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

Chen, X.

Chapter 3

Preliminaries

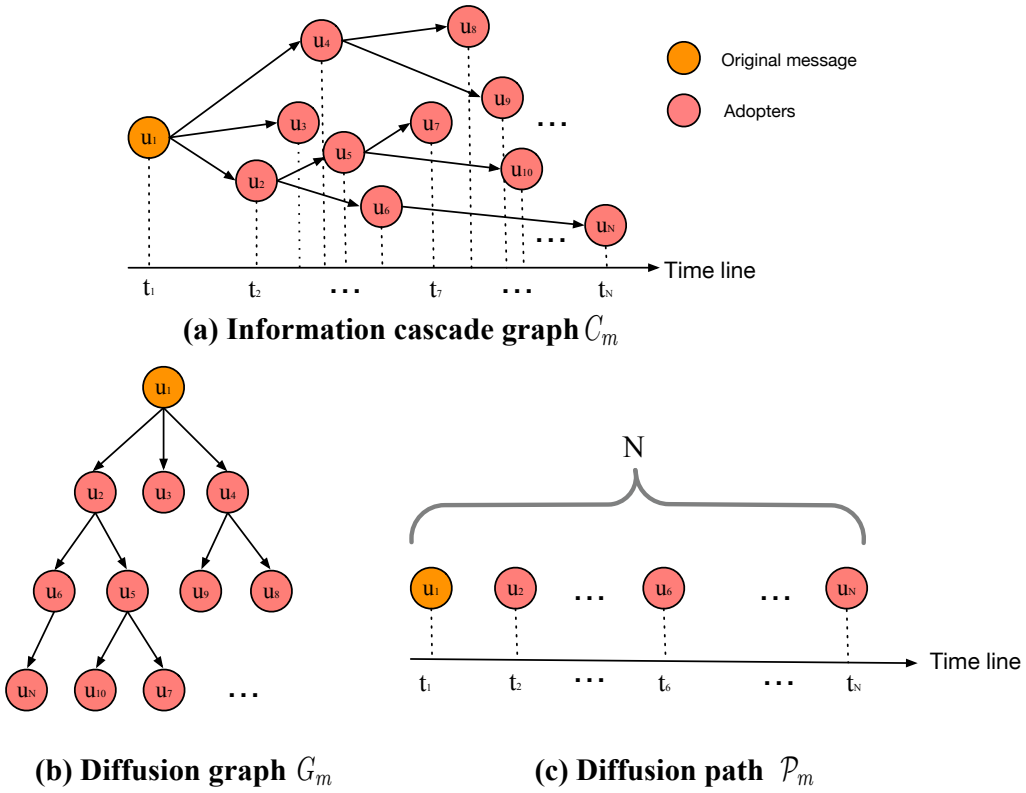
In this chapter, we introduce general definitions about information cascades, the basic formal problem statements forming the core of our research, and few technical concepts needed in the rest of the thesis. Table 3.1 summarizes the main notations used in this chapter.

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:

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

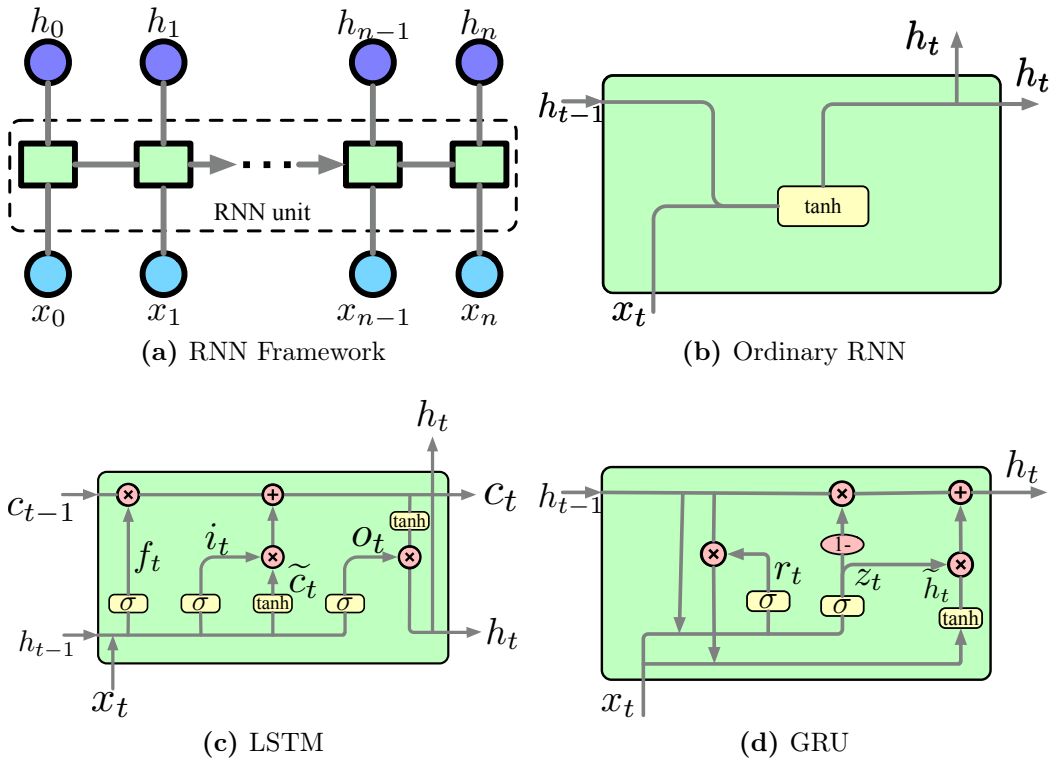
where $\mathbf{r}_t$ and $\mathbf{z}_t$ are reset gate and update gate, respectively. The reset gate $\mathbf{r}_t$ decides how the input $\mathbf{x}_t$ and previous memory $\mathbf{h}_{t-1}$ are combined, and the update gate $\mathbf{z}_t$ controls how much information from the previous memory and candidate 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].