저작자표시-비영리-변경금지 2.0 대한민국

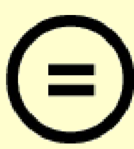
이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.


변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

A Study on Recommendation Systems Based on Time-Series GNN

Suji Lee

Department of Statistics

The Graduate School of Sungshin Women's University

A Study on Recommendation Systems Based on Time-Series GNN

A Master's Thesis
Submitted to the
Graduate School of Sungshin Women's University

in partial fulfillment of the requirements
for the degree of
Master of Statistics

Suji Lee

November, 2024

This is to certify that we have examined the

Master's Thesis of

Suji Lee

Submitted to Department of Statistics

Approved as to style and content:

Thesis Advisor	Hohyun Jung
Committee Chairman	Seongoh Park
Committee Member	Dongha Kim
Committee Member	Frederick Kin Hing Phoa

The Graduate School of Sungshin Women's University

Abstract

While LightGCN, a GCN model optimized for recommendation systems, has been developed, a specialized model for recommendation systems based on time-series GCN has not yet been proposed. LightGCN performs recommendations using only user-item interactions at a single time point. In response, we propose Evolving LightGCN (ELGCN), which extends LightGCN by incorporating a temporal update mechanism using GRU. In ELGCN, the initial embedding layer is updated over time using GRU, and each time point's layer is further updated using the LightGCN architecture. The final embedding is computed as the average of all embedding layers across time. The link connection probability between users and items is then calculated as the dot product of their final embedding vectors. We evaluated the proposed model using the MovieLens and Netflix Prize datasets and conducted a comparative study against temporal versions of GCN, LightGCN, and the EvolveGCN models. The results indicate that the proposed model consistently outperformed the others across both datasets.

Keywords : Recommendation system, Graph Neural Network, Time Series model, user-item interactions

Table of Contents

Table of Contents iii
List of Figures iv
List of Tables v

I. Introduction **1**

II. Related Works **4**

III. Preliminaries **6**
3.1 Notations 8
3.2 Graph Convolutional Network 9
3.3 LightGCN 11
3.4 EvolveGCN 13

IV. Proposed Method **18**
4.1 ELGCN 18
4.2 Loss Function 22

V. Experiments **24**
5.1 Dataset 24
5.2 Comparative Models 25
5.3 Performance Metrics 25
5.4 Experiment setting 28
5.5 Results 28

VI. Conclusion **31**

References **33**

List of Figures

Figure 1. An example graph illustrating the connections between nodes A to F . 7

Figure 2. An example directed graph illustrating the directional connections between nodes A to F, including self-loops 8

Figure 3. ELGCN architecture illustrating the temporal update process using GRU layers, followed by layer aggregation for final embeddings. . . . 19

Figure 4. Example of the ELGCN layer update process, illustrating the initial embeddings for users (u_1, u_3, u_4) and items (i_1, i_6) at the 0th layer . . 21

List of Tables

Table 1. The corresponding adjacency matrix representing the connections in the example graph 7

Table 2. The corresponding adjacency matrix for the directed graph, reflecting the directional edges and self-loops shown in Figure 2 8

Table 3. Notation used throughout the paper 8

Table 4. Example node feature matrix, where each row represents a node (A to F) and each column (f_1 to f_5) represents a feature 10

Table 5. *Recall@K*, *NDCG@K* ($K = 10, 20, 30$) scores for the Netflix Prize and MovieLens 25M datasets. The best performance is highlighted in **bold**, and the second-best performance is underlined. 29

Chapter 1

Introduction

The rapid advancement of internet technology has transformed the paradigm of how consumers access information. In the early days of the internet, limited information was provided in a static form (Cerf, 2004). However, with the development of high-speed networks, cloud computing, and data storage technologies, a vast amount of data began to be generated and distributed in real time (Leiner et al., 2009; Campbell-Kelly and Garcia-Swartz, 2013). As a result, the scope and speed of information access for consumers have expanded exponentially. Consumers can now search for and utilize necessary information instantly, without spatial or temporal constraints (Campbell-Kelly and Garcia-Swartz, 2013). Despite this, consumers have faced significant challenges in locating desired information within the massive volume of data. To address this issue, recommender systems were introduced (Adomavicius and Tuzhilin, 2005). These systems have allowed users to receive personalized recommendations with minimal effort. Consequently, extensive research has been conducted to enhance the performance of recommender systems (Ricci et al., 2021).

Researchers have made significant progress in developing efficient recommender models. LightGCN (He et al., 2020) stands out as a highly effective static recommender system that optimizes user-item interactions. However, LightGCN focuses only on static environments and depends solely on user interactions at a single point in time. It overlooks the temporal dynamics of user preferences, which often change over time due to shifting interests, seasonal trends, or evolving needs.

These limitations highlight a significant gap in current recommender systems. User preferences are not static. They evolve in response to new content, emerging trends, or ex-

ternal influences. Static models like LightGCN fail to fully capture these temporal changes, which can result in outdated or less relevant recommendations. Addressing this issue requires models that can dynamically adapt to changes over time and reflect the evolving nature of user behavior.

To address these limitations, we propose an advanced version of LightGCN that incorporates temporal updates to account for changes and trends over time. Therefore, we developed the Evolving LightGCN (ELGCN) model that integrates temporal dynamics into the original framework. This allows the model to adapt to users' shifting preferences and improve the accuracy and relevance of recommendations. It retains the strengths of LightGCN in capturing static user-item interactions while also leveraging sequential data to deliver more timely and personalized recommendations. This work addresses a critical challenge in the recommender system field, paving the way for adaptive and robust models in dynamic environments.

This study identifies three key contributions:

- **Incorporating a GRU-based temporal update mechanism**
  ELGCN addresses the limitations of static models like LightGCN by introducing a GRU-based temporal update mechanism. This enables the model to effectively capture the temporal evolution of user preferences, making it suitable for dynamic recommendation scenarios.

- **Proposing a novel embedding aggregation method combining temporal and layer-wise updates**
  ELGCN independently updates embeddings for each time step using LightGCN and then aggregates the embeddings across all time steps and layers through averaging. This method allows the model to comprehensively reflect both temporal patterns and structural relationships, enhancing recommendation accuracy.

- **Demonstrating superior performance compared to existing models**
  ELGCN consistently outperforms baseline models, GCN, LightGCN, and EvolveGCN, in Recall and NDCG metrics across the Netflix Prize and MovieLens 25M datasets.

The organization of this paper is outlined below.
Chapter 2 reviews traditional and graph-based recommendation system models and discussed thier advantages and limitations. Chapter 3 introduces the essential concepts, notations and foundatinal models such as GCN, LightGCN, and EvolveGCN to provide the necessary background for understanding the proposed method. Chapter 4 presents the proposed Evolving LightGCN (ELGCN) model, detailing its GRU-based temporal update mechanism, layer propagation process, embedding aggregation strategy, and the loss function. Chapter 5 describes the experimental setup, including datasets, evaluation metrics, and comparative models and provides a detailed analysis of the results. Finally, Chapter 6 concludes the paper by summarizing the main contributions, discussing their implications, and suggesting directions for future research.

Chapter 2

Related Works

Early recommendation systems relied on rule-based systems to deliver content that matched consumer preferences. These systems employed simple algorithms using static filtering methods. By analyzing past behavior patterns, they recommended content with similar characteristics (Adomavicius and Tuzhilin, 2005; Ricci et al., 2021). A rule-based system operates on 'if-then' rules to make decisions or draw inferences (Hayes-Roth, 1985). This approach offers the advantage of transparency and predictability in decision-making because it depends on explicitly defined rules. However, it faces limitations when addressing complex problems, as it is impractical to define every condition through rules. Moreover, it lacks flexibility during the reasoning process (Buchanan and Shortliffe, 1984; Grosan et al., 2011).

Subsequently, the introduction of collaborative filtering marked a significant shift in the approach to recommendation systems (Adomavicius and Tuzhilin, 2005; Ricci et al., 2021). Unlike rule-based systems, collaborative filtering analyzes not only an individual user's behavior but also the patterns of other users to deliver personalized recommendations (Sarwar et al., 2001). This method considers popular items within groups of users who exhibit similar preference patterns (Linden et al., 2003). By leveraging similarities among users, it became possible to generate more accurate recommendation results (Koren et al., 2009).

In recent years, the development of recommendation systems utilizing Graph Neural Network (GNN) and deep learning models has gained significant attention (Wang et al., 2019; Zhang et al., 2019b). These deep learning-based approaches have addressed the limi-

tations of earlier recommendation methodologies. Traditional recommendation systems analyzed user-item interactions using flat matrices, which failed to capture indirect relationships and complex network structures (Wang et al., 2019; Wu et al., 2019). In contrast, graph-based systems can identify not only direct user-item connections but also indirect associations through multi-hop links (Ying et al., 2018).

Moreover, traditional collaborative filtering methods relied on matrix factorization to extract latent factors of users and items. However, these approaches were insufficient in handling data sparsity and capturing non-linear user preferences (He et al., 2017). By employing deep learning models, it became possible to learn user-item interactions more effectively using non-linear activation functions and multi-layer neural networks (Zhang et al., 2019b; Covington et al., 2016).

The advancement of recommendation systems has significantly improved recommendation performance, enabling the delivery of personalized content to consumers. This progress effectively addresses the issue of information overload, helping to reduce search costs and save time. Additionally, it goes beyond merely suggesting previously preferred items. It also introduces new items and content, thereby enriching the consumer's experience with a wider range of choices.

Chapter 3

Preliminaries

A graph is a structure composed of points (nodes) and lines (edges) that represent relationships among these points (Zhou et al., 2020). This structure can describe various phenomena, such as the relationships between atoms in molecules, cities within transportation networks, and users within social networks (Bondy et al., 1976). In general, Euclidean data structures have fixed dimensions. In contrast, non-Euclidean data structures, like graphs, possess dynamic dimensions (Zhang et al., 2019a). Therefore, conventional Artificial Neural Networks, which assume inputs in a vector format, are not well-suited to handle these graph structures (Zhou et al., 2020).

GNNs are designed to model and learn the relationships between nodes within a graph. This allows for the prediction and representation of the graph's characteristics. These capabilities provide new perspectives for data analysis, driving advancements in analytical methods. For instance, GNNs have overcome the challenges of traffic prediction by handling the complex spatial dependencies in road networks and the non-linear temporal dynamics involved (Li et al., 2017; Yu et al., 2017).

Additionally, GNNs effectively utilize graph attributes and structural information for anomaly detection by learning and scoring node relationships (Kim et al., 2022; Wu et al., 2021). Research has also been conducted to enhance the interpretability of predictions made by GNN-based models (Yuan et al., 2022; Ying et al., 2019). In this section, we will provide a detailed explanation of GNNs, which form the foundation of the proposed method.

Before diving into further details, let's explain the concept of a graph structure with a simple example. Consider Figure 1. In general, a graph is defined as $G = (V, E)$, where V

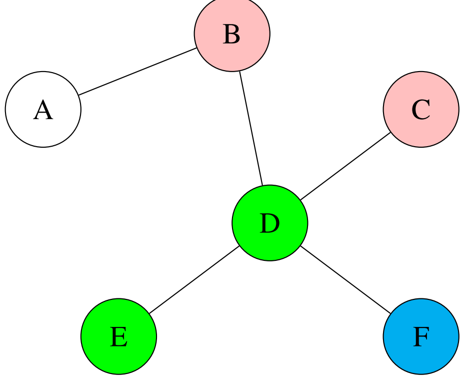


Figure 1: An example graph illustrating the connections between nodes A to F

	A	B	C	D	E	F
A						
B	1					
C	0	0				
D	0	1	1			
E	0	0	0	1		
F	0	0	0	1	0	

Table 1: The corresponding adjacency matrix representing the connections in the example graph

denotes the set of nodes, such that $V = \{A, B, \cdots, F\}$, and E represents the edges connecting pairs of nodes. Thus, the graph shown in Figure 1 can be defined as:

$$G = (\{A, B\}, \{B, D\}, \cdots, \{D, F\})$$

Graph structures are typically represented using an adjacency matrix. An adjacency matrix is a square matrix that indicates which nodes are connected by edges. Representing Figure 1 as an adjacency matrix results in Table 1. The size of the adjacency matrix is defined as (the number of nodes) $\times$ (the number of nodes), which in this case is 6×6.

If two nodes are connected, the corresponding entry is 1; if not, it is 0. Since nodes are not connected to themselves, the diagonal elements are all zeros. Because this is an undirected graph, the adjacency matrix is symmetric.

In contrast, Figure 2 shows a directed graph where edges have directions, and there are self-loops on nodes A and E. Due to this graph structure, the diagonal entries for A and E in Table 2 are set to 1, and the symmetry property of the adjacency matrix is lost.

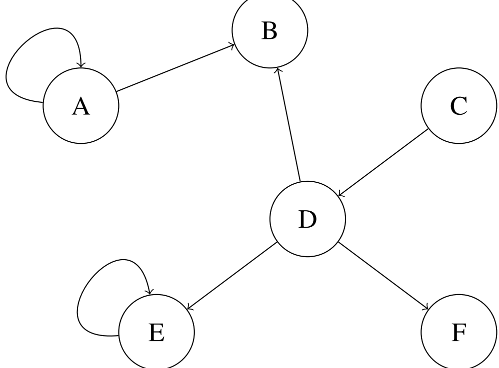


Figure 2: An example directed graph illustrating the directional connections between nodes A to F, including self-loops

	A	B	C	D	E	F
A	1	1	0	0	0	0
B	0	0	0	1	0	0
C	0	0	0	1	0	0
D	0	1	0	0	1	1
E	0	0	0	0	1	0
F	0	0	0	0	0	0

Table 2: The corresponding adjacency matrix for the directed graph, reflecting the directional edges and self-loops shown in Figure 2

3.1 Notations

To ensure consistency in explaining the GNN models and the proposed model, the notations that will be used throughout the following sections are organized below.

Notation	Description
u	users, $u = 1, 2, \cdots, U$
i	items, $i = 1, 2, \cdots, I$
k	lightGCN layers, $k = 0, 1, \cdots, K$
t	time points, $t = 1, 2, \cdots, T$
$e_{t,u}^{(k)}$	the embedding of user u in k^{th} layer at time t
$e_{t,i}^{(k)}$	the embedding of item i in k^{th} layer at time t
$E_t^{(k)}$	the kth embedding matrix at time t; $[e_{t,1}^{(k)}, \cdots, e_{t,U}^{(k)}, e_{t,1}^{(k)}, \cdots, e_{t,I}^{(k)}]$
$y_{t,u,i} \in \{0, 1\}$	user-item interaction is marked as 1 if present, otherwise 0 at time t
$\mathcal{N}_{t,u} = \{i : y_{t,u,i}{=}1\}$	the set of items that are interacted by user u at time t
$\mathcal{N}_{t,i} = \{u : y_{t,u,i}{=}1\}$	the set of users that are interacted by item i at time t

Table 3: Notation used throughout the paper

3.2 Graph Convolutional Network

The Graph Convolutional Network (GCN) (Kipf and Welling, 2016) is an adaptation of the Convolutional Neural Network (CNN) (LeCun et al., 1998) architecture applied to graph structures. CNNs were originally designed to effectively process image data. In image data, the spatial positions of pixels carry significant information, making it crucial to consider the data structure. CNNs utilize convolution layers to effectively capture the positional information of pixels, thereby enhancing classification and prediction performance (Voulodimos et al., 2018).

Similarly, in both images and graphs, it is essential to account for the data structure. This need led to research focused on generalizing CNNs for graph data (Hechtlinger et al., 2016), which eventually resulted in the development of GCNs.

In GCNs, node features are taken into account and thus a graph is redefined as follows:

$$G = (A, X), \tag{3.1}$$

where $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix, and N represents the total number of nodes. The matrix $X \in \mathbb{R}^{N \times d}$ denotes the node feature matrix, where d is the number of node features.

Assume that the color of each node represents a node feature in Figure 1. Let the number of node features including the node color be $d = 5$. In this case, a node feature matrix like Table 4 can be generated.

When the inputs A and X are provided, each convolution layer updates the latent node feature matrix $E \in \mathbb{R}^{N \times f}$ according to Eq. (3.2). Here, f refers to the dimension of the latent feature vectors, which can be considered analogous to the number of filters in a CNN.

$$E^{(l+1)} = \sigma\left(\widehat{A}E^{(l)}W^{(l)} + b^{(l)}\right) \tag{3.2}$$

	f_1	f_2	f_3	f_4	f_5
A	0	0	0	0	0
B	1	0	0	1	1
C	1	0	0	1	1
D	0	1	0	1	1
E	0	1	0	1	1
F	0	0	1	1	1

Table 4: Example node feature matrix, where each row represents a node (A to F) and each column (f_1 to f_5) represents a feature

In this equation, $\widehat{A}$ is the normalized adjacency matrix, defined as follows:

$$\widehat{A} = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}, \quad \tilde{A} = A + I, \quad \tilde{D} = \text{diag}\left(\sum_j \tilde{A}_{ij}\right)$$

Here, $\tilde{A}$ is the adjacency matrix A with added self-loops. While A reflects only the connections between neighboring nodes, this adjustment ensures that information about each node itself is also incorporated during the graph convolution process. And $\tilde{D}$ is the degree matrix of $\tilde{A}$, with its diagonal elements indicating the degree of each node.

By multiplying $\tilde{A}$ on both sides with $\tilde{D}^{-1/2}$, the computed features are normalized with respect to node connectivity. This normalization stabilizes backpropagation during the graph convolution process. Nodes with many neighbors may end up with disproportionately large feature representations without this normalization leading to the issue of gradient explosion during layer updates. Conversely, nodes with few neighbors may suffer from gradient vanishing.

The matrix $W \in \mathbb{R}^{d \times f}$ is a learnable weight matrix, and training the GCN is essentially equivalent to adjusting this weight matrix. The function σ denotes a non-linear activation function, commonly using sigmoid or ReLU functions.

3.3 LightGCN

LightGCN (He et al., 2020) is a model optimized for recommendation systems by removing feature transformation and non-linear activation functions from GCN. Originally, GCN was proposed for efficient node classification, assuming that graphs contained rich node features. However, the relationships between users and items play a more critical role in recommendation systems.

Before LightGCN, the Neural Graph Collaborative Filtering (NGCF) model (Wang et al., 2019) adopted a GCN structure that included both feature transformation and non-linear activation functions for item recommendation. However, experiments demonstrated that removing these two components actually led to better recommendation performance compared to retaining them (He et al., 2020). As a result, LightGCN was proposed as a lightweight recommendation system model, focusing solely on neighborhood aggregation.

In LightGCN, the user and item embeddings are updated as a weighted sum for each embedding, using a normalized constant as shown in Eq. (3.3).

$$\begin{aligned} e_u^{(k+1)} &= \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} e_i^{(k)} \\ e_i^{(k+1)} &= \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} e_u^{(k)} \end{aligned} \tag{3.3}$$

Here, u and i denote the user and item, respectively. The set $\mathcal{N}_u$ represents the items adjacent to user u, while $\mathcal{N}_i$ represents the users adjacent to item i. The term $e_u^{(k)}$ denotes the kth embedding layer of user u, and $e_i^{(k)}$ denotes the kth embedding layer of item i.

By multiplying the embeddings by the normalized constant $(\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|})^{-1}$, the model prevents the embedding values from becoming excessively large for nodes with a high number of neighbors during the graph convolution process. This normalization helps control the influence of densely connected nodes.

Given U users and I items, the user-item interaction matrix can be defined as $R \in \mathbb{R}^{U \times I}$, where the entries of R consist of 0s and 1s. The adjacency matrix can then be expressed as:

$$A = \begin{bmatrix} 0 & R \\ R^T & 0 \end{bmatrix}$$

Here, R represents the user embedding matrix, while R^T denotes the item embedding matrix. Let $E^{(k)} \in \mathbb{R}^{(U+I) \times d}$ denote the kth embedding layer matrix. In this context, Eq. (3.3) can be reformulated as shown in Eq. (3.4).

$$E^{(k+1)} = D^{-1/2} A D^{-1/2} E^{(k)} \tag{3.4}$$

Here, $D \in \mathbb{R}^{(U+I) \times (U+I)}$ is a diagonal matrix where each diagonal element indicates the number of adjacent nodes for each user and item.

In LightGCN, the model starts with an initial embedding layer (0th layer) and updates the embeddings according to Eq. (3.3). Therefore, the trainable parameters in LightGCN are the initial embeddings, $e_u^{(0)}$ for users and $e_i^{(0)}$ for items. After updating all K embedding layers, their representations are combined using a weighted sum as shown in Eq. (3.5).

$$e_u = \sum_{k=0}^{K} \alpha_k e_u^{(k)}, \quad e_i = \sum_{k=0}^{K} \alpha_k e_i^{(k)} \tag{3.5}$$

The weight α_k represents the contribution of the kth embedding layer to the final embedding. It is generally set to $\alpha_k = 1/(K+1)$, which has been shown to enhance performance (He et al., 2020).

Expanding Eq. (3.5) based on Eq. (3.4), we obtain the following:

$$\begin{aligned} E &= \alpha_0 E^{(0)} + \alpha_1 E^{(0)} + \cdots + \alpha_K E^{(K)} \\ &= \alpha_0 E^{(0)} + \alpha_1 \tilde{A} E^{(0)} + \cdots + \alpha_K \tilde{A}^K E^{(0)}, \quad \tilde{A} = D^{-1/2} A D^{-1/2} \end{aligned}$$

It can be seen that once the initial embedding matrix $E^{(0)}$ is define, the final embedding can be obtained by repeatedly multiplying with the powers of $\tilde{A}$.

3.4 EvolveGCN

Both GCN and LightGCN are designed for static graphs, where nodes and edges remain unchanged over time. However, reflecting changes in the appearance of new nodes and evolving connections between nodes over time is a more natural way to represent real-world scenarios. Dynamic graphs can capture richer, time-varying information in various contexts. For instance, in social networks, they can represent changes in friendships, post sharing, and comment interactions. In the financial sector, they can reflect evolving transaction relationships between users. In recommendation systems, they can capture changes in users' item preferences or viewing histories over time.

EvolveGCN (Pareja et al., 2020) extends the GCN framework to dynamic environments by using a Recurrent Neural Network (RNN) for updating the GCN parameters. When nodes frequently appear or disappear over time, the node embedding approach can destabilize the model during the RNN training process. To address this issue, EvolveGCN updates the entire GCN model itself at each time step using an RNN.

Given its temporal nature, EvolveGCN redefines Eq. (3.2) as shown in Eq. (3.6).

$$\begin{aligned} H_t^{(l+1)} &= \mathrm{GCONV}\left(A_t, H_t^{(l)}, W_t^{(l)}\right) \\ &= \sigma\left(\widehat{A}_t H_t^{(l)} W_t^{(l)}\right) \end{aligned} \tag{3.6}$$

At each time step t, $\widehat{A}_t$ represents the normalized adjacency matrix, $H_t^{(l)}$ denotes the node embedding matrix serving as the input, and $W_t^{(l)}$ is the weight matrix.

The core of EvolveGCN lies in updating the GCN parameters, specifically $W_t^{(l)}$, using two distinct methods. The first method treats $W_t^{(l)}$ as a hidden state and updates it using a

Gated Recurrent Unit (GRU) (Cho, 2014). In this case, the input information is set as the node embedding $H_t^{(l)}$. This approach is implemented in Step 2 of Algorithm 1.

The second method treats $W_t^{(l)}$ as both input and output, updating it with a Long Short-Term Memory (LSTM) network (Hochreiter, 1997). Unlike the previous approach, it avoids using node embeddings, which can destabilize the model during the RNN training process. This method is implemented in Step 1 of Algorithm 2.

Algorithms 1 and 2 present the entire process in pseudocode, covering both the update of $W_t^{(l)}$ using RNNs and the subsequent embedding layer update based on the newly updated $W_t^{(l)}$.

Although Algorithm 1 uses a standard GRU, several modifications have been applied. The first modification involves arranging the input and hidden state as column vectors side by side to form a matrix. The second modification aligns the column dimension of the input with that of the hidden state. This adjustment is implemented in Step 1 (Cangea et al., 2018).

The goal of Step 1 is to summarize the columns of $H_t^{(l)}$ into representative vectors that match the number of columns in $W_{t-1}^{(l)}$. To achieve this, it calculates the projection score of $H_t^{(l)}$ using a time-independent, trainable vector p. Based on these scores, it selects the indices of the top $\#\mathrm{col}(W_{t-1}^{(l)})$ scores. The hyperbolic tangent of the scores for these selected indices is then multiplied by $H_t^{(l)}$ to generate new embedding vectors.

In Step 2 of Algorithm 1, the previously computed embedding vector $\mathrm{emb}_t^{(l)}$ and the weight matrix $W_{t-1}^{(l)}$ are used as inputs to the GRU to compute $W_t^{(l)}$. In line 8, the update gate determines the proportion of past and present information to incorporate. Specifically, $\mathrm{update}_t^{(l)}$ controls how much of the current information should be utilized.

In line 9, the reset gate uses the previous hidden state and the current embedding vector to decide how much of the previous hidden state information to retain. It outputs a value between 0 and 1, where a higher value indicates more information to be preserved. This value is then multiplied with the hidden state $W_{t-1}^{(l)}$ in line 10.

Algorithm 1 EvolveGCN-H Layer

Input : Adjacency matrix A_t, hidden state $H_t^{(l)}$, weight matrix from previous time step $W_{t-1}^{(l)}$

Output : Updated hidden state $H_t^{(l+1)}$, updated weight matrix $W_t^{(l)}$

1: **EvolveGCN-H Update**

2: $[H_t^{(l+1)}, W_t^{(l)}] = \text{EGCN-H}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$

3: **Step 1: Summarization**

4: $\quad \text{score}_t^{(l)} = H_t^{(l)} p \, / \, ||p||$

5: $\quad \text{index}_t^{(l)} = \text{top_indices}(\text{score}_t^{(l)}, \#col(W_{t-1}^{(l)}))$

6: $\quad \text{emb}_t^{(l)} = [H_t^{(l)} \cdot \tanh(\text{score}_t^{(l)})]^T_{\text{index}_t^{(l)}}$

7: **Step 2: GRU-based Weight Update**

8: $\quad \text{update}_t^{(l)} = \text{sigmoid}(W_z \text{emb}_t^{(l)} + U_z W_{t-1}^{(l)} + B_z)$

9: $\quad \text{reset}_t^{(l)} = \text{sigmoid}(W_r \text{emb}_t^{(l)} + U_r W_{t-1}^{(l)} + B_r)$

10: $\quad \tilde{W}_t^{(l)} = \tanh(W_h \text{emb}_t^{(l)} + U_h(\text{reset}_t^{(l)} \cdot W_{t-1}^{(l)}) + B_h)$

11: $\quad W_t^{(l)} = (1 - \text{update}_t^{(l)}) \cdot W_{t-1}^{(l)} + \text{update}_t^{(l)} \cdot \tilde{W}_t^{(l)}$

12: **Step 3: Graph Convolutional Update**

13: $\quad H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$

14: **return** $H_t^{(l+1)}$, $W_t^{(l)}$

After processing both gates, line 11 aggregates the results to compute the updated hidden state $W_t^{(l)}$. In Step 3, the current adjacency matrix, the updated weight matrix $W_t^{(l)}$ from Step 2, and the current embedding vector $H_t^{(l)}$ are used to compute the embedding vector $H_{t+1}^{(l)}$ for the next time step.

The EvolveGCN-O version performs updates using only the hidden state information. As a result, it only requires expanding the vector into a matrix format. Thus, unlike the previous approach, it does not involve a summarization process. Instead, it directly applies the LSTM to the hidden state for updates.

Algorithm 2 EvolveGCN-O Layer

Input : Adjacency matrix A_t, hidden state $H_t^{(l)}$, weight matrix from previous time step $W_{t-1}^{(l)}$

Output : Updated hidden state $H_t^{(l+1)}$, updated weight matrix $W_t^{(l)}$

1: **EvolveGCN-O Update**

2: $[H_t^{(l+1)}, W_t^{(l)}] = \text{EGCN-O}(A_t, H_t^{(l)}, W_{t-1}^{(l)})$

3: **Step 1: LSTM-based Weight Update**

4: $H_t^{(l)} = W_{t-1}^{(l)}$

5: $\text{forget}_t^{(l)} = \text{sigmoid}(W_f H_t^{(l)} + U_f W_{t-1}^{(l)} + B_f)$

6: $\text{input}_t^{(l)} = \text{sigmoid}(W_i H_t^{(l)} + U_i W_{t-1}^{(l)} + B_i)$

7: $\text{output}_t^{(l)} = \text{sigmoid}(W_o H_t^{(l)} + U_o W_{t-1}^{(l)} + B_o)$

8: $\tilde{\text{input}}_t^{(l)} = \tanh(W_c H_t^{(l)} + U_c W_{t-1}^{(l)} + B_c)$

9: $C_t^{(l)} = \text{forget}_t^{(l)} \cdot C_{t-1}^{(l)} + \text{input}_t^{(l)} \cdot \tilde{\text{input}}_t^{(l)}$

10: $W_t^{(l)} = \text{output}_t^{(l)} \cdot \tanh(C_t^{(l)})$

11: **Step 2: Graph Convolutional Update**

12: $H_t^{(l+1)} = \text{GCONV}(A_t, H_t^{(l)}, W_t^{(l)})$

13: **return** $H_t^{(l+1)}$, $W_t^{(l)}$

In Algorithm 2, the input is $W_{t-1}^{(l)}$, which serves as both the node embedding and hidden state, as indicated in line 5. Let's examine how the hidden state is updated within the LSTM.

In line 5, the forget gate calculates a value between 0 and 1, which is then used in line 10 to determine which information from the previous cell state $C_{t-1}^{(l)}$ should be discarded. Line 6 represents the input gate, which decides which new information should be stored in the cell gate. In line 8, a new candidate value $\tilde{\text{input}}_t^{(l)}$ is generated. Both the input gate and candidate value contribute to the cell update in line 9.

The output gate, shown in line 7, determines which parts of the cell state to output. In line 10, the updated cell state $C_t^{(l)}$ is passed through a tanh layer, which scales its values

between -1 and 1. This result is then multiplied by $\text{output}_t^{(l)}$ to produce the final output $W_t^{(l)}$. Similar to the -H version, Step 2 utilizes the GCN layer to compute the next time step's embedding vector, $H_{t+1}^{(l)}$.

Both versions of the architecture can be modified to use other recurrent architectures if needed.

Chapter 4

Proposed Method

In this study, we propose a novel recommendation system model called Evolving LightGCN (ELGCN). The model integrates the RNN update mechanism of EvolveGCN into the LightGCN framework, known for its optimization in recommendation systems. This integration allows the model to capture temporal trends, enhancing its ability to reflect time-dependent patterns.

Research on recommendation systems that leverage temporal elements, such as the sequence and timing of user and item interactions, has been actively pursued (Choe et al., 2021; Wang and Han, 2021). Incorporating temporal dynamics into recommendation systems contributes to the effective detection of shilling attacks, where users intentionally input biased ratings to manipulate recommendation outcomes (Zhang et al., 2006; Zhou et al., 2018). Additionally, it improves the accuracy of Quality of Service (QoS) predictions, which are critical for consumers' service selection (Hu et al., 2015; Zhang et al., 2011).

4.1 ELGCN

The objective of ELGCN is to predict $y_{t+1,u,i}$ using the user-item interaction information from the previous t time steps. In other words, the goal is to estimate the probability that a user will select a specific item at the next time step $t+1$, thereby predicting whether an interaction will occur.

Figure 3 illustrates the architecture of ELGCN, which can be divided into three main components:

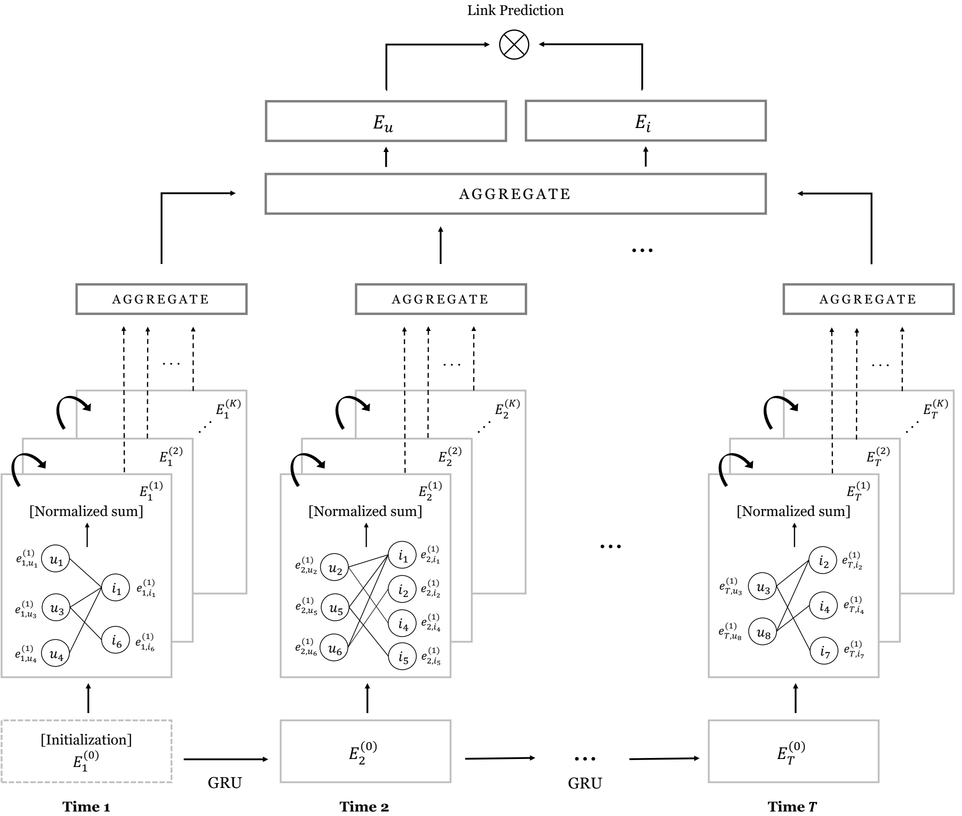


Figure 3: ELGCN architecture illustrating the temporal update process using GRU layers, followed by layer aggregation for final embeddings.

1) **Initial Embedding Layer**: This section incorporates temporal information by updating the embeddings using a GRU at each time point. Since we do not utilize node features, we determined that applying the EvolveGCN would be more suitable.

2) **Layer Propagation**: This component receives the initial embeddings and uses the LightGCN architecture to perform independent layer updates for each time step.

3) **Final Embedding Aggregation and Prediction**: Embeddings from all layers across all time steps are aggregated to generate the final embedding. This final embedding is then used to predict the link connection probability between users and items.

This is the only part of the ELGCN architecture where temporal information is incorporated. The initial embedding layer at the first time step is a trainable parameter matrix. If the node embedding size is D, an initial embedding matrix $E_1^{(0)}$ of size $D \times N$ is generated. This embedding matrix is updated for each time step according to Alg.3. This update process is inspired by the hidden state update mechanism of EvolveGCN. In the GRU, both the node embedding and hidden state are set identically as $E_{t-1}^{(0)}$. The matrix U and vector b are trainable parameters within the GRU. The bias matrix B, which is composed of the b vector, is set to be the same for all nodes.

Algorithm 3 ELGCN Initial Layer GRU Update

Input : Initial embedding matrix $E_{t-1}^{(0)}$
Output : Updated initial embedding matrix $E_t^{(0)}$

1: $Z_t = \sigma(U_z E_{t-1}^{(0)} + B_z), B_z = [b_z, \cdots, b_z]$
2: $R_t = \sigma(U_r E_{t-1}^{(0)} + B_r), B_r = [b_r, \cdots, b_r]$
3: $\tilde{H}_t = \tanh(U_h(R_t \odot E_{t-1}^{(0)}) + B_h), B_h = [b_h, \cdots, b_h]$
4: $E_t^{(0)} = (1 - Z_t) \odot E_{t-1}^{(0)} + Z_t \odot \tilde{H}_t$
5: **return** $E_t^{(0)}$

After updating the initial embeddings, layer updates are performed for each time step. The original layer update equation from LightGCN, shown in Eq. (3.3), is modified to incorporate temporal elements. This results in a new definition as shown in Eq. (4.1).

$$
\begin{aligned}
e_{t,u}^{(k+1)} &= \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} e_{t,i}^{(k)} \\
e_{t,i}^{(k+1)} &= \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} e_{t,u}^{(k)}
\end{aligned}
\tag{4.1}
$$

Let's examine the layer update process using Eq. (4.1) in more detail. Suppose there is a user-item interaction network at a specific time step, as illustrated in Figure 4.

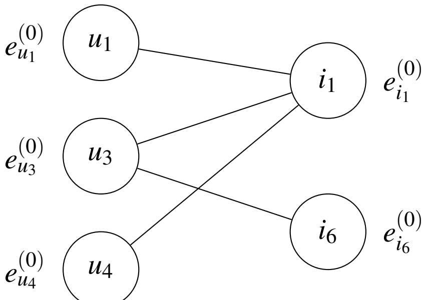


Figure 4: Example of the ELGCN layer update process, illustrating the initial embeddings for users (u_1, u_3, u_4) and items (i_1, i_6) at the 0th layer

Only the embedding vectors corresponding to the nodes at that specific time step are updated from the initial embedding matrix, defined as $E^{(0)} = \left[e_{u_1}^{(0)}, \cdots, e_{u_3}^{(0)}, e_{u_4}^{(0)}, \cdot, e_{i_1}^{(0)}, \cdots, e_{i_6}^{(0)}\right] \in \mathbb{R}^{N \times D}$. Let's take user 3 as an example to examine how its node embedding vector is updated. The neighboring nodes of user 3, i.e., the items with which user 3 has interacted, are item 1 and item 6. Thus, $|\mathcal{N}_{u_3}| = 2$. The number of neighbors for the adjacent items are $|\mathcal{N}_{i_1}| = 3$ and $|\mathcal{N}_{i_6}| = 1$, respectively. Thus, the calculation for updating the embedding layer from the initial embedding to the first embedding for user 3 at time t is as follows:

$$e_{t,u_3}^{(1)} = \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{3}} e_{i_1}^{(0)} + \frac{1}{\sqrt{1}} e_{i_6}^{(0)} \right).$$

Similarly, the embedding layer update for item 1 can be computed using the same approach.

$$e_{t,i_1}^{(1)} = \frac{1}{\sqrt{3}} \left(\frac{1}{\sqrt{1}} e_{u_1}^{(0)} + \frac{1}{\sqrt{2}} e_{u_3}^{(0)} + \frac{1}{\sqrt{1}} e_{u_4}^{(0)} \right)$$

In the layer update process described above, since the network structure remains the same across all K layers, updating each layer is equivalent to multiplying the initial embedding matrix by the normalized constants raised to the appropriate powers based on the number of neighboring nodes.

Once the layer update process for all time steps is completed, the final embedding

vector is generated. The final embedding vector is calculated as follows:

$$
\begin{aligned}
E_u &= \frac{1}{T(K+1)} \sum_{t=1}^{T} \sum_{k=0}^{K} e_{t,u}^{(k)} \\
E_i &= \frac{1}{T(K+1)} \sum_{t=1}^{T} \sum_{k=0}^{K} e_{t,i}^{(k)}
\end{aligned}
\tag{4.2}
$$

From Eq. (4.2), it can be seen that the final embedding vector is essentially the average of all the updated node embedding layers. By averaging over all embeddings, information from all time steps is evenly reflected.

The final recommendation score is computed as the dot product of the user final embedding vector and the item final embedding vector. The recommendation score is calculated as follows, which represents the probability of a link between the user and the item:

$$\hat{x}_{u,i} = E_u^T E_i.$$

4.2 Loss Function

For model training, we utilized the Bayesian Personalized Ranking (BPR) loss (Rendle et al., 2012). The BPR loss is a loss function specifically designed for learning personalized ranking in recommendation systems. It is based on the assumption that "items not consumed by a user are less preferred than items that have been consumed." The goal is to train the model to rank items that a user has interacted with higher than those they have not.

In other words, the model is trained to assign higher scores to items that users have previously interacted with compared to items they have not, thereby directly learning the user's preference ranking. The BPR loss function used to train ELGCN is defined in Eq. (4.3).

$$L_{BPR} = -\sum_{u=1}^{U} \sum_{i \in \mathcal{N}_u} \sum_{j \notin \mathcal{N}_u} \log \sigma(\hat{x}_{u,i} - \hat{x}_{u,j}) \quad (4.3)$$

The BPR loss operates by comparing items in pairs. It selects a positively interacted item i (positive item) and a non-interacted item j (negative item) for a given user. The model is trained to predict a higher score $\hat{x}_{u,i}$ for item i than for item j, i.e., $\hat{x}_{u,j}$. In this process, negative sampling is used to select item j.

The score calculation for the BPR loss is as follows. The function σ represents the sigmoid function, which maps the difference between the positive item's score and the negative item's score to a value between 0 and 1. For the model to achieve a higher score, the difference between $\hat{x}_{u,i}$ and $\hat{x}_{u,j}$ needs to be large. Thus, the model is trained to increase $\hat{x}_{u,i}$ while decreasing $\hat{x}_{u,j}$.

To optimize the model, a logarithm is applied to adjust the score range to be between $-\infty$ and 0, and a negative sign is added for gradient descent. As a result, the model is trained to maximize the difference between the two scores, thereby minimizing the overall loss.

Chapter 5

Experiments

5.1 Dataset

In this study, we used the Netflix Prize dataset (Bennett et al., 2007) and the MovieLens 25M dataset (Harper and Konstan, 2015), which are commonly employed for evaluating the performance of recommendation systems.

The Netflix Prize dataset consists of approximately 100,480,500 ratings given by 480,189 users on 17,770 movies. It includes the variables 'userid', 'movieid', 'timestamp', and 'rating', with ratings on a 5-point scale ranging from 1 to 5. The MovieLens 25M dataset comprises about 25,000,000 ratings provided by 162,541 users on 62,423 movies. It includes the variables 'userId', 'movieId', 'rating', and 'timestamp', with ratings also measured on a 5-point scale from 1 to 5.

We applied a common preprocessing procedure to both datasets. First, we removed any rows where at least one of the four variables had missing values. Then, to ensure meaningful analysis, we filtered for users and items with a minimum of 70 interactions.

From the Netflix Prize dataset, we extracted 600 users and 600 movies with the highest interaction counts. Similarly, from the MovieLens dataset, we selected 700 users and 700 movies. After extracting the data, we sorted it in chronological order and arbitrarily divided it into 15 timepoints. The time window was set to 5, with a train/validation/test split ratio of 5:2:3.

The time window of 5 indicates that each training session uses interactions from the previous 5 timepoints to predict the next one. This approach involves using data from the

previous 5 timepoints to predict interactions at the subsequent timepoint.

5.2 Comparative Models

In this study, we compare the performance of a total of three models in the context of a recommendation system: GCN, LightGCN, and EvolveGCN, as previously described. Among these, only EvolveGCN inherently incorporates temporal updates. For the other two models, we adjusted the training process to incorporate temporal elements. Instead of training GCN and LightGCN on the entire dataset at once, we sorted the data by timestep and constructed models using a set of data corresponding to a specific time window. The performance evaluation was conducted by predicting the subsequent time step using these models. The key difference from the original model construction is that the data for each time step remains independent, and no temporal update mechanism is applied as time progresses.

5.3 Performance Metrics

In this study, we compare the model performance using two metrics commonly employed in recommendation systems.

The first metric is the Recall@K measure, which is inspired by the Recall metric used in evaluating classification models. Recall, also known as sensitivity or True Positive Rate (TPR), is used to assess the performance of general classification models. It is calculated as follows:

$$Recall = \frac{TP}{TP+FN} \tag{5.1}$$

In Eq. (5.1), T denotes cases where the prediction is correct, while F represents cases where the prediction is incorrect. P indicates that the model predicted a positive outcome, and N indicates that the model predicted a negative outcome. Thus, TP stands for True Pos-

itive, which refers to instances where the model predicted positive and the actual outcome was also positive. On the other hand, FN stands for False Negative, which refers to instances where the model predicted negative but the actual outcome was positive. In summary, Eq. (5.1) represents the proportion of actual positives that were correctly predicted as positive by the model.

Based on this idea, Recall@K is defined as follows, where K represents the number of items recommended by the model.

$$Recall@K = \frac{\text{relevant recommended items}}{\text{all the possible relevant items}} \tag{5.2}$$

As shown in Eq. (5.2), $Recall@K$ calculates the recall for the top K recommended items. It represents the proportion of items that the user is actually interested in among all items, which are correctly included in the top K items recommended by the model. For example, suppose a user is interested in 6 items and we set $K = 5$. Let's assume there are a total of items labeled as A, B, $\cdots$, J. The user is interested in items A, B, C, H, I, J, while the model recommends items A, B, C, D, E. In this case, $Recall@5$ is calculated as follows: the denominator is the total number of items that the user is interested in 6, and the numerator is the number of items that the model correctly recommended among those of interest to the user 3. Thus, $Recall@5$ = 3/6 = 0.5.

The second metric is Normalized Discounted Cumulative Gain (NDCG). NDCG is designed to evaluate recommendation performance more precisely by taking into account the ranking order of the items recommended by the model.

To understand NDCG, it is important to first explain some preliminary concepts. Cumulative Gain (CG) is defined as the sum of relevance scores. The relevance score indicates how much a user prefers a particular item. In this study, the relevance score is defined as 1 if the recommended item has had an interaction with the user, and 0 otherwise. Thus, $CG@K$

is defined as follows:

$$CG@K = \sum_{k=1^K} rel_{i_k} \tag{5.3}$$

In Eq. (5.3), rel_{i_k} represents the relevance score of the kth recommended item i. Thus, $CG@K$ is the sum of the items recommended by the model that have had interactions with the user.

Next, we introduce Discounted Cumulative Gain (DCG), which takes the ranking order of recommendations into account. $DCG@K$ is defined as follows:

$$DCG@K = \sum_{k=1}^{K} \frac{rel_{i_k}}{\log_2(k+1)} \tag{5.4}$$

As the position of the recommended item increases, the denominator $\log_2(k+1)$ grows, reducing the contribution of that item's relevance score. In other words, it applies log normalization to the relevance score, giving lower weight to items recommended later in the list. Finally, the Ideal Discounted Cumulative Gain (IDCG) represents the score when items are recommended to the user in the optimal order. By ranking the items based on their relevance scores in descending order, IDCG assumes an ideal scenario where the model recommends items in the best possible sequence. Thus, it reflects the highest possible DCG value that can be achieved for a given set of K recommendations.

$IDCG@K$ is defined as follows:

$$IDCG@K = \sum_{k=1}^{K} \frac{rel_{i_k}^{opt}}{\log_2(k+1)} \tag{5.5}$$

Here, $rel_{i_k}^{opt}$ denotes the relevance scores sorted in descending order for the top K results.

Combining the above concepts, $NDCG@K$ is defined as the ratio of $DCG@K$ to

$IDCG@K$:

$$NDCG@K = \frac{DCG@K}{IDCG@K} \tag{5.6}$$

This metric measures how close the model's recommendation ranking is to the optimal ranking, with values ranging from 0 to 1. A higher $NDCG@K$ score indicates that the model's recommendations are closer to the ideal order.

5.4 Experiment setting

The factors and hyperparameters used during model training are as follows. The embedding dimension was set to 16, and the LightGCN model was built with 5 layers. The batch size was set to 512, and the learning rate was set to 0.001, using the Adam optimizer. The maximum number of epochs was set to 100, with an early stopping rule applied to terminate training if the validation loss did not decrease for 10 consecutive epochs. Upon completion of training, the model with the lowest validation loss was saved and used for model testing.

We calculated the two measures, $Recall@K$ and $NDCG@K$, from Section 5.3 for K values of 10, 20, and 30. These metrics were used to compare the recommendation performance of the models discussed in Section 5.2, namely GCN, LightGCN, EvolveGCN, and the proposed ELGCN.

5.5 Results

The results presented in Table 5 demonstrate that the proposed ELGCN model consistently outperforms other models in terms of both Recall and NDCG metrics across the Netflix Prize and MovieLens 25M datasets.

For the Netflix Prize dataset, ELGCN achieved the highest performance at all evaluated values of K, particularly excelling in $Recall@30$ with a score of 0.2295, which is

Data	Method	Performance metric					
		Recall			NDCG		
		K=10	K=20	K=30	K=10	K=20	K=30
Netflix Prize	GCN-T	<u>0.0625</u>	<u>0.0992</u>	<u>0.1468</u>	0.0581	0.0717	<u>0.0893</u>
	LightGCN-T	0.0513	0.0563	0.0813	0.0487	0.0684	0.0614
	EvolveGCN-H	0.0378	0.0519	0.0678	**0.0722**	<u>0.0733</u>	0.0815
	ELGCN	**0.0804**	**0.1514**	**0.2295**	<u>0.0694</u>	**0.098**	**0.126**
MovieLens 25M	GCN-T	<u>0.0497</u>	<u>0.0519</u>	<u>0.0556</u>	<u>0.0503</u>	0.0501	0.0510
	LightGCN-T	0.0440	0.0460	0.0475	0.0440	0.0447	0.0448
	EvolveGCN-H	0.0115	0.0332	0.0456	**0.0906**	**0.0931**	**0.1097**
	ELGCN	**0.0502**	**0.0527**	**0.0598**	0.0500	<u>0.0502</u>	<u>0.0516</u>

Table 5: *Recall@K*, *NDCG@K* ($K = 10, 20, 30$) scores for the Netflix Prize and MovieLens 25M datasets. The best performance is highlighted in **bold**, and the second-best performance is <u>underlined</u>.

substantially higher than the competing models. Additionally, ELGCN attained the highest NDCG score of 0.1260 at $K = 30$, showcasing its ability to effectively recommend items that align with user preferences. In comparison, while the LightGCN-T model showed relatively strong performance, particularly in $Recall@20$ and $Recall@30$, it was consistently outperformed by ELGCN. The GCN-T and EvolveGCN-H models exhibited notably lower scores for both Recall and NDCG on the Netflix Prize dataset, indicating that these models were less effective in capturing the temporal dynamics of user-item interactions.

On the MovieLens 25M dataset, ELGCN continued to demonstrate superior performance in terms of Recall, achieving the highest scores across all K values. For example, it achieved a $Recall@30$ of 0.0598, indicating its effectiveness in identifying relevant items for users. However, in terms of NDCG, the EvolveGCN-H model slightly outperformed ELGCN at all evaluated values of K. This suggests that while ELGCN excels in identifying items that users are likely to interact with, EvolveGCN-H may have a slight edge in optimizing the ranking of recommendations.

Overall, the results highlight the strength of ELGCN in capturing temporal patterns in user behavior, leading to higher recall performance. The integration of temporal updates

using GRU layers appears to significantly enhance the model's ability to adapt to changes in user preferences over time. While EvolveGCN-H demonstrated marginally better NDCG performance on the MovieLens dataset, ELGCN's consistently superior recall scores across both datasets underscore its robustness in delivering accurate recommendations. This indicates that ELGCN is particularly well-suited for recommendation scenarios where capturing time-evolving user-item interactions is critical.

Chapter 6

Conclusion

In this study, we introduced ELGCN to incorporate temporal dynamics in recommendation systems. Traditional static models like LightGCN are effective in capturing user-item interactions at a single point in time but fail to consider the temporal evolution of user preferences. By integrating temporal updates through GRU layers, ELGCN addresses this limitation, enabling it to adapt to changes in user interests over time.

Our empirical evaluation utilized two benchmark datasets, the Netflix Prize and MovieLens 25M datasets, to compare the performance of ELGCN against existing models, including GCN, LightGCN, and EvolveGCN. The results demonstrated that ELGCN consistently outperformed other models in terms of Recall across different values of K, especially excelling in $Recall@30$ on the Netflix Prize dataset with a score of 0.2295. This indicates that ELGCN is highly effective in identifying relevant items for users, even as their preferences evolve. Although EvolveGCN-H showed slightly better performance in NDCG on the MovieLens dataset, ELGCN achieved consistently superior recall scores, highlighting its strength in capturing time-sensitive user-item interactions.

However, our study also has certain limitations that suggest avenues for future improvements. The current implementation of ELGCN relies on a fixed GRU-based update mechanism, which, while effective for capturing sequential patterns, may not be sufficient to model the more complex and nuanced temporal dependencies present in real-world data. This limitation indicates that there is room for enhancing the model's capacity to adapt to variations in user behavior over time.

Additionally, the current approach does not yet incorporate contextual information

such as user profiles, item attributes, or external factors like trends and events that could significantly impact user preferences. These factors can provide richer insights into user behavior and help tailor recommendations more precisely.

To address these limitations, we plan to integrate an attention mechanism into the EL-GCN architecture to enable data-adaptive prediction. By leveraging an attention-based approach, the model can dynamically focus on the most relevant user-item interactions and historical patterns that are critical for predicting future behavior. This would allow the model to weigh different time steps and interactions based on their importance, rather than treating all past interactions equally. Such a data-adaptive method is expected to improve the accuracy and robustness of predictions, especially in scenarios where user preferences shift rapidly or are influenced by external factors.

References

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

Bennett, J., Lanning, S., et al. (2007). The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35. New York.

Bondy, J. A., Murty, U. S. R., et al. (1976). *Graph theory with applications*, volume 290. Macmillan London.

Buchanan, B. G. and Shortliffe, E. H. (1984). *Rule based expert systems: the mycin experiments of the stanford heuristic programming project (the Addison-Wesley series in artificial intelligence)*. Addison-Wesley Longman Publishing Co., Inc.

Campbell-Kelly, M. and Garcia-Swartz, D. D. (2013). The history of the internet: the missing narratives. *Journal of Information Technology*, 28(1):18–33.

Cangea, C., Veličković, P., Jovanović, N., Kipf, T., and Liò, P. (2018). Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*.

Cerf, V. G. (2004). On the evolution of internet technologies. *Proceedings of the IEEE*, 92(9):1360–1370.

Cho, K. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Choe, B., Kang, T., and Jung, K. (2021). Recommendation system with hierarchical recurrent neural network for long-term time series. *IEEE Access*, 9:72033–72039.

Covington, P., Adams, J., and Sargin, E. (2016). Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*, pages 191–198.

Grosan, C., Abraham, A., Grosan, C., and Abraham, A. (2011). Rule-based expert systems. *Intelligent systems: A modern approach*, pages 149–185.

Harper, F. M. and Konstan, J. A. (2015). The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19.

Hayes-Roth, F. (1985). Rule-based systems. *Communications of the ACM*, 28(9):921–932.

He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., and Wang, M. (2020). Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 639–648.

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. (2017). Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182.

Hechtlinger, Y., Chakravarti, P., and Qin, J. (2016). Convolutional neural networks generalization utilizing the data graph structure.

Hochreiter, S. (1997). Long short-term memory. *Neural Computation MIT-Press*.

Hu, Y., Peng, Q., Hu, X., and Yang, R. (2015). Web service recommendation based on time series forecasting and collaborative filtering. In *2015 ieee international conference on web services*, pages 233–240. IEEE.

Kim, H., Lee, B. S., Shin, W.-Y., and Lim, S. (2022). Graph anomaly detection with graph neural networks: Current status and challenges. *IEEE Access*, 10:111820–111829.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Leiner, B. M., Cerf, V. G., Clark, D. D., Kahn, R. E., Kleinrock, L., Lynch, D. C., Postel, J., Roberts, L. G., and Wolff, S. (2009). A brief history of the internet. *ACM SIGCOMM computer communication review*, 39(5):22–31.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. (2017). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*.

Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80.

Pareja, A., Domeniconi, G., Chen, J., Ma, T., Suzumura, T., Kanezashi, H., Kaler, T., Schardl, T., and Leiserson, C. (2020). Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 5363–5370.

Rendle, S., Freudenthaler, C., Gantner, Z., and Schmidt-Thieme, L. (2012). Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*.

Ricci, F., Rokach, L., and Shapira, B. (2021). Recommender systems: Techniques, applications, and challenges. *Recommender systems handbook*, pages 1–35.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001). Item-based collaborative filtering

recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295.

Voulodimos, A., Doulamis, N., Doulamis, A., and Protopapadakis, E. (2018). Deep learning for computer vision: A brief review. *Computational intelligence and neuroscience*, 2018(1):7068349.

Wang, X., He, X., Wang, M., Feng, F., and Chua, T.-S. (2019). Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174.

Wang, Y. and Han, L. (2021). Adaptive time series prediction and recommendation. *Information Processing & Management*, 58(3):102494.

Wu, S., Tang, Y., Zhu, Y., Wang, L., Xie, X., and Tan, T. (2019). Session-based recommendation with graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 346–353.

Wu, Y., Dai, H.-N., and Tang, H. (2021). Graph neural networks for anomaly detection in industrial internet of things. *IEEE Internet of Things Journal*, 9(12):9214–9231.

Ying, R., He, R., Chen, K., Eksombatchai, P., Hamilton, W. L., and Leskovec, J. (2018). Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983.

Ying, Z., Bourgeois, D., You, J., Zitnik, M., and Leskovec, J. (2019). Gnnexplainer: Generating explanations for graph neural networks. *Advances in neural information processing systems*, 32.

Yu, B., Yin, H., and Zhu, Z. (2017). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*.

Yuan, H., Yu, H., Gui, S., and Ji, S. (2022). Explainability in graph neural networks: A taxonomic survey. *IEEE transactions on pattern analysis and machine intelligence*, 45(5):5782–5799.

Zhang, Q., Chang, J., Meng, G., Xu, S., Xiang, S., and Pan, C. (2019a). Learning graph structure via graph convolutional networks. *Pattern Recognition*, 95:308–318.

Zhang, S., Chakrabarti, A., Ford, J., and Makedon, F. (2006). Attack detection in time series for recommender systems. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 809–814.

Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019b). Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)*, 52(1):1–38.

Zhang, Y., Zheng, Z., and Lyu, M. R. (2011). Wspred: A time-aware personalized qos prediction framework for web services. In *2011 IEEE 22nd international symposium on software reliability engineering*, pages 210–219. IEEE.

Zhou, J., Cui, G., Hu, S., Zhang, Z., Yang, C., Liu, Z., Wang, L., Li, C., and Sun, M. (2020). Graph neural networks: A review of methods and applications. *AI open*, 1:57–81.

Zhou, W., Wen, J., Qu, Q., Zeng, J., and Cheng, T. (2018). Shilling attack detection for recommender systems based on credibility of group users and rating time series. *PloS one*, 13(5):e0196533.

국문 초록

시계열 그래프 신경망에 기반한 추천시스템에 관한 연구

성신여자대학교
대학원
통계학과
이수지

추천 시스템에 최적화된 GCN 모형인 LightGCN이 개발되었으나, 시계열 GCN에 기반한 추천 시스템 특화 모형은 아직 개발되지 않았다. LightGCN은 단일 시점의 user-item 상호작용만을 활용하여 추천을 수행한다. 이에 본 연구에서는 LightGCN에 시계열 업데이트 메커니즘인 GRU를 통합한 Evolving LightGCN (ELGCN)을 제안한다. ELGCN에서는 GRU를 이용해 초기 임베딩 레이어를 시간에 따라 업데이트하고, 각 시점별 레이어는 LightGCN 아키텍처를 통해 추가적으로 업데이트한다. 최종 임베딩은 모든 시점의 임베딩 레이어 평균을 이용해 계산하며, 사용자와 아이템의 최종 임베딩 벡터의 내적을 통해 에지 연결 확률을 예측한다. 제안된 모형은 MovieLens 및 Netflix Prize 데이터셋을 활용하여 평가되었으며, 시계열 버전의 GCN, LightGCN, EvolveGCN 모형과 비교 연구를 수행하였다. 실험 결과, 제안된 모형이 두 데이터셋 모두에서 일관되게 높은 성능을 보이는 것으로 확인되었다.

핵심용어 : 추천 시스템, 그래프 신경망, 시계열 모형, 사용자-아이템 상호작용