder and neural network, respectively. Then the VAE-based uncertainty learning layer can be simply formalized as:

$$\begin{aligned} &\mathbf{z}_j = f_{\text{ENC}}(\bar{\mathbf{u}}_j), \bar{\mathbf{u}}_j^{'} = f_{\text{Dec}}(\mathbf{z}_j), j = 1, 2, \cdots, |U_i|. \\ &\mu_j = f_{\text{NN}}(\bar{\mathbf{u}}_j), \log \sigma_j^2 = f_{\text{NN}}(\bar{\mathbf{u}}_j), \mathbf{z}_j \sim \mathcal{N}(\mu_j, \sigma_j^2) \end{aligned} \tag{7.7}$$

where $\bar{\mathbf{u}}_j^{'}$ represents the reconstructed input features. $\mathbf{z}_j \in \mathbb{R}^{d_z}$ is the latent vector. Specifically, VAE gets μ and $\log \sigma^2$ from the encoder (we omit the subscript j for simplicity), and then samples latent representation $\mathbf{z}$ from Gaussian distribution via reparameterization trick, where $\mathbf{z} = \mu + \sigma\epsilon$ and $\epsilon \sim \mathcal{N}(0, 1)$. Then the decoder takes the latent representation $\mathbf{z}$ as input, and try to reconstruct the original input feature. In general, the marginal log-likelihood of $\bar{\mathbf{u}}$ – $\log p_\theta(\bar{\mathbf{u}}) = \log \int_{\mathbf{z}} p_\theta(\bar{\mathbf{u}}|\mathbf{z}) p(\mathbf{z}) d_{\mathbf{z}}$, which is intractable to compute effectively. Instead, we adopt variational inference by defining a simple parametric distribution over the latent variables $q_\phi(\mathbf{z}|\bar{\mathbf{u}})$ (a.k.a. f_{Enc} parameterized by ϕ), and maximizing the evidence lower bound (ELBO) on the marginal log-likelihood of each observation:

$$\begin{aligned} \log p_\theta(\bar{\mathbf{u}}) &= \log \int_{\mathbf{z}} p_\theta(\bar{\mathbf{u}}|\mathbf{z}) p(\mathbf{z}) d_{\mathbf{z}} \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\bar{\mathbf{u}})} \log[\frac{p_\theta(\bar{\mathbf{u}}, \mathbf{z})}{q_\phi(\mathbf{z}|\bar{\mathbf{u}})}] + \mathbb{KL}[q_\phi(\mathbf{z}|\bar{\mathbf{u}})||p_\theta(\mathbf{z}|\bar{\mathbf{u}})] \\ &\geq \mathbb{E}_{q_\phi(\mathbf{z}|\bar{\mathbf{u}})}[\log p_\theta(\bar{\mathbf{u}}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\bar{\mathbf{u}})] \triangleq \text{ELBO}(\bar{\mathbf{u}}) \end{aligned} \tag{7.8}$$

To optimize the ELBO, we use a parametric inference network and reparameterization of $q_\phi(\mathbf{z}|\bar{\mathbf{u}})$ to alternatively maximize the following reformulation:

$$\begin{aligned} \text{ELBO}(\bar{\mathbf{u}}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\bar{\mathbf{u}})}[\log p_\theta(\bar{\mathbf{u}}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\bar{\mathbf{u}})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\bar{\mathbf{u}})}[\log p_\theta(\mathbf{z}) + \log p_\theta(\bar{\mathbf{u}}|\mathbf{z}) - \log q_\phi(\mathbf{z}|\bar{\mathbf{u}})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\bar{\mathbf{u}})} p_\theta(\bar{\mathbf{u}}|\mathbf{z}) - \mathbb{KL}[q_\phi(\mathbf{z}|\bar{\mathbf{u}})||p_\theta(\mathbf{z})] \end{aligned} \tag{7.9}$$

where $p_\theta(\bar{\mathbf{u}}|\mathbf{z})$ denotes the decoder and the first term of Equation (7.9) is the reconstruction loss, which is used to measure the likelihood value of the reconstructed features. The second term is the Kullback-Leibler (KL) divergence between the variational distribution $q_\phi(\mathbf{z}|\bar{\mathbf{u}})$ and the prior $p_\theta(\mathbf{z})$ (which is always ≥ 0). Therefore, the objective of maximizing ELBO of $\log p_\theta(\bar{\mathbf{u}})$ turns to minimize the Kullback-Leibler (KL) divergence. Through this VAE-based uncertainty learning layer, the learned latent representation for all users form a user latent representation matrix $\mathbf{U}_i^z = \{\mathbf{u}_j^z | j \in \mathbb{R}^{|U_i| \times d_z}\}$.

7.1.3.8 User-level aggregation attention

We concatenate $\bar{\mathbf{U}}_i$ and $\mathbf{U}_i^z$ at user-level to form a new user representation matrix $\mathbf{U}_i^F \in \mathbb{R}^{|U_i| \times (F+d_z)}$. Then we try to merge the user-level information to form an unique representation $\mathbf{R}_i$ for tweet m_i through attention sum-pooling operation:

$$\begin{aligned} \hat{a}_j &= \frac{\exp(\langle \mathbf{w}, \tanh(\mathbf{W}_a \mathbf{u}_j^F + \mathbf{b}_a) \rangle)}{\sum_{*:=1}^{|U_i|} \exp(\langle \mathbf{w}, \tanh(\mathbf{W}_a \mathbf{u}_*^F + \mathbf{b}_a) \rangle)}, \\ \mathbf{R}_i &= \sum_{j=1}^{N} \hat{a}_j \mathbf{u}_j^F \end{aligned} \tag{7.10}$$

where $\mathbf{W}_a \in \mathbb{R}^{(F+d_z) \times d}$, $\mathbf{b}_a \in \mathbb{R}^d$ and $\mathbf{w} \in \mathbb{R}^d$.

7.1.3.9 Rumor detection

Our ultimate goal is to predict the rumor label $\hat{y}_i$ of tweet m_i. We calculate this through feeding $\mathbf{R}_i$ into several fully connected layers and a softmax output layer, which is denoted as:

$$\hat{\mathbf{y}}_i = \text{softmax}\left(\text{FC}\left(\mathbf{R}_i\right)\right) \tag{7.11}$$

where $\hat{\mathbf{y}}_i$ is a vector of predicted probabilities of all rumor categories for the tweet m_i.

In the implementation, we train PLRD to estimate all the model parameters by minimizing the *cross-entropy* of the predictions $\hat{\mathbf{Y}}$ and the ground truth labels $\mathbf{Y}$. The prediction loss is:

$$\mathcal{L}_{pre} = -\frac{1}{|B|} \sum_{i=1}^{|B|} \sum_{c=0}^{1} y_{i,c} \log \hat{y}_{i,c} \tag{7.12}$$

where $|B|$ is the batch size, $y_{i,c}$ and $\hat{y}_{i,c}$ are the ground truth and predicted results for the i-th sample. That is, if the sample belongs to c-th class, $\hat{y}_{i,c}$ is 1; otherwise it is 0.

The total loss of PLRD should take the ELBO into consideration, that is:

$$\mathcal{L} = \mathcal{L}_{pre} - \frac{1}{|B|} \sum_{i=1}^{|B|} \text{ELBO}(\bar{\mathbf{U}}_i) \tag{7.13}$$

During training, the well-known stochastic gradient descent is applied to update parameters. Specifically, we use the adaptive learning rate optimization algorithm Adam [139] for model training. All hyper-parameters are tuned using the standard grid search for optimal results. The next section provides the details of the computational experiments.

7.1.3.10 Computational complexity analysis

In this section, we give a brief analysis of the computational complexity of PLRD. (1) The complexity for social homophily learning from global graph: as analyzed in [155], the overall complexity of this layer is $\mathcal{O}(d^2|U| + |E|)$, where d, $|U|$ and $|E|$ are the dimensions of user social homophily, number of nodes and edges in global graph, respectively. (2) The complexity for users' influence and susceptibility learning: we use a multi-hop graph convolutional layer to learn the users' influence and susceptibility (cf. Eq 7.2). Recall that, the dimensions of the input and the output are d_X and F, respectively, the max-order is K, and the normalized adjacency matrix $\hat{\mathbf{A}}$ is a sparse matrix with m nonzero elements. Therefore, for a single MHGCN layer, the computational complexity is $\mathcal{O}(F \times K \times m \times d_X)$. (3) The other parts of the PLRD are implemented by GRUs and MLPs. The time and space complexity are related to the input dimensions of latent variables. Since the users' social homophily are computed in preprocessing phase, the computational complexity of whole PLRD is therefore $\mathcal{O}(F \times K \times m \times d_X)$.

7.1.4 Evaluation

In this section we evaluate our proposed PLRD framework and demonstrate its practical utility through quantitative experiments.

7.1.4.1 Evaluation metrics and baselines

In our work, we use Accuracy (ACC), Precision (Pre), Recall (Rec), and F1 as the evaluation protocols to measure the models' performance. In particular, ACC measures the proportion of correctly classified tweets, while F1 is the harmonic mean of the precision and recall.

We compare our method with a battery of baselines, they are:

- **DTC** [23]: A decision tree-based classification model that combines manually engineered characteristics of tweets to compute the information credibility.
- **SVM-TS** [85]: A linear SVM-based time series model, which can capture the variation of a broad spectrum of social context information over time by converting the continuous-time stream into fixed time intervals.
- **GRU** [33]: An RNN-based model has been employed to learn the sequential cascading effect of tweets with high-level feature representations extracted from relevant posts over time.
- **TD-RvNN** [34]: A tree-structured RNN model for rumor detection, which embeds hidden indicative signals in the tree-structures and explores the importance of comments for rumor detection.

- **PPC_RNN+CNN** [100]: An early-stage rumor detection model through classifying news propagation paths with RNN and CNN, which learns the rumor representations through the characteristics of users and source tweets (for brevity, the model name is abbreviated to PPC).
- **dEFEND** [87]: A co-attention-based fake news detection model that exploits both news contents and user comments for fake news detection.
- **Bi-GCN** [16]: A GCN-based model exploiting the bi-directional propagation structures and comments for rumor detection.
- **GCAN** [102]: A co-attention network that detects true and false rumors based on the content of the source tweet and its propagation-based users. We also provide a variant of GCAN, denoted as GCAN-Text, which removes the source tweet features in the original inputs.

7.1.4.2 Experimental setup

We implement DTC with Weka[1], SVM-based models with scikit-learn[2], and other neural network-based models with Tensorflow[3]. All baselines follow the parameter settings in the original papers.

For PLRD, the learning rate is initialized at 0.001 and gradually decreases as the training proceeds. We set the embedding size d for the social homophily to 40. As for time-embedding, we set the total number of time intervals to be 100 and each interval represents 10 minutes. Retweets with a latency of more than 1,000 minutes would fall into the last time interval. The size of time-embedding is d_{time} set to 50. The hidden size F of user influence, susceptibility, and temporal information are set to 64; the hidden size d_z of the VAE-based uncertainty learning layer is set to 32. The batch size is set to 32 for Twitter15/16, 128 for Science and 16 for RumourEval19, and the training process is iterated upon for 500 epochs but would be stopped earlier if the validation loss does not decrease after 10 epochs. And we randomly choose 70% data for training and the rest of 10% and 20% for validation and testing. The experiments are conducted on a machine with an Intel i7-6700 3.40GHZ CPU and a single NVIDIA GeForce GTX 2080Ti. The time cost for model training is less than 10 minutes on all datasets used in this work.

7.1.4.3 Study on Twitter15/16

Table 7.5 and 7.6 summarizes the overall performance on Twitter15 and Twitter16 datasets. The last two rows show the performance of the complete version of our model PLRD and the improvement percentage compare with the second-best method, which basically yields much better performance than the other baseline

[1]https://www.cs.waikato.ac.nz/ml/weka/
[2]https://scikit-learn.org/
[3]https://www.tensorflow.org/

Table 7.5: Overall performance comparison of rumor detection on Twitter15. The best method is shown in **bold**, and the second best one is underlined. The number of retweets is 40.

	Twitter15			
Method	Acc	Pre	Rec	F1
DTC	0.495	0.494	0.481	0.495
SVM-TS	0.519	0.519	0.518	0.519
GRU	0.580	0.544	0.545	0.544
TD-RvNN	0.678	0.671	0.674	0.672
PPC	0.691	0.674	0.686	0.679
dEFEND	0.738	0.658	0.661	0.654
Bi-GCN	0.748	0.731	0.759	0.745
GCAN	0.875	0.825	0.829	0.825
GCAN-Text	0.683	0.705	0.652	0.678
PLRD	**0.934**	**0.928**	**0.929**	**0.927**
Improvement	**8.98%**	**12.5%**	**12.1%**	**12.4%**

Table 7.6: Overall performance comparison of rumor detection on Twitter16. The best method is shown in **bold**, and the second best one is underlined. The number of retweets is 40.

	Twitter16			
Method	Acc	Pre	Rec	F1
DTC	0.561	0.575	0.537	0.562
SVM-TS	0.693	0.692	0.691	0.692
GRU	0.554	0.514	0.516	0.515
TD-RvNN	0.661	0.632	0.641	0.636
PPC	0.655	0.632	0.651	0.641
dEFEND	0.702	0.637	0.638	0.631
Bi-GCN	0.711	0.709	0.710	0.716
GCAN	0.823	0.803	0.841	0.822
GCAN-Text	0.664	0.716	0.579	0.648
PLRD	**0.875**	**0.876**	**0.874**	**0.855**
Improvement	**6.32%**	**9.09%**	**3.92%**	**4.01%**

methods across all metrics. Note that we conduct a McNemar's test [144] between our PLRD and the best baseline based on the prediction results on the testing set. The p-values are $p < 0.001$ on both Twitter15 and Twitter16. Therefore, we can conclude that the performance between PLRD and GCAN exists with statistical significance.

We make the following additional observations. *O1:* The feature-based approaches – DTC and SVM-TS use hand-crafted features based on the overall statistics of tweets, which perform poorly. These two methods not sufficient to capture the generalizable features associated with tweets and the diffusion process. Notably, SVM-TS achieves

comparatively better performance than DTC because it utilizes an extensive set of features and focuses on retweets' temporal features. *O2:* Deep learning-based models achieve better performance than feature-based methods. GRU is the first deep-learning-based method for rumor detection, which performs worst among deep-learning-based baselines because it only relies on the temporal-linguistics features of the post sequence but ignores other useful information such as diffusion structures and user profiles. Both TD-RvNN and PPC outperform GRU, which indicates the effectiveness of modeling the propagation structure and temporal information in rumor detection. Moreover, PPC proves user profile features as important as text features for rumor detection. dEFEND utilize a co-attention mechanism to learn the correlation between news contents and user comments, which performs better than TD-RvNN and PPC but worse than Bi-GCN and GCAN. Bi-GCN and GCAN claim that they can learn structure information from graphs, and their performance indeed exceeds other baseline methods. However, Bi-GCN constructs the structural tree based on the replies, which can not reflect the full process of rumor diffusion. As for GCAN, it captures the similarity between users rather than propagation structural features. According to the results, GCAN performs much better than Bi-GCAN, because it takes both text information and user profiles into consideration. By comparing GCAN with its variants GCAN-Text, we can find that after removing text information, the performance of GCAN remarkably decrees, demonstrating that GCAN is not efficient to capture user-related features. *O3:* PLRD consistently outperforms all baselines on both Twitter15 and Twitter16. Compare to the best baseline method Bi-GCN, PLRD learns rumor representation from a participant-level without any text information, demonstrate the primary motivations of this work – i.e., users are the main contributor to the rumor propagation.

7.1.4.4 Study on Science

Table 7.7: Overall performance comparison of rumor detection on Science. The best method is shown in **bold**. The number of retweets is 40.

	Science			
Method	Acc	Pre	Rec	F1
PPC	0.655	0.649	0.568	0.606
GCAN-Text	0.671	0.646	0.622	0.634
PLRD	**0.768**	**0.727**	**0.814**	**0.768**

In this section, we conduct an experiment on Science dataset. Specifically, from the original Science dataset, we first filter out (1) the tweets with less than 10 retweets and more than 100; and (2) the tweets' with a diffusion period exceed 24 hours. After that, we have $3,493$ tweets in total, and the processed dataset is still highly imbalanced, e.g., only 610 tweets were labeled as "non-rumor", while the majority (i.e., $2,883$) were classified as "rumor". We randomly select $1,000$ items from the rumor set to make sure the number of tweets labeled as non-rumors is 50% of the

rumors, and finally, we get a new experiment dataset with 1,000 rumors and 610 non-rumors. Table 7.3 summarizes the performance comparison between PLRD and two state-of-the-art propagation-based baselines, i.e., PPC and GCAN-Text[1]. We can observe that our PLRD outperforms both PPC and GCAN-Text on all metrics, which indicates that our model is more effective and stable in extracting diffusion patterns of users even without any textual information.

7.1.4.5 Study on RumourEval19

Table 7.8: Overall performance comparison of rumor detection on RumourEval19. The best method is shown in **bold**. The result of the "Base" model has referenced the best method from the paper [150]. "w/o Text" means without textual features. The number of comments is 100.

	RumourEval19			
Method	Acc	Pre	Rec	Macro-F1
Base	–	0.596	0.603	0.577
TD-RvNN	0.667	0.641	0.673	0.615
Bi-GCN	0.734	0.733	0.735	0.661
PLRD	**0.813**	**0.826**	**0.885**	**0.788**
PLRD w/o Text	0.750	0.806	0.842	0.692

Since the RumorEval19 dataset has rich textual features, in this section we conduct an experiment on RumourEval19 to test the PLRD performance when including textual features. Specifically, we first use a pre-trained model – BERTweet [164] – to generate the tweet embedding for each source tweet and its corresponding comments, and then concatenate the source tweet embedding and comments embedding to form a textual embedding matrix $\mathbf{C} \in \mathbb{R}^{|U_i| \times d_{text}}$ for each tweet. Finally, we combine $\mathbf{C}$ with $\mathbf{X}$ to form the input of PLRD. For a comparison, we choose TD-RvNN and Bi-GCN as baselines, and also provide the result of the best method in [150] denotes as "Base". From Table 7.8, we can find that under the situation of unbalanced label distribution, our PLRD can still achieve competitive performance compared with other baselines on rumor detection. After deleting the textual features, the performance of PLRD slightly drops, which demonstrate that textual features are powerful and can help improve the model performance.

7.1.4.6 Ablation study

In this section, we conduct an ablation study on Twitter15 and Twitter16 to explore the effect of each component in PLRD. Towards that, we derive the following variants of PLRD:

[1]The Science dataset only provides anonymous user profile characteristics and propagation threads. No textual features are available. So we compare with PPC and GCAN-Text.

- **w/o user profiles (UP)**: In "**w/o UP**", we do not consider the user profile characteristics, which means we do not use $\mathbf{U}_i$ and only keep $\mathbf{E}_i$ as input features.
- **w/o social homophily (SH)**: In "**w/o SH**", we ignore the social homophily of users, which means we do not use $\mathbf{E}_i$ and only keep $\mathbf{U}_i$ as input features.
- **w/o user influence (UI)**: In "**w/o UI**", we do not capture user influence, which means ignore $\mathbf{H}_i^{inf}$.
- **w/o user susceptibility (US)**: In "**w/o US**", we do not capture user susceptibility, which means ignore $\mathbf{H}_i^{sus}$..
- **w/o user temporal (UT)**: In "**w/o UT**", we do not take the user temporal information into consideration, which means ignore $\mathbf{H}_i^{time}$.
- **w/o graph information (GI)**: In "**w/o GI**", we do not utilize any information from the global graph and the diffusion graph, which means we ignore users' social homophily, influence, susceptibility and only keep the users' temporal learning component. The input of this part is the concatenation of user features and temporal information.
- **w/o feature uncertainty (FU)**: In "**w/o FU**", we remove the VAE-based uncertainty learning layer and use $\bar{\mathbf{U}}_i$ directly.
- **w/o feature-level attention (FA)**: In "**w/o FA**", we remove the feature-level aggregation attention in PLRD and concatenate the different user features directly, i.e., $\bar{\mathbf{U}}_i = \text{concat}(\mathbf{H}_i^{inf}, \mathbf{H}_i^{sus}, \mathbf{H}_i^{time}) \in \mathbb{R}^{|U_i| \times 3F}$.
- **w/o user-level attention (UA)**: In "**w/o UA**", we do not allocate different importance for each user and directly use a sum-pooling to form the rumor representation.

Table 7.9: performance comparison between PLRD and its variants.

	Twitter15				Twitter16			
Method	Acc	Pre	Rec	F1	Acc	Pre	Rec	F1
w/o UP	0.853	0.842	0.838	0.828	0.838	0.858	0.847	0.840
w/o SH	0.906	0.894	0.910	0.896	0.802	0.821	0.786	0.794
w/o UI	0.868	0.863	0.871	0.859	0.800	0.859	0.871	0.867
w/o US	0.841	0.877	0.857	0.864	0.781	0.794	0.775	0.782
w/o UT	0.873	0.914	0.935	0.920	0.795	0.792	0.802	0.788
w/o GI	0.806	0.755	0.811	0.782	0.758	0.716	0.732	0.724
w/o FU	0.913	0.911	0.914	0.911	0.854	0.847	0.859	0.848
w/o FA	0.811	0.842	0.843	0.848	0.790	0.831	0.837	0.816
w/o UA	0.896	0.906	0.877	0.886	0.854	0.838	0.831	0.829
PLRD	**0.934**	**0.928**	**0.929**	**0.927**	**0.875**	**0.876**	**0.874**	**0.855**

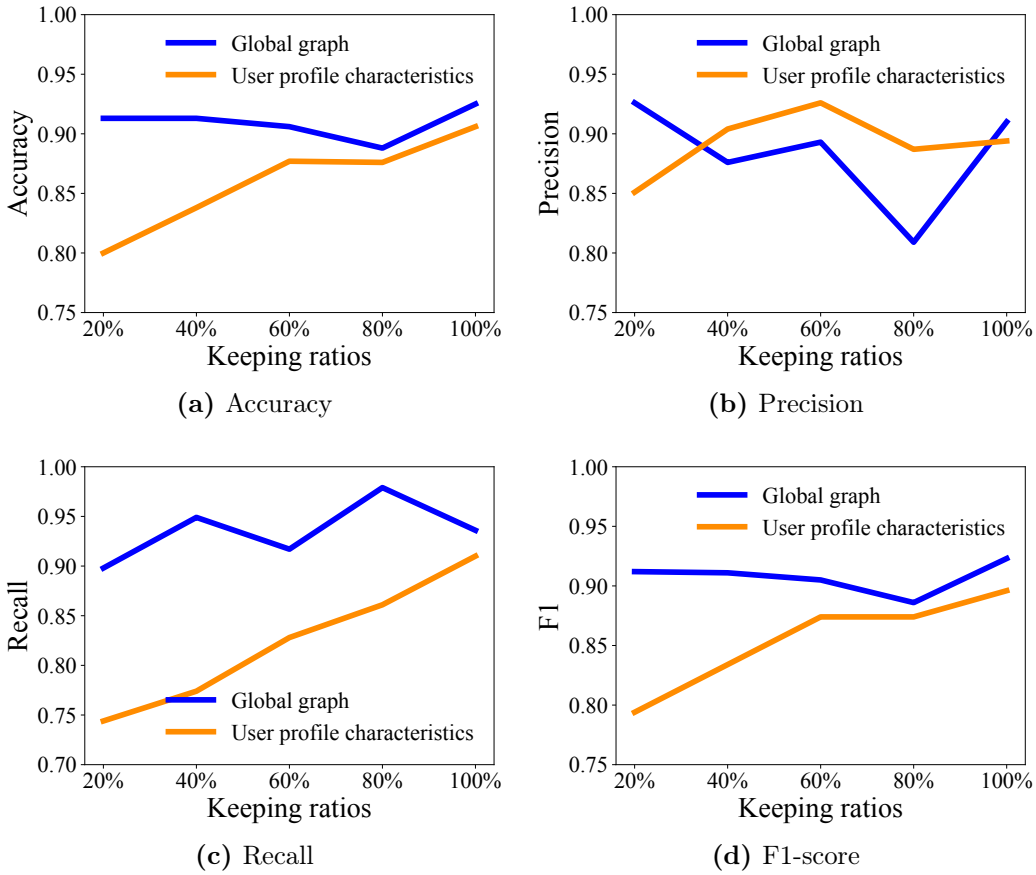
The results, shown in Table 7.9, indicate that the original PLRD outperforms these variants in terms of all metrics. From Table 7.9, we can observe that: (1) Both user profile features (w/o UP) and social homophily ('w/o SH) are reliable inputs of our model that because user profiles can be used to identify an individual, and social homophily can reflect the user preference. (2) User influence (w/o UI), susceptibility (w/o US), and temporal features (w/o UT) are indispensable for rumor detection. (3) The result of "w/o GI" performs worst among all variants, demonstrating that graph data (global graph and diffusion graph) provide considerable meaningful features, and are thus indispensable in rumor detection. (4) The fact that "w/o FU" provides lower performance compare with PLRD, reflects the benefit of modeling the feature uncertainty. (5) The two attention-based aggregation layers, i.e., feature-level aggregation attention (w/o FA) and user-level aggregation attention (w/o UA), play crucial roles in detecting rumors. Especially the feature-level aggregation attention (w/o FA), after removing it, the performance remarkably decreases, suggesting that distinguishing the importance of different scale of user features can improve detection performance. Similarly, "w/o UA" demonstrates that different users play different roles in rumor spreading.

7.1.4.7 Privacy-preserving study

In this section, we conduct experiments on Twitter15 dataset to test our PLRD's performance on privacy-preserving scenarios which can be summarized as: (1) randomly removing different proportions of edges in the global graph $\mathcal{G}$; (2) randomly masking different proportions of user characteristics $\mathbf{u}$, and (3) randomly removing different proportions of edges in the diffusion graph G. Note that, in scenario (1) and (2) we only keep related features as inputs, i.e., in scenario (1), we only use the user homophily $\mathbf{E}$ generated from the global graph as input, and in scenario (2) we only keep user characteristic $\mathbf{U}$ as input. Both scenarios are tested on 40 retweets.

Figure 7.4 plots the performance of PLRD in scenarios (1) and (2), which shows that even though we only keep 20% of edges in the global graph, the PLRD still achieves 90% accuracy. However, when masking 80% of user characteristics, the performance of PLDR drops significantly.

Figure 7.5 shows the performance of PLRD with the different numbers of retweets in scenario (3). We find that when we only observe few retweets, e.g., 10 and 40, the performance of the model decreases as the removal of edges in the diffusion graph. In contrast, when the number of observation retweets is sufficient enough such as 100, dropping a few edges would help improve the model performance. The reasons behind is that: (1) with the increase of observed retweets, there is a great possibility to introduce noise into the graph, which can be eliminated with random removal of graph edges; and (2) randomly drop a few of graph edges is widely used as a data augmentation method in graph representation learning field [165, 166], which can



(a) Accuracy **(b)** Precision

(c) Recall **(d)** F1-score

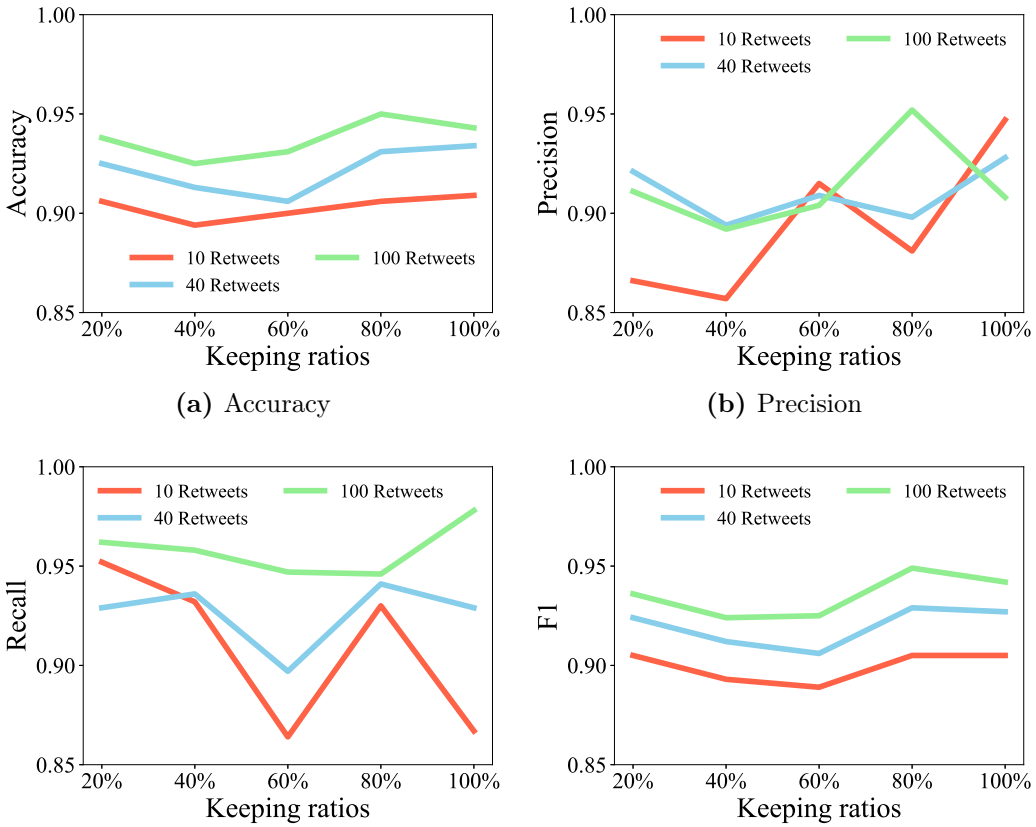
Figure 7.4: Evaluations on randomly removing different proportions of edges in the global graph and random masking different proportions of user characteristics.

improve model generalization and overcome the overfitting and over-smooth issues of graph neural networks.)

7.1.4.8 Early detection

Another critical goal of rumor detection is to detect rumors as early as possible that is essential to stop their spread in a timely fashion. Next we investigate the performance of models on identifying rumors at an early stage. Here, we consider the early 50 retweets.

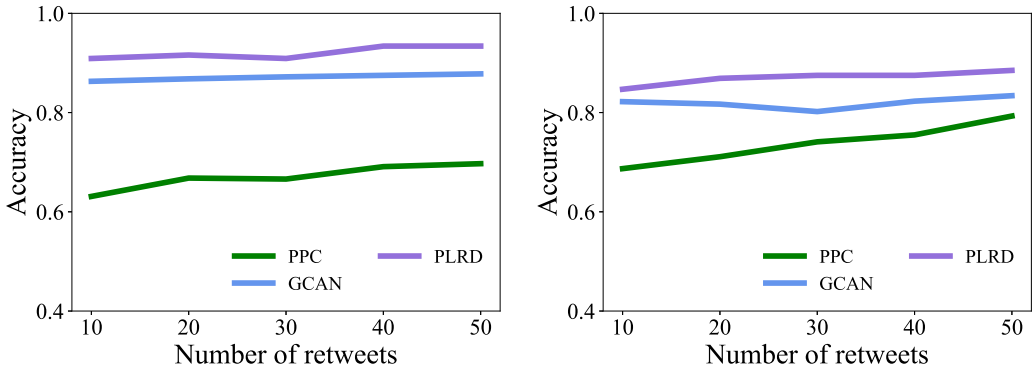
Figure 7.6 shows the performance comparison on early-stage detection between our PLRD and the selected baselines. Note that we omit the feature-based approaches (i.e., DTC and SVM-TS) and GRU since they did not show comparable performance, especially on early rumor detection. Moreover, we also ignore TvRvNN, dEFEND, and Bi-GCN, because these methods are built on the replies that may not exist in the early-stage. We observe that PLRD performs better than PPC and GCAN, especially when there are only a few observations. PLRD needs a short time to identify the misinformation because PLRD learns the rumor representation from a



(a) Accuracy **(b)** Precision

(c) Recall **(d)** F1-score

Figure 7.5: Evaluations on randomly removal of diffusion links based on Twitter15.

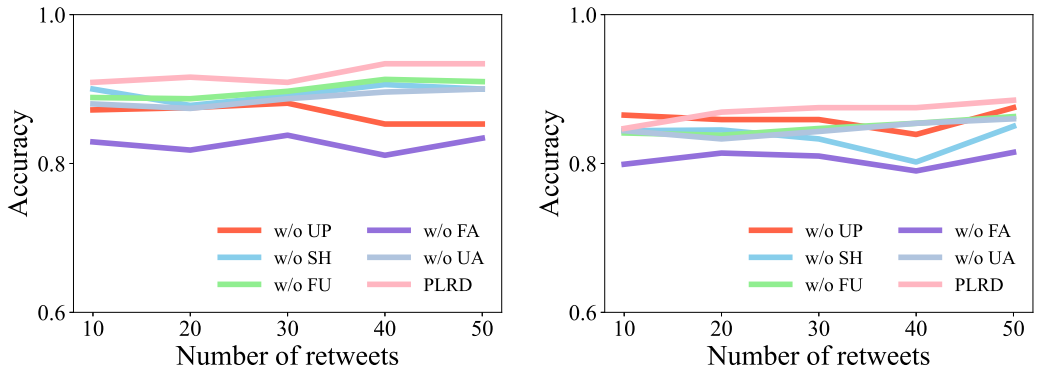


(a) Early 50 retweets (Twitter15). **(b)** Early 50 retweets (Twitter16).

Figure 7.6: Evaluations on early rumor detection.

participant-level and fuses users' multi-scale knowledge, such as user influence, user susceptibility, user temporal information, etc.

We also investigate the time-varying performance between PLRD and its variants. Specifically, we choose "w/o UP", "w/o SH", "w/o FU", "w/o FA" and "'w/o UA' as the comparison methods. The results show in Figure 7.7. We find that the performance



(a) Early 50 retweets (Twitter15). **(b)** Early 50 retweets (Twitter16).

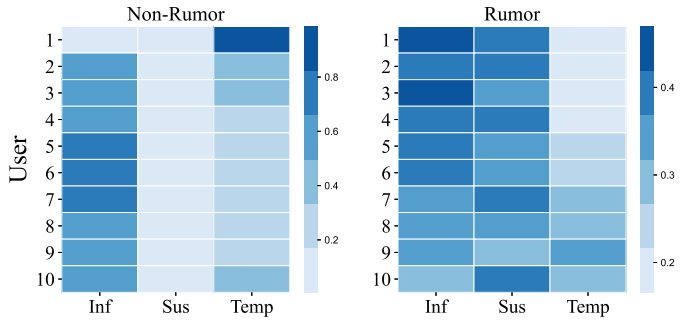
Figure 7.7: Evaluations on early rumor detection among variants of PLRD.

of PLRD surpasses all variants, and with the number of retweets increased, the accuracy of all methods grow to saturation.

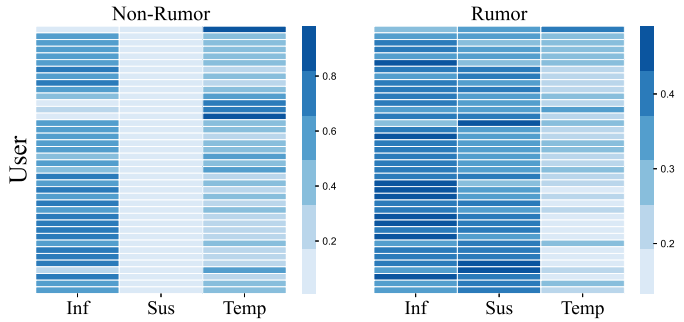
7.1.4.9 Interpretability analysis

The above ablation studies have shown the superiority of each component in PLRD. In this section, we provide more in-depth insights by visualizing features.

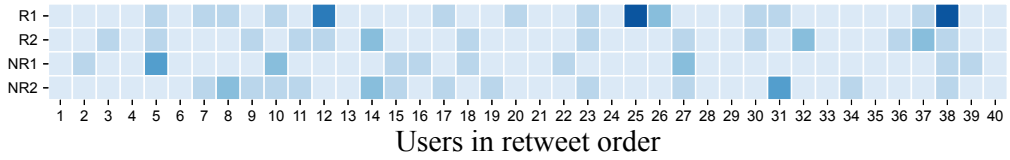
Figure 7.8 plots the importance of the feature-level aggregation layer and user-level aggregation layer. As for the feature-level aggregation attention, we randomly selected two different types of tweets in Twitter15 and plotted the importance of the different features. Figure 7.8a and Figure 7.8b show the results of previous 10 and 40 retweet users, respectively. Overall, we find that (1) the three types of features for each user have different importance; (2) attention distribution varies between rumor and non-rumor. Specifically, as for rumor tweets, participants try to affect others, while themselves are easier to expose in the misleading tweets. In contrast, in non-rumor tweets, participants are more influential compare with the susceptibility. Moreover, temporal information plays a crucial role in detecting both rumor and non-rumor. In Figure 7.8c, we investigate the role of the retweet users at the very beginning of the diffusion. As shown, the later users are more critical in rumor spreading, which confirms the hypothesis that rumors can spread deeper than non-rumors [43]. Moreover, to have an intuitive explanation regarding the superiority of each component in PLRD, we plot the learned latent representations (i.e.,$\bar{\mathbf{U}}$, $\mathbf{U}^z$, $\mathbf{U}^F$ and $\mathbf{R}$) using t-SNE [125]. Each point in the plot represents a tweet in the test set (tweets with similar latent vectors are closer in the plot), and different colors refer to different labels, i.e., green represents non-rumor, orange represents rumor. From Figure 7.9a, we see clear clustering phenomena by $\bar{\mathbf{U}}$. These latent vectors can already be used to predict directly. In contrast, Figure 7.9b "smoothes" this clustering effect by modeling the feature uncertainty, which should help explore more



(a) Feature-level aggregation attention (10 retweet user).



(b) Feature-level aggregation attention (40 retweet user).



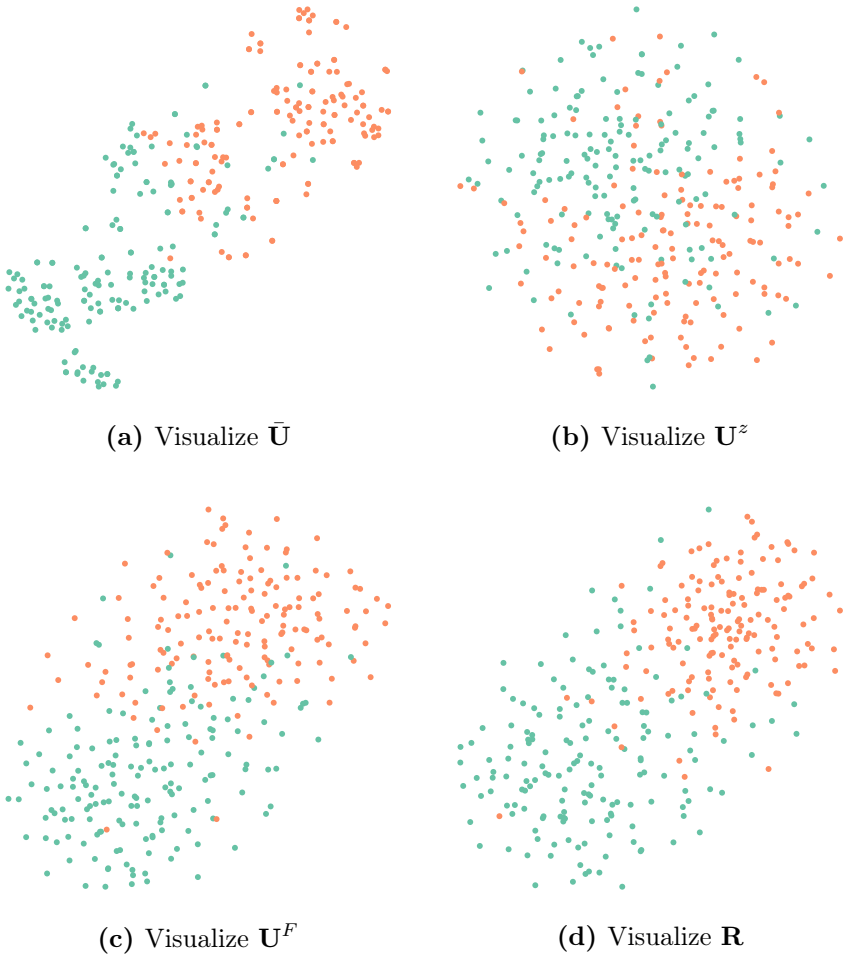
(c) User-level aggregation attention.

Figure 7.8: Attention visualization of PLRD. "Inf": influence; "Sus": susceptibility; "Temp": temporal.

possibilities. From Figure 7.9c – Figure 7.9d, we find that the model learned more suitable latent representations for prediction after a user-level attention layer.

7.1.5 Summary

In this first part of the chapter, we first provided empirical evidence that all participants in the diffusion chains of rumors exhibit different patterns than participants in the diffusion chains of non-rumors. Based on these findings, we proposed a novel fine-grained all- participant level rumor detection model, named PLRD (Participant-Level Rumor Detection). Specifically, PLRD learns fine-grained user representations, i.e., user influence, user susceptibility, and user temporal information from the propagation threads of a given post, and merges the learned features to form a unique rumor representation through a feature-level attention layer and a user-level attention layer. Moreover, a variational autoencoder used to capture uncertainty



(a) Visualize $\bar{\mathbf{U}}$ **(b)** Visualize $\mathbf{U}^z$

(c) Visualize $\mathbf{U}^F$ **(d)** Visualize $\mathbf{R}$

Figure 7.9: Visualization of the learned latent representation on Twitter15 using t-SNE. Each point is a sample from the test set. The color green represents non-rumor, and the orange one represents rumor.

from features further improves the learned rumor representation. Compared with existing rumor detection methods, PLRD makes predictions only based on user-level features learned from the diffusion process of posts, which overcomes the problem of overemphasizing the text features. We conducted experiments on four real-world datasets, Twitter15, Twitter16, Science and RumourEval19. The experiment results not only demonstrate that our model significantly outperforms the baselines regarding effective early detection, but also supports the hypothesis that the combination of various user information at a participant level in a diffusion chain will improve the performance of rumor discovery. Besides, our ablation study further demonstrates that each part in our model is indispensable for rumor detection.

7.2 UMLARD: Multi-view Learning with Distinguishable Feature Fusion for Rumor Detection

7.2.1 Section Overview

In section 7.1, we proposed a novel participant-level rumor detection model – PLRD, which can learn fine-grained all-participant patterns throughout the whole diffusion process, including social patterns (user social homophily), diffusion patterns (user influence and susceptibility), temporal patterns (how fast the news is propagated in the social network), all-participant profile patterns. Based on the experimental results compared with existing state-of-the-art deep learning-based rumor detection methods, we have seen that PLRD achieves significant improvements. In this section, we propose a new model UMLARD – User-aspect Multi-view Learning with Attention for Rumor Detection model, which can make even better predictions than PLRD and other recent works. UMLARD inherits the advantages of PLRD, i.e., solving the same limitations that are also considered by PLRD:

(L1) *Lack of systematic user-aspect rumor modeling*: Recent research [43] reveals that humans are the principal "culprits" in spreading false news. Existing studies either directly aggregate users' profile information as model inputs [85, 86, 102], which only pay attention to the local structure correlations among propagated users and the sequential propagation patterns [100] (i.e., user temporal features), or focus on learning the global structure of rumor diffusion [16] (i.e., user structural features). Essentially, these works learn the rumor representation from an event-level, and still lacks a unified framework that can learn rumor diffusion while extracting meaningful features from user-aspect.

(L2) *Indistinguishable importance of both features and users*: Different features play different roles in rumor detection at different phases of propagation. As information spreads, for example, the effect of structural information and temporal information on discriminating rumors becomes different [84, 85]. Also, users may either unconsciously forward some unproven news, or deliberately propagate the fake news in the information spread [26]. Understanding the efficacy of features and individual users at the same time would help detect rumors, which, however, has not been well investigated in existing studies.

(L3) *Limited interpretability*: Most existing studies focus on explaining the news content, e.g., discover the important sentence in the articles or emotional words in comments [87], to interpret the detection results. However, these works cannot explain critical features beyond text and determine user's roles in rumor propagation.

Besides all those points above, UMLARD also improves on PLRD by overcoming the following limitation is not solved by PLRD:

(L4) *Entangled high-level feature learning*: Existing works learn high-level representations (e.g., structural or temporal) for rumor detection by exploiting user profiles or pretrained textual features as model inputs. While improving detection performance, it is difficult to demonstrate the effectiveness and high-level representations of the model, because (1) the learned representations are entangled with the original input [102], and (2) the models use the same input [44] to learn different high-level features.

To develop UMLARD and address the limitations L1–L4 above, inspired by recent progress in multi-view learning [167, 168, 169], we initiate the attempts to capture the principal characteristics of users and rumors by learning multiple distinct features. Multi-view learning is a promising learning paradigm that jointly models different views of the same input data for improving learning performance [170]. For example, a web page can be described in forms of text, video, and image [171] simultaneously. By exploring the complementarity and consistency of different views, it can further improve the model performance [172]. Specifically, we exploit different views to represent an instance for comprehensively describing the information of the instance. We first abstract the user-aspect features of the users engaged in the diffusion process as user profile-view, user structural-view, and user temporal-view, and then incorporate different views to predict the credibility of the given information. Specifically, UMLARD exploits different embedding methods to learn the view-specific high-level representations of a given post from the hierarchical diffusion process and user profiles. To understand the importance of each view and the role of the user, UMLARD employs a view-wise attention network and a capsule attention network to incorporate both view-level and user-level features. It allows us to better discriminate feature influence and the effect of user behaviors in spreading rumors.

Our main contributions towards rumor detection problem provide:

- **User-aspect feature extraction (L1)**: We conceptualize user-aspect features as different views, including profile-view, structural-view, and temporal-view, and present a novel model to learn different views for each user who engaged in the information diffusion.
- **View-specific embedding methods (L2)**: UMLARD utilizes different embedding methods to learn view-specific high-level representations based on different inputs: (1) an attention-based layer aims to learn user profile-view by assigning different importance to features in user profiles; (2) an improved GCN-based network to learn structural-view from the diffusion network while considering the direction of information dissemination, taking the adjacency matrices of diffusion networks as input; and (3) a time-decay LSTM considers the influence of users and is used for temporal-view learning based on the diffusion path taking two types of embeddings as inputs, i.e., static-embedding and dynamic-embedding.

- **Distinguishable hierarchical feature fusion (L3)**: We design a hierarchical feature fusion mechanism to unify the knowledge from different perspectives, which consists of two components: (1) a view-wise attention layer to capture the features from different views; and (2) a capsule attention layer to differentiate the most related users.
- **Explainable prediction results (L4)**: UMLARD explains the significance of features according to the learned attention values. Specifically: (1) the dimensional-wise attention network shows the importance of different characteristics in the user profiles; (2) the view-wise attention results tell how the users play different roles in different phases of rumor propagation; and (3) from the capsule attention results, one can easily understand which users play critical roles in detecting the rumors.

This section is based on the following publication [46]:

- **Chen, X.**, Zhou, F., Trajcevski, G., Bonsangue, M.: Multi-view Learning with Distinguishable Feature Fusion for Rumor Detection. Knowledge-Based Systems 240 (2022) 108085

7.2.2 Problem Statement

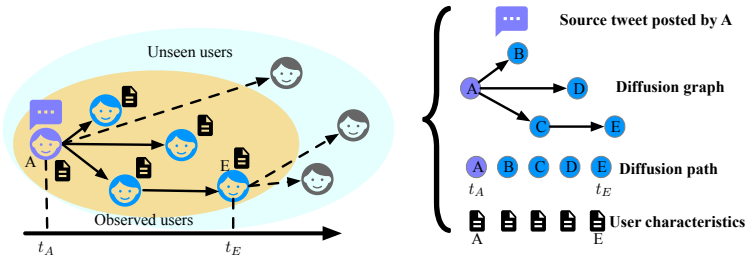


Figure 7.10: An example of the extracted information from a tweet diffusion.

Suppose we have a set of tweets $M = \{m_i, i \in [1, |M|]\}$, where each tweet m_i is a quadruplet representing the corresponding diffusion process and the users enrolled: $m_i = \{G_i, \mathcal{P}_i, \mathbf{U}_i, \mathbf{C}_i\}$, where $G_i, \mathcal{P}_i, \mathbf{U}_i, \mathbf{C}_i$ are diffusion graph, diffusion path, user characteristic matrix and the content vector of source tweet, respectively. The concepts of diffusion graph, diffusion path and user characteristic matrix are formally defined in Definition. 2, 3 and 5. And the definition of tweet content is shown as follow:

Definition 15 ***Tweet Content.*** *For a tweet m_i, the text content $\boldsymbol{C}_i$ is considered to be a sequence of words – i.e., $\boldsymbol{C}_i = [\boldsymbol{w}_{i1}, \boldsymbol{w}_{i2}, \ldots, \boldsymbol{w}_{iL}] \in \mathbb{R}^{L \times d_{word}}$, where L is the number of words in source tweet.*

We note that each word is represented by a d_{word}-dimension vector using a particular word embedding technique, e.g., word2vec.

We summarize the (definitions of the) symbols used in this section in Table 7.10. We note that, in the sequel, whenever there is no ambiguity, we may omit the double-subscript from the notation (i.e., whenever we are unambiguously working with one specific tweet m_i, we may drop i from the sequences denoting users, time-stamps, etc.).

Table 7.10: Main notations used throughout this chapter.

Symbol	Description
$\mathbf{p}_*$	the user profile vector of each user.
$\mathbf{g}_*$	The pre-trained node embedding of each user.
$\mathbf{e}_*^s$	The static embedding of each user.
$\mathbf{e}_*^d$	The dynamic embedding of each user.
d_{user}	The hidden size of the profile-view.
d_{stru}	The hidden size of the structural-view.
d_{temp}	The hidden size of the temporal-view.
d_{word}	The hidden size of the word embedding.
d_{view}	The hidden size of the multi-view layer.
$\mathbf{H}_i^{User}$	The representations of the profile-view.
$\mathbf{H}_i^{Stru}$	The representations of the structural-view.
$\mathbf{H}_i^{Temp}$	The representations of the temporal-view.
$\mathbf{H}_i^{Text}$	The representations of the content feature.
$\mathbf{V}_i^{'}, \mathbf{s}_{in}$	the representation after view-wise attention and capsule attention for tweet m_i.
$\mathbf{H}_i^{Rumor}$	the final representation of tweet m_i.
$\hat{\mathbf{Y}}/\hat{\mathbf{y}}_*$	The predicted label.
$\mathbf{Y}/\mathbf{y}_*$	The ground truth.

We now formally define the rumor detection problem that we study as follows:

Definition 16 ***Rumor Detection.*** *Given a tweet $m_i = \{G_i, \mathcal{P}_i, \boldsymbol{U}_i, \boldsymbol{C}_i\}$ within an observation window t_o, our rumor detection goal is to learn a function f from labeled claims, i.e., $f(\hat{\boldsymbol{y}}_i | G_i, \mathcal{P}_i, \boldsymbol{U}_i, \boldsymbol{C}_i; t_o)$, where the predicted result $\hat{\boldsymbol{y}}_i$ takes one of the four finer-grained classes: non-rumor, false rumor, true rumor, and unverified rumor (as introduced in [84]).*

7.2.3 Methodology

In this section, we first introduce the preliminaries and basic notations, and then formalize the problem studied in this paper. Subsequently, we present the details of the proposed UMLARD framework.

As illustrated in Figure 7.11, UMLARD consists of three main components: (1) *Representation learning layer* that simultaneously extracts user-aspect features from the profile-view, structural-view, and temporal-view, while embedding the source tweet content into low-dimensional space; (2) *Hierarchical fusion layer* that fuses the learned representation at both view-level and user-level; and (3) *Rumor detection*

layer that makes use of a fully connected layer to predict the labels of tweets, based on the learned user-aspect knowledge and tweet content.

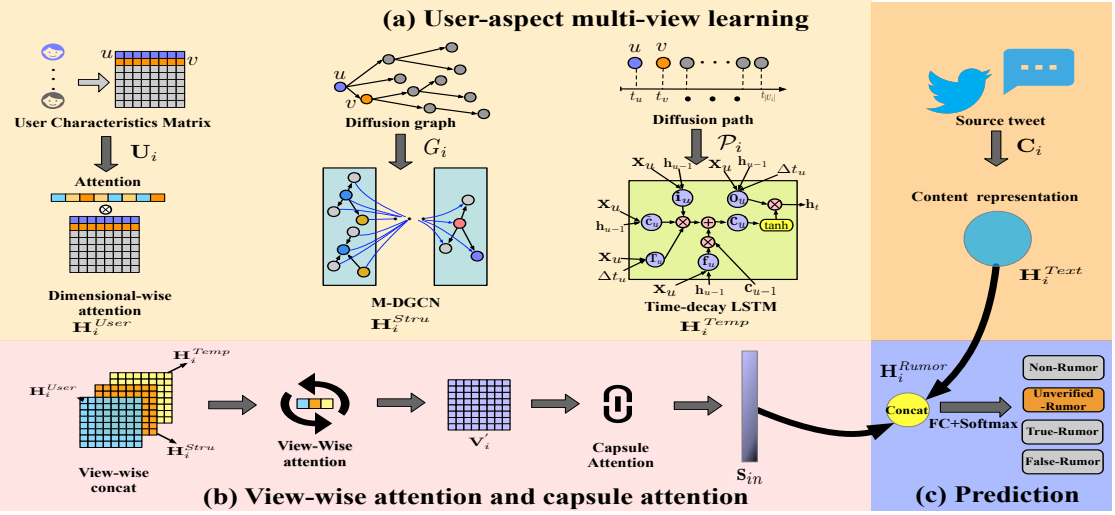


Figure 7.11: An overview of UMLARD. (a) The inputs of UMLARD are the observed diffusion network, the diffusion path, the user characteristic matrix, and the content of the source tweet. It uses a dimensional-wise attention layer, a multi-layer diffusion graph convolutional network (M-DGCN), and a time-decay LSTM to learn the latent representations from the three kinds of inputs, respectively. (b) It learns to discriminate the role of three-views and the importance of users in identifying misinformation. (c) Finally, we concatenate the learned features with text content to perform classification.

7.2.3.1 Learning the User Profile-View

User profiles have been demonstrated to be strong indicators when detecting rumors [26, 83]. The user profile characteristics are either explicit (e.g., username and geolocations) or implicit (e.g., gender and age). However, accessing the implicit features may not always be feasible due to the privacy concerns of many OSNs. Therefore, we consider the following eight explicit features, grouped in two major categories, which can be typically accessed in most OSNs:

- **Profile-Related features** include five basic user description fields: the screen name that the user identify herself; the user's self description; the attribute indicating whether the account has been verified by the platform; the geographical location of the user; and the UTC time that the user account was created on the social platform.
- **Influence-Related features** include three attributes describing user activities and social relations: the number of posts issued by the user, the number of followers, and the mutual follower-ship.

For each user u_j in a tweet m_i, we concatenate the profile characteristics into one feature vector, and then form the user characteristic matrix $\mathbf{U}_i \in \mathbb{R}^{|U_i| \times d_{user}}$ by

concatenating all user vectors for the users involved in spreading the tweet.

To provide explanations on which characteristics are useful for rumor detection, we design a dimensional-wise attention layer to assign weights to each dimension of user profiles. Its aim is to learn how to discriminate the importance of different characteristics. First, we expand $\mathbf{U}_i$ as a sequence of 1-dimensional "channels" for the features, i.e., $\mathbf{U}_i \in \mathbb{R}^{|U_i| \times 1 \times d_{user}}$, where $|U_i|$, 1 and d_{user} can be regarded as the height, width and channel of an image (similarly to the channels for each of the primitive colors – red, green and blue – in image processing). Then, we use a global average pooling (GAP) to aggregate the global information into a dimensionality-wise descriptor $\mathbf{z} \in \mathbb{R}^{d_{user}}$, where $\mathbf{z} = \frac{1}{|U_i| \times 1} \sum_{h=1,w=1}^{|U_i|,1} \mathbf{U}_i(h, w)$. To capture the dimensional-wise dependencies, we employ two fully connected layers with non-linearity – i.e., dimensionality-reduction layer and dimensionality-increasing layer:

$$\begin{aligned} \mathbf{f}_{red} &= \tanh(\mathbf{W}_1 \mathbf{z} + \mathbf{b}_1), \\ \mathbf{f}_{inc} &= \text{softmax}(\mathbf{W}_2 \mathbf{f}_{red} + \mathbf{b}_2), \end{aligned} \tag{7.14}$$

where $\mathbf{W}_1 \in \mathbb{R}^{\frac{d_{user}}{r} \times d_{user}}$ and $\mathbf{W}_2 \in \mathbb{R}^{d_{user} \times \frac{d_{user}}{r}}$ are parameter matrices, $\mathbf{b}_1 \in \mathbb{R}^{\frac{d_{user}}{r}}$ and $\mathbf{b}_2 \in \mathbb{R}^{d_{user}}$ are biases, and r is the reduction ratio. Thus, the final output of the user profile-view becomes:

$$\mathbf{H}_i^{User} = \mathbf{U}_i \mathbf{f}_{inc} + \mathbf{U}_i, \tag{7.15}$$

where $\mathbf{H}_i^{User} \in \mathbb{R}^{|U_i| \times d_{user}}$, $\mathbf{f}_{inc}$ denotes the attention score allocating different importance to each dimension of $\mathbf{U}_i$ through the multiplication operation, i.e., $\mathbf{U}_i \mathbf{f}_{inc}$. The operation of plus $\mathbf{U}_i$ is borrowed from the idea of skip connections [66].

The objective of dimensional-wise attention layer is to obtain a new user characteristic matrix through correlation training between the user profile's different characteristics by assigning different dimensions of the matrix with the different weights during training the model. In general, the contributing characteristics would be strengthened. Since the trivial characteristics should be weakened, we can also reduce the noise brought by non-critical characteristics, thereby improving the accuracy of the detection task. This effect is especially valuable for early-stage rumor detection. For example, when the number of participating users and the corresponding profiles are limited, it is particularly important to encourage the fundamental characteristics to explain rumor identification decisions. We will provide visual explanations in Sec. 7.2.4.

7.2.3.2 Learning the User Structural-View

The structural information of users who participate in spreading a tweet is extracted from the diffusion graph, which aims to capture the degree of connection, similarity, distance, and even community, etc., between users [99]. Inspired by the recent successes of network representation learning methods in processing graph-structured

data [41, 61, 115, 173], we define a multi-layer diffusion graph convolutional network (M-DGCN) as user structural-view encoder, in which the propagation rule of diffusion convolutional network is defined as:

$$\mathbf{H}^{(l+1)} = \sigma((\theta_O(\mathbf{D}_O^{-1}\mathbf{A}) + \theta_I(\mathbf{D}_I^{-1}\mathbf{A}^T))\mathbf{H}^{(l)}), \tag{7.16}$$

where θ_O and θ_I are filter parameters; $\mathbf{D}_O^{-1}\mathbf{A}$ and $\mathbf{D}_I^{-1}\mathbf{A}^T$ are transition matrices of the forward diffusion process and the reverse one, respectively – $\mathbf{D}_O$ and $\mathbf{D}_I$ represent out-degree diagonal matrix and in-degree diagonal matrix, respectively; $\sigma(\cdot)$ denotes activation function, i.e., ReLU$(\cdot)$ here; $\mathbf{H}^{(l)} \in \mathbb{R}^{|U|\times F}$ is the matrix of activation in the l-th layer – $|U|$ is the number of users in the diffusion graph and F is the dimension of the output. The difference between our M-DGCN and previous graph convolutional network [41, 61] is that the Chebyshev kernel in M-DGCN is equal to 1, whereas we stack a couple of such layers to aggregate the information from the distant nodes rather than the K-localized convolutions. In this layer, the initial input $\mathbf{H}^{(0)}$ is obtained from a pre-trained network embedding layer which maps a user u_j to it's D-dimensional representation $\mathbf{g}_j \in \mathbb{R}^D$, which allows the varying-size diffusion networks learning.

In order to reduce over-fitting for diffusion convolutional network, we employed a recently developed technique *DropEdge* (cf. [165]) for robust structural-view learning. That is, we randomly drop edges from the input diffusion graphs to generate different copies with a certain ratio in each training epoch. More specifically, suppose the total number of edges in the diffusion graph is $|E|$ and the dropping rate is r_{drop}. The adjacency matrix after dropout is computed as $\hat{\mathbf{A}} = \mathbf{A} - \mathbf{A}_{drop}$, where $\mathbf{A}_{drop}$ is the matrix constructed using $|E| \times r_{drop}$ edges randomly sampled from the original edge set E. After the diffusion convolutional layer, the diffusion graph G_i is represented as a vector matrix $\mathbf{H}_i^{Stru} \in \mathbb{R}^{|U_i|\times d_{stru}}$.

The structural-view $\mathbf{H}_i^{Stru}$ learned through M-DGCN represents the role of a node (i.e., a user) in the information spreading. M-DGCN not only models the propagation direction of information between spreaders but also aggregates high-order structural details, including the cascade virality, spreading patterns, etc., which may facilitate the rumor identification. We note that in [84] it has been demonstrated that the rumors have similar propagation patterns.

7.2.3.3 Learning the User Temporal-View

Users' engagement time and the sequential patterns of retweets also play an essential role in detecting rumors [33, 100]. We capture this view of users based on the diffusion path. Each user in the diffusion path would be assigned two types of embeddings: a static-embedding and a dynamic-embedding.

- **Static-embedding** refers to the relative position j $(1 \leq j \leq |U_i|)$ for each user u_j in the sequence. We encode this information based on the chronological

order of retweet times, and the users with the same retweet time will have the same position embedding. Inspired by the self-attention [131], we obtain the static-embedding $\mathbf{e}_j^s$ using a positional-encoding technique based on sine and cosine functions of frequencies:

$$\begin{aligned} \mathbf{PE}(j)_{2d} &= \sin(j/10000^{2d/d_e}), \\ \mathbf{PE}(j)_{2d+1} &= \cos(j/10000^{2d/d_e}) \end{aligned} \tag{7.17}$$

where d_e is an adjustable dimension and $1 \leq d \leq d_e/2$ denotes the dimension index in $\mathbf{e}_j^s$. The basic idea of this choice is to allow the model attending the relative position of the users. For details of this formula, refer to [131].

- **Dynamic-embedding** initializes user representations as one-hot vector $\mathbf{q} \in \mathbb{R}^N$, where N denotes the total number of users in the dataset. All users are associated with a specific embedding matrix $\mathbf{E} \in \mathbb{R}^{N \times d_e}$, where d_e is an adjustable dimension. Matrix $\mathbf{E}$ converts each user u_j into a unique representation vector as $\mathbf{e}_j^d = \mathbf{qE}, \mathbf{e}_j^d \in \mathbb{R}^{d_e}$. In this way, the user embedding matrix $\mathbf{E}$ can be learned during training, supervised by the downstream task, i.e., rumor detection in this work.

Subsequently, we use an RNN model (e.g., LSTM [40]) to learn the temporal dependence of the diffusion. However, the influence of retweet users will diminish over time, and the "vanilla LSTM" is not capable of capturing this time-decay effect of information diffusion. To address this issue, we introduce a time-gate inspired by [174] into the LSTM.

The time-gate not only controls the influence of $\mathbf{x}_j$ – the combination of static and dynamic embeddings – on the current step, but also caches the time interval between consecutive retweets to model the time-decay effect. Specifically, a time-decay LSTM unit takes: $\mathbf{x}_j$, previous hidden state $\mathbf{h}_{j-1}$, and time interval Δ_{t_j} as inputs – and outputs the current hidden state $\mathbf{h}_j$ using:

$$\begin{aligned} \mathbf{x}_j &= \mathbf{e}_j^s + \mathbf{e}_j^d, \\ \mathbf{i}_j &= \sigma\left(\mathbf{W}_{xi}\mathbf{x}_j + \mathbf{U}_{hi}\mathbf{h}_{j-1} + \mathbf{b}_i\right), \\ \mathbf{f}_j &= \sigma\left(\mathbf{W}_{xf}\mathbf{x}_j + \mathbf{U}_{hf}\mathbf{h}_{j-1} + \mathbf{b}_f\right), \\ \mathbf{T}_j &= \sigma\left(\mathbf{W}_{xT}\mathbf{x}_j + \tanh\left(\mathbf{W}_{tt}\Delta t_j\right) + \mathbf{b}_T\right), \\ \mathbf{o}_j &= \sigma\left(\mathbf{W}_{xo}\mathbf{x}_j + \mathbf{U}_{ho}\mathbf{h}_{j-1} + \mathbf{W}_{to}\Delta t_j + \mathbf{b}_o\right), \\ \widetilde{\mathbf{c}}_j &= \tanh\left(\mathbf{W}_{xz}\mathbf{x}_j + \mathbf{U}_{hz}\mathbf{h}_{j-1} + \mathbf{b}_z\right), \end{aligned} \tag{7.18}$$

where $\sigma(\cdot)$ is the sigmoid function; $\mathbf{i}_j, \mathbf{f}_j, \mathbf{T}_j, \mathbf{o}_j, \widetilde{\mathbf{c}}_j, \mathbf{b}_*$ are the input gate, forget gate, time gate, output gate, new candidate vector and bias vector, respectively. The matrices $\mathbf{W}_{x*} \in \mathbb{R}^{d_e \times d_{temp}}$, $\mathbf{W}_{t*} \in \mathbb{R}^{1 \times d_{temp}}$ and $\mathbf{U}_{h*} \in \mathbb{R}^{d_h \times d_{temp}}$ represent the different gate parameters. In particular, the memory cell $\mathbf{c}_j$ is updated by replacing

the existing memory unit with a new cell $\mathbf{c}_j$ as:

$$\mathbf{c}_j = \mathbf{f}_j \odot \mathbf{c}_{j-1} + \mathbf{i}_j \odot \mathbf{T}_j \odot \widetilde{\mathbf{c}}_j, \tag{7.19}$$

where $\odot$ denotes the element-wise multiplication. The hidden state is then updated by:

$$\mathbf{h}_j = \mathbf{o}_j \odot \tanh\left(\mathbf{c}_j\right), \tag{7.20}$$

Finally, the representation vector for the temporal-view is $\mathbf{H}_i^{Temp} = \left\{\mathbf{h}_j^{Temp} | j \in [1, |U_i|]\right\}$, where $\mathbf{H}_i^{Temp} \in \mathbb{R}^{|U_i| \times d_{temp}}$. Note that the temporal-view of the user obtained by the time-decay LSTM reflects each user's influence on the subsequent participators in the message diffusion.

7.2.3.4 View-Wise Attention for View-Level Feature Fusion

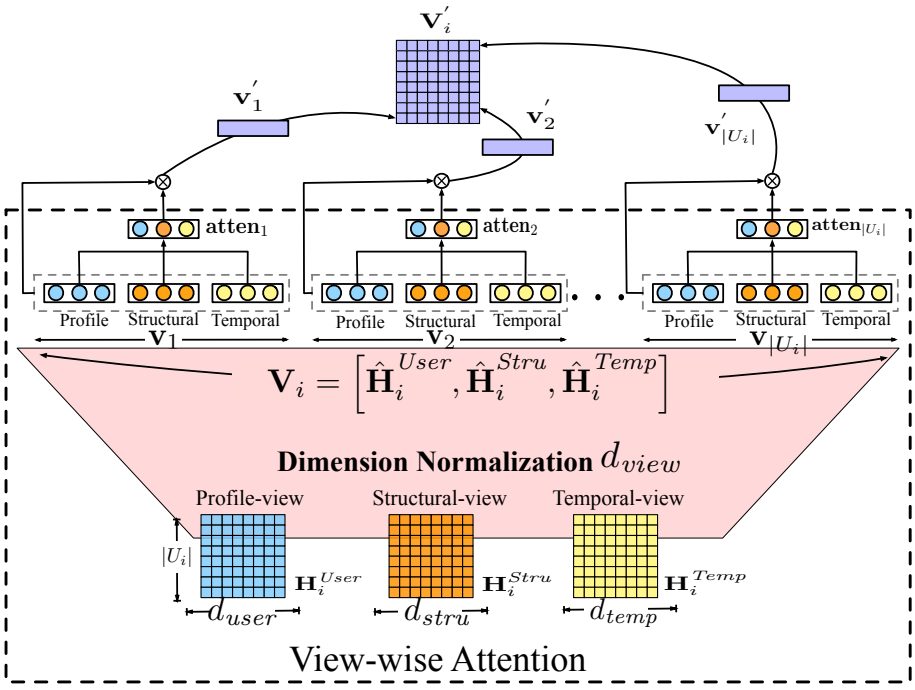


Figure 7.12: Illustration of view-wise attention.

After obtaining the latent representation for each view, we need to fuse the multi-view features. Rather than directly concatenating different aspects, as often done in the existing solutions [91, 94, 175], we present a method to capture the differences between different views. The primary motivation stems from the observation that various views are not equally relevant in the task of rumor identification. Towards that, we propose a view-wise attention layer to prioritize the fundamental views for each user.

As depicted in Figure 7.12, the view-wise attention layer takes profile-view, structural-view, and temporal-view as input and generates the attention score for each view at

the user-level. Specifically, it first normalizes the dimensions of the three views' vectors as d_{view} via a fully connected layer. Let $\mathbf{V}_i = \left[\hat{\mathbf{H}}_i^{User}, \hat{\mathbf{H}}_i^{Stru}, \hat{\mathbf{H}}_i^{Temp}\right]$ denote the feature set after dimension normalization. Each vector $\mathbf{v}_j = \left[\hat{\mathbf{h}}_j^{User}, \hat{\mathbf{h}}_j^{Stru}, \hat{\mathbf{h}}_j^{Temp}\right] \in \mathbf{V}_i$ represents a view feature set for a specific user u_j engaged in spreading tweet m_i. Then, the view-wise attention layer calculates the attention score $\mathbf{atten}_j \in \mathbb{R}^{1\times 3}$ for each view of the user-level feature set $\mathbf{v}_j \in \mathbb{R}^{d_{view}\times 3}$ as:

$$\bar{\mathbf{v}}_j = \tanh\left(\mathbf{W}_v \cdot \mathbf{v}_j\right), \tag{7.21}$$

$$\mathbf{atten}_j = \mathrm{softmax}\left(\mathbf{w}_v^T \cdot \bar{\mathbf{v}}_j\right), \tag{7.22}$$

where $\mathbf{W}_v \in \mathbb{R}^{d_{view}\times d_{view}}$, $\mathbf{w}_v \in \mathbb{R}^{d_{view}}$ are learnable projection parameters during training, $\bar{\mathbf{v}}_j = \left[\bar{\mathbf{h}}_j^{User}, \bar{\mathbf{h}}_j^{Stru}, \bar{\mathbf{h}}_j^{Temp}\right]$. Here, the view-wise attention layer first computes the hidden representation of $\mathbf{v}_j$ through multiplying it with $\mathbf{W}_v$ to get $\bar{\mathbf{v}}_j$, which is implemented with a fully connected layer without bias. It measures the weight of a view as the similarity of $\bar{\mathbf{h}}_j^*$ ($* \in \{User, Stru, Temp\}$) with a view-level context vector $\mathbf{w}_v$ and finally obtains a normalized weight through a softmax function. Each entry of $\mathbf{atten}_j$ represents an importance score for a specific view of user j.

Finally, the fused multi-view feature vector $\mathbf{v}_j^{'}$ for user u_j can be calculated as:

$$\mathbf{v}_j^{'} = \mathbf{atten}_j \cdot \mathbf{v}_j, \tag{7.23}$$

where $\mathbf{v}_j^{'} \in \mathbb{R}^{d_{view}}$. The fused multi-view feature vector for each user forms the multi-view matrix, denoted as $\mathbf{V}_i^{'} = \left\{\mathbf{v}_1^{'}, \mathbf{v}_2^{'}, \cdots, \mathbf{v}_{|U_i|}^{'}\right\}$, where $\mathbf{V}_i^{'} \in \mathbb{R}^{|U_i|\times d_{view}}$.

7.2.3.5 Capsule Attention for User-level Feature Fusion

Most of existing works [16, 100, 102] would directly use $\mathbf{V}_i^{'}$ for rumor detection. However, that does not properly discriminate different users, contrary to the fact that different users in a tweet propagation network may contribute differently to classifying the tweet. In our UMLARD, we introduce a capsule attention layer inspired by the recent success of capsule networks [133, 176, 177]. The Capsule network was first proposed in [133] and the main idea is to replace the scalar-output feature detectors in traditional neural networks with vector-output capsules, and train the model by the dynamic routing algorithm. It can be regarded as a parallel attention mechanism that allows each underlying capsule to attend to higher capsules at different importance.

In UMLARD, the capsule attention chooses the most related underlying vectors dynamically to form the only upper capsule via an unsupervised routing-by-agreement mechanism, which also avoids the intensive computation raised by a huge amount of parameters used in multi-layer attention. More precisely, in the n-th iteration,

the upper capsule $\mathbf{s}_{in}$ is calculated by:

$$\mathbf{s}_{in} = \sum_{j}^{|U_i|} \mathbf{a}_j \hat{\mathbf{v}}_j, \hat{\mathbf{v}}_j = \mathbf{W}\mathbf{v}_j^{'}, \tag{7.24}$$

where the coupling coefficient $\mathbf{a}_j$ indicates the contributions of a user capsule to the upper capsule – namely, the attention score of each user. $\mathbf{W} \in \mathbb{R}^{d_{view} \times d_{caps}}$ is the transform matrix that guarantees the feature representation ability of the center vector after clustering, and identifies the order of input features. Note that before the last iteration we add a normalization $\widetilde{\mathbf{s}}_{in} = \mathbf{s}_{in} / \|\mathbf{s}_{in}\|$ in $\mathbf{s}_{in}$ to overcome the information loss caused by the original CapsAtt [176].

The coupling coefficient $\mathbf{a}_j \in \mathbb{R}^{|U_i| \times 1}$ is determined by a "routing softmax" whose initial logit is denoted as $\mathbf{b}_j$, where $\mathbf{b}_j$ is the log prior probability that the j-th user capsule should be coupled to the upper capsule $\mathbf{s}_{in}$. The coefficient is calculated by:

$$\mathbf{a}_j = \frac{\exp(\mathbf{b}_j)}{\sum_k^{|U_i|} \exp(\mathbf{b}_k)}, \tag{7.25}$$

The log prior is initialized with zero and then updated by adding agreements between the user capsule and the upper capsule:

$$\mathbf{b}_j = \mathbf{b}_j + \hat{\mathbf{v}}_j \cdot \widetilde{\mathbf{s}}_{in}, \tag{7.26}$$

These agreements are added to log priors after each routing, i.e., the output capsule $\mathbf{s}_{in}$ represents the feature matrix after correlation learning, which can be easily coupled into the model for downstream tasks, in our case the rumor detection.

7.2.3.6 Tweet Content Representation

Tweet content is one of the most important features in rumor detection [23, 29, 78], and has been extensively studied in the literature [16, 33, 80, 102, 178], where various natural language processing (NLP) techniques have been exploited for learning informative signals from the textual content. Though content learning is beside the scope of this thesis, we describe a simple CNN layer for text representation learning from the input of word embedding matrix for completeness. A single CNN layer is denoted as:

$$\mathbf{h}_m = \sigma(\mathbf{W} * \mathbf{w}_{m:m+d-1}), \tag{7.27}$$

where $\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_{L-d+1}\}$ is the extracted feature map, and $\mathbf{W} \in \mathbb{R}^{d \times d_{word}}$ is the convolutional kernel with d as size of the receptive filed, and σ as non-linearity. Then max-pooling operation is used over the feature map to generate the output representation $\hat{\mathbf{H}}$. In our work, we use multiple CNN layers with different receptive fields to obtain multiple features, and then concatenate all outputs to form the tweet content representation $\mathbf{H}_i^{Text}$.

7.2.3.7 Training Objective

Finally, we concatenate content representation $\mathbf{H}_i^{Text}$ and capsule attention $\mathbf{s}_{in}$ to merge the information as:

$$\mathbf{H}_i^{Rumor} = \text{concat}(\mathbf{H}_i^{Text}, \mathbf{s}_{in}) \tag{7.28}$$

which is subsequently used for predicting the label $\hat{\mathbf{y}}_i$ of tweet m_i via a fully connected layer and the softmax function:

$$\hat{\mathbf{y}}_i = \text{softmax}\left(\text{FC}\left(\mathbf{H}_i^{Rumor}\right)\right). \tag{7.29}$$

We train all the parameters by minimizing the *cross-entropy* of the predictions $\hat{\mathbf{Y}}$ and the ground truth labels $\mathbf{Y}$ as:

$$\mathcal{L}(\mathbf{Y}, \hat{\mathbf{Y}}) = -\sum_{i \in |M|} \mathbf{y}_i \log \hat{\mathbf{y}}_i + \lambda \left\|\Theta\right\|_2^2, \tag{7.30}$$

where $\|\Theta\|_2^2$ is the L_2 regularizer over all the model parameters Θ, and λ is the trade-off coefficient. In this work, we use *RAdam* [140] as optimizer. The whole training process of UMLARD is outlined in Algorithm 10.

Algorithm 10: Training of UMLARD.

Input: A set of tweets $M = \left\{m_i\right\}_{i=1}^{|M|}$, each tweet $m_i = \left\{G_i, \mathcal{P}_i, \mathbf{U}_i, \mathbf{C}_i\right\}$.
Output: Predicted labels $\hat{\mathbf{Y}}$ for all tweets.

repeat
  for m_i in a batch **do**
    Profile-view learning: $\mathbf{H}_i^{User} \leftarrow \mathbf{U}_i$ via Eq.(7.14) and Eq.(7.15);
    Structural-view learning $\mathbf{H}_i^{Stru} \leftarrow G_i$ via Eq.(7.16);
    Temporal-view Learning $\mathbf{H}_i^{Temp} \leftarrow \mathcal{P}_i$ via Eq.(7.18) - Eq.(7.20);
    Content representation: $\mathbf{H}_i^{Text} \leftarrow \mathbf{C}_i$ via Eq.(7.27);
    Nomalize dimensions:
    $\mathbf{V}_i = [\hat{\mathbf{H}}_i^{User}, \hat{\mathbf{H}}_i^{Stru}, \hat{\mathbf{H}}_i^{Temp}] \leftarrow [\mathbf{H}_i^{User}, \mathbf{H}_i^{Stru}, \mathbf{H}_i^{Temp}]$;
    View-wise attention learning: $\mathbf{V}_i^{'} \leftarrow \mathbf{V}_i$ via Eq.(7.21) to Eq.(7.23);
    Capsule attention learning: $\mathbf{s}_{in} \leftarrow \mathbf{V}_i^{'}$ via Eq.(7.24);
    Merge $\mathbf{H}_i^{Text}$ and $\mathbf{s}_{in}$ via Eq.(7.28);
    Estimate the probability $\hat{\mathbf{y}}_i$ via Eq.(7.29);
    Compute loss $\mathcal{L}(\mathbf{y}_i, \hat{\mathbf{y}}_i)$, via Eq.(7.30);
    Update parameters using RAdam.
  end for
until convergence;

7.2.3.8 Computational Complexity

We finalize this section with a discussion of the computational complexity of UMLARD, analyzed in two categories.

— *Complexity of multi-view representation learning* is influenced by four main components:

(1) As for **profile-view** that only uses dimensional-wise attention to allocate varying weights to each dimension, the computational complexity stems from the two fully connected layers, i.e., $\mathcal{O}(2d_{user}^2/r)$. Because the dimension of user characteristic d_{user} is very small, this computational cost is typically negligible.

(2) We use a multi-layer diffusion convolutional network for the **structural-view** learning (cf. Eq.(7.16)), which can be decomposed into two parts with the same time complexity, i.e., $\mathbf{D}_I^{-1}\mathbf{A}$ and $\mathbf{D}_O^{-1}\mathbf{A}^T$. Since the two matrices are very sparse, the time complexity is $\mathcal{O}(|E|)$, i.e., linear with the number of edges. Specifically, in a two-layer M-DGCN, the computational complexity is $\mathcal{O}(|E|DF_1F_2)$, where D, F_1 and F_2 are the input feature size, and the hidden size for the first and the last M-DGCN layer, respectively.

(3) The **temporal-view** is learned through a time-decay LSTM. The computational complexity of original LSTM per time step is $\mathcal{O}(1)$ due to LSTM is local in space and time [40]. Compared with LSTM, the only difference of our time-decay LSTM is an extra time-gate that controls the influential decreasing with time. This operator introduced extra parameters that requires $4(d_e d_{temp} + d_{temp}^2 + d_{temp}) + d_e d_{temp} + 3d_{temp}$ complexity. Besides, the dynamic embedding in UMLARD needs $N \times d_e$ parameters.

(4) For the **source tweet** representation learning, the CNN layers have the time complexity of $\mathcal{O}(\sum_{l=1}^{L}(M_l^2 K_l^2 C_{l-1} C_l))$, where L is the total number of CNN layers; K_l C_{l-1}, C_l are kernel size, input channel number and output channel number for l-th layer; output size is $M_l = (X_l - K_l)/Stride + 1$ and X_l is the input feature size. Overall, this component requires $\sum_{l=1}^{L}(K_l^2 C_{l-1} C_l)$ parameters.

— *Complexity of fusion layers.* In the hierarchical fusion layers, the time and space complexities of both view-wise attention and capsule attention are related to the input and output dimensions of the latent variables. In view-wise attention, it introduces $d_{view} \times d_{view} + |U_i| \times d_{view}$ parameters. As for the capsule attention layer, the parameter size is $d_{view} \times d_{caps}$, where d_{view} and d_{caps} represent view size and capsule size, respectively.

7.2.4 Evaluation

We now present the findings from our experimental evaluations. We compare the performance of our UMLARD with the state-of-art baselines on rumor detection, and we also investigate the effects of different components by comparing several variants of UMLARD.

Specifically, we would aim at providing quantitative characterization of the following research-related questions:

Table 7.11: Statistics of the datasets.

Statistic	Twitter15	Twitter16	Weibo
# source tweets	1,482	809	4,664
# users	477,009	286,657	2,746,818
# non-rumors	370	199	2,351
# false-rumors	369	205	2,313
# true-rumors	372	207	–
# unverified-rumors	371	198	–
Max. # retweets	2989	3058	59,318
Min. # retweets	55	73	10
Avg. # retweets	398	422	816
Avg. # time length	1,268 Hours	828 Hours	1,811 Hours

- **Q1**: How does UMLARD perform on rumor detection compare with the state-of-the-art baselines?
- **Q2**: What is the effect of each component of UMLARD?
- **Q3**: Can UMLARD detect rumors in early stages of their propagation?
- **Q4**: Can UMLARD explain the model behavior and the predicted results?

7.2.4.1 Experimental Settings

Following is the description of the main aspects of our experimental setup.

1) Datasets: We conduct our experiments on the three real-world datasets[1]: *Twitter15*, *Twitter16* [84] and *Weibo* [33]. In each dataset, a group of widespread source tweets along with their propagation threads with time stamps are provided. We construct propagation paths and diffusion networks from the propagation threads, which are also used for user temporal-aspect embedding and user structural-aspect embedding.

Different from the experiment settings in previous PLRD (Section 7.1), in this section, we consider the two Twitter datasets as multi-class datasets, i.e., each source tweet is annotated with one of the four class labels, i.e., *non-rumor*, *false-rumor*, *true-rumor*, and *unverified-rumor*, while the Weibo dataset contains binary labels: *false-rumor*, *non-rumor* – the labeling rules follow the method in [33]. The statistics of the three datasets are shown in Table 7.11. We extract the same user characteristics as PLRD for both Twitter and Weibo dataset, as shown in Table 7.2. Specifically, as for the Weibo dataset, we directly extract these eight characteristics from the JSON files in the original dataset. And the way to split the dataset into training, validation, and testing set follows the same setting in PLRD (see Section 7.1.4.2).

[1]https://www.dropbox.com/s/7ewzdrbelpmrnxu/rumdetect2017.zip?dl=0

2) Baselines: We compare UMLARD with following state-of-the-art rumor detection baseline models:

- **DTC** [23], **SVM-TS** [85], **GRU** [108], **TD-RvNN** [34], **PPC_RNN+CNN (PPC)**, **Bi-GCN** [16], **GCAN** [102], the description of these baseline methods can be found in Section 7.1.4.1. We also compared UMLARD with **PLRD** [45] (Section 7.1).
- **SVM-RBF** [86]: A support vector machine (SVM) based model that uses radius basis function (RBF) as the kernel and leverages the handcrafted features of posts for rumor detection.
- **PLAN** [179]: A hierarchical token- and post-level attention model for rumor detection, which models pairwise interactions between tweets via the self-attention mechanism.
- **Bi-GCN-U** [16]: A variant of Bi-GCN, which uses user profile characteristics to replace the comment features.
- **STS-NN** [180]: A rumor detection model based on spatial-temporal neural networks. It treats the spatial structure and temporal structures as a whole to learn a fine-grained rumor representation.
- **GCAN-G** [102]: A variant of GCAN, which uses the diffusion graph to replace the user similarity graph.
- **RDEA** [103]: A self-supervised rumor detection model. On the basis of Bi-GCN [16], RDEA improves the rumor representations and alleviates limited data issues through event augmentation and contrastive learning.

3) Implementation details: We implement DTC with Weka[1], SVM-based models with scikit-learn[2], and other neural network-based models with Tensorflow[3]. All baselines follow the parameter settings in the original papers. For UMLARD, the learning rate is initialized at 0.001 and gradually decreases as the training proceeds. We use word2vec to initialize the word embeddings with $d_{word} = 300$ dimensions, and the convolution kernel size is set to $[3, 4, 5]$, and per size with 100 kernels. The embedding size for structural view d_{stru} and temporal view d_{temp} of users are both set to 64; the view size d_{view} is also set to 64, as is the the capsule size; and the iteration number varies between 2 and 4. The batch size is 64; and the rate of dropout in the main neural networks is 0.5; the dropout rate in DropEdge is 0.2. The training process is iterated upon for 200 epochs, but would be stopped earlier if the validation loss does not decrease after 10 epochs.

4) Evaluation metrics: We use accuracy (ACC) and F-measure (F1) as the evaluation protocols to measure the models' performance. Specifically, ACC measures

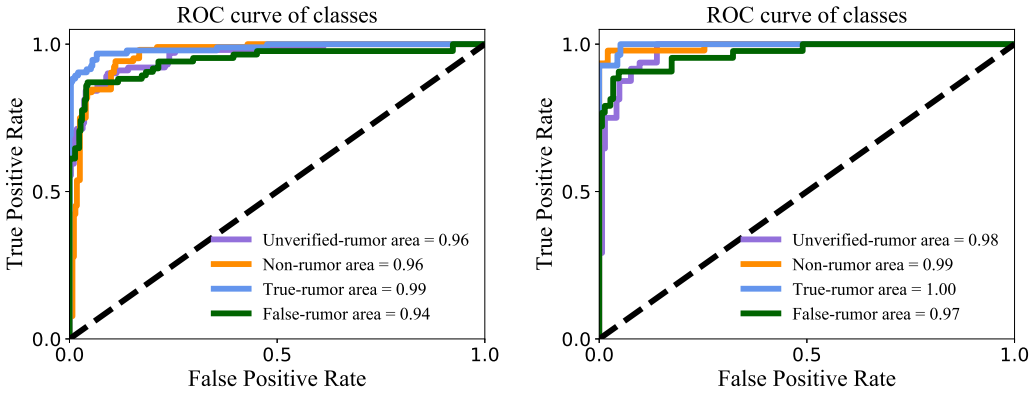
[1]https://www.cs.waikato.ac.nz/ml/weka/
[2]https://scikit-learn.org/
[3]https://www.tensorflow.org/

the proportion of correctly classified tweets, while F1 is the harmonic mean of the precision and recall values averaged across four classes. As for the Weibo dataset, we also report the precision and recall results.

Table 7.12: Overall performance comparison of rumor detection on Twitter15 (the observation window is set to the previous 40 retweets). "UR": unverified-rumor; "NR": non-rumor; "TR": true-rumor; "FR": false-rumor. The best method is shown in **bold**, and the second best is shown as underlined. A paired t-test is performed and $*$ indicates a statistical significance $p < 0.05$ compared to the best baseline method (RDEA).

Model	Twitter15				
	ACC.	F1			
		UR	NR	TR	FR
DTC	0.454	0.415	0.733	0.317	0.355
SVM-RBF	0.318	0.218	0.225	0.455	0.082
SVM-TS	0.544	0.483	0.796	0.404	0.472
GRU	0.646	0.608	0.592	0.792	0.574
TD-RvNN	0.723	0.654	0.682	0.821	0.758
PPC	0.697	0.689	0.760	0.696	0.645
PLAN	0.787	0.775	0.7754	0.768	0.807
Bi-GCN	0.829	0.752	0.772	0.885	0.847
Bi-GCN-U	0.778	0.764	0.741	0.853	0.752
GCAN	0.808	0.690	**0.930**	0.812	0.758
GCAN-G	0.750	0.731	0.754	0.823	0.678
STS-NN	0.808	0.779	0.786	0.860	0.808
RDEA	0.835	0.819	0.786	0.887	0.837
PLRD	0.622	0.519	0.832	0.438	0.596
UMLARD	**0.857***	**0.835***	0.840*	**0.906***	**0.848***

7.2.4.2 Overall Performance (Q1)



(a) Twitter15 **(b)** Twitter16

Figure 7.13: ROC curve comparison for each information type. Area under curve of ROC (AUC) is presented after the legend.

Table 7.12, Table 7.13, and Table 7.14 report the performance comparison among

Table 7.13: Overall performance comparison of rumor detection on Twitter16 (the observation window is set to the previous 40 retweets). "UR": unverified-rumor; "NR": non-rumor; "TR": true-rumor; "FR": false-rumor. The best method is shown in **bold**, and the second best is shown as underlined. A paired t-test is performed and $*$ indicates a statistical significance $p < 0.05$ compared to the best baseline method (RDEA).

Model	Twitter16				
	ACC.	F1			
		UR	NR	TR	FR
DTC	0.465	0.403	0.643	0.419	0.393
SVM-RBF	0.321	0.419	0.037	0.423	0.085
SVM-TS	0.574	0.526	0.755	0.571	0.420
GRU	0.633	0.686	0.593	0.772	0.489
TD-RvNN	0.737	0.708	0.662	0.835	0.743
PPC	0.702	0.608	0.711	0.816	0.664
PLAN	0.799	0.779	0.754	0.836	0.821
Bi-GCN	0.837	0.818	0.772	0.885	0.847
Bi-GCN-U	0.786	0.733	0.783	0.875	0.767
GCAN	0.765	0.784	0.848	0.678	0.754
GCAN-G	0.721	0.642	0.690	0.799	0.732
STS-NN	0.829	0.838	0.775	0.899	0.809
RDEA	0.848	0.868	0.729	0.922	0.823
PLRD	0.646	0.618	0.698	0.609	0.445
UMLARD	**0.901***	**0.822***	**0.965***	**0.960***	**0.855***

UMLARD and baselines on three datasets, from which we have the following observations:

O1: Feature-based approaches such as SVM-TS, SVM-RBF, and DTC perform poorly. These methods use hand-crafted features based on the overall statistics of tweets, but are not sufficient to capture the generalizable features associated with tweets and the process of information diffusion. Notably, SVM-RBF performs worse than the other two methods on two Twitter datasets. However, it achieves the best performance among the feature-based modes on Weibo dataset, because it selects the features based on Weibo that are hard to be generalized to other social platforms such as Twitter. SVM-TS achieves relatively better performance because it utilizes an extensive set of features and primarily focuses on retweets' temporal traits.

O2: Deep learning-based models perform significantly better than feature-based methods. As the first work exploiting RNN for efficient rumor detection, GRU only relies on temporal-linguistics of the repost sequence while ignoring other useful information such as diffusion structures and user profiles. TD-RvNN and PPC_RNN+CNN outperform GRU, which indicates the effectiveness of modeling the propagation structure and temporal information in rumor detection. The performance of PLAN slightly exceeds TD-RvNN and PPC_RNN+CNN, because it still mainly focuses on textual information and ignores structural features of rumor propagation.

Table 7.14: Overall performance comparison of rumor detection on Weibo (the observation window is set to the previous 40 retweets). "NR": non-rumor; "FR": false-rumor. The best method is shown in **bold**, and the second best is shown as underlined. A paired t-test is performed and ** indicates a statistical significance $p < 0.01$ compared to the best baseline method (RDEA).

Model	Weibo						
	ACC	NR			FR		
		Prec.	Rec.	F1	Prec.	Rec.	F1
DTC	0.731	0.715	0.747	0.730	0.747	0.715	0.731
SVM-RBF	0.741	0.738	0.747	0.742	0.745	0.735	0.740
SVM-TS	0.780	0.801	0.753	0.780	0.762	0.808	0.784
GRU	0.762	0.803	0.715	0.757	0.728	0.809	0.767
TD-RvNN	0.832	0.832	0.812	0.821	0.821	0.861	0.841
PPC	0.845	0.870	0.810	0.839	0.810	0.883	0.844
PLAN	0.857	0.829	0.904	0.857	0.893	0.805	0.835
Bi-GCN	0.891	0.892	0.892	0.890	0.891	0.891	0.890
Bi-GCN-U	0.864	0.896	0.818	0.860	0.830	0.910	0.868
GCAN	0.880	0.911	0.861	0.885	0.866	0.929	0.896
GCAN-G	0.831	0.815	0.824	0.819	0.847	0.815	0.831
STS-NN	0.875	0.881	0.866	0.865	0.851	0.872	0.852
RDEA	0.911	0.902	0.923	0.907	**0.913**	0.899	0.901
PLRD	0.899	0.936	0.863	0.900	0.862	**0.946**	0.904
UMLARD	**0.928****	**0.942****	**0.965****	**0.924****	**0.894****	0.944**	**0.928****

O3: Bi-GCN, GCAN, STN-SS, and RDEA have considered structural or temporal information, and thus outperform other baselines. In particular, Bi-GCN constructs the diffusion graph based on user replies, i.e., the retweets with comments, which may not reflect the whole structure of rumor dispersion. In contrast, GCAN models the structural information from the user similarity matrix rather than propagation network. Therefore, according to the results, Bi-GCN performs much better than GCAN, because it takes the comments information into consideration. Besides, the bi-directional GCN is more effective in learning propagation structures than vanilla GCN used in GCAN. Although STS-NN extracts both structural and temporal features for rumor detection, STS-NN still performs worse than Bi-GCN, because it fails to discriminate the spatial structures and the temporal patterns. RDEA improves the performance of Bi-GCAN via introducing contrastive learning and event augmentations, which alleviate the influence of limited data issue. However, this method still faces the same problem as Bi-GCN, i.e., reply network is not enough to represent the full information diffusion process. Through comparing UMLARD with Bi-GCN-U and GCAN-G, we find that the performance of Bi-GCN-U and GCAN-G drops significantly. This result indicates that these methods heavily depend on the input features and are ineffective in extracting diffusion patterns as our method. PLRD does not perform very well at Twitter15 and Twitter16 while showing competitive results on the Weibo dataset, since the label of Twitter15 and Twitter16 becomes more fine-grained, and PLRD is not sensitive to the un-verified and true rumors.

O4: UMLARD consistently outperforms all other baselines across all datasets. Compare to the best baseline RDEA, UMLARD models rumor diffusion from multi-view perspective that allows the model to discriminate the importance of features and users in spreading the tweets. These results also validate one of our primary motivations, i.e., various features play different roles in spreading the rumors, and users are the main contributor to the misinformation propagation.

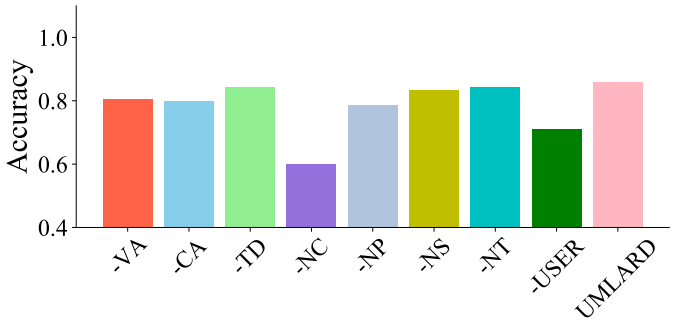
Finally, we scrutinize the performance of UMLARD on discriminating against the individual type of information on Twitter15 and Twitter16. Figure 7.13 plots the ROC curves of the model performance on four different kinds of tweets. We find that our model achieves the best identification results on true-rumors, which indicates that the characteristics of true-rumors are more distinctive from other types of messages. This result also implies that our model is more expressive on a binary classification task that only needs to classify tweets as rumors or truths (cf. the results on Weibo in Table 7.13). In practice, however, unverified-rumors and false-rumors are noisy signals that require careful treatment, which is a promising way of further improving the detection accuracy.

7.2.4.3 Ablation Experiments (Q2)

In this section, we conduct an ablation study to explore the effect of each component in UMLARD. Towards that, we derive the following variants of UMLARD:

- **-VA**: In -VA, ignores the different importance of different views, i.e., it removes the view-wise attention layer.
- **-CA**: In -CA, replaces the capsule attention layer with a fully connected layer.
- **-TD**: In -TD, neglects the time decay effect of retweet behaviors which is replaced by a vanilla LSTM [40] to learn sequential retweet behavior.
- **-NC**: In -NC, removes the content feature of the source tweet but keeps the temporal, profile, and structural features.
- **-NP**: In -NP, disregards the profile features of users but retains temporal, structural, and content features.
- **-NS**: In -NS, ignores the structural features of users but keeps temporal, profile, and content features.
- **-NT**: In -NT, ignores the temporal features of users but keep structural, profile, and content features.
- **-USER**: In -USER, ignores the user-aspect features (i.e., temporal, structural, and profile) that only retains the content feature.

Figure 7.14 illustrates the performance of the variants, where we can observe that:



(a) Twitter15

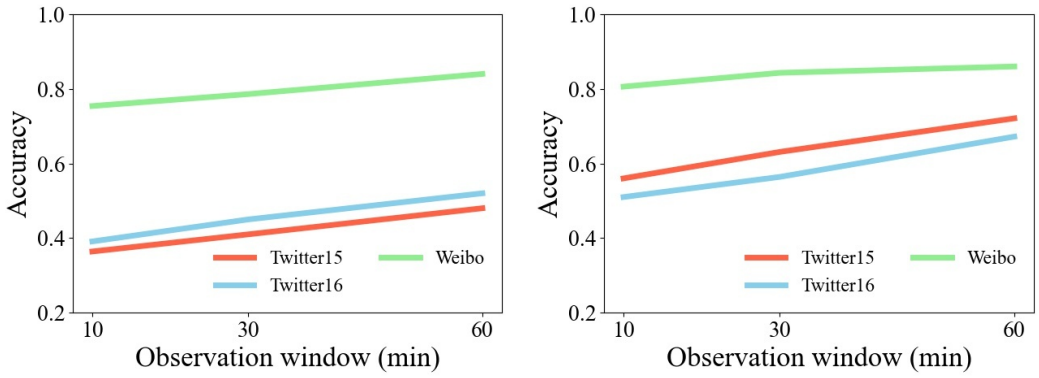
(b) Twitter16

(c) Weibo

Figure 7.14: Ablation study of UMLARD. Two attention mechanisms can significantly improve the detection performance by distinguishing the importance of features and users. Tweet content and profile information are two most informative features on rumor detection.

(1) The content of tweet (**-NC** and **-USER**) is still the most critical signal of discriminating rumors among various features. Without it, the model performance would significantly drop, as observed in many previous works [16, 102]. However, only based on the content feature is insufficient to develop an effective rumor detection model that can identify different types of rumors with high accuracy.

(2) Profile information (**-NP**) is another reliable indicator to detect the rumors because it is a straightforward but useful method to identify the users that spread the misinformation intentionally [26, 83].



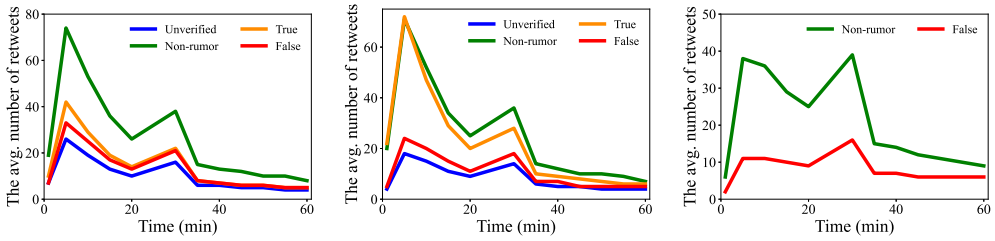
(a) Structural view **(b)** Temporal view

Figure 7.15: Study on structural and temporal view. We only use structural or temporal views to detect rumors in 10, 30, and 60 mins. The results demonstrate the importance of both views in rumor detection.

(3) Though both structural (**-NS**) and temporal information (**-NT**) are informative, they are not as important as contents of tweets and user profiles. This result also explains why the methods proposed in [100], and [16] do not show comparable performance as ours – the former mainly focuses on modeling the temporal information of retweets, whereas the latter one relies on graph neural networks to exploit the diffusion structures. We also conduct additional experiments to demonstrate the importance of structural and temporal features once the input contains enough information, especially in a binary classification task (e.g., Weibo). The results are shown in Figure 7.15. We find that as for Twitter datasets, the detection performance based on structural features grows slightly but is still not good enough as the type of rumors is fine-grained, making it challenging to learn discriminative structural features. As for the Weibo dataset, both structural and temporal features are helpful for rumor detection even in a short time.

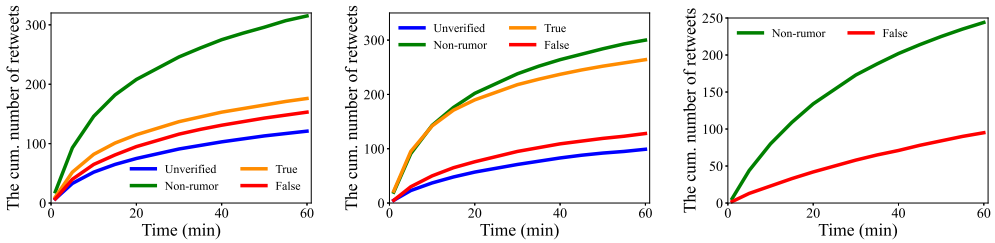
In order to demonstrate our findings in Figure 7.15, we conduct statistical analysis of the datasets and plot the temporal and structural propagation patterns in Figure 7.16–7.19. We find that the differences in temporal patterns are more obvious compared with the structural patterns. In addition, the differences between true and false rumors in Weibo are more significant than the discrepancy between the fine-grained types of rumors in Twitter datasets.

(4) The two attention mechanisms proposed in this work, i.e., view-wise attention (**-VA**) and capsule attention (**-CA**), play a crucial role on identifying the misinformation – the importance of which even exceed temporal features and diffusion patterns. This result also suggests that distinguishing the significance of different views of users can improve classification performance. Similarly, different users play different roles in spreading misinformation, e.g., users may intentionally mislead others or unknowingly retweet doubtful news. However, examining users' purposes is beyond the scope of this work and is left as our future work.



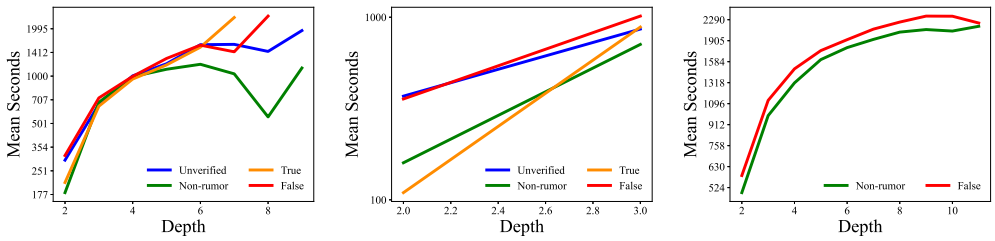
(a) Twitter15 **(b)** Twitter16 **(c)** Weibo

Figure 7.16: The average number of retweets for different types of rumors at different timestamps.



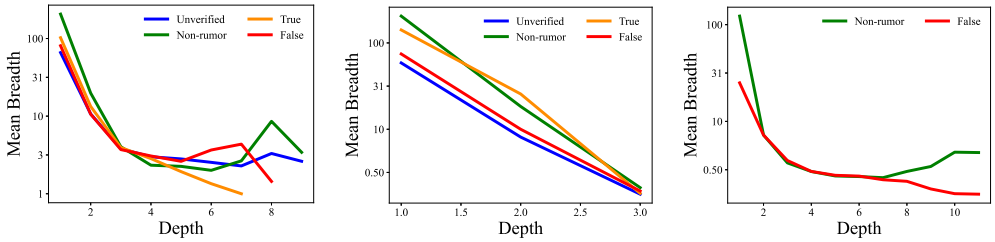
(a) Twitter15 **(b)** Twitter16 **(c)** Weibo

Figure 7.17: The cumulative number of retweets for different types of rumors at different timestamps.



(a) Twitter15 **(b)** Twitter16 **(c)** Weibo

Figure 7.18: The average time (in seconds) required to reach the same network depth. The observation window 60 minutes.

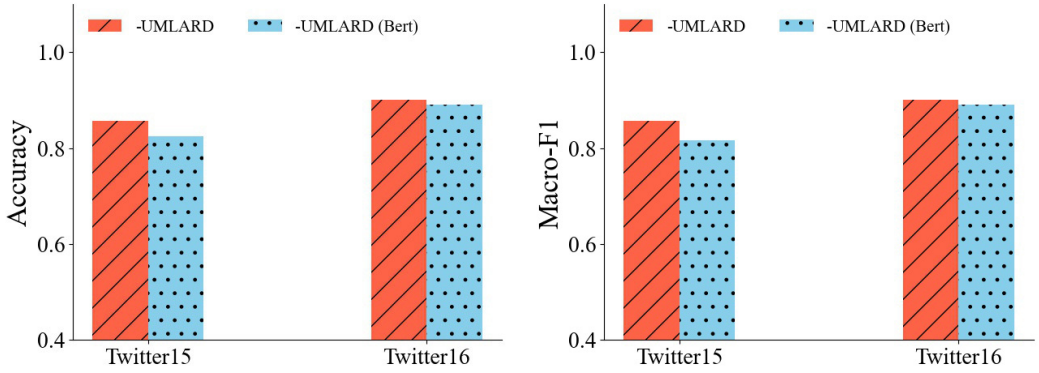


(a) Twitter15 **(b)** Twitter16 **(c)** Weibo

Figure 7.19: The average network breadth for different types of rumors. The observation window is 60 minutes.

(5) Finally, the discrepancy between UMLARD and **-TD** indicates the gain of modeling time decay in retweet cascades. In other words, both real information and false

information will significantly reduce their influence over time.



(a) Accuracy **(b)** Macro-F1

Figure 7.20: Content-aspect study of UMLARD.

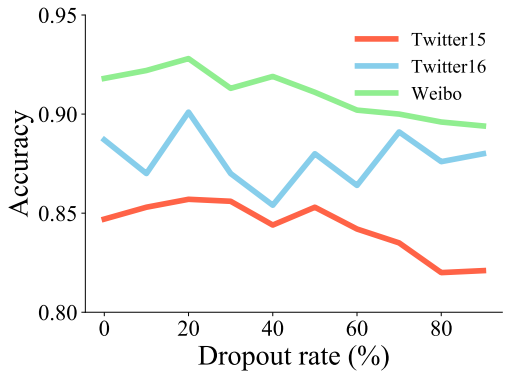


Figure 7.21: DropEdge Study of UMLARD.

To further investigate the content-aspect effect, we examine the influence of different word embedding methods. Specifically, we use the state-of-the-art Bert-based pretraining model [181] to replace the word2vec and then compare the performance on accuracy and macro-F1. In our work, we choose BERT-Base[1], which was trained on a large text corpus (e.g., Wikipedia). The results are shown in Figure 7.20. We can observe that the performance of Bert-based UMLARD is surprisingly lower. This happens due to the characteristics of tweet text, which are *short*, *sparse*, *sporadic* and written *casually*. Therefore, the Bert-based pretraining techniques that are usually trained on large-scale language corpus are difficult to directly used for short-text tasks such as Twitter content embedding. This conjecture is in accordance with some recent observations on [164].

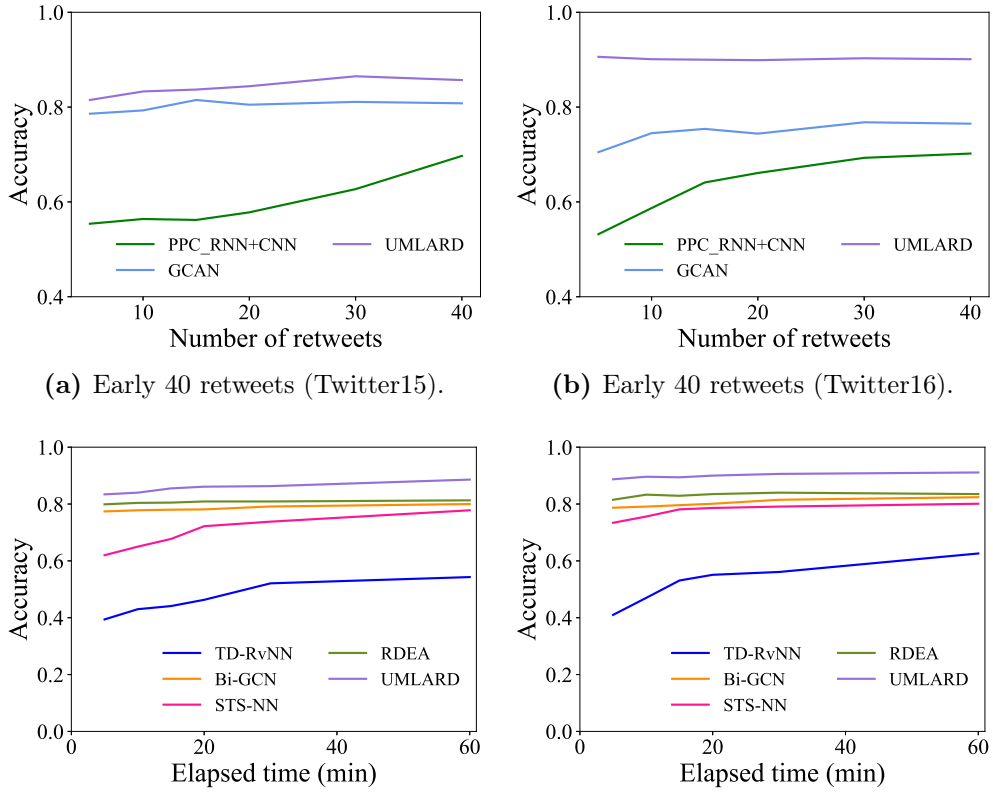
Furthermore, at the end of this section, we conduct an extra experiment to demonstrate the effectiveness of the "DropEdge" technical used in the data prepossessing. The dropout rate is set from 0 to 0.9, and the experimental results are shown in

[1]https://github.com/google-research/bert

Figure 7.21. We find that slightly dropping the edges in the diffusion graph would improve the model performance.

7.2.4.4 Performance on Early Detection (Q3)

Another important goal of rumor detection is to detect misinformation as early as possible and stop its spread in a timely fashion. Now we investigate the performance of models on identifying rumors at early-stage. Here, we consider two metrics for gauging the observation windows of information spread, i.e., the previous 40 retweets and the propagation in the first hour. In this section, the experiments of early detection are conducted on Twitter15 and Twitter16.



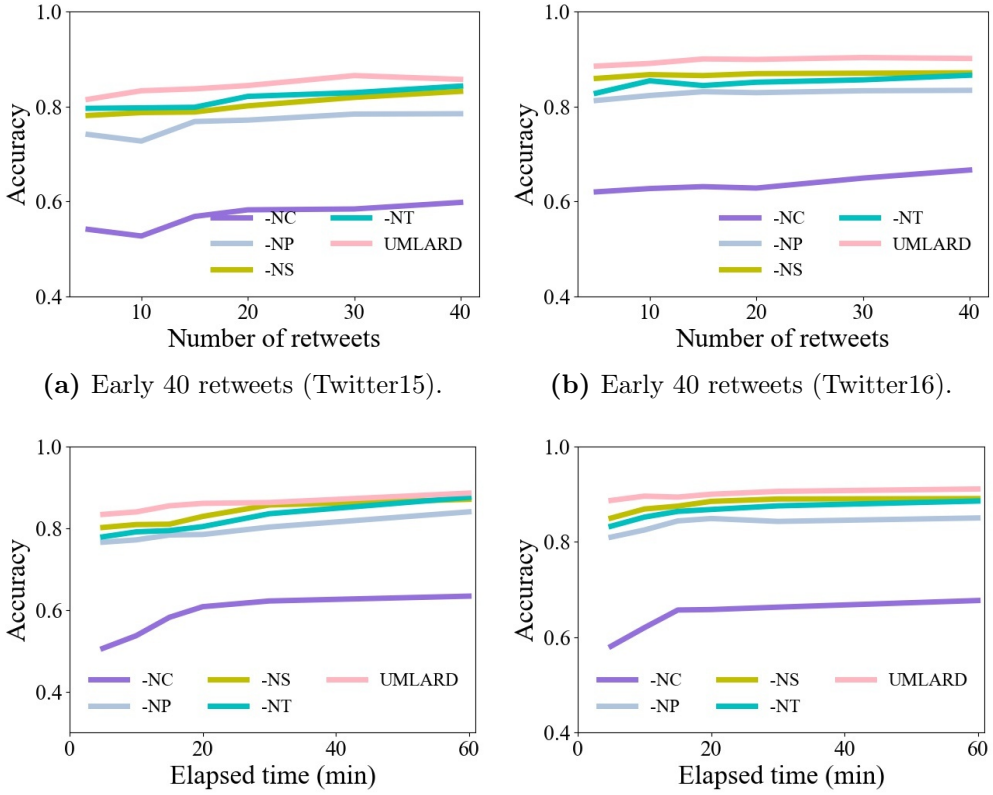
(a) Early 40 retweets (Twitter15). **(b)** Early 40 retweets (Twitter16).

(c) Early 1-hour retweets (Twitter15). **(d)** Early 1-hour retweets (Twitter16).

Figure 7.22: Evaluations on early rumor detection. (a) and (b): PPC_RNN+CNN and GCAN are cascade length-based methods. (c) and (d): Tv-RvNN, Bi-GCN, STS-NN and RDEA are built on the user comments that may not exist in early-stage retweets – hence, we observe their performance over time.

Figure 7.22 shows the performance comparison on early-stage detection between our UMLARD and the baselines. Note that we omit the feature-based methods and credibility-based approaches since they did not show comparable performance, especially on early rumor detection. We observe that UMLARD performs better, especially when there are only a few observations. UMLARD needs a short time to identify the misinformation because it fuses the multi-view knowledge of users.

For example, understanding the role of a user in spreading information is vital since tweets' size, spread speed and patterns are different. Moreover, UMLARD is capable of discriminating the importance of features even with few observations, which means the interference caused by the trivial or useless features would be dampened during training the model. In all cases, their early detection accuracy grows at the early stage of propagation. However, we find that the performance of our model demonstrates obvious advantage as time goes on.



(a) Early 40 retweets (Twitter15). **(b)** Early 40 retweets (Twitter16).

(c) Early 1-hour retweets (Twitter15). **(d)** Early 1-hour retweets (Twitter16).

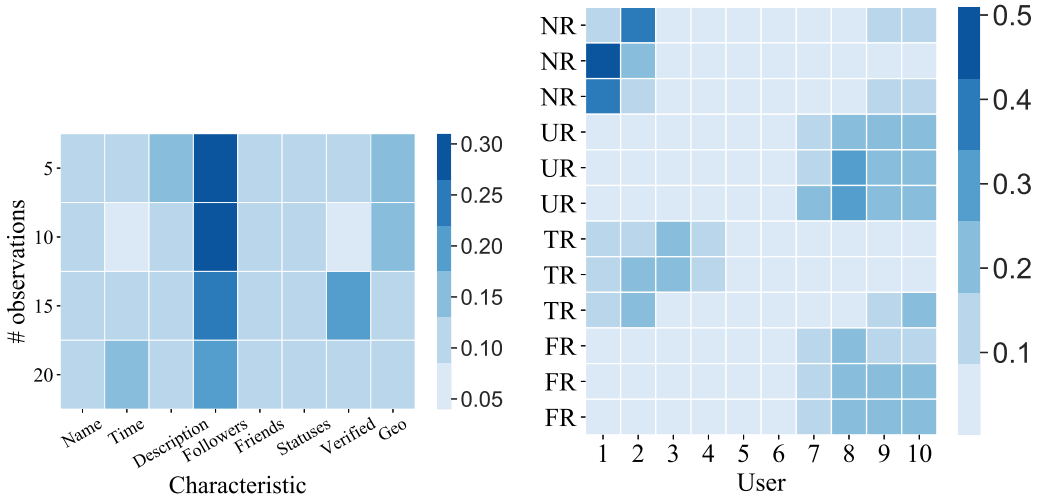
Figure 7.23: Evaluations on early rumor detection among variants of UMLARD. (a) and (b): The model performance using early 40 retweets. (c) and (d): The models are trained with early one hour observations.

We also investigate the time-varying performance between the variants and the full UMLARD. As shown in Figure 7.23, we find that the accuracy of all methods grows to saturation with increasing the number of retweets or time elapsed. Moreover, from Figure 7.23c and 7.23d, we can observe that the performance of -NP, -NT, and -NS is very close to the full UMLARD, because the models have acquired enough knowledge to detect rumors within a short observation time.

7.2.4.5 Interpretability Analysis (Q4)

The above experimental results have shown the superiority of the proposed hierarchical attentions. Namely, they can effectively discriminate the importance of

multi-views of users and the roles of users in spreading the (mis)information. Here, we provide more in-depth insights into the two components by visualizing the hierarchical attention layers in UMLARD.



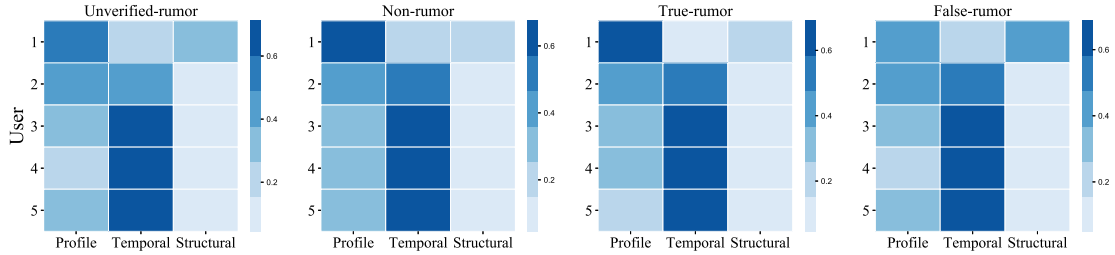
(a) User profile attention. **(b)** Capsule attention.

Figure 7.24: Visualization of user profiles importance and the role of earlier spreaders (Twitter15).

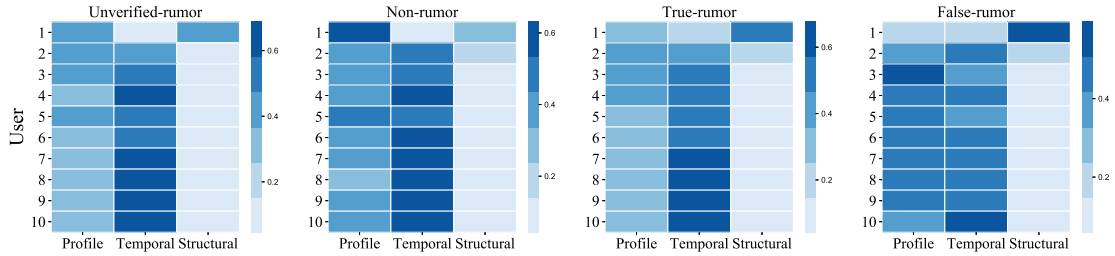
Figure 7.24 shows the importance of user-profiles and users themselves – the higher the value, the more important the feature or the user. Figure 7.24a plots the importance of eight user profile characteristics, where we vary the number of observed retweets between $\{5, 10, 15, 20\}$. We can observe that the follower counts is the most informative feature, followed by the register time, verified account, and geo-enabled features, consistent with the findings in [26, 83], i.e., the users enrolled in spreading of rumors have fewer followers.

In Figure 7.24b, we investigate the role of the retweet users at the very beginning of the cascade. As shown, the earlier users are more important for detecting *non-rumors* (NR) and *true-rumors* (TR). To the contrary, the later participators are important for detecting *unverified-rumors* (UR) and *false-rumors* (FR). This phenomenon shows that authoritative users usually spread TRs and NRs at the beginning of spreading information. URs and FRs, after the false information spread a while, will see an influx of massive malicious users, who would pretend these tweets as real information.

We now discuss the impact of the different views of users in rumor detection. We randomly selected four different types of tweets in Twitter15 and plots the importance of different views. Figure 7.25a and Figure 7.25b show the results of previous 5 and 10 retweet users, respectively. Overall, we can see that the three views of each user in these tweets have different importance. Specifically, when there are few observations (e.g., only 5 retweet users), the profile view and the temporal view of the



(a) View-wise Attention, observation length = 5.



(b) View-wise Attention, observation length = 10.

Figure 7.25: Visualization of the different user-aspect importance (Twitter15).

users dominate the rumor detection performance. As the number of retweet users increases, the structural information becomes more and more important. This result can be understood intuitively: In reality, at the very beginning the participants directly retweet the information from the source spreader, which leads to the similar propagation structures of information cascades. However, users are different from each other in profile and the time of retweeting, which are, consequently, the most important views for early-stage misinformation detection. Besides, by comparing different types of information, the non-rumor and the true-rumor have very similar weight distribution over different users' views, as observed in Figure 7.24b.

7.2.5 Summary

In this work, we presented UMLARD – a novel model for rumor detection which fuses multiple information contexts pertaining to users of social networks. Combining multiple views of users aspects and discriminating the importance of spreaders and user-aspect information, we successfully identified users' roles in different stages of rumor diffusion. UMLARD significantly outperforms previous methods in terms of misinformation classification and rapid rumor detection. Our approach is also notable in its strength of interpreting model behaviors and the predicted results. The experiments conducted on real Twitter datasets and Weibo dataset support the hypothesis that characteristics of user-profiles, aspects view of participants, as well as user's engagement time and tweets' diffusion patterns, can contribute to the misinformation prediction from the collective signals. Besides, our experimental results on early-detection discern several vital features of false information.

Chapter 8

Conclusions and Future work

In this thesis, we focused on two specific tasks in the field of information diffusion analysis in online social networks, namely, information cascades modeling and rumor detection. In this chapter, we first briefly summarize the main contributions of this thesis, and then conclude by discussing some important future directions.

8.1 Summary of Contributions

In Chapter 1, we stated four different research questions that we now recall so to summarize for each question the contribution of this thesis.

Research Question 1 (RQ 1) *Can we develop an effective deep learning-based model to capture structural and temporal features from the observed cascade graph for macro-level information cascade prediction?*

To the best of our knowledge, the proposed *Recurrent Cascades Convolutional Network* (CasCN) [10] (Chapter 4) is the first neural diffusion model answering this question. Besides that, CasCN also provides an important first step to sample the observed cascade graph as a sequence of subgraphs rather than random walks and diffusion paths. Based on the experimental results and feature visulizations, we find that (1) the way to sample an observed cascade as a sequence of subgraphs is much informative than other node sequence-based sampling methods; (2) CasCN can effectively learn the structural-temporal features from a sequence of subgraphs; (3) CasCN significantly outperform earlier deep learning-based methods for the task of macro-level information cascade prediction.

Research Question 2 (RQ 2) *How can we improve upon earlier deep learning-based models that learn the latent representation for the observed cascade graph from a multi-scale perspective to predict the future size of this cascade?*

We have introduced the *Multi-scale Cascades* (MUCas) model [42] (Chapter 5) to address this question. MUCas abstracts the multi-scale information for cascade graphs as a collection of direction-scale, high-order-scale, position-scale, and dynamic-scale

features, and it uses a multi-scale graph capsule network and an influence attention to learn and fuse the multi-scale information to form a unique cascade representation. This marks an important first step towards solving RQ 2. Different from previous works, MUCas innovatively propose a time interval-based sampling method, which significantly reduces the number of subgraphs, and increases differences between each subgraph as much as possible. Moreover, we find that: (1) each of the scale information is vital for information cascades; (2) aggregate features in a more fine-grained way by introducing routing mechanism can be a more effective and simple way to replace multi-head attention from prior works.

Research Question 3 (RQ 3) *Can we detect rumors at an early stage by learning various diffusion patterns from rumor spreading threads?*

The *Macroscopic and Microscopic-aware Rumor Detection* (MMRD) model [44] (Chapter 6) addresses this research question by two newly designed encoding components MacroE and MicroE, which were used for modeling rumor diffusion from macroscopic and microscopic perspectives, respectively. To the best of our knowledge, this is the first work to solve the rumor detection task by only exploring different levels of diffusion patterns. Also, MMRD innovatively introduces cross-learning and hybrid aggregation mechanisms to improve the model ability in feature learning and feature fusion. Furthermore, MMRD successfully leverages knowledge distillation to increase detection performance, which is a meaningful attempt that can be further considered in future research. We conduct our experiments on two real-world datasets, the experimental results demonstrate MMRD outperforms other state-of-the-art methods and can be applied in an extremely early rumor detection, i.e., with only 10 retweet user observations.

Research Question 4 (RQ 4) *Can we improve the model performance by developing effective rumor detection models at the participant level?*

We have proposed two complementary models to solve the **RQ 4** in Chapter 7: the *Participant-level Rumor Detection* (PLRD) model [45] and the *User-aspect Multi-view Learning with Attention for Rumor Detection* (UMLARD) model [46]. Both of them aim to extract features at the user level rather than the event level. PLRD serves as the first deep learning-based participant-level rumor detection model rooted in social influence and propagation theory, which provides demonstrations of the importance of users in rumor spreading from both theoretical and model performance perspectives. PLRD comprehensively exploits various fine-grained user features from the diffusion threads, i.e., the users' social homophily, influence, susceptibility, temporal features, and then uses these features to determine whether the information is true or false. PLRD also introduces a variational autoencoder (VAE) to handle the uncertainty which exists in the feature learning phase. Moreover, Compared with prior deep learning-based rumor detection models, PLRD can make good predictions only based on user-level features and also provides explainability from both feature-level and user-level. UMLARD solves one burning limitation left by PLRD and

other existing rumor detection methods, i.e., input features entangled with learned high-level features, by using three view-specific embedding methods with distinct inputs. Compared with PLRD, UMLARD innovatively proposes a capsule-based attention layer to replace the original attention mechanism in PLRD, which is more effective from both performance and time cost. UMLARD classifies information into more fine-grained labels, which is rarely considered in existing works because it increases the difficulty in detection.

8.2 Future Work

The research presented in this thesis achieved some interesting results, opens some potential research directions that we leave for future work.

Data collection and processing: Current open benchmark datasets (such as Weibo [9], Twitter [33], etc.) face several challenges. First, due to the strict privacy protection policies in online social platforms, the benchmark datasets can not open access to all resources, such as user profiles, which leads to difficulty in reproducing the same results as the state-of-the-art methods provided. Secondly, the data collection APIs of online social platforms do not provide true retweet paths. In fact, the raw data looks like all retweets point to the original post, which is generally not the case in reality. To deal with this problem, the current method infers the true retweet path based on the follower relationship between all users enrolled in the diffusion process and the timestamps for all retweets, however, this method is time-consuming and biased. Last, the collection of the complete diffusion graph and the social graph is difficult for researchers due to the access limitation of data collection APIs. There is the needs to develop an open-source data collection platform rather than only provide public available datasets, such as FakeNewsNet[1]. To construct the diffusion graph, we can further improve the quality of inferred edges by learning edge uncertainty. And in order to acquire the complete graphs, a corporation with online social platforms is indispensable.

Developing self-supervised and unsupervised model: Most existing efforts on information cascade modeling and rumor detection have been mainly focused on developing supervised and semi-supervised models, both of which require a dataset with sufficient labeled data. However, deciding on a label for a specific message requires a lot of manual labor, especially when it comes to rumor detection. For example, the label of each tweet in Twitter15/16 was confirmed from the fact-checking systems (e.g., Snopes [2], Factcheck [3], etc.). Hence, designing a model that can reduce dependence on labeled data while nevertheless doing well in information diffusion

[1]https://github.com/KaiDMML/FakeNewsNet
[2]https://www.snopes.com/
[3]https://www.factcheck.org/

analysis tasks is an urgent task. Ideally, one could design self-supervised and unsupervised frameworks to encode the information diffusion including structural and temporal features and facilitate the downstream tasks, such as information cascade prediction and rumor detection. Moreover, self-supervised learning always asks for a data augmentation operating, different from the augmentation on images, existing augmentation on graphs, such as edge-delete/add, node-delete/add, etc., which will harm the real diffusion networks and cause information biases. Therefore, how to design a diffusion-aware augmentation method or develop an augmentation-free self-supervised model is another interesting research topic.

Incorporating multi-modal and external information: To improve the model performance for both information cascade modeling and rumor detection, incorporating the knowledge from different aspects is demonstrated to be indispensable in Chapter 7. Apart from the previously used graph, sequence, and text, how to comprehensively extract features from other related sources, such as images, videos, websites, and so on, in a unified framework and efficiently fuse multiple features is an interesting research topic. Incorporating the external information from knowledge graphs in information diffusion prediction or rumor detection can introduce interpretability and inference ability into models, how to combine multi-modal data and external information is absolutely an interesting research direction.

Learning robust embedding for tail-nodes in the graph: The long-tail distribution phenomena can be discovered not only in datasets, such as the label imbalance, but also in graphs, where the majority of nodes are tail-nodes (with small degree) and only a small fraction have a big degree (head-node). Most of the existing works of graph neural networks treat all nodes equally, and do not pay more attention to the difference between tail-nodes and head-nodes, which have limited ability in learning distinguishable and robust embedding for the most vulnerable tail-nodes. As future work, one could borrow the idea from the fields of meta-learning and transfer learning to design a more unified graph neural network for tail-node embedding.

Developing interpretable deep learning-based models: Despite the significant achievements made by employing deep learning methods in information analysis tasks, compared with the traditional hand-crafted feature-based methods, the deep learning-based models do not provide enough interpretability due to their “black-box" models. However, besides looking for improvements of model performance, researchers also want to know the reason behind a message going viral or the intentions behind rumors. Thus, developing interpretable deep learning-based models without significantly sacrificing model performance, is another interesting direction to explore. For example, we plan in future work to keep designing new attention-based and disentangled models, as well as introducing causality into model learning.

.

Bibliography

[1] Cheng, J., Adamic, L., Dow, P.A., Kleinberg, J.M., Leskovec, J.: Can cascades be predicted? In: Proceedings of the 23rd International Conference on World Wide Web. WWW '14 (2014) 925–936

[2] Shen, H.W., Wang, D., Song, C., Barabási, A.L.: Modeling and predicting popularity dynamics via reinforced poisson processes. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. AAAI '14 (2014) 291–297

[3] Gruhl, D., Guha, R., Liben-Nowell, D., Tomkins, A.: Information diffusion through blogspace. In: Proceedings of the 13th International Conference on World Wide Web. WWW '04 (2004) 491–501

[4] Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N., Hurst, M.: Cascading behavior in large blog graphs: Patterns and a model. Society of Applied and Industrial Mathematics: Data Mining (2007) 551–556

[5] Liben-Nowell, D., Kleinberg, J.: Tracing information flow on a global scale using internet chain-letter data. Proceedings of the national academy of sciences **105** (2008) 4633–4638

[6] Golub, B., Jackson, M.O.: Using selection bias to explain the observed structure of internet diffusions. Proceedings of the National Academy of Sciences **107** (2010) 10833–10836

[7] Jenders, M., Kasneci, G., Naumann, F.: Analyzing and predicting viral tweets. In: Proceedings of the 22nd International Conference on World Wide Web. WWW '13 (2013) 657–664

[8] Li, C., Ma, J., Guo, X., Mei, Q.: Deepcas: An end-to-end predictor of information cascades. In: Proceedings of the 26th International Conference on World Wide Web. WWW '17 (2017) 577–586

[9] Cao, Q., Shen, H., Cen, K., Ouyang, W., Cheng, X.: Deephawkes: Bridging the gap between prediction and understanding of information cascades. In: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. CIKM '17 (2017) 1149–1158

[10] Chen, X., Zhou, F., Zhang, K., Trajcevski, G., Zhong, T., Zhang, F.: Information diffusion prediction via recurrent cascades convolution. In: 2019 IEEE 35th International Conference on Data Engineering. ICDE '19 (2019) 770–781

[11] Zhou, F., Xu, X., Trajcevski, G., Zhang, K.: A survey of information cascade analysis: Models, predictions, and recent advances. ACM Computing Surveys (CSUR) **54** (2021) 1–36

[12] Leskovec, J., Adamic, L.A., Huberman, B.A.: The dynamics of viral marketing. ACM Transactions on the Web (TWEB) (2007) 5–es

[13] Wang, Y., Shen, H., Liu, S., Cheng, X.: Learning user-specific latent influence and susceptibility from information cascades. In: Twenty-Ninth AAAI Conference on Artificial Intelligence. AAAI '15 (2015)

[14] Li, H., Ma, X., Wang, F., Liu, J., Xu, K.: On popularity prediction of videos shared in online social networks. In: Proceedings of the 22nd ACM International Conference on Information and Knowledge Management. CIKM '13 (2013) 169–178

[15] Wu, Q., Gao, Y., Gao, X., Weng, P., Chen, G.: Dual sequential prediction models linking sequential recommendation and information dissemination. In: Proceedings of the 25th ACM

SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '19 (2019) 447–457

[16] Bian, T., Xiao, X., Xu, T., Zhao, P., Huang, W., Rong, Y., Huang, J.: Rumor detection on social media with bi-directional graph convolutional networks. In: Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence. AAAI '20 (2020) 549–556

[17] Shu, K., Sliva, A., Wang, S., Tang, J., Liu, H.: Fake news detection on social media: A data mining perspective. ACM SIGKDD explorations newsletter **19** (2017) 22–36

[18] Allcott, H., Gentzkow, M.: Social media and fake news in the 2016 election. Journal of economic perspectives **31** (2017) 211–36

[19] DiFonzo, N., Bordia, P.: Rumor and prediction: Making sense (but losing dollars) in the stock market. Organizational Behavior and Human Decision Processes **71** (1997) 329–353

[20] Rosenberg, H., Syed, S., Rezaie, S.: The twitter pandemic: The critical role of twitter in the dissemination of medical information and misinformation during the covid-19 pandemic. Canadian Journal of Emergency Medicine **22** (2020) 418–421

[21] Banerjee, D., Rao, T.S., et al.: Psychology of misinformation and the media: Insights from the covid-19 pandemic. Indian Journal of Social Psychiatry **36** (2020) 131

[22] Petrovic, S., Osborne, M., Lavrenko, V.: Rt to win! predicting message propagation in twitter. In: Proceedings of the International AAAI Conference on Web and Social Media. ICWSM'11 (2011) 586–589

[23] Castillo, C., Mendoza, M., Poblete, B.: Information credibility on twitter. In: Proceedings of the 20th International Conference on World Wide Web. WWW '11 (2011) 675–684

[24] Gupta, M., Zhao, P., Han, J.: Evaluating event credibility on twitter. In: Proceedings of the 2012 SIAM International Conference on Data Mining. SIAM '12 (2012) 153–164

[25] Bakshy, E., Hofman, J.M., Mason, W.A., Watts, D.J.: Everyone's an influencer: quantifying influence on twitter. In: Proceedings of the fourth ACM international conference on Web search and data mining. WSDM '11 (2011) 65–74

[26] Shu, K., Wang, S., Liu, H.: Understanding user profiles on social media for fake news detection. In: 2018 IEEE Conference on Multimedia Information Processing and Retrieval. MIPR '18 (2018) 430–435

[27] Bao, P., Shen, H.W., Huang, J., Cheng, X.Q.: Popularity prediction in microblogging network: A case study on sina weibo. In: Proceedings of the 22nd International Conference on World Wide Web. WWW '13 (2013) 177–178

[28] Szabo, G., Huberman, B.A.: Predicting the popularity of online content. Communications of the ACM **53** (2010) 80–88

[29] Kwon, S., Cha, M., Jung, K., Chen, W., Wang, Y.: Prominent features of rumor propagation in online social media. In: 2013 IEEE 13th International Conference on Data Mining. ICDM '13 (2013) 1103–1108

[30] Gao, S., Ma, J., Chen, Z.: Modeling and predicting retweeting dynamics on microblogging platforms. In: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining. WSDM '15 (2015) 107–116

[31] Jin, Z., Cao, J., Jiang, Y.G., Zhang, Y.: News credibility evaluation on microblog with a hierarchical propagation model. In: 2014 IEEE International Conference on Data Mining. ICDM '14 (2014) 230–239

[32] Jin, Z., Cao, J., Zhang, Y., Luo, J.: News verification by exploiting conflicting social viewpoints in microblogs. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence. AAAI '16 (2016) 2972–2978

[33] Ma, J., Gao, W., Mitra, P., Kwon, S., Jansen, B.J., Wong, K.F., Cha, M.: Detecting rumors from microblogs with recurrent neural networks. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence. IJCAI '16 (2016) 3818–3824

[34] Ma, J., Gao, W., Wong, K.F.: Rumor detection on twitter with tree-structured recursive neural networks. In: Proceedings of the 56th Annual Meeting of the Association for Com-

putational Linguistics. ACL '18 (2018) 1980–1989

[35] Wang, J., Zheng, V.W., Liu, Z., Chang, C.C.: Topological recurrent neural network for diffusion prediction. In: 2017 IEEE International Conference on Data Mining. ICDM '17 (2017) 475–484

[36] Wang, Y., Shen, H., Liu, S., Gao, J., Cheng, X., Wang, Y., Shen, H., Liu, S., Gao, J., Cheng, X.: Cascade dynamics modeling with attention-based recurrent neural network. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence. IJCAI '17 (2017) 2985–2991

[37] Weng, L., Menczer, F., Ahn, Y.Y.: Predicting successful memes using network and community structure. In: Eighth international AAAI conference on weblogs and social media. AAAI '14 (2014)

[38] Tsur, O., Rappoport, A.: What's in a hashtag? content based prediction of the spread of ideas in microblogging communities. In: Proceedings of the Fifth ACM International Conference on Web Search and Data Mining. WSDM '12 (2012) 643–652

[39] Cui, P., Jin, S., Yu, L., Wang, F., Zhu, W., Yang, S.: Cascading outbreak prediction in networks: A data-driven approach. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '13 (2013) 901–909

[40] Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9** (1997) 1735–1780

[41] Defferrard, M., Bresson, X., Vandergheynst, P.: Convolutional neural networks on graphs with fast localized spectral filtering. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. NIPS'16 (2016) 3844–3852

[42] Chen, X., Zhang, F., Zhou, F., Bonsangue, M.: Multi-scale graph capsule with influence attention for information cascades prediction. International Journal of Intelligent Systems **37** (2022) 2584–2611

[43] Vosoughi, S., Roy, D., Aral, S.: The spread of true and false news online. Science **359** (2018) 1146–1151

[44] Chen, X., Zhou, F., Zhang, F., Bonsangue, M.: Modeling microscopic and macroscopic information diffusion for rumor detection. International Journal of Intelligent Systems **36** (2021) 5449–5471

[45] Chen, X., Zhou, F., Zhang, F., Bonsangue, M.: Catch me if you can: A participant-level rumor detection framework via fine-grained user representation learning. Information Processing & Management **58** (2021) 102678

[46] Chen, X., Zhou, F., Trajcevski, G., Bonsangue, M.: Multi-view learning with distinguishable feature fusion for rumor detection. Knowledge-Based Systems **240** (2022) 108085

[47] Crane, R., Sornette, D.: Robust dynamic classes revealed by measuring the response function of a social system. Proceedings of the National Academy of Sciences **105** (2008) 15649–15653

[48] Gao, J., Shen, H., Liu, S., Cheng, X.: Modeling and predicting retweeting dynamics via a mixture process. In: Proceedings of the 25th International Conference Companion on World Wide Web. WWW '16 (2016) 33–34

[49] Mishra, S., Rizoiu, M.A., Xie, L.: Feature driven and point process approaches for popularity prediction. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. CIKM '16 (2016) 1069–1078

[50] Rizoiu, M.A., Xie, L., Sanner, S., Cebrian, M., Yu, H., Van Hentenryck, P.: Expecting to be hip: Hawkes intensity processes for social media popularity. In: Proceedings of the 26th International Conference on World Wide Web. WWW '17 (2017) 735–744

[51] Hong, L., Dan, O., Davison, B.D.: Predicting popular messages in twitter. In: Proceedings of the 20th International Conference Companion on World Wide Web. WWW '11 (2011) 57–58

[52] Ma, Z., Sun, A., Cong, G.: On predicting the popularity of newly emerging hashtags in twitter. Journal of the American Society for Information Science and Technology **64** (2013)

1399–1410

[53] Lerman, K., Galstyan, A.: Analysis of social voting patterns on digg. In: Proceedings of the First Workshop on Online Social Networks. WOSN '08 (2008) 7–12

[54] Romero, D.M., Tan, C., Ugander, J.: On the interplay between social and topical structure. In: Seventh International AAAI Conference on Weblogs and Social Media. AAAI '13 (2013)

[55] Pinto, H., Almeida, J.M., Gonçalves, M.A.: Using early view patterns to predict the popularity of youtube videos. In: Proceedings of the Sixth ACM International Conference on Web Search and Data Mining. WSDM '13 (2013) 365–374

[56] Shulman, B., Sharma, A., Cosley, D.: Predictability of popularity: Gaps between prediction and understanding. In: Proceedings of the International AAAI Conference on Web and Social Media. ICWSM '16 (2016)

[57] Li, C., Guo, X., Mei, Q.: Joint modeling of text and networks for cascade prediction. In: Proceedings of the International AAAI Conference on Web and Social Media. ICWSM '18, Proceedings of the International AAAI Conference on Web and Social Media (2018)

[58] Gou, C., Shen, H., Du, P., Wu, D., Liu, Y., Cheng, X.: Learning sequential features for cascade outbreak prediction. Knowledge and Information Systems **57** (2018) 721–739

[59] Liu, Z., Wang, R., Liu, Y.: Prediction model for non-topological event propagation in social networks. In: International Conference of Pioneering Computer Scientists, Engineers and Educators. ICPCSEE '19, Springer (2019) 241–252

[60] Tang, X., Liao, D., Huang, W., Xu, J., Zhu, L., Shen, M.: Fully exploiting cascade graphs for real-time forwarding prediction. In: Proceedings of the AAAI Conference on Artificial Intelligence. AAAI '21 (2021) 582–590

[61] Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International conference on learning representations. ICLR '17 (2017)

[62] Huang, Z., Wang, Z., Zhang, R.: Cascade2vec: Learning dynamic cascade representation by recurrent graph neural networks. IEEE Access (2019) 144800–144812

[63] Xu, Z., Qian, M., Huang, X., Meng, J.: Casgcn: Predicting future cascade growth based on information diffusion graph. arXiv preprint arXiv:2009.05152 (2020)

[64] Wang, Y., Wang, X., Michalski, R., Ran, Y., Jia, T.: Casseqgcn: Combining network structure and temporal sequence to predict information cascades. arXiv preprint arXiv:2110.06836 (2021)

[65] Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations. ICLR '19 (2019)

[66] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. CVPR '16 (2016) 770–778

[67] Huang, Z., Wang, Z., Zhang, R., Zhao, Y., Zheng, F.: Learning bi-directional social influence in information cascades using graph sequence attention networks. In: Companion Proceedings of the Web Conference 2020. WWW '20, Companion Proceedings of the Web Conference 2020 (2020) 19–21

[68] Cao, Q., Shen, H., Gao, J., Wei, B., Cheng, X.: 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 (2020) 70–78

[69] Zhou, F., Xu, X., Zhang, K., Trajcevski, G., Zhong, T.: Variational information diffusion for probabilistic cascades prediction. In: IEEE Conference on Computer Communications. INFOCOM '20 (2020)

[70] Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: International conference on learning representations. ICLR '14 (2014)

[71] Chen, X., Zhang, K., Zhou, F., Trajcevski, G., Zhong, T., Zhang, F.: Information cascades modeling via deep multi-task learning. In: Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '19 (2019)

885–888

[72] Yang, C., Tang, J., Sun, M., Cui, G., Liu, Z.: Multi-scale information diffusion prediction with reinforced recurrent networks. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence. IJCAI '19 (2019) 4033–4039

[73] Zhou, F., Jing, X., Xu, X., Zhong, T., Trajcevski, G., Wu, J.: Continual information cascade learning. In: 2020 IEEE Global Communications Conference. GLOBECOM '20 (2020) 1–6

[74] Zhou, F., Yu, L., Xu, X., Trajcevski, G.: Decoupling representation and regressor for long-tailed information cascade prediction. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '21, Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval (2021) 1875—-1879

[75] Cao, Q., Shen, H., Liu, Y., Gao, J., Cheng, X.: Pre-training of temporal convolutional neural networks for popularity prediction. arXiv preprint arXiv:2108.06220 (2021)

[76] Xu, X., Zhou, F., Zhang, K., Liu, S.: CCGL: contrastive cascade graph learning. arXiv preprint arXiv:2107.12576 **abs/2107.12576** (2021)

[77] Zhang, Z., Zhang, Z., Li, H.: Predictors of the authenticity of internet health rumours. Health Information & Libraries Journal **32** (2015) 195–205

[78] Zhao, Z., Resnick, P., Mei, Q.: Enquiring minds: Early detection of rumors in social media from enquiry posts. In: Proceedings of the 24th International Conference on World Wide Web. WWW '15 (2015) 1395–1405

[79] Hassan, A., Qazvinian, V., Radev, D.: What's with the attitude? identifying sentences with attitude in online discussions. In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing. EMNLP '10 (2010) 1245–1255

[80] Ma, B., Lin, D., Cao, D.: Content representation for microblog rumor detection. In: Advances in Computational Intelligence Systems. (2017) 245–251

[81] Wu, K., Yang, S., Zhu, K.Q.: False rumors detection on sina weibo by propagation structures. In: 2015 IEEE 31st International Conference on Data Engineering. ICDE '15 (2015) 651–662

[82] Jin, Z., Cao, J., Zhang, Y., Zhou, J., Tian, Q.: Novel visual and statistical image features for microblogs news verification. IEEE transactions on multimedia (2016) 598–608

[83] Shu, K., Zhou, X., Wang, S., Zafarani, R., Liu, H.: The role of user profiles for fake news detection. In: Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. ASONAM '19 (2019) 436–439

[84] Ma, J., Gao, W., Wong, K.F.: Detect rumors in microblog posts using propagation structure via kernel learning. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics. ACL '17 (2017) 708–717

[85] Ma, J., Gao, W., Wei, Z., Lu, Y., Wong, K.: Detect rumors using time series of social context information on microblogging websites. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. CIKM '15 (2015) 1751–1754

[86] Yang, F., Liu, Y., Yu, X., Yang, M.: Automatic detection of rumor on sina weibo. In: Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics. MDS '12 (2012)

[87] Shu, K., Cui, L., Wang, S., Lee, D., Liu, H.: Defend: Explainable fake news detection. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '19 (2019) 395–405

[88] Hossain, T., Logan IV, R.L., Ugarte, A., Matsubara, Y., Young, S., Singh, S.: COVIDLies: Detecting COVID-19 misinformation on social media. In: Proceedings of the 1st Workshop on NLP for COVID-19 (Part 2) at EMNLP 2020. (2020)

[89] Dharawat, A., Lourentzou, I., Morales, A., Zhai, C.: Drink bleach or do what now? covidhera: A dataset for risk-informed health decision making in the presence of covid19 misinformation. arXiv preprint arXiv:2010.08743 (2020)

[90] Suryadikara, R., Verberne, S., Takes, F.W., Stefan Conrad, S., Tiddi, I.: False news classification and dissemination: the case of the 2019 indonesian presidential election. In: Proceed-

ings of the CIKM 2020 Workshops, co-located with 29th ACM International Conference on Information and Knowledge Management. CIKM '20 (2020) 1–9

[91] Jin, Z., Cao, J., Guo, H., Zhang, Y., Luo, J.: Multimodal fusion with recurrent neural networks for rumor detection on microblogs. In: Proceedings of the 25th ACM International Conference on Multimedia. MM '17 (2017) 795–816

[92] Khattar, D., Goud, J.S., Gupta, M., Varma, V.: Mvae: Multimodal variational autoencoder for fake news detection. In: The World Wide Web Conference. WWW '19 (2019) 2915–2921

[93] Ma, J., Gao, W., Wong, K.: Detect rumor and stance jointly by neural multi-task learning. In: Companion Proceedings of the The Web Conference 2018. WWW '18 (2018) 585–593

[94] Schwarz, S., Theóphilo, A., Rocha, A.: Emet: Embeddings from multilingual-encoder transformer for fake news detection. In: ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing. ICASSP '20 (2020) 2777–2781

[95] Tian, L., Zhang, X., Wang, Y., Liu, H.: Early detection of rumours on twitter via stance transfer learning. In: European Conference on Information Retrieval. (2020) 575–588

[96] Kar, D., Bhardwaj, M., Samanta, S., Azad, A.P.: No rumours please! a multi-indic-lingual approach for covid fake-tweet detection. In: 2021 Grace Hopper Celebration India (GHCI). (2020) 1–5

[97] Kaliyar, R.K., Goswami, A., Narang, P.: Fakebert: Fake news detection in social media with a bert-based deep learning approach. Multimedia tools and applications **80** (2021) 11765–11788

[98] Cui, L., Seo, H., Tabar, M., Ma, F., Wang, S., Lee, D.: Deterrent: Knowledge guided graph attention network for detecting healthcare misinformation. In: Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining. KDD '20 (2020) 492–502

[99] Shu, K., Mahudeswaran, D., Wang, S., Liu, H.: Hierarchical propagation networks for fake news detection: Investigation and exploitation. In: Proceedings of the International AAAI Conference on Web and Social Media. Volume 14 of ICWSM '20. (2020) 626–637

[100] Liu, Y., Wu, Y.F.B.: Early detection of fake news on social media through propagation path classification with recurrent and convolutional networks. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence. AAAI '18 (2018) 354–361

[101] Song, C., Shu, K., Wu, B.: Temporally evolving graph neural network for fake news detection. Information Processing & Management **58** (2021) 102712

[102] Lu, Y.J., Li, C.T.: Gcan: Graph-aware co-attention networks for explainable fake news detection on social media. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. ACL '20 (2020) 505–514

[103] He, Z., Li, C., Zhou, F., Yang, Y.: Rumor detection on social media with event augmentations. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '21 (2021) 2020–2024

[104] Dou, Y., Shu, K., Xia, C., Yu, P.S., Sun, L.: User preference-aware fake news detection. In: Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '21 (2021) 2051—-2055

[105] Cao, J., Guo, J., Li, X., Jin, Z., Guo, H., Li, J.: Automatic rumor detection on microblogs: A survey. arXiv preprint arXiv:1807.03505 (2018)

[106] Elman, J.L.: Finding structure in time. Cognitive science **14** (1990) 179–211

[107] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. IEEE transactions on neural networks **5** (1994) 157–166

[108] Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)

[109] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y.: Graph attention networks. In: International Conference on Learning Representations. ICLR '18 (2018)

[110] Ying, R., You, J., Morris, C., Ren, X., Hamilton, W.L., Leskovec, J.: Hierarchical graph representation learning with differentiable pooling. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. NIPS'18 (2018) 4805—-4815

[111] Ma, Y., Wang, S., Aggarwal, C.C., Tang, J.: Graph convolutional networks with eigenpooling. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '19 (2019) 723–731

[112] Lei, F., Liu, X., Jiang, J., Liao, L., Cai, J., Zhao, H.: Graph convolutional networks with higher-order pooling for semisupervised node classification. Concurrency and Computation: Practice and Experience (2020) e5695

[113] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20** (2008) 61–80

[114] Wegrzycki, K., Sankowski, P., Pacuk, A., Wygocki, P.: Why do cascade sizes follow a power-law? In: Proceedings of the 26th International Conference on World Wide Web. WWW '17 (2017) 569–576

[115] Li, Y., Yu, R., Shahabi, C., Liu, Y.: Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. In: International Conference on Learning Representations. ICLR '18 (2018)

[116] Atwood, J., Towsley, D.: Diffusion-convolutional neural networks. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. NIPS'16 (2016) 2001–2009

[117] Chung, F.: Laplacians and the cheeger inequality for directed graphs. Annals of Combinatorics **9** (2005) 1–19

[118] Li, Y., Zhang, Z.L.: Digraph laplacian and the degree of asymmetry. Internet Mathematics **8** (2012) 381–401

[119] Filimonov, V., Sornette, D.: Apparent criticality and calibration issues in the hawkes self-excited point process model: application to high-frequency financial data. Quantitative Finance **15** (2015) 1293–1314

[120] Wallinga, J., Teunis, P.: Different epidemic curves for severe acute respiratory syndrome reveal similar impacts of control measures. American Journal of epidemiology **160** (2004) 509–516

[121] Gehrke, J., Ginsparg, P., Kleinberg, J.: Overview of the 2003 kdd cup. Acm SIGKDD Explorations Newsletter **5** (2003) 149–151

[122] Grover, A., Leskovec, J.: Node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '16 (2016) 855–864

[123] Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '14 (2014) 701–710

[124] Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., Mei, Q.: Line: Large-scale information network embedding. In: Proceedings of the 24th International Conference on World Wide Web. WWW '15 (2015) 1067–1077

[125] Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008** (2008) P10008

[126] Abu-El-Haija, S., Perozzi, B., Kapoor, A., Harutyunyan, H., Alipourfard, N., Lerman, K., Steeg, G.V., Galstyan, A.: Mixhop: Higher-order graph convolution architectures via sparsified neighborhood mixing. In: Proceedings of the 36th International Conference on Machine Learning. ICML '19 (2019) 21–29

[127] Xinyi, Z., Chen, L.: Capsule graph neural network. In: International conference on learning representations. ICLR '19 (2019)

[128] Seo, Y., Defferrard, M., Vandergheynst, P., Bresson, X.: Structured sequence modeling with

graph convolutional recurrent networks. In: International Conference on Neural Information Processing. ICONIP '18 (2018) 362–373

[129] Yao, L., Mao, C., Luo, Y.: Graph convolutional networks for text classification. In: Proceedings of the AAAI Conference on Artificial Intelligence. AAAI '19 (2019) 7370–7377

[130] Lee, J.B., Rossi, R.A., Kong, X., Kim, S., Koh, E., Rao, A.: Higher-order graph convolutional networks. arXiv preprint arXiv:1809.07697 (2018)

[131] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NeurIPS'17 (2017) 6000–6010

[132] Liu, S., Chen, L., Dong, H., Wang, Z., Wu, D., Huang, Z.: Higher-order weighted graph convolutional networks. In: arXiv. (2019)

[133] Sabour, S., Frosst, N., Hinton, G.E.: Dynamic routing between capsules. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS '17 (2017) 3859–3869

[134] Peer, D., Stabinger, S., Rodríguez-Sánchez, A.: Limitation of capsule networks. Pattern Recognition Letters **144** (2021) 68–74

[135] Lin, Z., Feng, M., Santos, C.N.d., Yu, M., Xiang, B., Zhou, B., Bengio, Y.: A structured self-attentive sentence embedding. In: International Conference on Learning Representations. ICLR '17, International Conference on Learning Representations (2017)

[136] Li, J., Li, S., Zhao, W.X., He, G., Wei, Z., Yuan, N.J., Wen, J.R.: Knowledge-enhanced personalized review generation with capsule graph neural network. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management. CIKM '20, Proceedings of the 29th ACM International Conference on Information & Knowledge Management (2020) 735—744

[137] Yang, Y., Niu, K., He, Z.: Exploiting the topology property of social network for rumor detection. In: 12th International Joint Conference on Computer Science and Software Engineering (JCSSE). JCSSE'15 (2015) 41–46

[138] Shu, K., Wang, S., Liu, H.: Beyond news contents: The role of social context for fake news detection. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining. WSDM '19 (2019) 312–320

[139] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: International Conference on Learning Representations. ICLR '15 (2015)

[140] Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J., Han, J.: On the variance of the adaptive learning rate and beyond. In: Proceedings of the Eighth International Conference on Learning Representations. ICLR '20 (2020)

[141] Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)

[142] Hou, Y., Ma, Z., Liu, C., Loy, C.C.: Learning lightweight lane detection cnns by self attention distillation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. ICCV '19 (2019) 1013–1021

[143] Zhang, L., Song, J., Gao, A., Chen, J., Bao, C., Ma, K.: Be your own teacher: Improve the performance of convolutional neural networks via self distillation. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. ICCV '19 (2019) 3713–3722

[144] Dietterich, T.G.: Approximate statistical tests for comparing supervised classification learning algorithms. Neural Computation **10** (1998) 1895–1923

[145] Spohr, D.: Fake news and ideological polarization: Filter bubbles and selective exposure on social media. Business information review **34** (2017) 150–160

[146] Shore, J., Baek, J., Dellarocas, C.: Network structure and patterns of information diversity on twitter. MIS Q. **42** (2018) 849–872

[147] Nickerson, R.S.: Confirmation bias: A ubiquitous phenomenon in many guises. Review of general psychology **2** (1998) 175–220

[148] McPherson, M., Smith-Lovin, L., Cook, J.M.: Birds of a feather: Homophily in social networks. Review of Sociology **27** (2001) 415–444

[149] Kelman, H.C.: Compliance, identification, and internalization three processes of attitude change. Journal of conflict resolution **2** (1958) 51–60

[150] Gorrell, G., Kochkina, E., Liakata, M., Aker, A., Zubiaga, A., Bontcheva, K., Derczynski, L.: Semeval-2019 task 7: Rumoureval, determining rumour veracity and support for rumours. In: Proceedings of the 13th International Workshop on Semantic Evaluation. SemEval '19 (2019) 845–854

[151] Derczynski, L., Bontcheva, K., Liakata, M., Procter, R., Hoi, G.W.S., Zubiaga, A.: Semeval-2017 task 8: Rumoureval: Determining rumour veracity and support for rumours. In: Proceedings of the 11th International Workshop on Semantic Evaluation. SemEval '17 (2017) 69–76

[152] Gilani, Z., Farahbakhsh, R., Tyson, G., Wang, L., Crowcroft, J.: Of bots and humans (on twitter). In: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017. (2017) 349–354

[153] Sankar, A., Zhang, X., Krishnan, A., Han, J.: Inf-vae: A variational autoencoder framework to integrate homophily and influence in diffusion prediction. In: Proceedings of the 13th International Conference on Web Search and Data Mining. WSDM '20 (2020) 510–518

[154] Cao, S., Lu, W., Xu, Q.: Grarep: Learning graph representations with global structural information. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management. CIKM '15 (2015) 891–900

[155] Zhang, J., Dong, Y., Wang, Y., Tang, J., Ding, M.: Prone: Fast and scalable network representation learning. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence. IJCAI '19 (2019) 4278–4284

[156] Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM review **53** (2011) 217–288

[157] Tao, T.: Topics in random matrix theory. Volume 132. American Mathematical Soc. (2012)

[158] Levy, O., Goldberg, Y.: Neural word embedding as implicit matrix factorization. In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. NIPS'14 (2014) 2177–2185

[159] Wang, Z., Chen, C., Li, W.: Information diffusion prediction with network regularized role-based user representation learning. ACM Trans. Knowl. Discov. Data **13** (2019)

[160] Aral, S., Walker, D.: Identifying influential and susceptible members of social networks. Science **337** (2012) 337–341

[161] Hoang, T.A., Lim, E.P.: Virality and susceptibility in information diffusions. In: Sixth international AAAI conference on weblogs and social media. ICWSM '12 (2012)

[162] Shu, K., Mahudeswaran, D., Wang, S., Lee, D., Liu, H.: Fakenewsnet: A data repository with news content, social context, and spatiotemporal information for studying fake news on social media. Big Data (2020) 171–188

[163] Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)

[164] Nguyen, D.Q., Vu, T., Tuan Nguyen, A.: BERTweet: A pre-trained language model for English tweets. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. EMNLP '20 (2020) 9–14

[165] Rong, Y., Huang, W., Xu, T., Huang, J.: Dropedge: Towards deep graph convolutional networks on node classification. In: International Conference on Learning Representations. ICLR '20 (2020)

[166] Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., Shah, N.: Data augmentation for graph neural networks. arXiv preprint arXiv:2006.06830 (2020)

[167] Tang, J., Tian, Y., Liu, D., Kou, G.: Coupling privileged kernel method for multi-view

learning. Information Sciences **481** (2019) 110–127

[168] Tang, J., Xu, W., Li, J., Tian, Y., Xu, S.: Multi-view learning methods with the linex loss for pattern classification. Knowledge-Based Systems **228** (2021) 107285

[169] Wang, C., Chen, X., Chen, B., Nie, F., Wang, B., Ming, Z.: Learning unsupervised node representation from multi-view network. Information Sciences **579** (2021) 700–716

[170] Zhao, D., Gao, Q., Lu, Y., Sun, D., Cheng, Y.: Consistency and diversity neural network multi-view multi-label learning. Knowledge-Based Systems **218** (2021) 106841

[171] Zhu, C., Wang, P., Ma, L., Zhou, R., Wei, L.: Global and local multi-view multi-label learning with incomplete views and labels. Neural Computing and Applications **32** (2020) 15007–15028

[172] Jia, K., Lin, J., Tan, M., Tao, D.: Deep multi-view learning using neuron-wise correlation-maximizing regularizers. IEEE Transactions on Image Processing **28** (2019) 5121–5134

[173] Xie, Y., Yao, C., Gong, M., Chen, C., Qin, A.: Graph convolutional networks with multi-level coarsening for graph classification. Knowledge-Based Systems **194** (2020) 105578

[174] Zhu, Y., Li, H., Liao, Y., Wang, B., Guan, Z., Liu, H., Cai, D.: What to do next: Modeling user behaviors by time-lstm. In: Proceedings of the 26th International Joint Conference on Artificial Intelligence. IJCAI '17 (2017) 3602–3608

[175] Wang, Y., Ma, F., Jin, Z., Yuan, Y., Xun, G., Jha, K., Su, L., Gao, J.: Eann: Event adversarial neural networks for multi-modal fake news detection. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '18 (2018) 849–857

[176] Zhou, Y., Ji, R., Su, J., Sun, X., Chen, W.: Dynamic capsule attention for visual question answering. In: Proceedings of the AAAI Conference on Artificial Intelligence. AAAI '19 (2019) 9324–9331

[177] Liu, X., Chen, Q., Liu, Y., Siebert, J., Hu, B., Wu, X., Tang, B.: Decomposing word embedding with the capsule network. Knowledge-Based Systems **212** (2021) 106611

[178] Liu, Z., Xiong, C., Sun, M., Liu, Z.: Fine-grained fact verification with kernel graph attention network. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. ACL '20 (2020) 7342–7351

[179] Khoo, L.M.S., Chieu, H.L., Qian, Z., Jiang, J.: Interpretable rumor detection in microblogs by attending to user interactions. In: Proceedings of the AAAI Conference on Artificial Intelligence. AAAI '20 (2020) 8783–8790

[180] Huang, Q., Zhou, C., Wu, J., Liu, L., Wang, B.: Deep spatial–temporal structure learning for rumor detection on twitter. Neural Computing and Applications (2020) 1–11

[181] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. NAACL '19 (2019) 4171–4186

English Summary

With the emergence of online social networks (OSNs), the way people create and share information has changed, which becomes faster and broader than traditional social media. Understanding how information (both good and harmful) spreads through OSNs, as well as what elements drive the success of information diffusion, has significant implications for a wide range of real-world applications. In this thesis, we conduct research to analysis the diffusion of information in OSNs via using deep representation learning. Specifically, we aim to develop deep learning-based models to solve two specific tasks, i.e., information cascades modeling and rumor detection.

Our contributions are as follows:

We propose recurrent cascades convolutional network (CasCN) in Chapter 4, a graph-based neural model for macro-level information cascade prediction. CasCN utilizes a combination of graph convolutional network and recurrent neural network to predict the incremental size of a cascade by extracting structural-temporal features from a sequence of timestamp-based subgraphs. CasCN also introduces CasLaplacian for directed graphs, which overcomes the limitations of previous graph neural networks when dealing with directed graphs. The experimental results conducted on two real-world datasets, show that CasCN is well-suited for modeling structural-temporal features from cascades.

We propose multi-scale cascades model (MUCas) in Chapter 5, which aims at capturing multi-scale features for macro-level information cascade prediction. MUCas utilizes a multi-scale graph capsule network and an influence attention to learn and fuse the multi-scale information (i.e., dynamic-scale, direction-scale, position-scale, and high-order-scale) to form a unique cascade representation. MUCas also improves the sampling methods in CasCN by sampling subgraphs based on time intervals rather than timestamps. We conduct experiments on real-world datasets and demonstrate that MUCas is particularly effective at extracting features on cascades from different scales, and multi-scale features are vital for improving prediction accuracy.

We propose macroscopic and microscopic-aware rumor detection model (MMRD) in Chapter 6, a diffusion-based rumor detection model that detects rumors by only

exploring different levels of diffusion patterns. MMRD leverages graph neural networks to learn the macroscopic diffusion of rumor propagation and capture microscopic diffusion patterns using bidirectional recurrent neural networks while taking into account the user-time series. MMRD also leverages knowledge distillation technique to create a more informative student model and further improve the model performance. MMRD is evaluated on the well-known Twitter data sets and could obtain good detection results in the early stage of rumor spreading.

We propose participant-level rumor detection model (PLRD) and multi-view learning with attention for rumor detection model (UMLARD) in Chapter 7. Both PLRD and UMLARD are designed at user-level. PLRD exploits various fine-grained user features from the diffusion threads, i.e., the users' social homophily, influence, susceptibility, temporal features, and then uses these features to determine whether the information is true or false. PLRD also introduces a variational autoencoder (VAE) to handle the uncertainty which exists in the feature learning phase. UMLARD extends PLRD and solves one burning limitation left by PLRD, i.e., input features entangled with learned high-level features, by using three view-specific embedding methods with distinct inputs. UMLARD also innovatively proposes a capsule-based attention layer to replace the original attention mechanism in PLRD, which is more effective in both performance and time cost. Both PLRD and UMARD outperform other non-user-level models, which demonstrates that developing models at the user level indeed improves detection performance.

Nederlandse Samenvatting

Met de opkomst van online sociale netwerken (OSNs) is de manier waarop mensen informatie delen veranderd, wat sneller en breder wordt dan traditionele media. Begrijpen hoe informatie (zowel goede als schadelijke) zich via OSNs verspreidt, en welke elementen het succes van informatieverspreiding stimuleren, heeft belangrijke implicaties voor een breed scala aan toepassingen in de echte wereld. In dit proefschrift doen we onderzoek om de diffusie van informatie in OSNs te leren kennen door gebruik te maken van Deep Representation Learning. Concreet willen we twee specifieke taken oplossen, namelijk het modelleren van informatiecascades en het detecteren van geruchten.

We dragen het volgende bij:

We introduceren recurrent cascades convolutional network (CasCN) in Hoofdstuk 4, een op grafieken gebaseerd neuraal model voor voorspelling van informatiecascades op macroniveau. CasCN gebruikt een combinatie van een graafconvolutienetwerk en een terugkerend neuraal netwerk om de incrementele grootte van een cascade te voorspellen door structurele en tijdelijke kenmerken te extraheren uit een reeks op tijdstempels gebaseerde subgrafieken. CasCN introduceert ook CasLaplace voor gerichte grafieken, waarmee de beperkingen van eerdere neurale netwerken voor graafgrafieken worden overwonnen bij het omgaan met gerichte grafieken. De experimentele resultaten die zijn uitgevoerd op twee real-world datasets, laten zien dat CasCN zeer geschikt is voor het modelleren van structurele-tijdelijke kenmerken van cascades.

We introduceren multi-scale cascades model (MUCas) in Hoofdstuk 5, die gericht is op het vastleggen van multi-scale functies voor voorspelling van informatiecascades op macroniveau. MUCas gebruikt multi-scale graph capsule network en influence attention om de multi-scale informatie te leren en samen te smelten om een unieke cascade representatie. MUCas verbetert ook de bemonsteringsmethoden in CasCN door subgrafieken te bemonsteren op basis van tijdsintervallen in plaats van tijdstempels. We voeren experimenten uit met real-world datasets en tonen aan dat MUCas bijzonder effectief is in het extraheren van kenmerken op cascades van verschillende schalen, en multi-scale functies zijn van vitaal belang voor het verbeteren van de nauwkeurigheid van voorspellingen.

We introduceren macroscopic and microscopic-aware rumor detection model (MMRD) in Hoofdstuk 6, een op diffusie gebaseerd geruchtendetectie model dat geruchten detecteert door alleen verschillende niveaus van diffusie patronen te onderzoeken. MMRD maakt gebruik van graph neural networks om de macroscopische diffusie van geruchten propagatie te leren en microscopische diffusie patronen vast te leggen met behulp van bidirectional recurrent neural networks, rekening houdend met de user-time serie. MMRD maakt ook gebruik van de knowledge distillation techniek om een meer informatief studentenmodel te creëren en de modelprestaties verder te verbeteren. MMRD wordt geëvalueerd op de bekende Twitter-datasets en zou goede detectieresultaten kunnen opleveren in het vroege stadium van geruchtenverspreiding.

We introduceren participant-level rumor detection model (PLRD) en multi-view learning with attention for rumor detection model (UMLARD) in Hoofdstuk 7. Zowel PLRD als UMLARD zijn ontworpen op participant-level. PLRD maakt gebruik van verschillende fijnmazige gebruikersfuncties van de diffusiethreads, i.e., de sociale homofilie, invloed, gevoeligheid, temporele kenmerken van de gebruikers, en gebruikt deze kenmerken vervolgens om te bepalen of de informatie waar of onwaar is. PLRD introduceert ook een Variational Autoencoder (VAE) om de onzekerheid aan te pakken die bestaat in de leerfase van functies. UMLARD breidt PLRD uit en lost één brandbeperking op die is achtergelaten door PLRD, i.e., invoerfuncties verstrengeld met geleerde functies op hoog niveau, door gebruik te maken van drie view-specific inbeddingsmethoden met verschillende invoer. UMLARD stelt ook op innovatieve wijze een capsule-based attention layer voor ter vervanging van het oorspronkelijke attention mechanism in PLRD, dat effectiever is in zowel prestaties als tijdskosten. Zowel PLRD als UMARD presteren beter dan andere non-user-level modellen, wat aantoont dat het ontwikkelen van modellen op user-level inderdaad de detectieprestaties verbetert.

Acknowledgements

This thesis would not have been possible without the help of everyone who has joined me on my PhD journey. So now I will express my deepest gratitude to them.

Foremost, I would like to thank my promoters, Prof. Marcello Bonsangue and Prof. Fengli Zhang. Really appreciate them giving me an opportunity to pursue my doctoral degree at Leiden University and the University of Electronic Science and Technology of China. Thank them for giving me large freedom through my PhD study, allowing me to conduct research that I really interested in, and helping me without hesitation when I felt confused or faced difficulties. I'm very fortunate to have Marcello and Fengli as my PhD promoters, from whom I learned not only how to explore and solve research problems, but also how to maintain patience and optimism attitude in the face of difficulties.

Dr. Fan Zhou, my co-supervisor, deserves special gratitude. Thank you for your patience, understanding, and guidance, as well as your trust and encouragement. When I started my PhD journey, you told me that you would always help me without hesitation whenever and wherever I needed it. These sweet words supported me a lot when I was facing difficulties doing research, and I will treasure them forever.

I also want to express my thankfulness to my co-authors, Dr. Goce Trajcevski, Dr. Kunpeng Zhang, and Dr. Ting Zhong. Thanks to their positive feedback and proofreading. Without their help, my papers would not have been accepted so successfully.

I would like to thank my committee members, Prof. Ashfaq Khokhar, Prof. Aske Plaat, Prof. Fons Verbeek, Dr. Suzan Verberne, and Dr. Elvin Isufi, for their helpful suggestions and insightful comments.

Also, my friends and colleagues supported and encouraged me during my PhD study, and they deserve special thanks. First, thanks to my colleagues at Leiden University and UESTC: Xu Jing, Qiang Gao, Jianfei, Zeng Xu, Nan Liu, Hui Feng, and Wenjing, who work with me give me substantial support and urge me to do better in research. Then I would like to thank my lovely friends: Ke Gong, Yin Pu, Xingyu, Lijunnan, Ming Ren, Fang Liu, Yuhua, Zijing, Xiaoyan, Yizhen, Yufan, Qiqi, and Xiaoyao, who accompany me, support me, help me, and share with me my worries, frustrations, and happiness, especially during the dark period of coronavirus.

Finally, I want to express my deepest love and heartfelt gratitude to my parents–Wensheng Chen and Chunhong Peng, as well as my grandparents–Xiuming Chen and Qirong Luo, who have always been supportive of my every decision. They provide me with the most tremendous strength to pursue my dreams.

Curriculum Vitae

Xueqin Chen was born on 28th July 1993 in Mianyang, Sichuan, People's Republic of China. He received his Bachelor degree at Dalian Neusoft University of Information in June 2015. In the same year, he studied as a Master student in Software Engineering under the supervision of Prof. Fengli Zhang at University of Electronic Science and Technology of China (UESTC). In 2017, he was allowed to participate in a Master-Doctor Programme, and started his PhD study in September 2017. From September 2017 to October 2019, he studied at UESTC, under the supervision of Prof. Fengli Zhang and Dr. Fan Zhou. In 2019, he obtained a grant from China Scholarship Council (CSC), then participate in a joint PhD training programme at the Leiden Institute of Advanced Computer Science (LIACS), Leiden University, and under the supervision of Prof. Marcello Bonsangue.

His research broadly spans applied deep learning and social computing, with a focus on information diffusion modeling, graph representation learning, and fake news detection. He has published several papers in international conferences and journals, including ICDE, SIGIR, International Journal of Intelligent Systems, Knowledge-based Systems, and Information Processing & Management. Also, during his PhD, he has served as an external reviewer for some conferences, as well as an invited reviewer for some journals, such as KDD, ACL, SIGIR, BigData, and International Journal of Intelligent Systems, etc.